Easy Tourney

Application de gestion et planification de tournois multisports

Travail De Master

Julien Richoz

Février 2025

Supervisé par:

Prof. Dr. Jacques Pasquier-Rocha
Software Engineering Group

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Software Engineering Group
Department of Informatics
University of Fribourg (Switzerland)

Software Engineering Group

Remerciements

Je tiens à exprimer ma profonde gratitude au Prof. Dr. Jacques Pasquier-Rocha pour sa supervision tout au long de cette dernière année. Sa confiance en mon autonomie, ainsi que sa compréhension de ma situation malgré ses nombreuses responsabilités, ont été essentielles pour mener à bien ce projet. Je lui souhaite une retraite bien méritée, riche en sérénité et en découvertes.
Je remercie également chaleureusement Jan Christophe Lehmann, Maître de Sport du Service de Sport Universitaire (SSU) de Fribourg. Ses questions pertinentes et son intérêt sincère pour mon projet ont été une source de motivation précieuse, m'incitant à approfondir et structurer des fonctionnalités clés, telles que l'implémentation des *design patterns Strategy*.
Enfin, je suis profondément reconnaissant envers mes proches et mon entourage, qui m'ont soutenu moralement et parfois nourri durant les longues heures de travail. Un merci tout particulier à Andreas Ruppen pour le template LaTeX, qui a rendu la rédaction de ce mémoire nettement moins intimidante.

Résumé

Easy Tourney est une application web dédiée à la gestion et à la planification de tournois sportifs dans les environnements scolaires et universitaires. Développée avec **Vue.js**, **Node.js**, **Express.js** et **MySQL** via **Sequelize**, elle suit le design pattern architectural **MVC** (Modèle-Vue-Contrôleur).
Face à la complexité de l'organisation de tournois éducatifs impliquant des centaines de joueurs, Easy Tourney propose une solution complète. Elle simplifie la gestion des inscriptions, des équipes, des plannings de matchs et du suivi des résultats grâce à des fonctionnalités avancées. L'application offre une création personnalisée de tournois, une gestion autonome des inscriptions via des liens d'invitation, des algorithmes de planification automatique et d'assignation des joueurs. Elle permet également l'exportation des plannings au format Excel et offre une visualisation en temps réel des scores sur une interface responsive. Actuellement, un seul type de tournoi est supporté, avec des extensions prévues pour divers formats dans les futures versions. En résumé, *Easy Tourney* offre une solution pratique permettant aux utilisateurs de gagner du temps et de se concentrer sur le succès des tournois sportifs.
Mots clés : *Tournoi sportif, Planification automatique, Vue.js, Node.js, Express.js, MySQL, MVC, Inscription en ligne, ORM Sequelize, Application web, Easy Tourney*

Table des matières

Liste des Figures

1
Introduction

1.1. Motivations et Objectifs

À 31 ans, après plusieurs années en informatique, j'ai choisi de me réorienter professionnellement pour devenir professeur de sport. Mon parcours inclut des expériences variées, telles que le soutien en soins intensifs pour l'armée et des stages auprès de personnes âgées, qui m'ont permis de développer une forte appréciation des contacts humains et de l'impact positif que l'on peut avoir sur les autres. Cette reconversion reflète mon désir de combiner mes compétences en informatique avec ma passion pour le sport et le développement personnel des jeunes.

Durant ma convalescence à la suite d'une blessure au tendon d'Achille, j'ai pu concrétiser un projet associant mes deux domaines d'intérêt : *Easy Tourney*, une application destinée à simplifier la gestion des tournois multisports dans un contexte scolaire. Ce projet répond à des défis réels rencontrés par les enseignants, tels que la planification des matchs pour des centaines d'élèves, l'organisation des terrains ou encore le suivi des inscriptions. En créant une solution intuitive et pratique, mon objectif est d'alléger leur charge de travail et d'améliorer leur expérience dans l'organisation de ces événements.

Ce projet est également l'occasion de mettre en pratique les notions apprises durant mes études à l'université de Fribourg. Par exemple, la gestion de la planification des matchs repose sur le *Strategy Pattern*, un concept enseigné en cours de Génie Logiciel, qui a permis de développer une solution modulaire et adaptable aux différents types de tournois. De plus, l'architecture de l'application suit le modèle *MVC* (Modèle-Vue-Contrôleur), garantissant une séparation claire entre la logique métier, la présentation et la gestion des données. Ces choix conceptuels reflètent mon engagement envers des solutions robustes et évolutives, tout en répondant aux besoins pratiques des utilisateurs. *Easy Tourney* incarne ainsi une synthèse des compétences techniques et des connaissances méthodologiques acquises tout au long de mon parcours académique et professionnel.

Origine du nom Easy Tourney

Le choix du nom *Easy Tourney* reflète l'objectif principal du projet : simplifier la gestion des tournois sportifs (tourney, abréviation anglophone de *tournament*). Ce nom exprime également la volonté de proposer une application intuitive et accessible, adaptée aux besoins des enseignants.

1.2. Structure du Rapport

Le présent rapport est structuré en cinq chapitres principaux :

Chapitre 1 : Introduction

Ce chapitre présente les motivations ayant conduit au choix de ce projet, les objectifs poursuivis, ainsi que la structure générale du rapport.

Chapitre 2 : Analyse, Fonctionnalités et Modélisation

Ce chapitre explore le contexte du projet, analyse les besoins des utilisateurs et présente les principales fonctionnalités de l'application.

Chapitre 3 : Point de vue utilisateur

Ce chapitre illustre l'expérience utilisateur à travers divers scénarios, en détaillant les rôles principaux (Admin, Assistant, Joueur, Invité) ainsi que les fonctionnalités associées à la création et à la gestion d'un tournoi.

Chapitre 4 : Point de vue développeur

Ce chapitre examine les choix technologiques effectués, les défis rencontrés et les solutions mises en œuvre pour les surmonter. Une attention particulière est portée aux aspects techniques tels que l'authentification, le pattern *Strategy* pour la génération de plannings, et les bonnes pratiques de développement.

Chapitre 5 : Conclusion

La conclusion résume les contributions du projet, ses impacts potentiels et propose des perspectives pour son amélioration future.

1.3. Conventions et Notations

- **Langue et style** : Le rapport est rédigé en français, avec des anglicismes liés à l'informatique lorsque cela est pertinent. Le code est écrit en anglais, accompagné de commentaires en français dans la plupart des cas.

- **Philosophie open-source** : L'intégralité du projet est disponible en open-source à l'adresse suivante : `https://github.com/JulienRichoz/easy-tourney/tree/0.8.1-thesis`. Le tag `0.8.1-thesis` correspond à la version de l'application utilisée lors de l'écriture de ce document.
- **Extraits de code** : Les exemples de code sont présentés à l'aide de l'environnement `listings`, offrant une mise en page claire et structurée.
  Les références aux fichiers sources apparaissent sous forme de notes de bas de page, avec un numéro violet redirigeant vers le code exact versionné sur GitHub.
- **Figures et tableaux** : Toutes les figures, tableaux et extraits de code sont numérotés par chapitre (par exemple, *Figure 2.1* pour la première figure du chapitre 2).
- **Rédaction non-genrée** : Bien que le masculin soit utilisé par souci de simplicité, il inclut toutes les identités de genre.

2

Analyse, Fonctionnalités et Modélisation

2.1. Contexte

Ce chapitre présente l'origine et les objectifs du projet, analyse des solutions ou plateformes existantes, et présente les choix technologiques retenus. Il se termine par des diagrammes d'utilisation représentant les fonctionnalités principales ainsi que la modélisation de la base de données.

Des maquettes, réalisées lors de la phase d'analyse, sont également présentées. Ces dernières ont servi de guide principal pour le développement de l'interface utilisateur. Elles assurent une cohérence entre l'idée initiale et l'implémentation finale, illustrant comment les concepts envisagés ont été traduits en fonctionnalités concrètes tout en tenant compte des contraintes techniques et des retours d'expérience.

2.2. Origine et Objectifs

2.2.1. Origine du projet

Ce projet s'inscrit dans un master en informatique dans le cadre du DEEM (Diplôme d'enseignement pour les écoles de maturité), qui constitue ma branche secondaire. Il vise à répondre à un besoin concret exprimé par les enseignants de sport, souvent confrontés à des défis liés à la gestion et à la planification des tournois multisports.

L'idée a émergé grâce aux retours de plusieurs amis enseignants et a été renforcée par l'intérêt direct d'un organisateur de l'Unisport, Mr. Jan Christophe Lehmann, qui a validé la pertinence et l'utilité des fonctionnalités proposées. L'objectif principal du projet est de développer une application intuitive permettant d'automatiser les tâches complexes tout en offrant une flexibilité totale aux utilisateurs.

2.2.2. Objectifs principaux

Les principaux objectifs du projet sont les suivants :

- Mettre en place une gestion complète des entités principales (*CRUD* : Create, Read, Update, Delete) telles que les sports, utilisateurs, tournois, terrains, groupes, pools, matchs et plannings.
- Implémenter un système d'inscription basé sur des tokens d'invitation, avec des fonctionnalités d'expiration et de validation.
- Automatiser l'assignation des utilisateurs sans groupe à des équipes pour simplifier la gestion.
- Générer automatiquement les plannings de pools et de matchs à l'aide d'algorithmes efficaces.

- Offrir la possibilité de modifier manuellement toutes les entités et les plannings pour garantir une flexibilité maximale.
- Permettre aux administrateurs et assistants d'arbitrer et commenter les matchs, avec une visualisation des scores en temps réel et un système d'historique.
- Proposer aux utilisateurs une interface claire pour consulter le planning, les matchs, les scores et les informations relatives au tournoi.

2.3. Analyse Comparative des Applications Existantes

Différents sites ont été analysés. Cependant, un problème est rapidement apparu : la plupart des sites sont payants, soit complexe à tester car il est nécessaire d'avoir des utilisateurs pour créer les matchs et utiliser les diverses fonctionnalités. Deux sites ont toutefois été retenus:

2.3.1. Tournify

Tournify [1] est un planificateur de tournoi en ligne conçu pour des événements monosports. Il offre une interface intuitive, des options de personnalisation avancées, et des outils de présentation publique, mais présente des limites dans le cadre d'une gestion multisport ou scolaire.

Interface et navigation

Tournify propose une interface épurée et intuitive, bien adaptée aux tournois simples. Cependant, la multiplicité des menus et sous-menus peut compliquer les configurations avancées (Fig. 2.1).

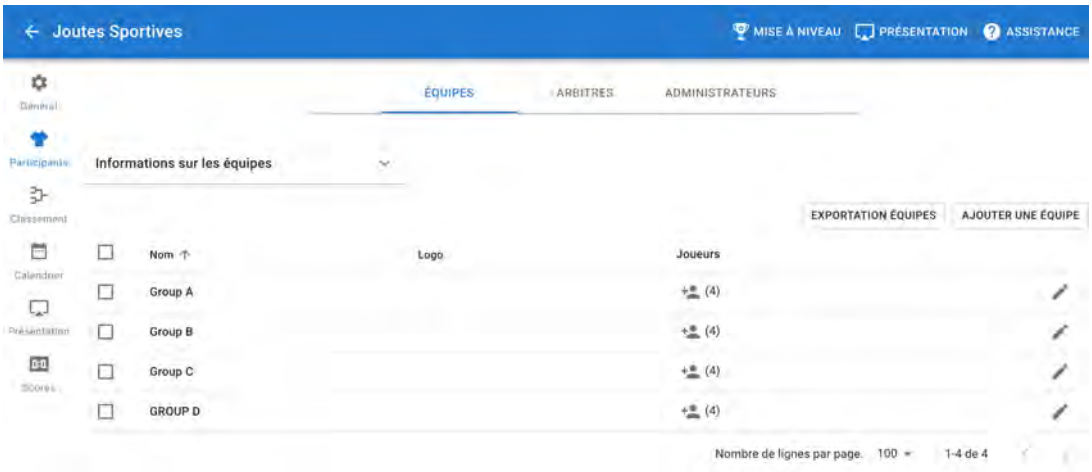


Fig. 2.1.: Interface de Tournify

Points positifs : Navigation claire et interface légère.
Points négatifs : Menus imbriqués, complexité pour les configurations avancées.
À retenir : Simplifier la structure des menus dans Easy Tourney pour éviter une surcharge d'options.

Système de calendrier

Le calendrier de Tournify permet une planification interactive via *drag and drop* et la configuration des terrains et horaires. Cependant, l'absence d'automatisation rend la gestion manuelle des plannings contraignante pour des événements complexes (Fig. 2.2).

Fig. 2.2.: Calendrier de Tournify

Points positifs : Flexibilité dans la configuration, interface interactive.
Points négatifs : Absence d'automatisation pour les grands tournois.
À retenir : Planification automatique avec flexibilité manuelle.

Personnalisation et partage

Tournify se distingue par des fonctionnalités avancées de personnalisation pour la présentation publique des tournois, permettant de générer des liens personnalisés et d'adapter l'affichage selon le support (Fig. 2.3).

Fig. 2.3.: Configuration avancée pour le partage de tournoi

Points positifs : Système de partage très complet et personnalisable.
Points négatifs : Trop orienté vers la présentation publique.
À retenir : Privilégier des fonctionnalités internes adaptées aux besoins des utilisateurs.

Synthèse

Tournify est une solution solide pour les tournois monosports, mais son orientation vers la présentation publique et l'absence d'automatisation limitent son usage pour des événements multisports ou scolaires. Contrairement à *Easy Tourney*, Tournify met l'accent sur la visibilité externe, tandis que mon projet se concentre sur l'optimisation des processus internes. Les éléments à retenir sont une **navigation simple**, intégrer une **planification** automatique adaptée aux besoins multisports et se concentrer sur des **outils internes pratiques**, au lieu d'une présentation publique avancée.

2.3.2. Challenge Place

Challenge Place [2] est une application dédiée à l'organisation de tournois sportifs. Elle propose des fonctionnalités intéressantes comme un système d'arbitrage efficace et un historique d'événements, qui ont inspiré certaines parties de mon projet.

Interface d'Arbitrage

L'interface d'arbitrage (Fig. 2.4) est bien pensée et permet une gestion efficace des matchs en cours. Elle inclut :

- Un **timer** en temps réel pour suivre le déroulement du match.
- Une visualisation claire et intuitive des équipes, scores et sets.

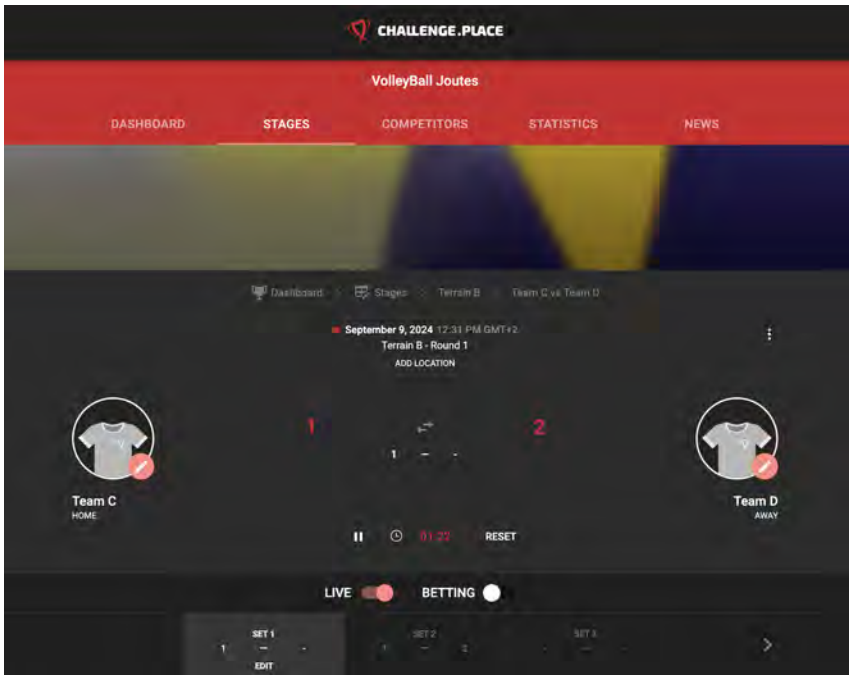


Fig. 2.4.: Exemple d'interface d'arbitrage

Points positifs : Interface Utilisateur (UI) intuitive, suivi en temps réel, gestion des scores et sets intégrée.
Points négatifs : Navigation lente pour passer d'un match à l'autre.
À retenir : Adapter l'interface pour simplifier la navigation tout en conservant la gestion en temps réel.

Historique d'Événements

Challenge Place inclut une timeline claire (Fig. 2.5) pour visualiser les événements d'un match, avec des options pour ajouter des actions comme fautes, points, ou remplacements via des modals intuitives.

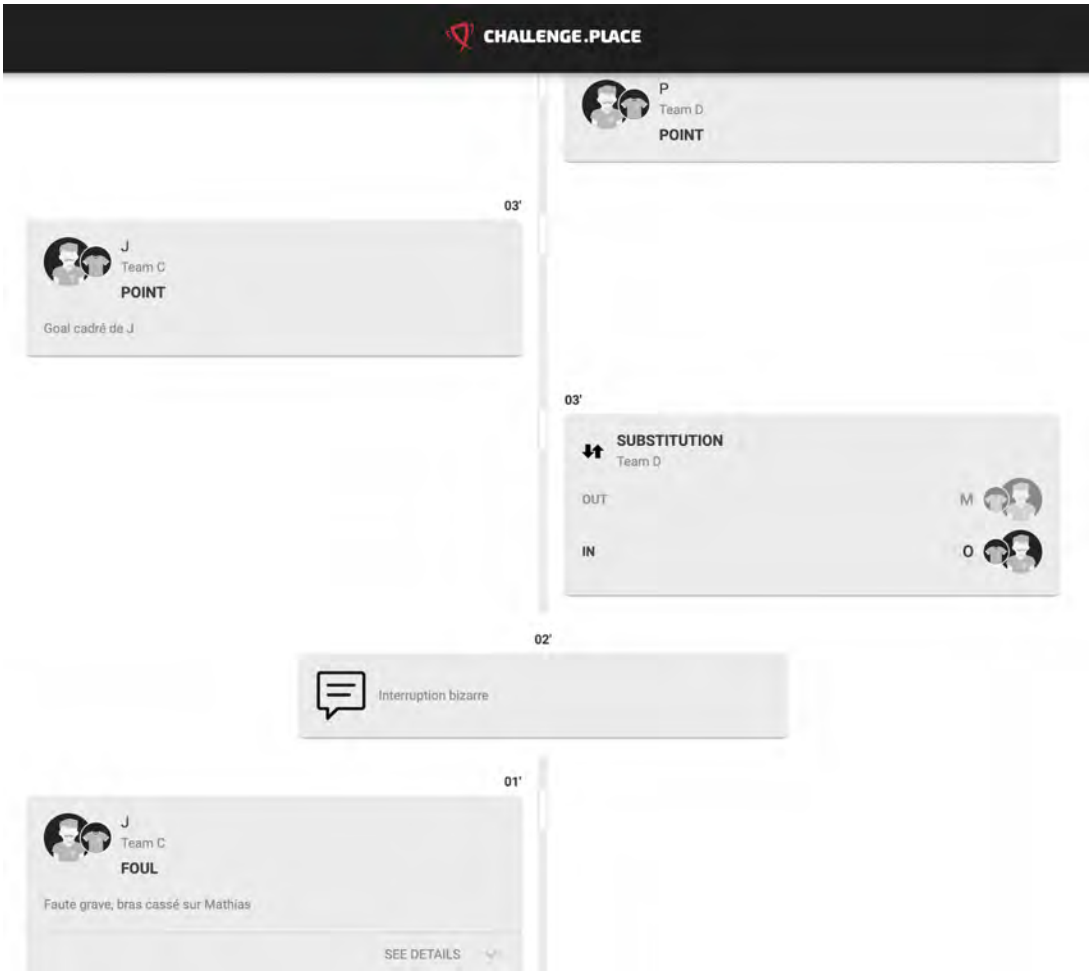


Fig. 2.5.: Historique des événements d'un match

Points positifs : Timeline élégante, actions variées, modals intuitives.
Points négatifs : Besoin d'une simplification pour un usage scolaire.
À retenir : Intégrer un système de timeline simplifié, adapté aux besoins des enseignants.

Guidage utilisateur et vue générale

Challenge Place offre un guidage utilisateur via des modals (Fig. 2.6) et une vue d'ensemble claire des tournois (Fig. 2.7). Ces fonctionnalités améliorent l'expérience utilisateur et simplifient l'administration.

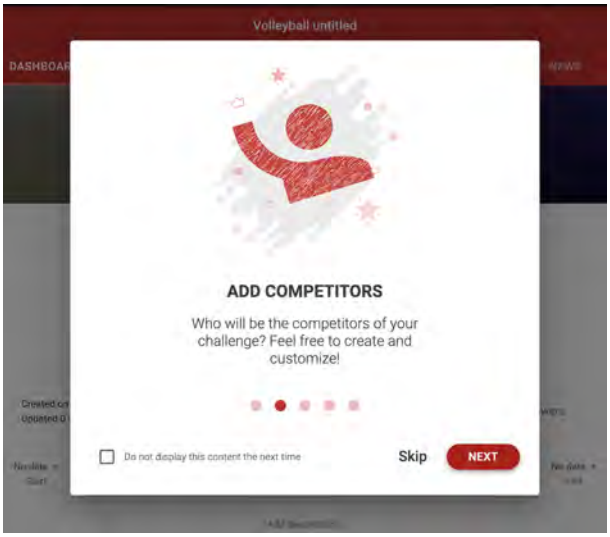


Fig. 2.6.: Modal pour guider l'utilisateur lors de la création d'un tournoi

Fig. 2.7.: Vue d'ensemble des états des tournois

Points positifs : Guidage clair et vue récapitulative utile.
Points négatifs : Complexité inutile pour un usage interne.
À retenir : Proposer un guidage léger pour la création des tournois.

Conclusion sur Challenge Place

Challenge Place se distingue par son système d'arbitrage intuitif, son suivi des événements en temps réel et son guidage utilisateur. Ces fonctionnalités inspirent mon projet, notamment pour :

- Intégrer une timeline simplifiée.
- Simplifier la navigation entre les matchs.
- Proposer un guidage léger pour la configuration des tournois.

2.3.3. Conclusion comparative des applications

Les deux applications analysées, Tournify et Challenge Place, partagent une limitation majeure : elles sont conçues pour des tournois monosport. Trouver une solution multisport s'est avéré difficile, et les contraintes liées à la création de comptes ou aux fonctionnalités limitées dans les versions gratuites compliquent encore plus l'expérience.

Malgré ces limites, ces analyses m'ont permis de constater l'importance d'une interface utilisateur claire et intuitive. Planifier un tournoi reste une tâche complexe qui nécessitera probablement une formation utilisateur, peu importe la simplicité de l'application.

Mon projet se concentrera sur :

- La création de tournois multisports avec une configuration intuitive.
- L'intégration d'un système de génération de plannings automatisé, capable de gérer les conflits.
- Des fonctionnalités spécifiques au cadre scolaire, comme l'assignation automatique des étudiants sans groupe.
- Un arbitrage en temps réel pour le défi technologique.

2.4. Design et Mockups

Suite à l'analyse des applications existantes, des prototypes visuels ont été conçus pour assurer une cohérence entre l'idée initiale et l'implémentation finale [3]. Ces visuels ont joué un rôle essentiel dans la définition de l'interface utilisateur, des flux de navigation et de l'organisation des fonctionnalités. Ils ont également permis d'anticiper des problèmes d'ergonomie et de structure en amont, réduisant ainsi les risques de refactorisation coûteuse durant le développement.

Imaginer une structure d'interface intuitive adaptée à un contexte multisport a représenté un défi important. Une réflexion approfondie sur l'expérience utilisateur (UX) et les interactions des différents acteurs (administrateurs, joueurs, assistants) a guidé cette phase de conception. Quelques exemples clés sont présentés ci-dessous.

2.4.1. Tableau de bord administrateur

Le tableau de bord (Fig. 2.8) constitue la page centrale de l'application pour les administrateurs, leur permettant de gérer les tournois, les sports et les utilisateurs à travers des actions CRUD (Create, Read, Update, Delete).

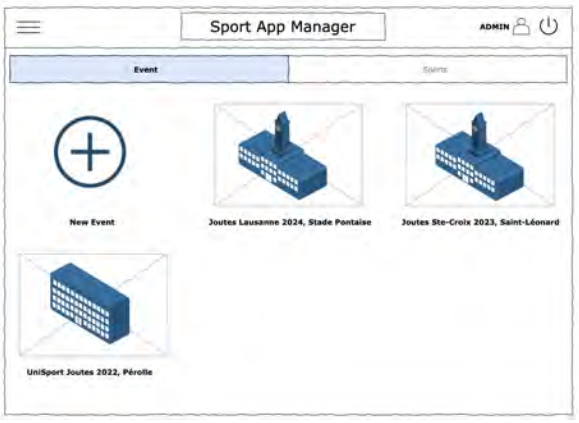


Fig. 2.8.: Gestion des tournois

Fig. 2.9.: Gestion des sports

Ces visuels définissent une navigation simple, avec un menu principal donnant accès aux différents modules (tournois, sports, etc.) et les entités présentées par des cartes.

2.4.2. Gestion des groupes et inscriptions

Cette section vise à simplifier la gestion des inscriptions et des groupes. Un système de filtres, combiné à une vue claire de l'état des groupes, permet aux administrateurs d'identifier les utilisateurs sans groupe et de les assigner automatiquement (Fig. 2.10).

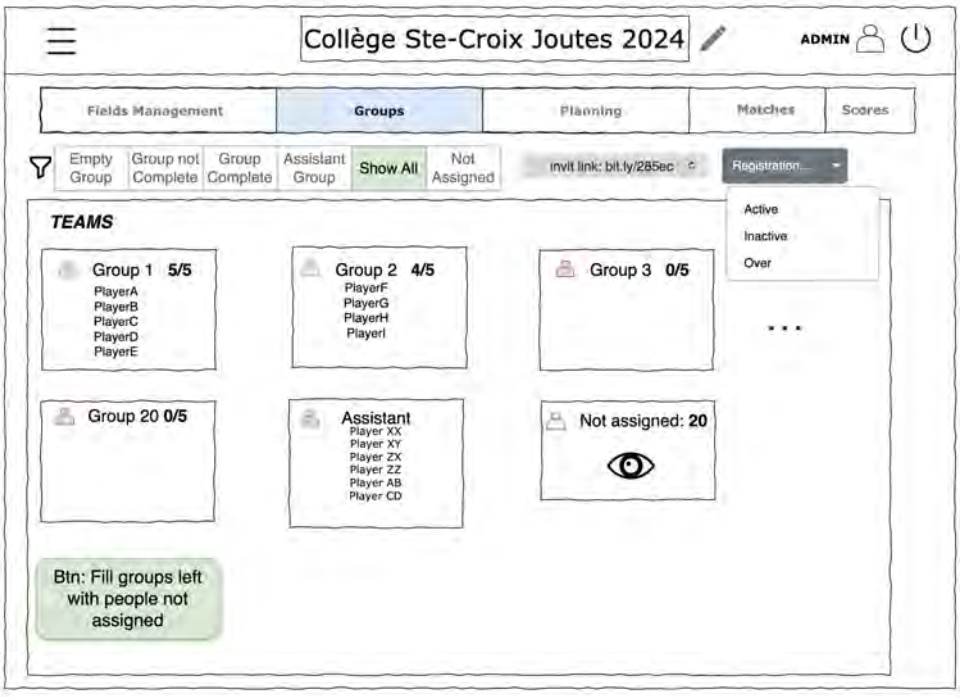


Fig. 2.10.: Gestion des groupes et inscriptions

Les fonctionnalités principales incluent :

- Activation et désactivation des inscriptions.
- Visualisation des utilisateurs sans groupe.

- Assignation automatique des groupes vides ou incomplets.

2.4.3. Planification des matchs

La planification des matchs est une des fonctionnalités les plus complexes de l'application. Elle combine visualisation, modification et génération automatique des plannings (Fig. 2.11).

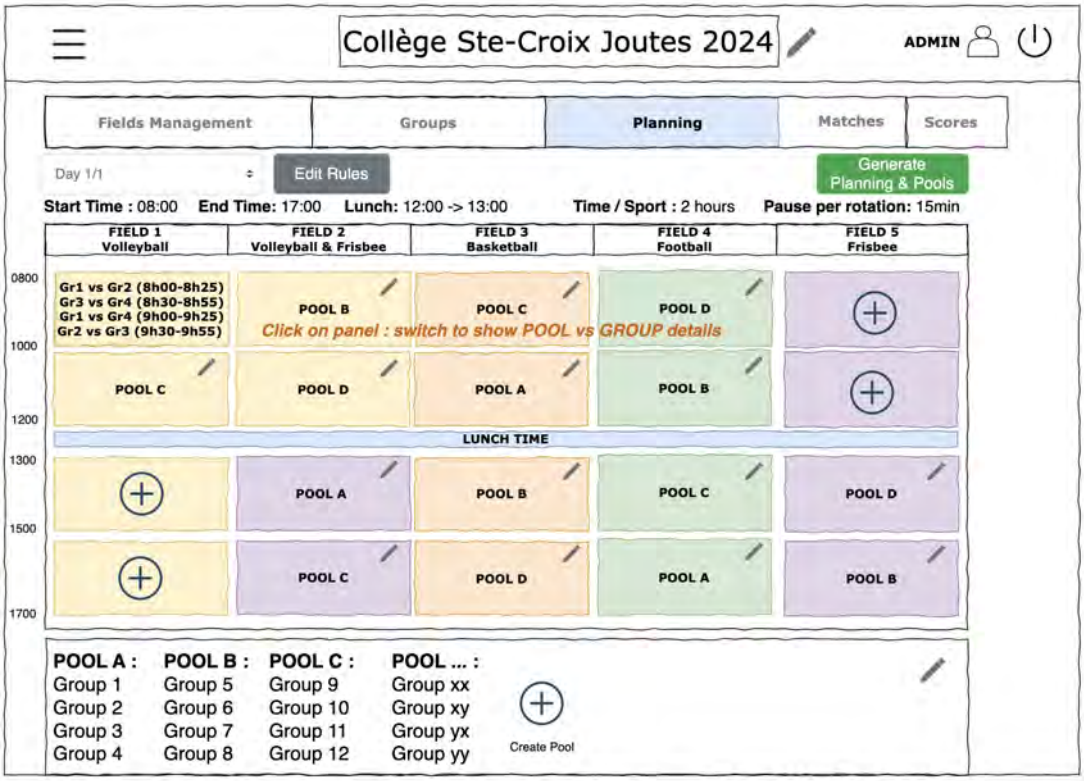


Fig. 2.11.: Gestion du planning

Les principales fonctionnalités incluent :

- **Visualisation détaillée :** Calendrier affichant les matchs par terrain et tranche horaire, avec distinction des pauses.
- **Modification manuelle :** Ajustement direct des matchs, pools et horaires via l'interface.
- **Génération automatique :** Création de plannings basée sur des règles prédéfinies (durée des matchs, pauses, etc.).
- **Gestion des pools :** Interface dédiée en bas de la page pour organiser ou recréer les pools.

Ces concepts ont guidé le choix d'utiliser une bibliothèque spécialisée comme FullCalendar pour l'implémentation.

2.4.4. Conclusion des prototypes

Les prototypes visuels présentés dans cette section illustrent les principales fonctionnalités de l'application et les choix ergonomiques qui les accompagnent. Ces modèles ont permis de poser des bases solides pour le développement tout en réduisant les risques de révisions importantes en cours de projet. Grâce à cette réflexion préalable, l'application bénéficie d'une structure claire et adaptée aux besoins des utilisateurs finaux.

2.5. Choix Technologiques

2.5.1. Introduction

Le choix des technologies pour ce projet a nécessité une phase d'expérimentation et de réflexion approfondie. Ce chapitre présente les explorations initiales réalisées à travers des prototypes CRUD [29], permettant d'évaluer différentes approches et d'identifier une solution technologique adaptée pour servir de socle au développement.

Ces choix se concentrent sur la comparaison entre un framework intégré (Adonis.js) et une solution modulaire plus ouverte (Node.js, Express.js, Sequelize). L'objectif était d'identifier une architecture flexible et intuitive, capable de répondre aux besoins spécifiques d'Easy Tourney. Les détails plus approfondis sur l'implémentation et les ajustements techniques seront développés dans le chapitre 4.2.2, consacré aux aspects techniques du développement.

2.5.2. Prototypes CRUD : Comparaison entre Adonis.js et une solution légère

Dans le cadre de ce projet, deux approches ont été explorées pour développer des prototypes CRUD (Create Read Update Delete). L'objectif était d'évaluer leurs avantages et inconvénients afin d'orienter les choix technologiques pour le projet final. Le premier prototype a été réalisé avec le framework **Adonis.js** [6], tandis que le second utilisait une solution plus modulaire avec **Node.js**, **Express.js**, et **Sequelize**.

Prototype avec Adonis.js

Adonis.js offre une architecture complète et intégrée, comprenant un ORM natif (**Lucid**), une gestion des middlewares, et un moteur de vue (**EdgeJS** [5]). Ce prototype a permis d'explorer ses fonctionnalités principales à travers la création d'une application de gestion de films (*CRUD Movie Management*) basée sur les tutoriels **Adocasts** [7].

Points positifs :

- **Structure intégrée :** Une organisation bien définie des fichiers et fonctionnalités.
- **Outils prêts à l'emploi :** L'ORM Lucid, le système d'authentification intégré, et les middlewares simplifient le développement.

Points négatifs :

- **Courbe d'apprentissage :** Les conventions strictes et les outils spécifiques (InertiaJS, EdgeJS) nécessitent un temps d'adaptation.
- **Documentation limitée :** Bien que correcte, elle manque d'exemples pour résoudre des problématiques avancées.

Prototype avec une solution légère

La seconde approche reposait sur une solution modulaire, combinant **Node.js**, **Express.js**, et **Sequelize** pour le backend, avec **Vue3** pour le frontend. Ce prototype, inspiré du tutoriel `https://www.bezkoder.com/vue-js-node-js-express-mysql-crud-example/`, consistait en une application de gestion de tutoriels (*CRUD Tutorial Management*).

Points positifs :

- **Flexibilité :** Une architecture adaptable aux besoins spécifiques du projet.
- **Popularité :** Une vaste communauté et de nombreuses ressources facilitent la résolution des problèmes.
- **Simplicité :** Une courbe d'apprentissage douce pour une implémentation rapide des bases.

Points négatifs :

- **Absence de cadre strict :** La permissivité de cette solution peut conduire à un code désorganisé si une architecture rigoureuse n'est pas appliquée.
- **Implémentation manuelle :** La personnalisation nécessite un effort supplémentaire, mais offre un meilleur contrôle sur les fonctionnalités.

Conclusion sur les prototypes

Après comparaison des deux prototypes, j'ai opté pour la solution légère avec une stack composée de **Vue3** pour le frontend et de **Node.js**, **Express.js**, et **Sequelize** pour le backend. Cette décision repose sur plusieurs facteurs :

- **Simplicité et flexibilité :** L'approche modulaire permet d'adapter l'architecture aux besoins spécifiques du projet.
- **Expérience :** Ma familiarité avec ces technologies a permis de limiter la courbe d'apprentissage.
- **Complexité du projet :** L'ampleur de l'application rendait risqué l'adoption d'un framework moins connu comme Adonis.js, ce qui aurait pu ralentir le développement.

L'objectif de cette approche n'est pas seulement de mettre en œuvre des fonctionnalités, mais également d'explorer pleinement les défis qu'elles posent. En choisissant une solution plus ouverte, l'application se construit à partir de principes fondamentaux, offrant une opportunité d'affronter directement les complexités du développement full-stack.

Bien que cette approche demande une organisation rigoureuse, notamment dans la structure du code, elle encourage également une meilleure compréhension des interactions entre les différentes couches d'une application web. La mise en place d'un modèle *MVC* [20]

(Modèle-Vue-Contrôleur) est une étape essentielle pour maintenir une séparation claire entre la logique métier, les interactions avec la base de données, et la présentation.

Ce choix technologique pose ainsi des bases solides pour le développement d'une application robuste et personnalisée, permettant de se concentrer sur les fonctionnalités clés à travers les cas d'utilisation.

2.6. UseCases

2.6.1. Introduction

Ce chapitre présente les cas d'utilisation (UseCases) [11] de l'application Easy Tourney, en les organisant par acteur principal : l'Admin, les Utilisateurs (Player, Assistant, Guest) et le Système. Chaque acteur est accompagné de son diagramme UML suivi d'une explication des fonctionnalités associées. L'Admin partage les cas d'utilisation des utilisateurs standards tout en bénéficiant de droits supplémentaires.

2.6.2. Diagrammes UseCase

Admin

L'Admin (Fig. 2.12) est le rôle principal dans l'application, chargé de gérer l'ensemble des fonctionnalités liées à l'organisation des tournois et à l'administration des utilisateurs. Il dispose de droits étendus pour superviser les tournois, gérer les inscriptions, configurer les terrains et plannings, et assurer le bon déroulement des événements sportifs.

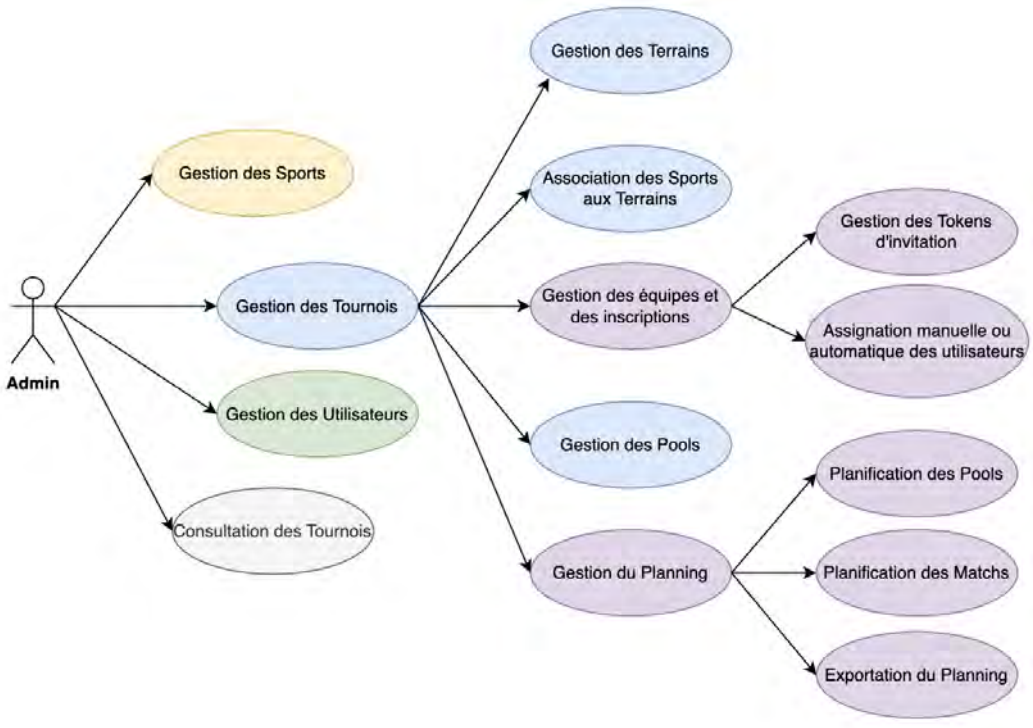


Fig. 2.12.: Diagramme UseCase - Admin

Cas d'utilisation - Admin 1. **Gestion des sports** : Permet d'ajouter, modifier ou supprimer des sports avec des règles, descriptions, images, couleurs et systèmes de points. Ces sports peuvent ensuite être associés à des terrains pour les tournois.

2. **Gestion des tournois** : Fonctionnalité centrale permettant de créer, modifier ou supprimer des tournois, de gérer leurs statuts (brouillon, actif, terminé) et d'organiser les terrains, les équipes, les pools, et le planning.

- **Gestion des terrains** : Définit et gère les terrains d'un tournoi.
- **Association des sports aux terrains** : Permet d'attribuer des sports à des terrains selon des créneaux horaires définis.
- **Gestion des équipes et des inscriptions** : Offre des outils pour créer et organiser les équipes, gérer les inscriptions via des tokens, envoyer des emails et filtrer/trier les groupes selon différents critères.
- **Gestion des pools** : Automatise la création et l'organisation des pools selon des règles configurées (ex : nombre d'équipes par pool).
- **Gestion du planning** : Génère automatiquement un calendrier des matchs et permet des ajustements manuels. Le planning peut être exporté en format Excel.

3. **Gestion des utilisateurs** : Gère les comptes, les rôles (admin ou utilisateur) et les permissions. Permet d'ajouter, modifier ou supprimer des utilisateurs, de les associer ou retirer d'un tournoi, et de rechercher des utilisateurs selon des critères précis.

4. **Consultation des tournois** : Offre une vue utilisateur pour simuler et tester les tournois configurés. L'Admin peut consulter tous les tournois et vérifier les paramètres.

Exemple : L'admin crée un tournoi multisport, configure les terrains, génère automatiquement les pools et les plannings, puis invite des utilisateurs via des tokens d'inscription.

User

Les utilisateurs standards (Player, Assistant, Guest) participent aux tournois avec des fonctionnalités adaptées à leur rôle (Fig. 2.13). Un utilisateur peut avoir des rôles différents selon le tournoi.

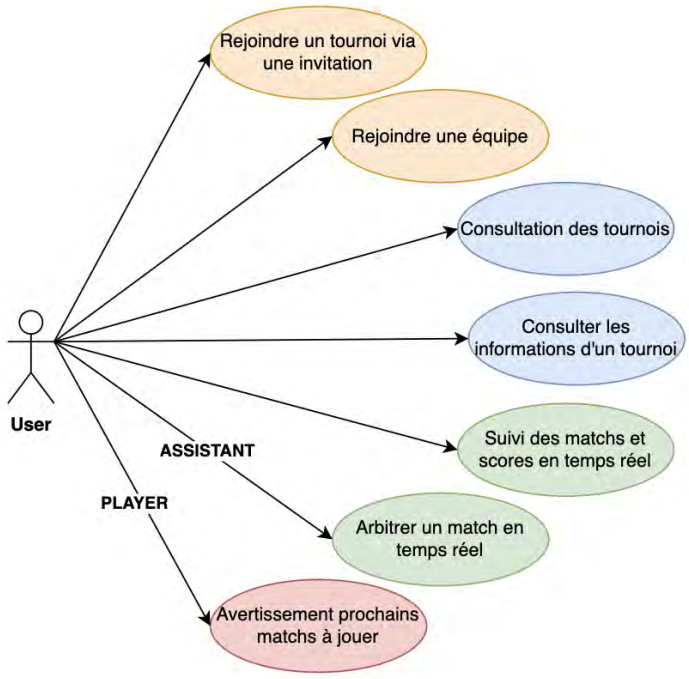


Fig. 2.13.: Diagramme UseCase - User

Cas d'utilisation - User

1. **Rejoindre un tournoi** : Les utilisateurs peuvent s'inscrire à un tournoi via un lien d'invitation, soit en créant un compte soit en utilisant un compte existant.

2. **Rejoindre une équipe** : Permet aux utilisateurs d'intégrer ou quitter une équipe (si les inscriptions sont ouvertes). Ils peuvent également se retirer du tournoi dans les mêmes conditions.

3. **Consultation des tournois et plannings** : Les utilisateurs peuvent visualiser les informations générales des tournois auxquels ils participent et suivre le planning des matchs.

4. **Suivi des matchs et scores en temps réel** : Fournit un accès en direct aux scores et événements des matchs. Les Players sont informés pour leurs prochains matchs.

5. **Arbitrage (Assistants uniquement)** : Les assistants enregistrent les événements des matchs (buts, fautes, cartons) et mettent à jour les scores. Ils gèrent également l'état global du match (prévu, en cours, terminé).

Exemple : Un utilisateur reçoit un lien d'invitation, s'inscrit au tournoi, rejoint une équipe, puis consulte le planning pour connaître ses matchs lorsque le tournoi est actif.

Système

Le "Système" (Fig. 2.14) regroupe des fonctionnalités techniques globales qui ne dépendent pas directement de l'organisation des tournois. Ces fonctionnalités, telles que l'installation en PWA ou le switch entre les modes clair et sombre, visent à améliorer l'expérience utilisateur au niveau de la plateforme.

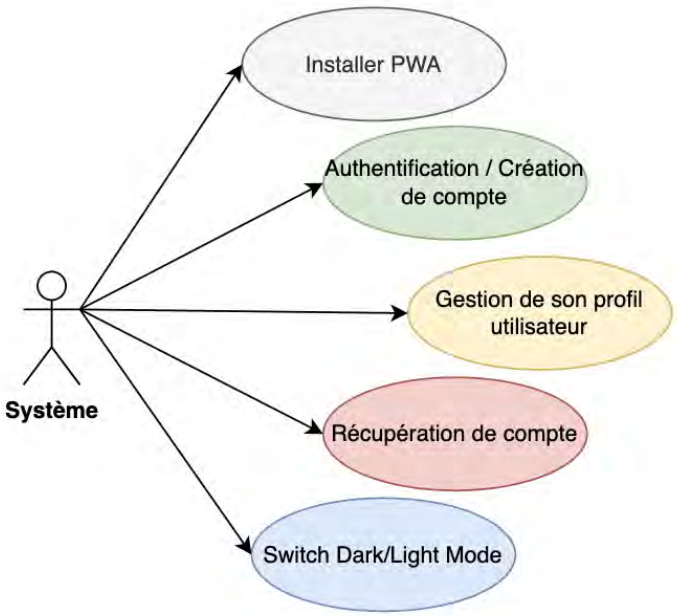


Fig. 2.14.: Diagramme UseCase - Système

Cas d'utilisation - Système

1. **Installation PWA** : Permet un accès partiel hors ligne à l'application, facilitant la consultation du planning.

2. **Authentification et gestion de compte** : Gère la création et la connexion des comptes, ainsi que la sécurité des accès via un système JWT.

3. **Gestion de profil utilisateur** : Les utilisateurs peuvent mettre à jour leurs informations personnelles (nom, mot de passe).

4. **Switch Dark/Light Mode** : Offre la possibilité de basculer entre les modes clair et sombre pour une meilleure accessibilité.

Exemple : Le système permet à un utilisateur d'accéder au planning hors ligne via une installation PWA et de personnaliser l'affichage en mode clair ou sombre.

2.6.3. Conclusion UseCases

Les cas d'utilisation décrits illustrent les interactions entre les différents acteurs et le système. L'Admin dispose de tous les droits nécessaires pour gérer les tournois et les utilisateurs, tandis que les utilisateurs standards bénéficient de fonctionnalités ciblées pour leur rôle dans les tournois. Cette répartition garantit une expérience fluide et une gestion optimale des événements sportifs. Le chapitre 3 détaillera les fonctionnalités de chaque acteur.

2.7. Modélisation ERM Simplifiée

2.7.1. Introduction

La conception de la base de données repose sur un modèle relationnel conçu pour gérer efficacement les relations complexes liées à un tournoi sportif. Ce modèle utilise le **Modèle Entité-Relation (ERM)**, une méthodologie couramment utilisée pour représenter les modèles conceptuels des données [24]. Une réflexion approfondie a été menée dès la phase d'analyse pour définir les entités et leurs relations, garantissant ainsi un design robuste et évolutif. Ce travail préalable a permis d'éviter des refontes coûteuses et a assuré une intégration fluide entre le backend et le frontend.

Importance de la conception en amont : Investir du temps dans la conception d'un modèle relationnel bien pensé est une étape cruciale dans tout projet logiciel. Une modélisation précise limite les besoins de refactorisation ultérieure et facilite les évolutions durant le développement. Dans le cadre de ce projet, cette approche structurée a été essentielle pour respecter les délais tout en gérant la complexité des fonctionnalités attendues.

2.7.2. Choix de la Base de Données Relationnelle

Le choix d'un modèle relationnel pour la gestion des données d'Easy Tourney a été motivé par plusieurs facteurs inhérents aux exigences du projet. Les bases de données relationnelles, telles que MySQL, offrent une structure robuste pour gérer des données fortement interconnectées, ce qui est essentiel pour les fonctionnalités complexes d'un système de gestion de tournois multisports. Les raisons principales de ce choix sont les suivantes :

1. **Intégrité et Consistance des Données :** Les bases de données relationnelles garantissent l'intégrité des données grâce à des contraintes telles que les clés primaires et étrangères, ce qui est crucial pour éviter les incohérences dans les relations complexes entre entités comme les équipes, les matchs et les utilisateurs.
2. **Requêtes Complexes :** SQL permet d'exécuter des requêtes complexes et optimisées, facilitant la récupération et l'analyse des données nécessaires pour le fonctionnement efficace de l'application.
3. **Support et Communauté :** MySQL bénéficie d'une vaste communauté et d'un support étendu, ce qui facilite la résolution de problèmes et l'accès à des ressources et des outils complémentaires.
4. **Compatibilité et Intégration :** MySQL s'intègre facilement avec divers frameworks et langages de programmation utilisés dans le projet, assurant une intégration fluide entre le backend et le frontend.
5. **Scalabilité :** Bien que les besoins actuels se concentrent sur la fiabilité et la consistance, MySQL offre également des options de scalabilité pour répondre à des besoins futurs en termes de volume de données et de nombre d'utilisateurs.

2.7.3. Diagramme ERM Simplifié

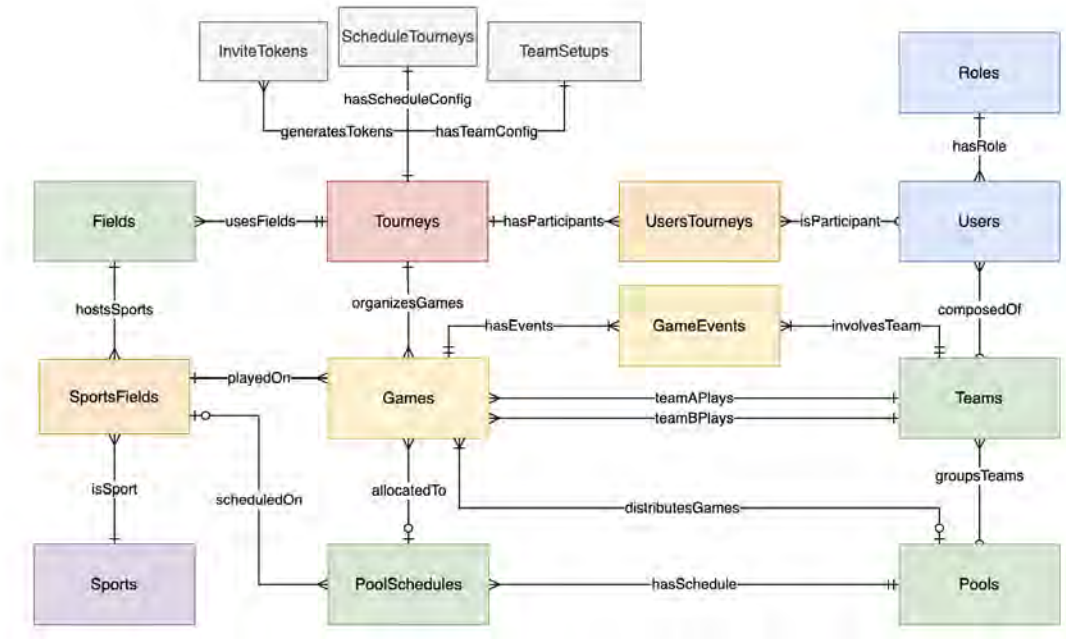


Fig. 2.15.: Diagramme ERM Simplifié - Easy Tourney

2.7.4. Description des Entités Principales

Le modèle ERM d'Easy Tourney (Fig. 2.15) repose sur des entités clés qui reflètent les besoins métiers liés à l'organisation de tournois sportifs. Voici une description détaillée des entités principales et de leurs relations :

- **Tourneys** : Cette entité centrale représente les tournois. Elle est reliée aux Users, Fields, Teams, et Games. Des tables supplémentaires, telles que 'ScheduleTourneys' et 'TeamSetups', gèrent les configurations spécifiques liées à chaque tournoi. Cette structure modulaire permet de simplifier la table 'Tourneys' tout en offrant une flexibilité pour adapter les paramètres des tournois à différents contextes.
- **Users** : Cette table représente les utilisateurs de l'application, qu'ils soient Admins, Players ou Assistants. Les rôles globaux sont définis dans la table 'Roles', tandis que leurs participations spécifiques à un tournoi sont gérées via 'UsersTourneys'. Cette séparation permet de donner à un utilisateur des rôles différents selon le tournoi, offrant une grande flexibilité pour la gestion des permissions.
- **Teams** : Les teams regroupent les utilisateurs pour les compétitions. Une team peut appartenir à un pool ou participer à des matchs. Cette entité est reliée à 'Games' pour représenter les équipes opposées dans un match (à travers 'teamAId' et 'teamBId').
- **Games** : Représente les matchs organisés dans un tournoi, en liant les teams, terrains, et horaires. Les événements clés des matchs (buts, fautes, etc.) sont capturés dans la table 'GameEvents'.

- **Pools** : Organise les équipes en groupes pour structurer les tournois. Les pools sont associés à un planning via 'PoolSchedules', permettant de distribuer les matchs de manière cohérente.
- **Fields** : Représente les terrains où se déroulent les matchs. Chaque terrain est lié à des sports spécifiques via 'SportsFields' avec des créneaux horaires. Cette relation permet de planifier plusieurs sports différents par terrain.
- **Sports** : Cette entité définit les sports disponibles dans l'application et leurs règles associées.
- **GameEvents** : Capture les événements clés des games (points, fautes, etc.) et les associe aux équipes concernées. Cette table est essentielle pour suivre les performances et les statistiques en temps réel.
- **UsersTourneys** : Cette table pivot associe les utilisateurs aux tournois. Elle permet à un utilisateur d'avoir un rôle différent par tournoi.
- **InviteTokens** : Gère les tokens d'invitation générés pour permettre l'inscription des utilisateurs à un tournoi. Chaque token contient des informations sur son état (valide ou expiré) et sa durée de validité.
- **ScheduleTourneys** et **TeamSetups** : Ces tables de configuration permettent respectivement de gérer les horaires et les règles des équipes pour chaque tournoi. Cette modularité simplifie la table centrale 'Tourneys' tout en facilitant l'évolution des règles et des plannings.

2.7.5. Choix de Modélisation et Remarques

- **Gestion des Terrains** : Chaque tournoi dispose de ses propres terrains, même si cela peut paraître non optimal d'un point de vue réel. Cette approche simplifie la gestion dans l'application en évitant des chevauchements entre tournois. Les enseignants doivent savoir le nombre de terrains disponibles et, au besoin, diviser un terrain en plusieurs parties. Cette décision facilite l'implémentation.
- **Double relation Games-Teams** : Cette double relation est indispensable pour représenter les deux équipes impliquées dans un match (à travers les champs 'teamAPlays' et 'teamBPlays').
- **GameEvent et Team** : Lors d'une faute par exemple, l'événement sera directement attribué à une équipe et non à un joueur. La raison est que les arbitres seront des élèves qui n'auront pas forcément connaissance du nom des joueurs de chaque équipe. Il faudra ajouter dans les événements de jeu un champ 'description' afin de pouvoir commenter l'événement.

2.7.6. Conclusion de la Modélisation ERM

La modélisation ERM présentée constitue une base solide pour la gestion des données dans Easy Tourney. Elle reflète les besoins métiers tout en anticipant les évolutions futures et les contraintes techniques. Bien que ce modèle comporte un nombre relativement élevé de tables, cette quantité est justifiée par la complexité des fonctionnalités et des interactions nécessaires à la gestion des tournois multisports. Chaque table joue un rôle bien défini, garantissant une modularité et une évolutivité optimales.

Cette structure appuie également le choix d'un modèle relationnel basé sur SQL, qui s'avère particulièrement adapté pour des données fortement structurées, avec des relations complexes et des règles métiers strictes. Contrairement aux bases de données non relationnelles (NoSQL), un modèle relationnel offre :

1. Une garantie d'intégrité des données grâce aux contraintes (clés primaires, clés étrangères, etc.).
2. Une facilité de modélisation des relations, particulièrement pertinente pour les cas d'utilisation tels que les tournois multisports où chaque entité (joueurs, équipes, matchs, etc.) est fortement interconnectée.
3. La possibilité d'exécuter des requêtes complexes et de maintenir des données normalisées pour éviter les redondances.

Bien que les bases NoSQL puissent être mieux adaptées à certains cas d'utilisation, comme les applications nécessitant une scalabilité horizontale extrême ou des schémas dynamiques, les besoins d'Easy Tourney se concentrent davantage sur la fiabilité, la cohérence et la gestion précise des relations. Ainsi, le choix d'un modèle relationnel permet de répondre efficacement aux exigences fonctionnelles et de garantir la robustesse du système.

En complément, parmi les bases de données relationnelles disponibles, MySQL a été privilégié pour ses performances éprouvées, sa compatibilité avec les outils et frameworks utilisés dans le projet, ainsi que sa facilité de gestion et de maintenance. Cette décision est détaillée dans le chapitre 4.

2.8. Conclusion de l'Analyse

Une attention particulière a été portée à la phase d'analyse, essentielle pour structurer les objectifs, anticiper les difficultés et orienter les choix technologiques. L'étude des sites existants, tels que Tournify et Challenge Place, a permis d'identifier des inspirations clés pour répondre aux besoins spécifiques des tournois multisports, tandis que les maquettes ont joué un rôle central pour clarifier les flux utilisateurs et prévenir les problèmes d'ergonomie.

Le choix des technologies a également été affiné après une expérimentation initiale avec Adonis.js. Bien que prometteur, ce framework s'est révélé trop complexe et chronophage dans le contexte du projet, ce qui a conduit à adopter une solution plus modulaire et intuitive avec Node.js, Express.js et Sequelize. Ce choix a facilité une mise en œuvre progressive et adaptée aux besoins spécifiques d'Easy Tourney.

Enfin, la modélisation soignée de la base de données a permis de répondre aux exigences métier tout en maintenant une organisation claire et évolutive. En somme, cette phase d'analyse a établi des fondations réfléchies qui ont permis de développer l'application de manière fluide et structurée. Le chapitre suivant explorera la mise en œuvre du projet du point de vue de l'utilisateur.

3

Point de vue utilisateur

3.1. Introduction

Ce chapitre propose une présentation détaillée de l'application *Easy Tourney* du point de vue de l'utilisateur. À travers des scénarios concrets, nous explorerons les fonctionnalités principales, en mettant particulièrement l'accent sur leur utilité et leur simplicité d'utilisation. Les différents rôles (administrateur, utilisateur standard) seront présentés de manière succincte et accompagnés de captures d'écran pour une meilleure compréhension.

La structure du chapitre s'articule autour d'une description préalable du type de tournoi, suivie de trois scénarios, puis de la présentation de quelques fonctionnalités clés du système :

1. **Scénario 1 :** Préparation d'un tournoi
2. **Scénario 2 :** Gestion des inscriptions
3. **Scénario 3 :** Déroulement du tournoi (arbitrage et scores)
4. **Fonctionnalités systèmes :** Mode Light/Dark, PWA, récupération de mot de passe, design responsive

Les captures d'écran présentées dans ces scénarios ont été réalisées sur un environnement macOS à l'aide du navigateur Google Chrome.

3.2. Type de tournoi : *CustomRoundRobin*

(L'implémentation est détaillée au chapitre 4.7)
Avant de présenter les scénarios, il est important de clarifier le type de tournoi actuellement géré par l'application *Easy Tourney* : le **CustomRoundRobin**, qui est une version personalisée du RoundRobin [16]. Ce format a été conçu pour maximiser la participation des équipes tout en tenant compte des contraintes de temps et de la diversité des sports. Voici ses grandes lignes de fonctionnement :

3.2.1. Structure en pools

Les équipes sont réparties en «pools». Chaque pool reste le même tout au long de la compétition et dispute des matchs dans différents sports, selon le planning configuré par l'administrateur.

3.2.2. Répartition des sports

Chaque pool est associé à un ou plusieurs créneaux horaires dans des sports distincts. Par exemple, un pool peut participer au football de 8h à 10h, au basketball de 10h à 12h, au volleyball de 13h à 15h et au badminton de 15h à 17h. Cette organisation permet aux équipes de jouer dans un maximum de sports différents.

3.2.3. Matchs au sein des pools

Les équipes d'un même pool s'affrontent entre elles, avec pour objectif de maximiser la variété des oppositions. Le système vise à équilibrer les rencontres de sorte qu'aucune équipe ne se retrouve à affronter systématiquement les mêmes adversaires.

3.2.4. Optimisation et flexibilité

- Contrairement à un *RoundRobin* classique (où chaque équipe affronte toutes les autres), le **CustomRoundRobin** adapte le nombre de matchs au temps disponible. Par exemple, si un tournoi prévoit cinq sports mais seulement quatre créneaux horaires, chaque pool jouera dans quatre sports, assurant ainsi une rotation efficace.
- La taille des pools peut être ajustée pour accueillir davantage d'équipes, offrant ainsi plus de temps de pause aux joueurs.

3.2.5. Avantages

Ce format est particulièrement adapté aux tournois scolaires ou multisports, où l'accent est mis sur la participation et la variété des disciplines. Il facilite également l'organisation logistique en gardant les mêmes groupes d'équipes ensemble tout au long de la compétition.

3.3. Contexte et problématique des scénarios

M. Dupont, professeur de sport au collège Saint-Michel, souhaite organiser un tournoi multisport pour des classes de première et deuxième année, soit environ 200 élèves au total.

La mise en place d'un tournoi multisport soulève plusieurs défis majeurs :

- **Gestion des inscriptions :** Enregistrer et organiser les informations de plusieurs centaines d'élèves, en veillant à attribuer correctement chaque participant à une équipe.

- **Planification :** Établir un planning cohérent pour un tournoi multisport tout en prenant en compte diverses contraintes (conflits d'horaires, répartition équitable des matchs, limitations des créneaux disponibles, etc.).
- **Gestion des scores et résultats :** Mettre à jour les scores en temps réel afin de suivre facilement l'avancée de chaque rencontre.

Ces problématiques rendent l'organisation manuelle particulièrement fastidieuse et susceptible d'entraîner des erreurs. L'application *Easy Tourney* propose des outils répondants à ces problématiques, lesquels seront présentés au fil des scénarios détaillés dans la suite du chapitre.

3.4. Scénario 1 : Préparation d'un tournoi

3.4.1. Connexion Administrateur

M. Dupont se connecte à l'application *Easy Tourney* (Fig. 3.1) pour préparer un nouveau tournoi avec un compte administrateur (créé en amont par le responsable informatique).

Connexion en tant qu'administrateur

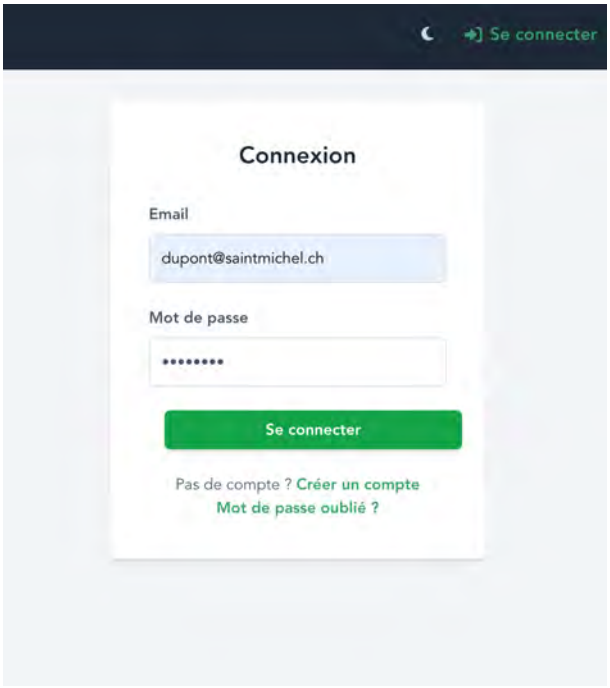


Fig. 3.1.: Interface de connexion

Dashboard Administrateur

Une fois connecté, M. Dupont est redirigé vers la page listant les tournois existants (Fig. 3.2).

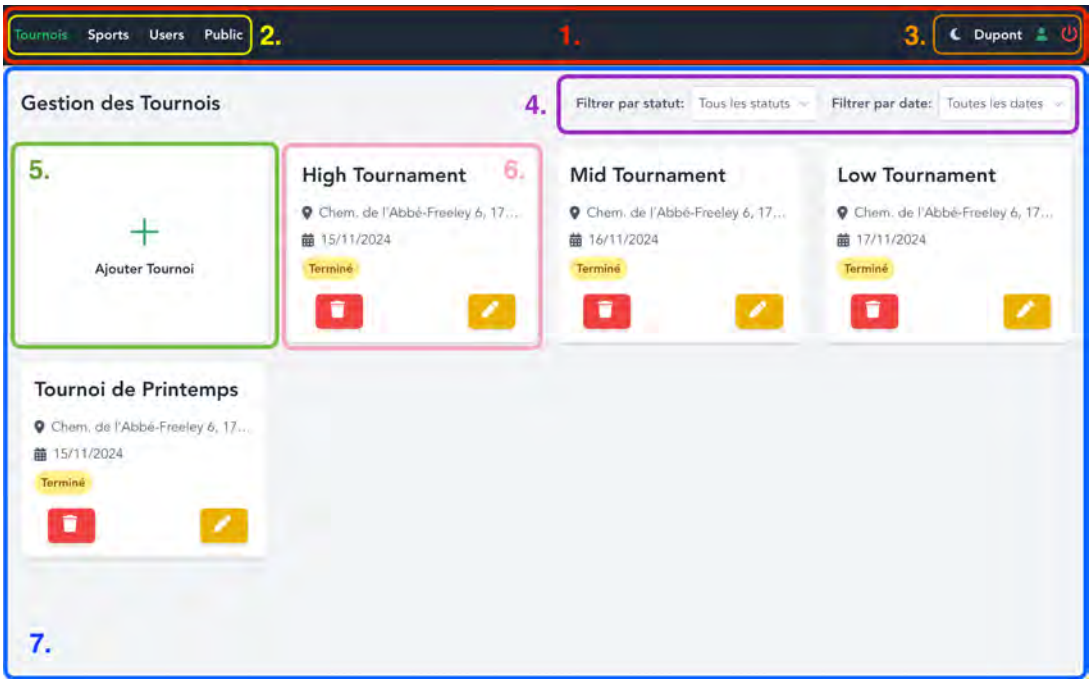


Fig. 3.2.: Présentation Dashboard Administrateur

1. **Menu Principal (rouge)** : Situé en haut, il permet d'accéder rapidement aux principales sections de l'application.
2. **Navigation Latérale (jaune)** : Accès direct aux sections essentielles, adaptées au rôle de l'utilisateur connecté.
3. **Système (orange)** : Actions système (mode sombre, profil, déconnexion).
4. **Filtres de Recherche (violet)** : Tri par statut (tournois en édition, prêts, terminés) ou par date (à venir, en cours, passés).
5. **Création de Tournoi (vert)** : Carte permettant l'ajout d'un nouveau tournoi.
6. **Liste des Tournois (rose)** : Cartes récapitulant les tournois existants avec leurs statuts et options (modifier, supprimer, accéder au tournoi).
7. **Contenu Principal (bleu)** : Zone principale affichant les informations contextuelles.

3.4.2. Gestion des sports

M. Dupont accède à la page **Sports**, où plusieurs disciplines sont préconfigurées (Fig. 3.3). Il ajoute alors le sport manquant : le Unihockey.

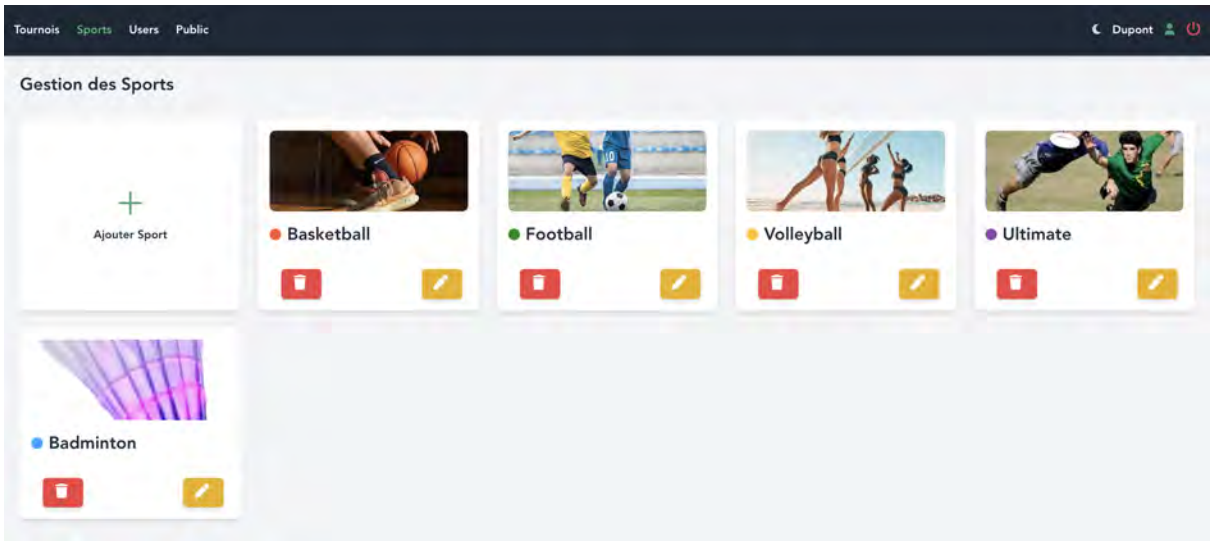


Fig. 3.3.: Gestion des sports

Il ouvre la fenêtre modale et ajoute les informations nécessaires pour le sport **Unihockey**.

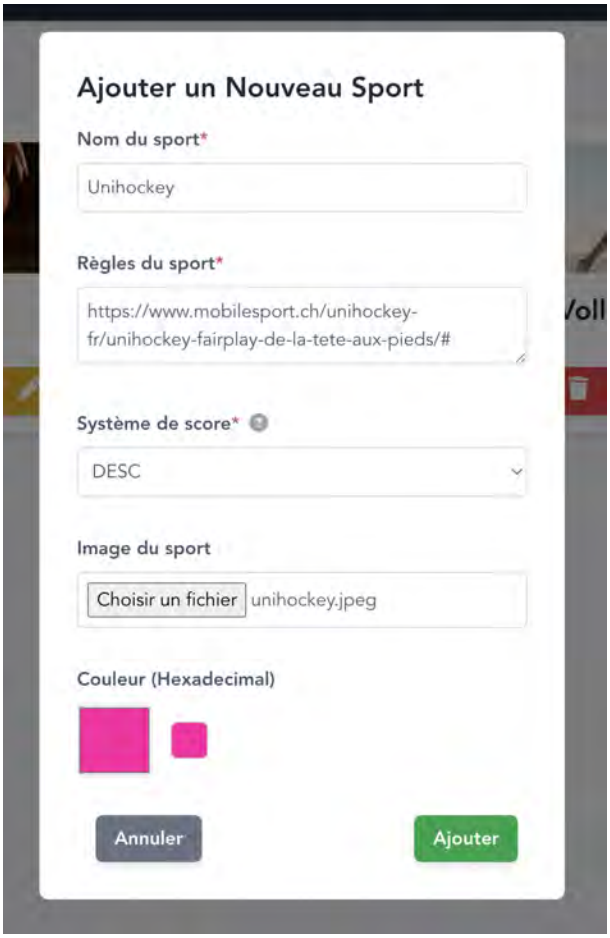


Fig. 3.4.: Ajout d'un nouveau sport

Note : Le système de scores de la figure 3.4 propose deux options : **DESC** (descendant) et **ASC** (ascendant). Le mode DESC est utilisé lorsque le score le plus élevé est synonyme

de victoire (p. ex., le nombre de points marqués), tandis que le mode ASC convient lorsque le score le plus faible l'emporte (p. ex., réaliser le meilleur temps sur 100 m).

Bien que cette fonctionnalité soit intégrée, elle n'est pas encore pleinement implémentée. Le système prend actuellement en charge uniquement les sports collectifs avec un système de points en mode DESC, ce qui couvre la majorité des sports d'équipe dans un contexte scolaire.

Le paramètre **Couleur** détermine la couleur du sport affichée sur les plannings.

3.4.3. Création et configuration du tournoi

Tous les sports étant présents, M. Dupont revient sur la page **Tournois** et clique sur **Ajouter tournoi** pour démarrer la configuration d'un nouveau tournoi (Fig. 3.5).

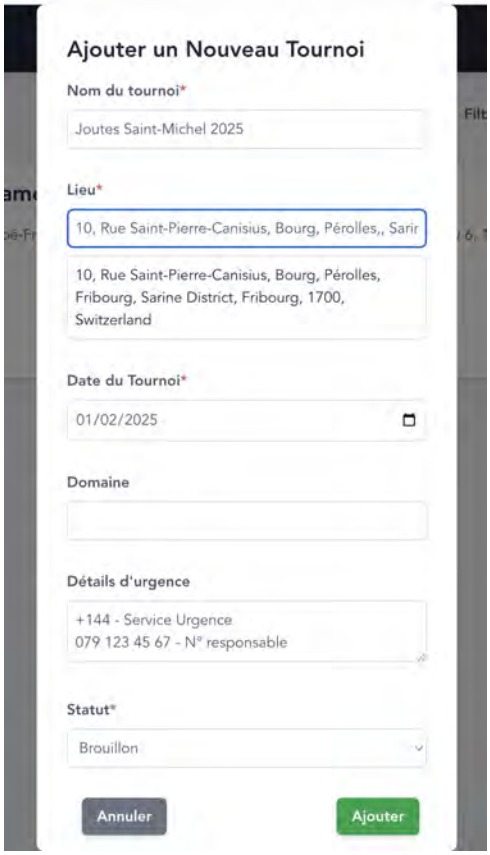


Fig. 3.5.: Ajout d'un nouveau tournoi

- **Nom du tournoi :** Nom (unique) du tournoi.
- **Lieu :** Propose des suggestions d'adresses.
- **Date du tournoi :** Le tournoi n'est, pour l'instant, configurable que sur une seule journée.
- **Domaine :** Limite les inscriptions à un domaine spécifique (p. ex. `@unifr.ch`), mais pas encore fonctionnel.
- **Détails d'urgence :** Contacts d'urgence ou tout autre détail.
- **Statut :** État du tournoi (Brouillon, Prêt, En cours, Terminé).

Détails Tournoi

Après avoir créé le tournoi, M. Dupont accède à la page de configuration (Fig. 3.6) :

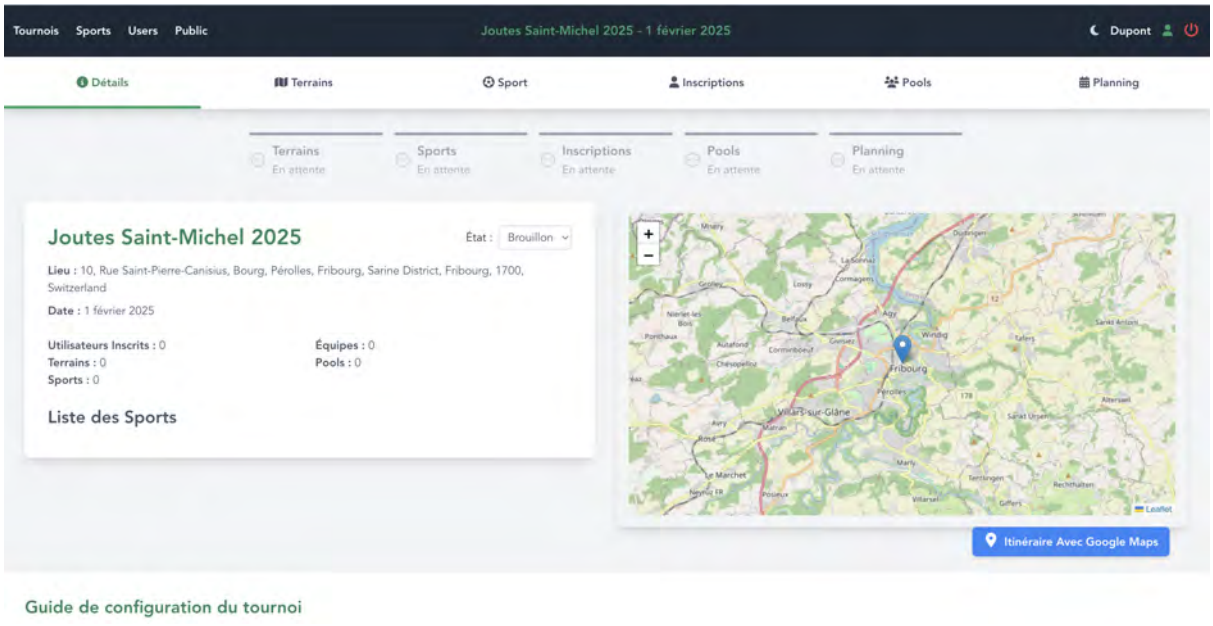


Fig. 3.6.: Configuration du tournoi

Cette page affiche l'état global de la configuration. Un sous-menu permet de naviguer entre les différentes sections, avec des statuts de progression en en-tête pour servir de checklist. L'administrateur peut ainsi suivre l'avancement en un coup d'œil.

En outre, un guide pratique au bas de la page explique comment configurer un tournoi selon les règles métier (Fig. 3.7).

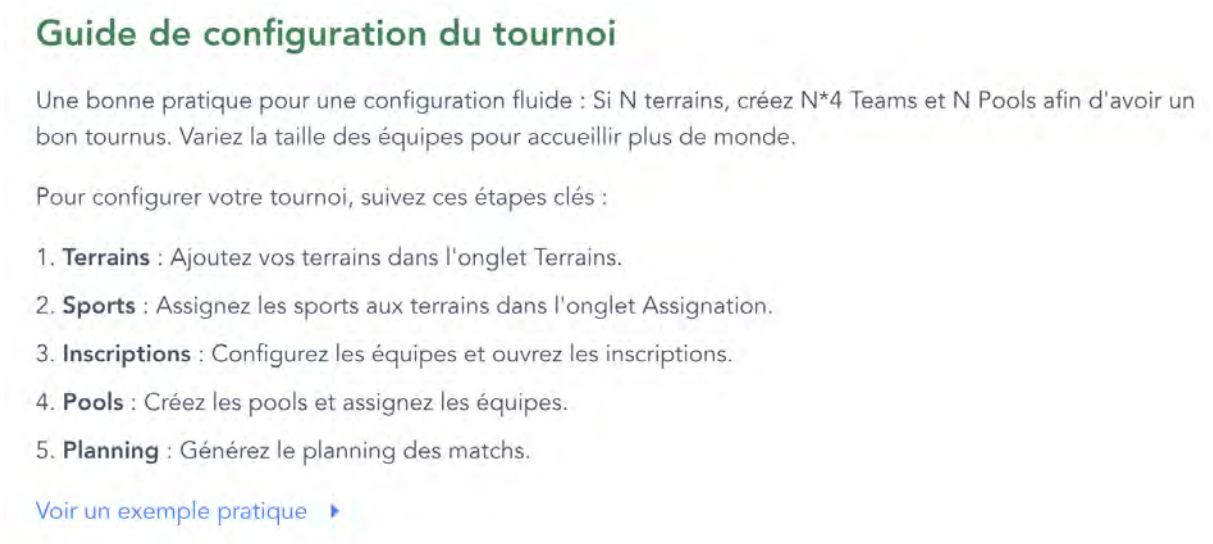


Fig. 3.7.: Guide du tournoi

L'exemple pratique détaille les étapes pour ajouter des terrains et leur associer des sports, définir les équipes, créer des pools et générer un planning fluide. Certaines décisions restent toutefois à la charge de l'utilisateur, notamment la réservation effective de terrains en fonction du nombre de participants.

Dans le cas de 200 élèves, M. Dupont évalue qu'il lui faut 10 terrains pour accueillir 10 pools (chacune composée de 4 équipes de 5 joueurs). De cette manière, 40 équipes de 5 joueurs (200 élèves) pourront tourner efficacement sur les différents créneaux.

Configuration des terrains

M. Dupont commence par créer les 10 terrains sur la page *Terrains* (Fig. 3.8) :

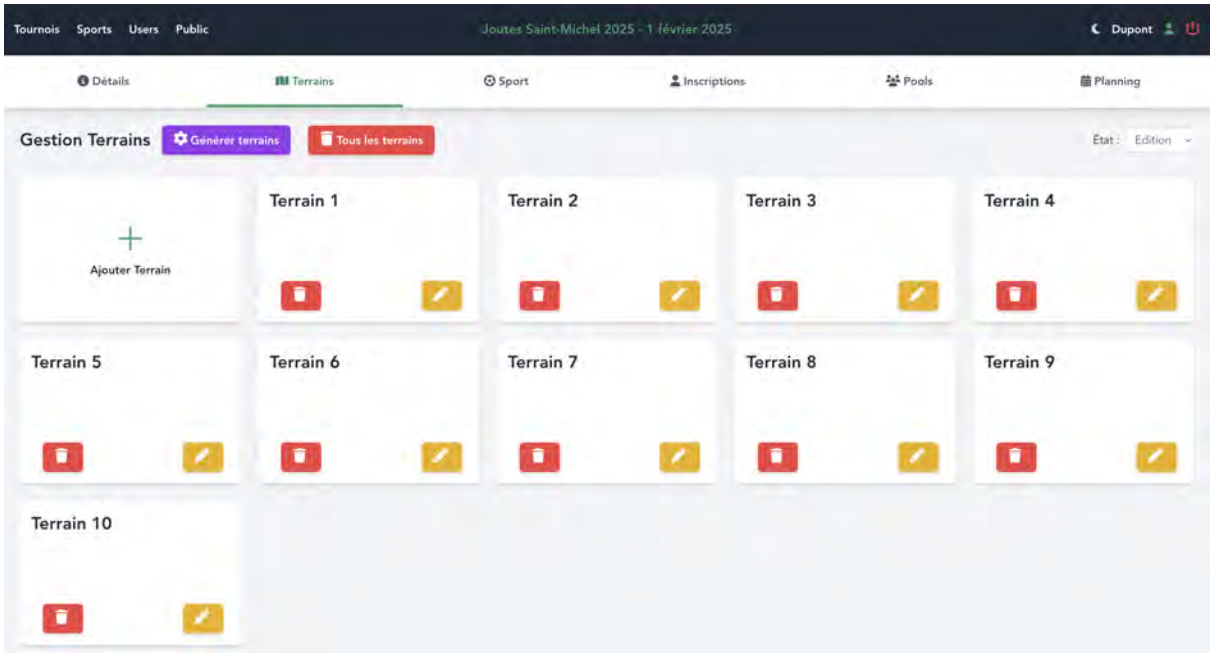


Fig. 3.8.: Configuration des terrains

Il est possible d'ajouter un terrain manuellement ou d'en générer plusieurs via **Générer terrains**. Chaque terrain reste modifiable (nom) ou supprimable. Le bouton **Supprimer tous les terrains** supprime tous les terrains ainsi que leurs associations (sports, plannings).

Note : Dans le coin supérieur droit, un sélecteur de statut permet de définir l'état de progression (par ex. **Édition** ou **Terminé**). Une fois l'état *Terminé* activé, il n'est plus possible de faire des modifications. Chaque entité (Terrains, Sports, Inscriptions, Pools, Planning) propose un sélecteur similaire pour suivre la configuration.

Assignation des sports aux terrains

Une fois les terrains créés, M. Dupont assigne les sports souhaités aux créneaux horaires correspondants (Fig. 3.9).

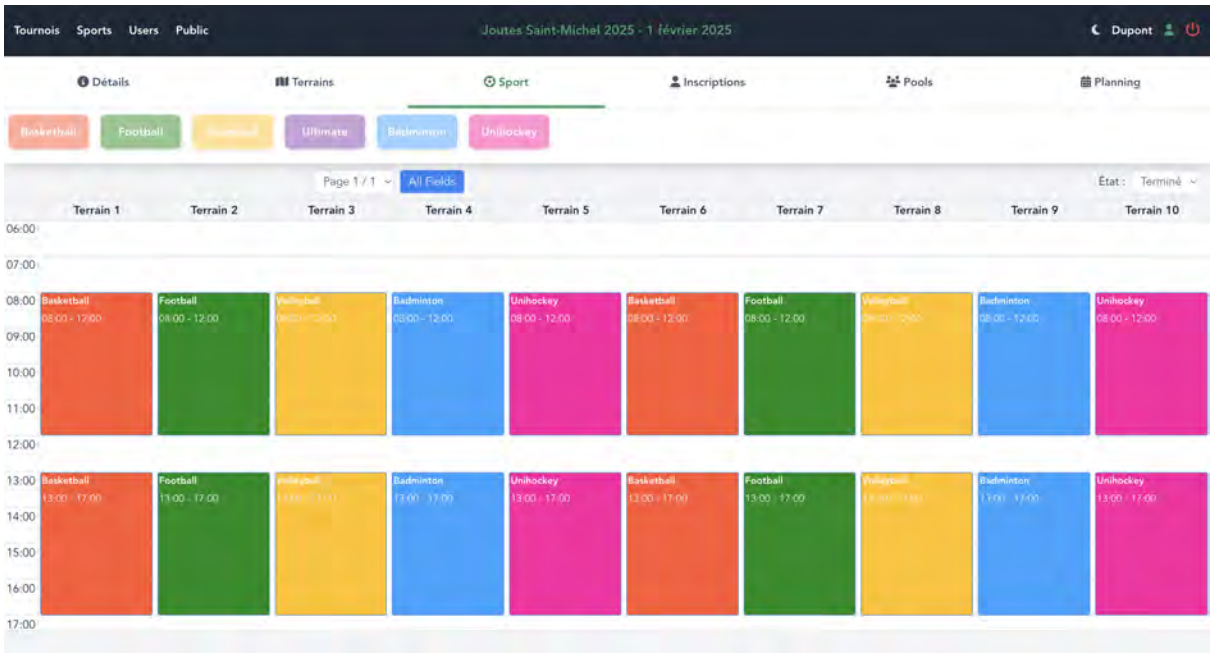


Fig. 3.9.: Assignation des sports aux terrains

Pour associer un sport à un terrain, il suffit de faire glisser un sport depuis l'en-tête blanche et de le déposer sur la plage horaire d'un terrain. Plusieurs sports peuvent être assignés à un même terrain, et la durée d'un créneau peut s'ajuster en étirant ou en réduisant l'entité correspondante. Une suppression d'association s'effectue par double-clic sur l'icône en forme de croix. Lorsque la page est en mode **Terminé**, toute modification est désactivée afin d'éviter tout changement accidentel.

Configuration des équipes et gestion des inscriptions

Pour préparer les inscriptions, M. Dupont doit tout d'abord créer des équipes. L'application autorise une configuration complète du tournoi (y compris le planning) avant d'inviter les joueurs, mais il est également possible de démarrer les inscriptions dès que les équipes sont créées. Le système d'inscription sera détaillé dans le scénario 2 (chapitre 3.5).

Fig. 3.10.: Page de gestion des équipes

Un message informe des actions à réaliser. M. Dupont clique sur **Config Équipes** (Fig. 3.11) :

Fig. 3.11.: Modal Config Équipes

Ces options permettent d'automatiser la création d'équipes. Après avoir défini les paramètres (nombre d'équipes, joueurs par équipe, etc.), un message l'informe qu'il doit procéder à la création des équipes (Fig. 3.12) :

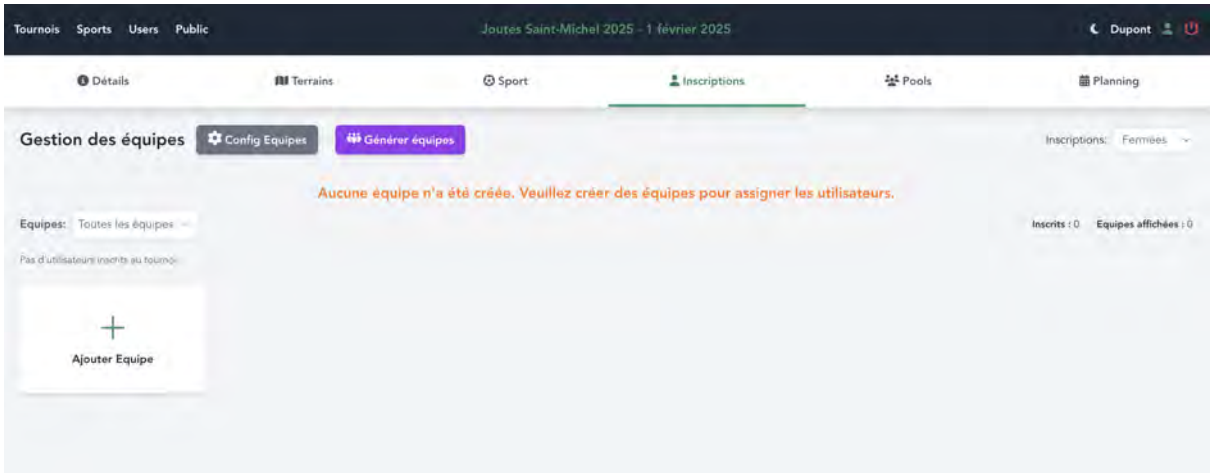


Fig. 3.12.: Équipes manquantes

Note : Tous les processus et messages d'information ne seront pas forcément affichés. Cette page illustre simplement l'existence d'un système d'assistance intégré.

M. Dupont crée alors automatiquement ses équipes, car le nombre est trop important pour une création manuelle (Fig. 3.13) :

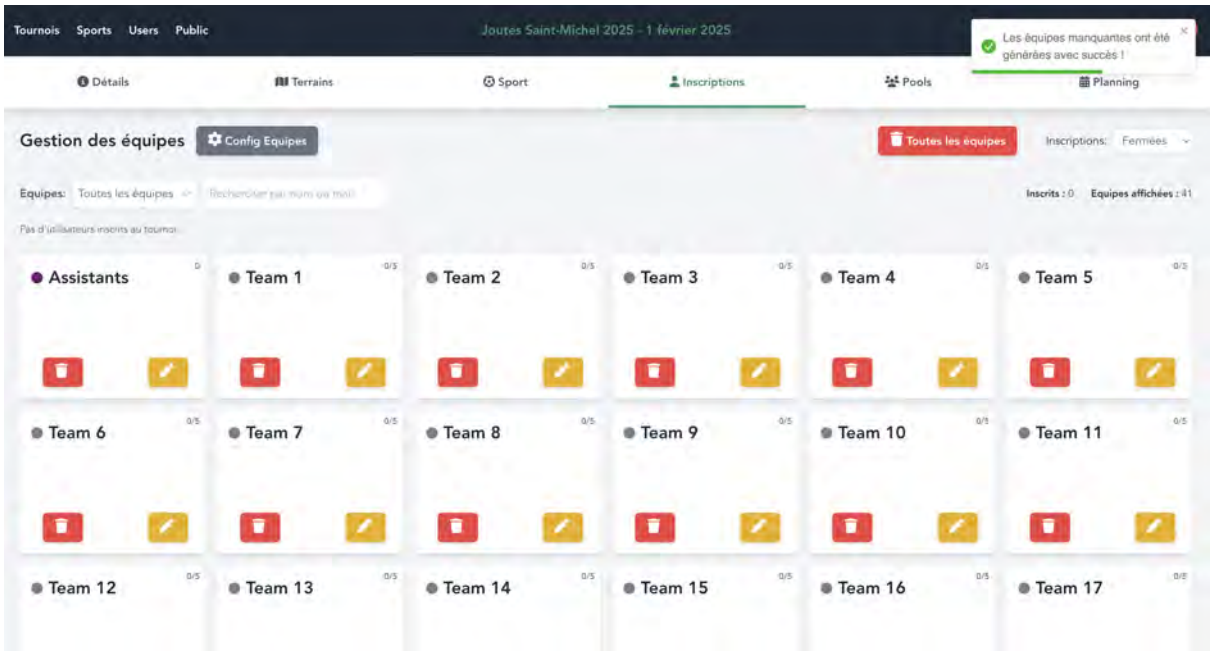


Fig. 3.13.: Génération d'équipes

La génération aboutit à 41 équipes, dont 40 pour les joueurs et 1 nommée **Assistants**, destinée à l'organisation/arbitrage. Cette dernière n'est pas prise en compte dans les pools ou les matchs.

Création des pools

Une fois les équipes configurées, il faut les répartir en pools. M. Dupont effectue les étapes ci-dessous pour assigner rapidement les équipes aux pools.

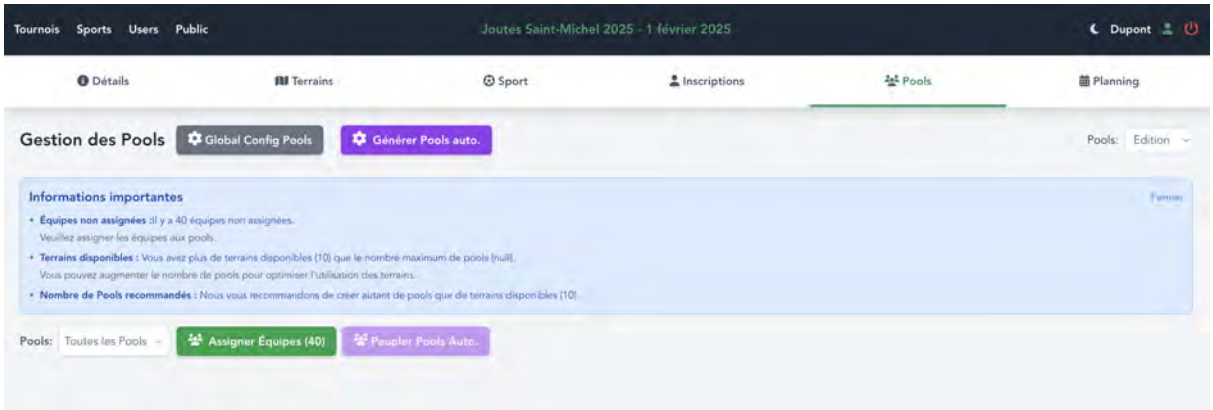


Fig. 3.14.: Message introductif pour la création des pools

La page des pools (Fig. 3.14) comprend :

1. Un bouton pour configurer les pools.
2. Un bouton pour générer automatiquement les pools selon le type de tournoi.
3. Une section informative sur les éléments manquants.
4. Un filtre pour afficher les pools pleins, vides ou invalides.

5. Des boutons pour attribuer manuellement ou automatiquement les équipes aux pools.

M. Dupont configure les pools, puis clique sur **Générer pools auto.**, qui crée 10 pools et y associe les équipes configurées.

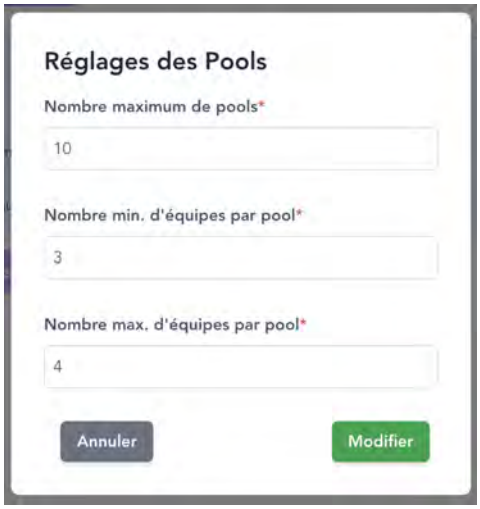


Fig. 3.15.: Configuration des pools

Fig. 3.16.: Génération automatique des pools

Les pools sont créés avec leurs équipes :

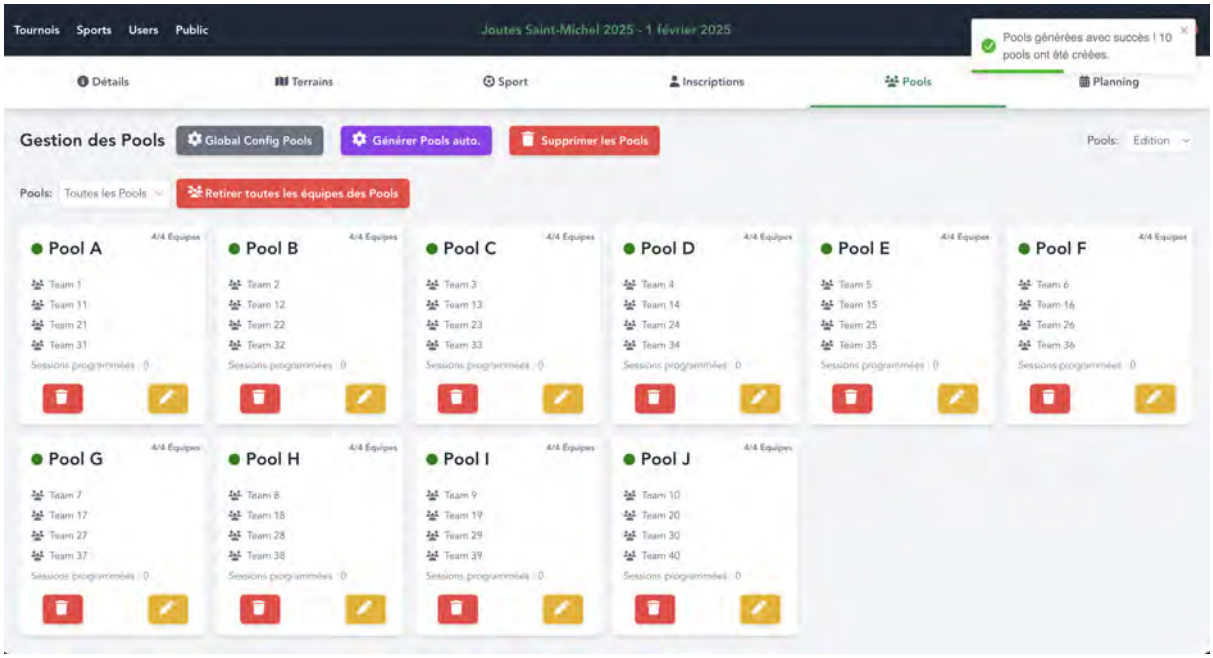


Fig. 3.17.: Pools générés automatiquement

Gestion des pools

Plusieurs fonctionnalités permettent ensuite de gérer les pools:

Suppression d'une équipe d'un pool: M. Dupont peut décider de retirer une équipe si nécessaire (Fig. 3.18).

Fig. 3.18.: Suppression d'une équipe d'un pool

Gestion des équipes non assignées: Si une équipe n'est pas assignée à un pool, elle apparaît dans la section **Assigner Équipes**.

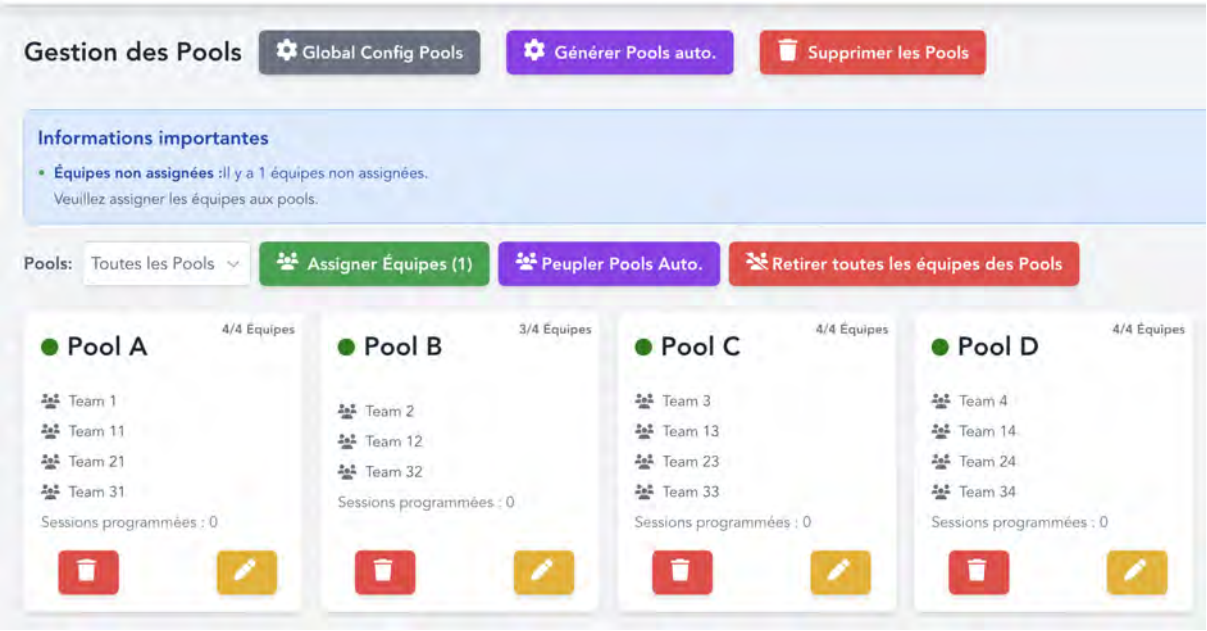


Fig. 3.19.: Équipes non assignées

Assignation d'une équipe à un pool: L'interface (Fig. 3.20) autorise l'ajout rapide d'une équipe non assignée à une pool existante.

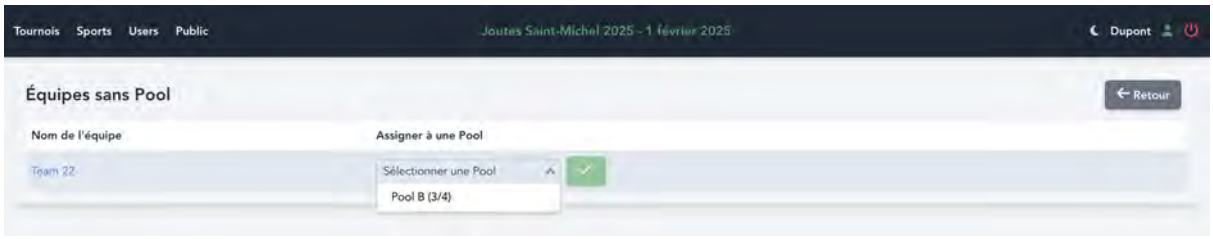


Fig. 3.20.: Assignation d'une équipe à un pool

Une fois toutes les modifications effectuées, l'état final des pools s'affiche :

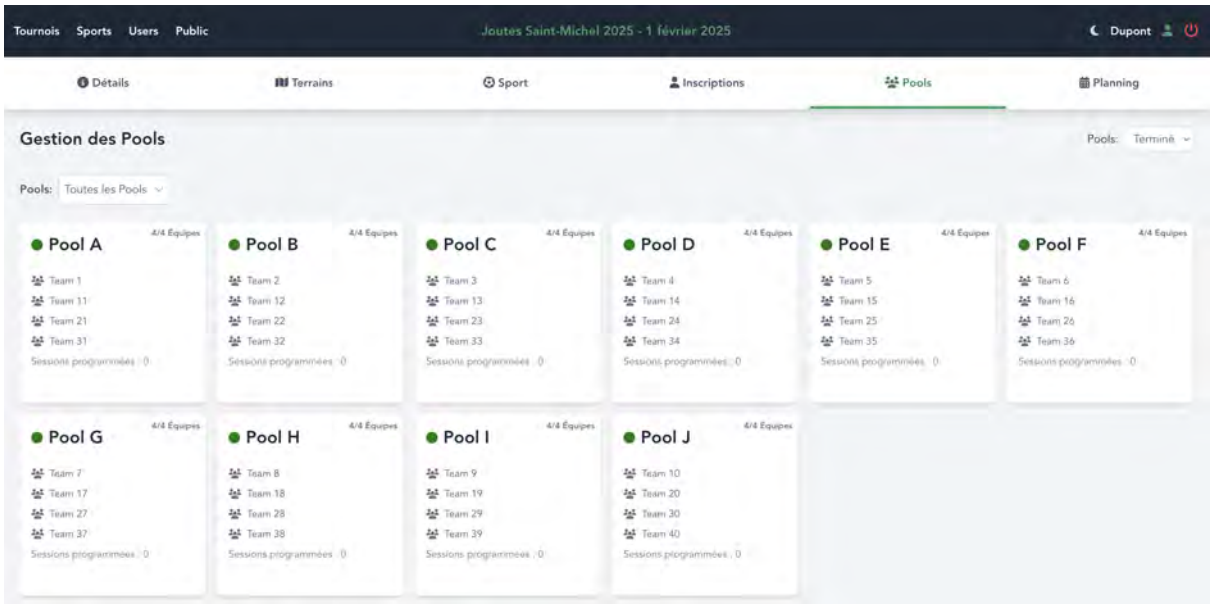


Fig. 3.21.: État final des pools

Des informations de synthèse permettent une visualisation rapide de chaque pool :

1. Capacité x/y, où x est le nombre d'équipes présentes et y le nombre maximum autorisé.
2. Statut de couleur (vert = valide, orange = invalide, gris = vide).
3. Liste des équipes du pool.
4. Nombre de sessions du pool (chaque session correspondant à un créneau horaire dans le planning).

3.4.4. Gestion du planning

La dernière étape consiste à générer le planning. Pour ce type de tournoi, il faut d'abord créer le planning des pools, puis celui des matchs. Une configuration manuelle est possible pour chaque session, mais l'application propose aussi une option de génération automatique.

Planning des pools

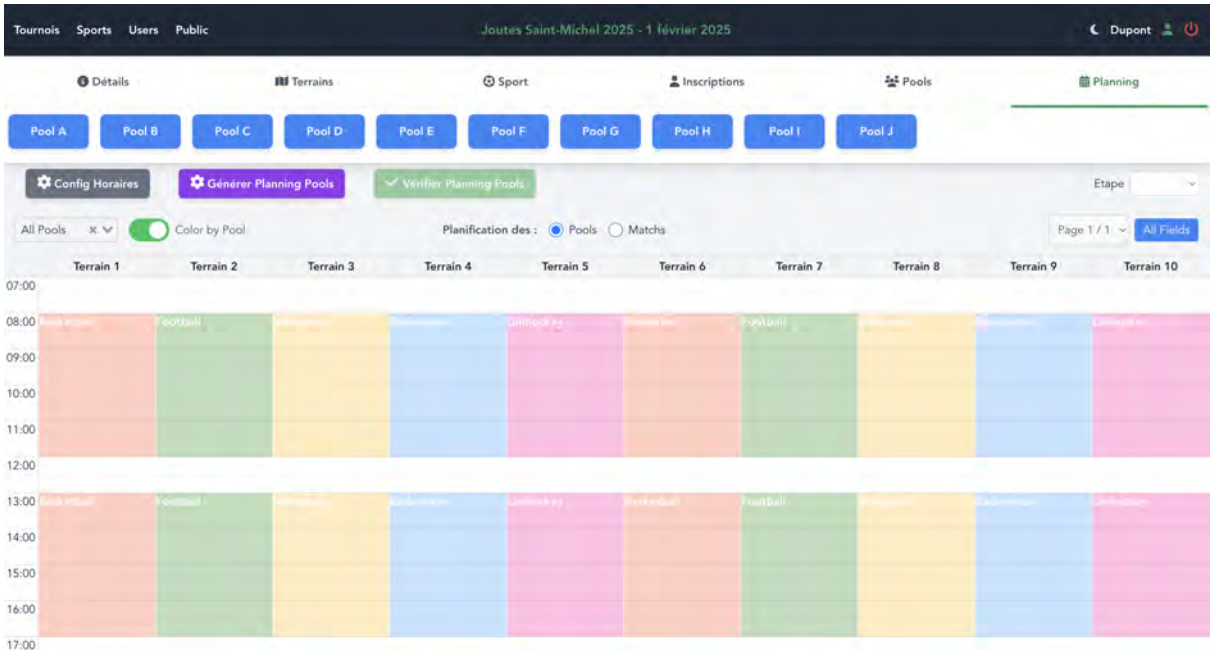


Fig. 3.22.: Page de Planning des Pools vide

La Figure 3.22 montre un système similaire à l'assignation sport-terrains: on peut glisser et déposer les pools sur le calendrier. Le calendrier présente, en arrière-plan, les créneaux horaires et sports déjà associés aux terrains pour une vue intuitive. Sur cette page, il est possible de:

1. **Configurer les horaires :** Définir l'horaire général du tournoi (début, fin, pauses, etc.).
2. **Générer Planning Pools :** Générer automatiquement un planning de pools selon les configurations horaires et les sports prévus.
3. **Vérifier Planning Pools :** S'assurer qu'il n'y a pas de conflits (même pool ou même terrain sur le même créneau).
4. **Filtres et couleurs :** Filtrer par pool ou par sport, et changer le code couleur.
5. **Planification des Pools/Matchs :** Permettre de basculer entre la configuration des pools et celle des matchs. Pour un *CustomRoundRobin*, on configure d'abord les pools, puis les matchs.
6. **Système de pagination :** Pour s'adapter à la taille de l'écran ou au nombre de terrains. Il reste possible d'afficher tous les terrains, mais l'ergonomie peut alors s'en trouver réduite.

Configuration des horaires

M. Dupont vérifie les horaires préconfigurés pour chaque tournoi et ajuste, si nécessaire, la durée des matchs, des pauses et des transitions (Fig. 3.23).

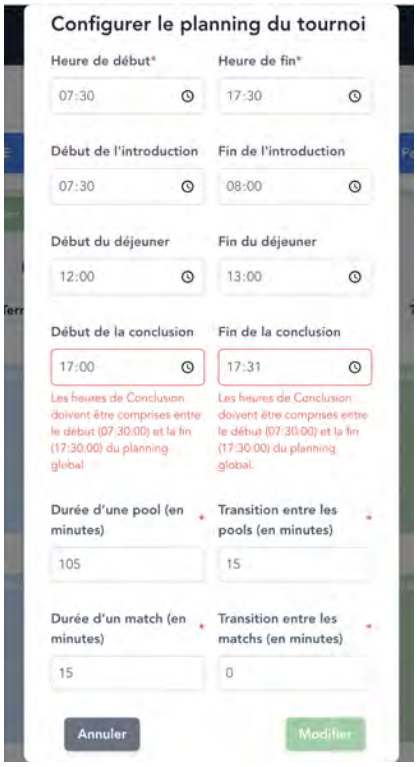


Fig. 3.23.: Configuration des horaires

- **Heure de début/fin :** Plage horaire globale du tournoi.
- **Introduction :** Temps pour accueillir les élèves, présenter le tournoi et expliquer les règles.
- **Déjeuner :** Pause de midi.
- **Conclusion :** Remise des résultats, discours de fin.
- **Durée d'un pool :** Temps alloué aux pools.
- **Transition entre pools :** Temps de pause ou de déplacement.
- **Durée d'un match :** Durée moyenne d'un match au sein d'un pool.
- **Transition entre matchs :** Intervalle entre deux matchs.

Note : Un système de validation empêche par exemple de placer la conclusion en dehors des bornes horaires du tournoi.

Vérification de validation

Pour illustrer l'outil de validation, M. Dupont effectue volontairement une planification erronée (Fig. 3.24) :

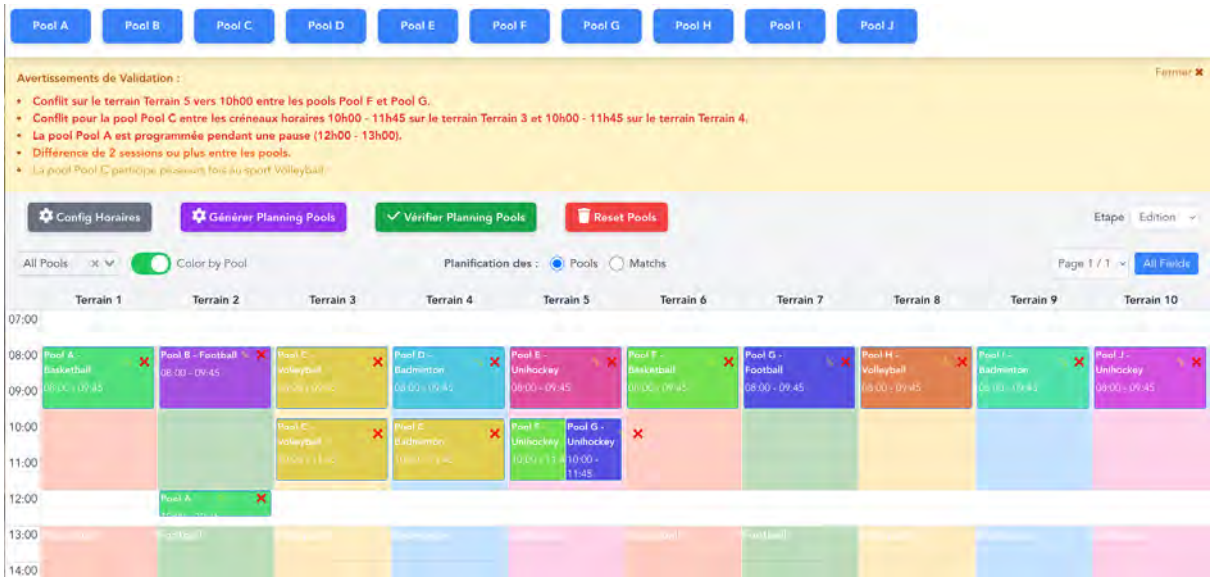


Fig. 3.24.: Conflits dans le Planning

L'algorithme détecte par exemple :

1. Les pools F et G sur le même terrain au même créneau (erreur critique).
2. Le pool C jouant simultanément sur deux terrains (erreur critique).
3. Le pool A configurée pendant une pause (erreur critique).
4. Le pool C ayant deux sessions de plus que les autres (erreur moyenne).
5. Le pool C participant plusieurs fois au même sport (erreur mineure).

Des oublis surviennent facilement dans les plannings complexes. Cet outil les détecte, mais n'empêche pas l'utilisateur de faire des choix.

Génération automatique du Planning des Pools

M. Dupont comprend qu'il lui sera difficile de tout planifier manuellement et utilise alors la fonctionnalité de génération automatique via **Générer Planning Pools**. Il obtient un planning sans erreur (Fig. 3.25) :

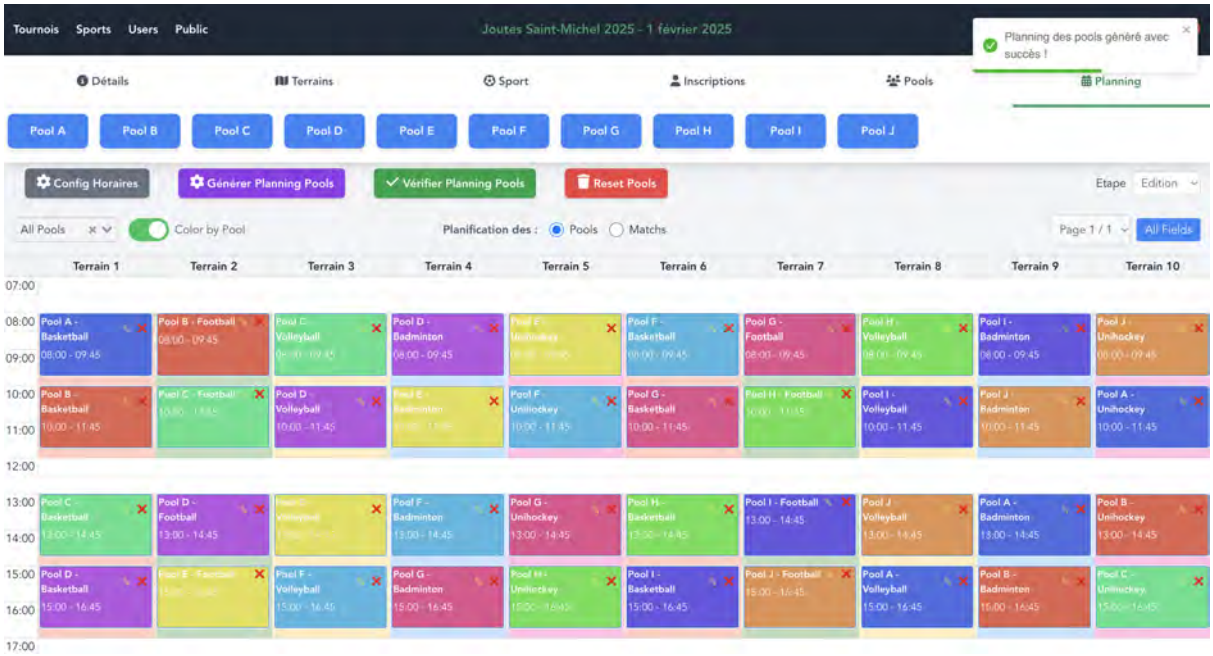


Fig. 3.25.: Planning des Pools généré

Les temps de transition sont correctement insérés, et la vérification ne signale aucune erreur. Les pools sont aussi différenciées par couleurs pour une visibilité optimale.

Planning des matchs

Une fois le planning des pools configuré, M. Dupont passe à la planification des matchs (Fig. 3.26):

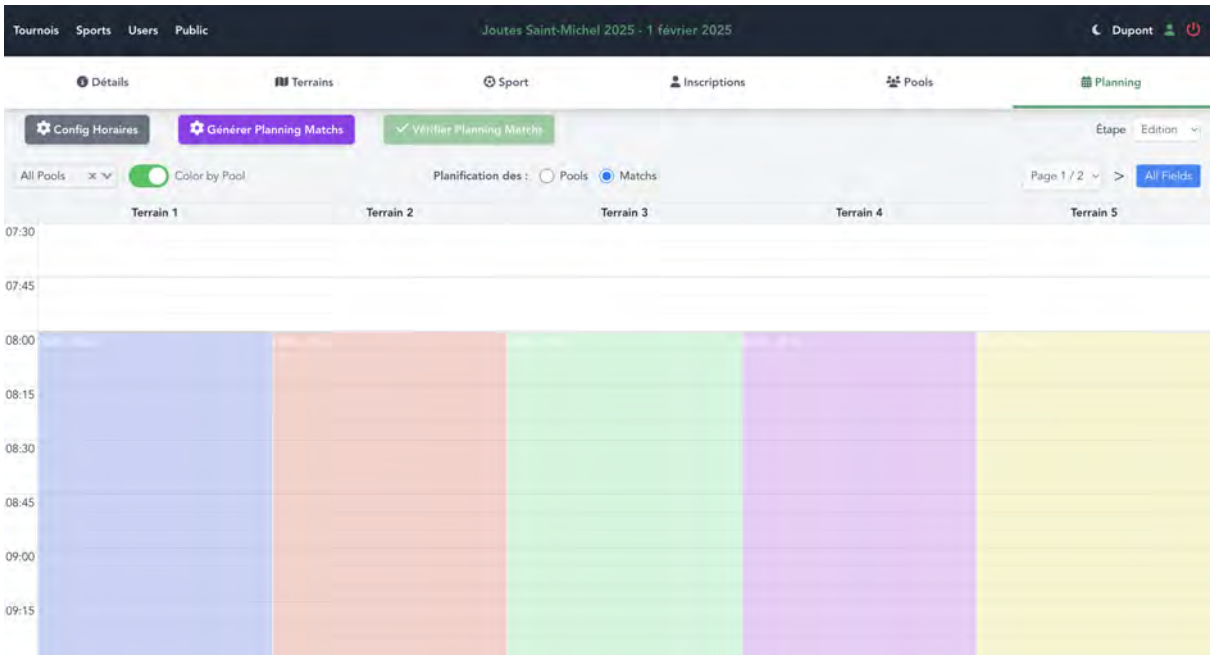


Fig. 3.26.: Planning des Matchs vide

Le calendrier est désormais plus précis (tranches de 15 minutes). Il est possible de créer un match (*game*) en cliquant sur un créneau disponible :

Fig. 3.27.: Création d'un match dans un pool

L'application reconnaît le pool cliqué et suggère les équipes associées. Il est aussi possible de créer un match en dehors d'un créneau pool (Fig. 3.28); dans ce cas, l'utilisateur choisit librement les équipes, le terrain et le sport (pratique pour des demandes hors standard) :

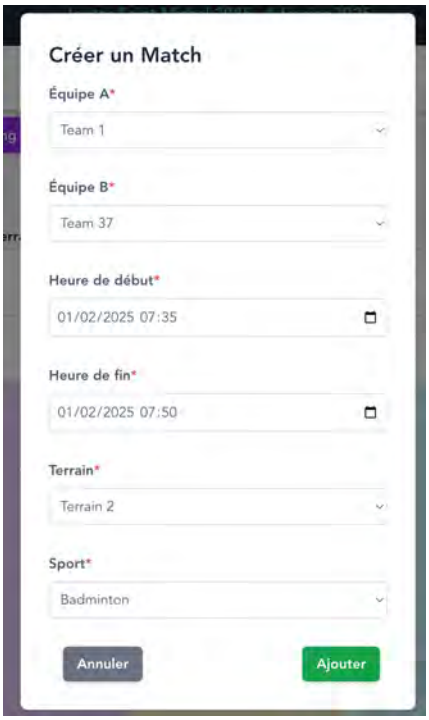


Fig. 3.28.: Création d'un match général

Note : La création manuelle d'un grand nombre de matchs peut devenir fastidieuse et source d'erreurs. C'est pourquoi l'application propose un module de génération automatique de matchs, assurant un équilibrage optimal et limitant les conflits dans la mesure du possible.

M. Dupont finalise donc la préparation du tournoi en cliquant sur **Générer Planning Matchs** (Fig. 3.29) :

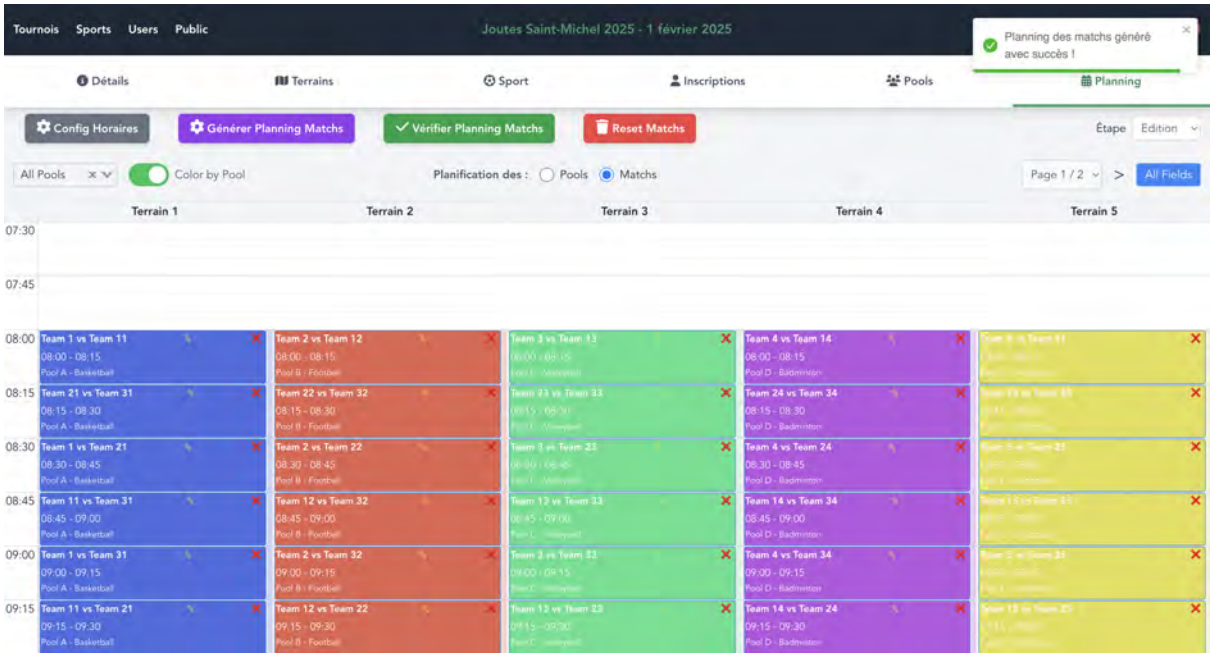


Fig. 3.29.: Génération des matchs

Les matchs sont alors correctement planifiés sur l'ensemble de la journée.

3.4.5. Finalisation de la préparation

Après s'être assuré que toutes les configurations sont complètes (passage de l'état **Édition** à **Terminé**), M. Dupont vérifie la page d'informations globales du tournoi (Fig. 3.30) :

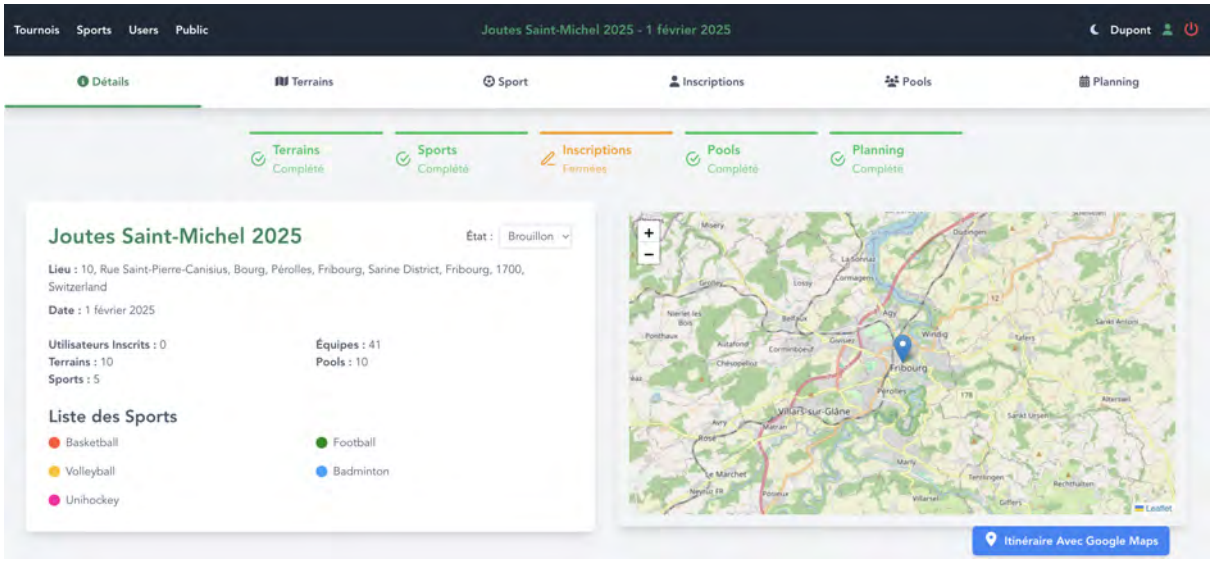


Fig. 3.30.: État de progression du tournoi

Le tournoi étant correctement configuré, la phase d'inscription peut débuter.

3.4.6. Conclusion du Scénario 1

Ce premier scénario a couvert l'ensemble du processus de planification d'un tournoi. L'administrateur a créé un tournoi en y associant terrains, sports, équipes et pools, et en établissant un planning détaillé. Les fonctions de génération automatique facilitent grandement l'organisation et réduisent les risques d'erreur. Il en résulte un gain de temps notable et un déroulement plus fluide, essentiel à la réussite de l'événement.

3.5. Scénario 2 : Gestion des inscriptions

3.5.1. Contexte

Après avoir terminé la configuration du tournoi, M. Dupont ouvre les inscriptions en générant un lien d'invitation à partager avec les élèves afin qu'ils puissent rejoindre une équipe.

3.5.2. Génération de liens d'invitation

M. Dupont retourne sur la page *Inscriptions* et change le statut des inscriptions de **Fermées** à **Ouvertes**. Un nouveau bouton **Lien d'invitation** apparaît. Il clique dessus, ouvrant une modale de gestion des liens d'invitation (Fig. 3.31).

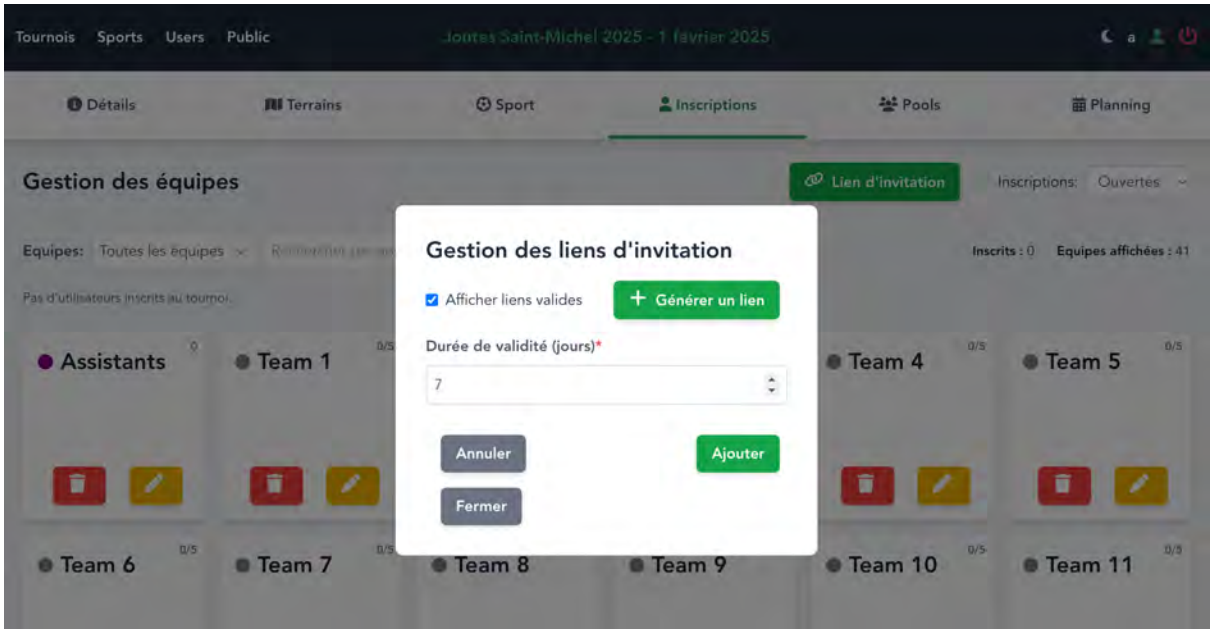


Fig. 3.31.: Génération de liens d'invitation

Cette interface permet de générer un lien d'invitation en précisant sa durée de validité en jours.

Gestion de plusieurs liens d'invitation

Il est possible de générer plusieurs liens d'invitation pour un même tournoi (Fig. 3.32). Chaque lien correspond à une URL (*Uniform Resource Locator*) contenant un *token* d'accès. Toute personne disposant de ce lien peut alors s'inscrire au tournoi. Pour les tournois où la liste des participants est connue à l'avance, il serait envisageable de créer des liens uniques, de sorte que chaque participant dispose de son propre accès sécurisé. Toutefois, dans le cadre d'événements plus ouverts, tels qu'un tournoi universitaire rassemblant un public varié, un lien générique est privilégié afin de simplifier la diffusion.

Fig. 3.32.: Gestion des liens d'invitation

Un administrateur peut activer ou désactiver un lien, par exemple en cas de fuite. Les liens s'affichent en rouge s'ils sont expirés, en jaune s'ils expirent dans les trois jours, et en vert s'ils restent valides au-delà. Ces liens d'invitation, conçus pour être partagés, permettent aux élèves de créer ou de se connecter à un compte afin de rejoindre le tournoi et une équipe.

Partage du lien

M. Dupont génère un lien d'invitation et le partage par email avec les élèves. Voici un exemple typique de mail qu'il pourrait envoyer aux classes de 1re année et 2e année du collège :

Chers élèves,
Nous sommes ravis d'organiser et d'accueillir les joutes sportives pour les classes de 1re et 2e année gymnasiales !
Pour participer, voici la marche à suivre :

1. ***Cliquer sur le lien suivant*** *pour accéder à la plateforme d'inscription :* `https://easytourneyXYZ.ch/register?/inviteToken=xXxXFakeTokenXxXx`
2. ***Créer un compte ou vous connecter*** *si vous en avez déjà un avec votre adresse email de l'école.*
3. ***Rejoindre une équipe*** *: une équipe valide se compose de 3 à 5 joueurs maximum. Vous disposez de* ***2 semaines*** *pour constituer vos équipes avant la fermeture des inscriptions.*

4. *Si vous bénéficiez d'une dispense ou n'êtes pas en mesure de jouer, rejoignez le groupe* ***"Assistants"****. Votre rôle sera d'aider à l'organisation.*
5. *Attention, les* ***étudiants qui ne seront pas inscrits à une équipe dans les délais*** *seront automatiquement affectés à une équipe.*

Nous comptons sur votre engagement, votre bonne humeur et votre esprit sportif ! Inscrivez-vous dès maintenant. Des informations sur la journée des joutes vous seront communiquées une fois les inscriptions terminées.
Votre professeur de sport, M. Dupont

3.5.3. Point de vue élève - lien d'invitation

Le scénario se déroule maintenant du point de vue d'un élève nommé *Demo1*. Il a reçu le mail de M. Dupont et a cliqué sur le lien d'invitation pour accéder à une page de création de compte (Fig. 3.33).

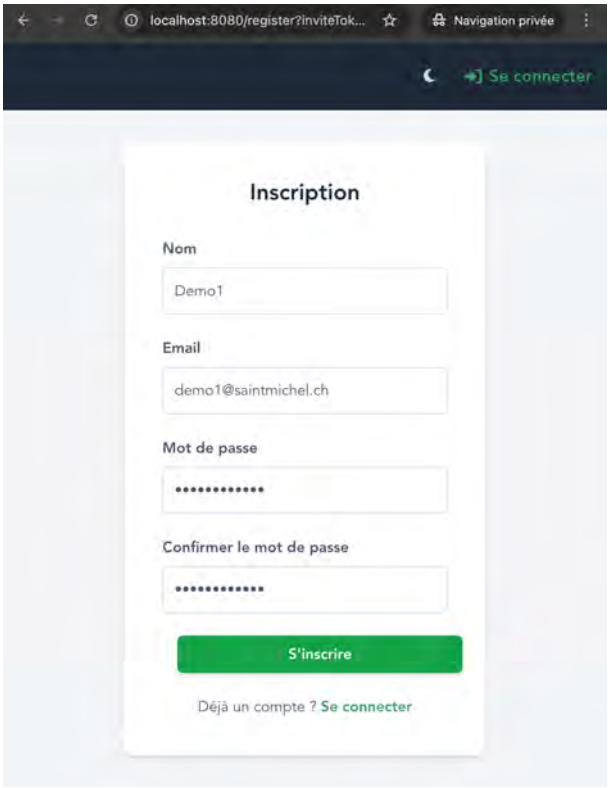


Fig. 3.33.: Création de compte

L'élève crée son compte en renseignant les informations demandées. Un système de vérification de boîte mail n'a pas été mis en place afin de simplifier les tests durant la phase de développement.
Une fois inscrit, l'élève est redirigé directement vers la liste des équipes du tournoi.

3.5.4. Rejoindre une équipe

Afin de simuler un scénario réel, des utilisateurs ont été créés en amont via un seeder. Certains élèves ont rejoint des équipes, d'autres sont sans équipe et 10 élèves ont rejoint le

groupe "Assistant". Ceci permet de mieux comprendre le scénario et les fonctionnalités du tournoi.

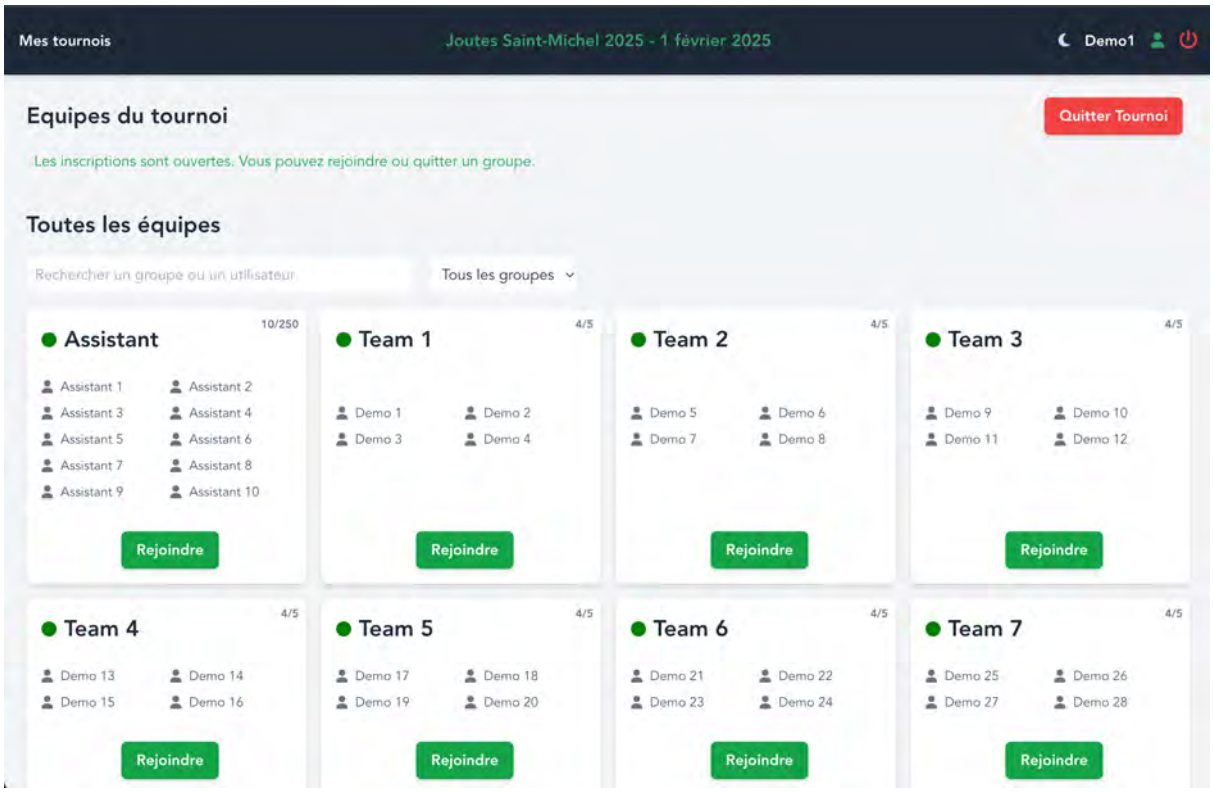


Fig. 3.34.: Rejoindre une équipe

A ce stade, l'utilisateur est inscrit au tournoi. Il doit encore rejoindre une équipe. La Figure 3.34 permet de visualiser toutes les équipes du tournoi avec leurs membres. Voici les fonctionalités présentes:

- Visualiser et rejoindre une équipe.
- Quitter le tournoi lorsque les inscriptions sont ouvertes.
- Indication d'un message informatif selon le statut des inscriptions (ici qu'il faut rejoindre un groupe).
- Un système de filtres permet de chercher un groupe spécifique (par exemple, si un ami mentionne qu'il est dans la team 38) ou de rechercher directement un utilisateur.

L'utilisateur Demo1 choisit de rejoindre le groupe "Assistant" afin de démontrer les fonctionnalités d'arbitrage par la suite (Fig. 3.35) :

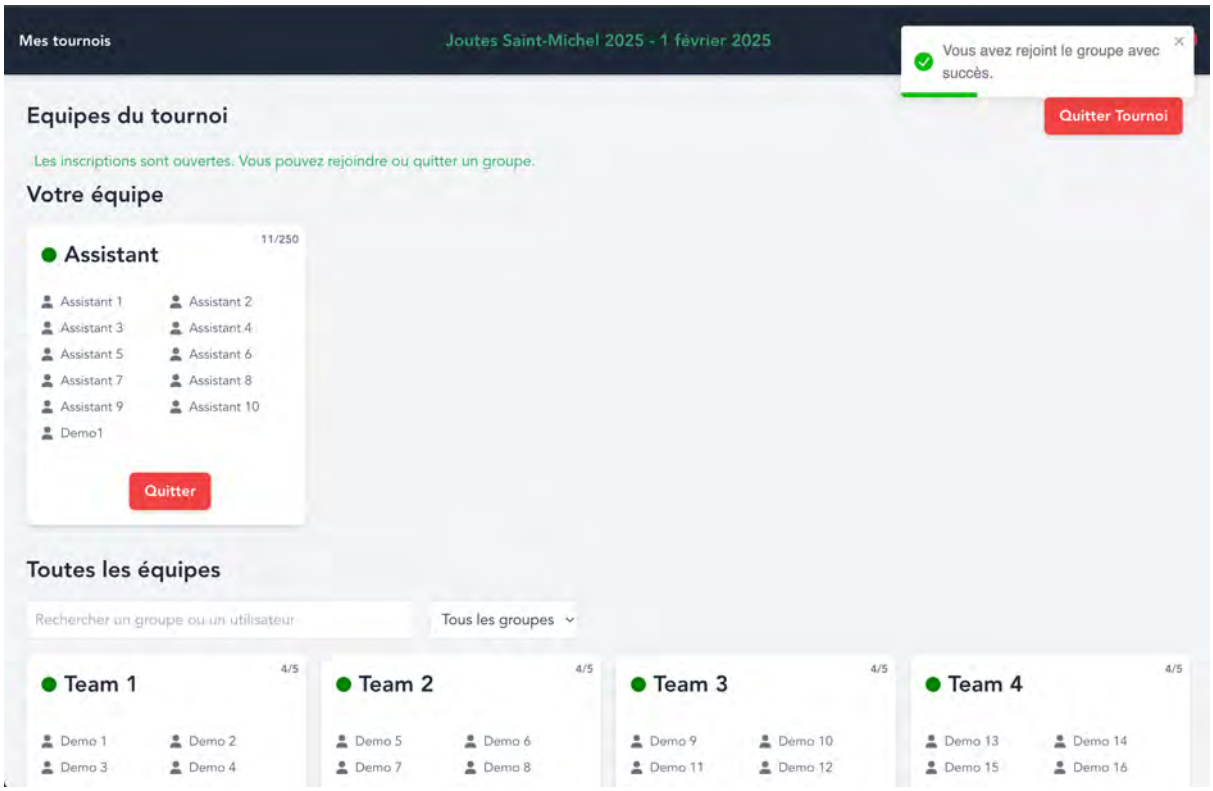


Fig. 3.35.: Équipe rejointe

Il est possible de quitter l'équipe pour en rejoindre une autre.
En cliquant sur une équipe, plus d'informations s'affichent (Fig. 3.36) :

Fig. 3.36.: Liste d'une équipe

3.5.5. Fonctionnalités annexes utilisateur

Demo1 ayant rejoint une équipe, il ne lui reste plus qu'à patienter jusqu'à la fin des inscriptions et le début du tournoi. L'utilisateur peut accéder à ses tournois en cliquant sur le menu *Mes tournois* (Fig. 3.37) afin de visualiser ses différentes inscriptions:

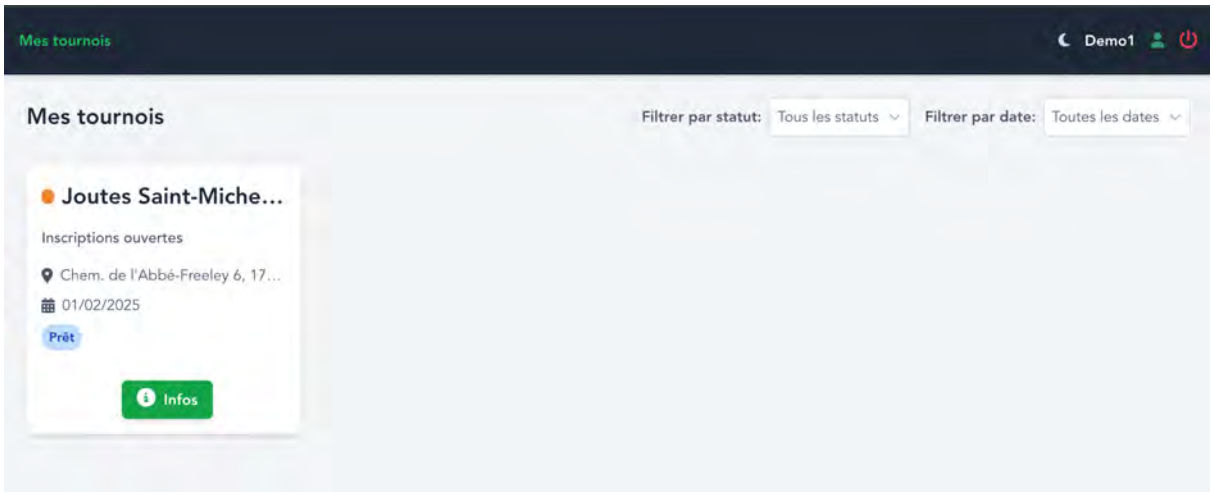


Fig. 3.37.: Liste des tournois utilisateur

Demo1 s'est inscrit à un seul tournoi. Plus de détails sont disponibles en cliquant sur *Infos* (Fig. 3.38):

Détails du tournoi

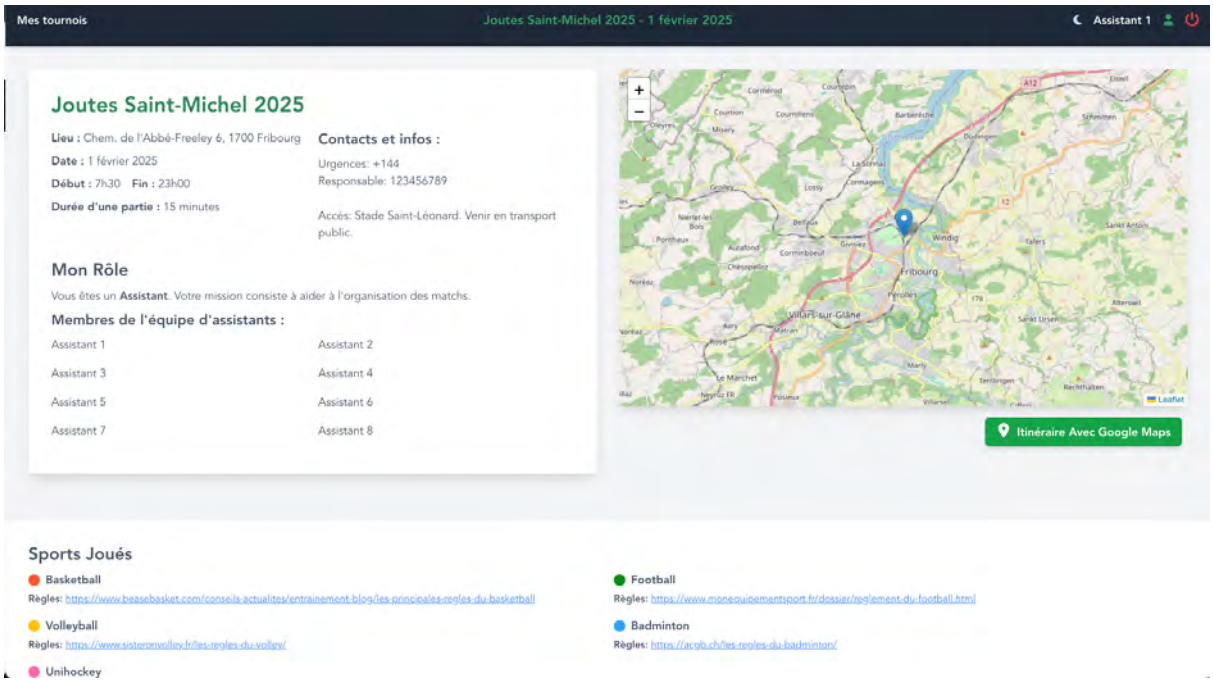


Fig. 3.38.: Détails du tournoi

Diverses informations sont présentées :

1. Le lieu, la date, l'heure de début et de fin du tournoi.
2. Les contacts et autres informations diverses du tournoi.
3. Le rôle de l'utilisateur ainsi que les membres de son équipe.
4. Un carte avec le lieu et un accès à l'itinéraire via Google Maps.
5. Les différents sports du tournoi ainsi que leurs règlements. L'élève pourra déjà prendre connaissance des différents règlements, notamment s'il est arbitre.

Cette page est importante pour fournir à l'utilisateur un aperçu complet du tournoi et lui permettre de mieux se préparer. Qu'il s'agisse de repérer l'emplacement, d'étudier le règlement ou d'organiser son emploi du temps, toutes les informations essentielles sont centralisées ici.

Profil utilisateur

L'utilisateur a également accès à son profil :

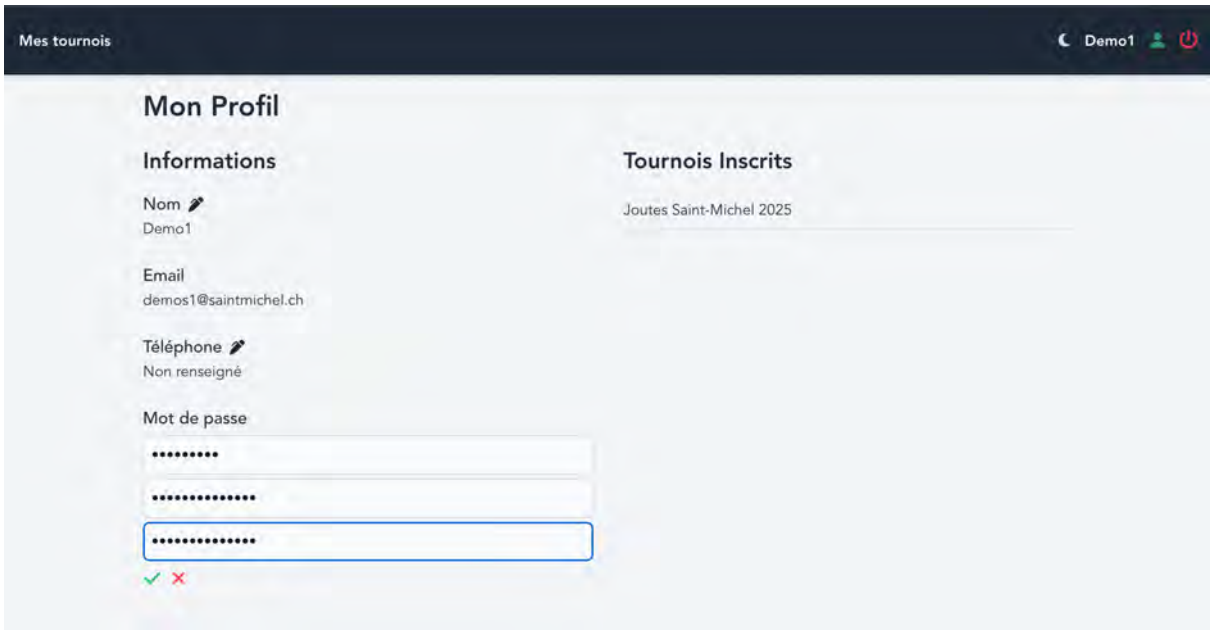


Fig. 3.39.: Page profil utilisateur

La page de la figure 3.39 indique les tournois auxquels l'élève s'est inscrit. Il peut modifier son nom, son téléphone ou son mot de passe. L'adresse email ne peut être modifiée que par l'administrateur, car l'application est destinée à un usage scolaire avec les emails de l'école.

3.5.6. Fin des inscriptions - Administrateur

Les inscriptions sont désormais terminées. L'administrateur vérifie l'état et configure les derniers réglages. M. Dupont se rend à nouveau sur la page *Inscriptions* (Fig. 3.40), change le statut des inscriptions de **Ouvertes** à **Fermées**, ce qui bloque toutes inscriptions même avec un lien d'invitation valide, et filtre pour afficher les équipes invalides :

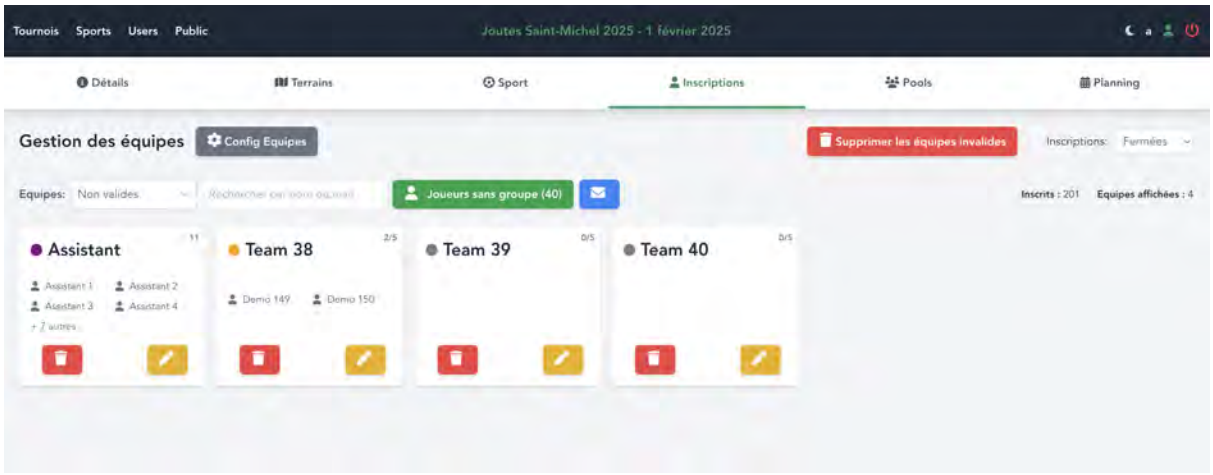


Fig. 3.40.: Gestion des équipes invalides

201 utilisateurs sont inscrits. Les groupes étant limités à 200 joueurs, le surplus pourra rejoindre le groupe "Assistant" ou rester sans groupe.
Le groupe "Assistant" est toujours visible quel que soit le filtre. On peut voir que 40 élèves n'ont pas rejoint de groupe alors qu'il reste apparemment 13 places disponibles. Cependant, ceci est trompeur car certaines équipes valides non affichées sont pleines à 3/5 ou 4/5 joueurs, par exemple. Un nouveau bouton **Joueurs sans groupe (40)** redirige vers une page permettant de gérer les élèves sans groupe (Fig. 3.41) :

Assignation automatique des utilisateurs

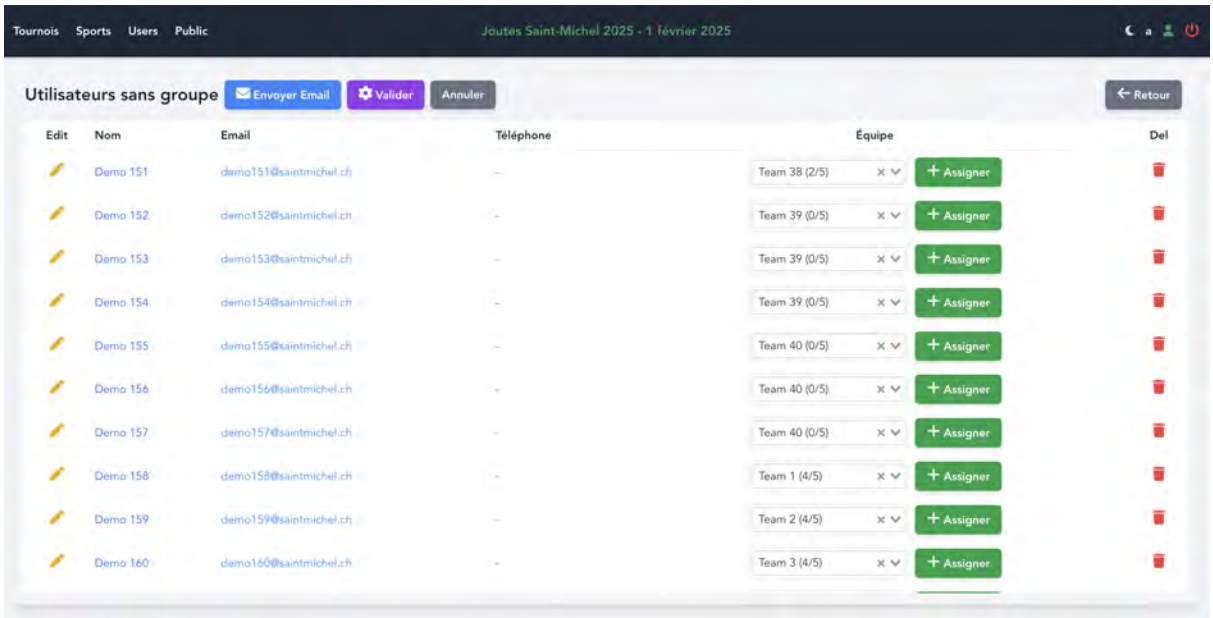


Fig. 3.41.: Assignation automatique des utilisateurs

Cette page liste tous les utilisateurs sans groupe. M. Dupont clique sur le bouton **Assignation automatique**, qui se transforme en bouton **Valider**. Un algorithme prévisualise les assignements des utilisateurs aux équipes, comme illustré sur la Figure

3.41. L'administrateur peut également envoyer un email à tous les élèves pour les informer qu'ils ont été assignés de manière aléatoire à un groupe puis cliquer sur **Valider** pour les assigner.
De nombreuses autres opérations sont disponibles sur cette page, telles que :

1. Accéder directement au profil d'un utilisateur en cliquant sur le bouton *Edit* ou sur son nom.
2. Envoyer un email spécifique à un utilisateur en cliquant sur son adresse email.
3. Assigner manuellement un utilisateur à une équipe en l'associant puis en cliquant sur le bouton **Valider**.
4. Supprimer un utilisateur du tournoi en cliquant sur l'icône de poubelle rouge.

Gestion manuelle des équipes

La gestion manuelle permet de retirer une personne de l'équipe, en l'associant directement à une autre équipe, ou en ajoutant directement un utilisateur sans groupe au sein de l'équipe :

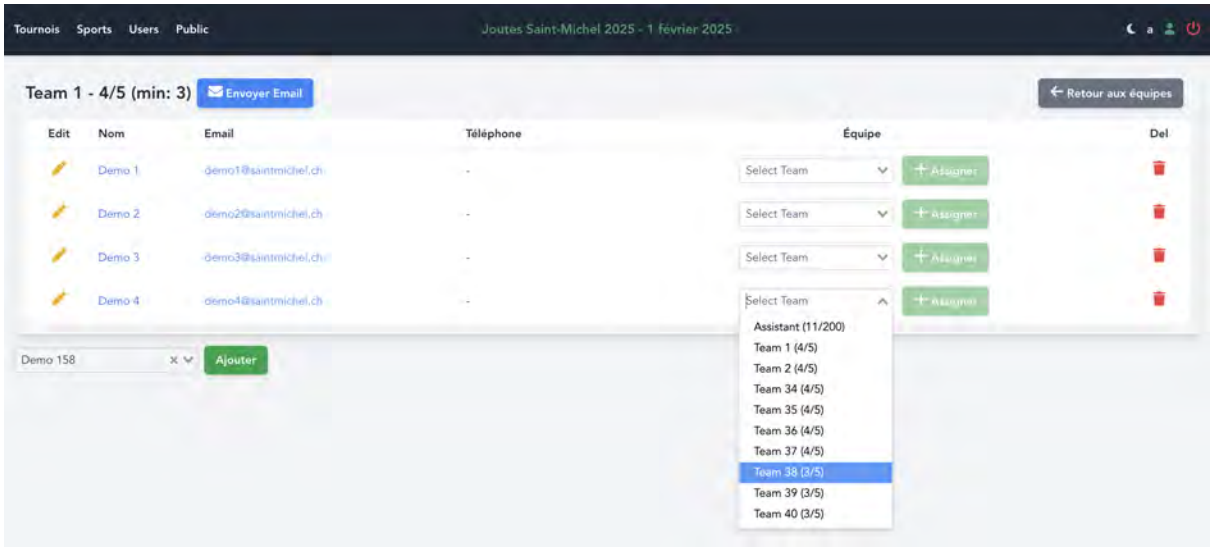


Fig. 3.42.: Gestion manuelle d'une équipe

Le menu déroulant affiche les équipes non pleines où il est possible d'associer directement un utilisateur à cette équipe. Ces différents panneaux de configuration offrent une administration totale et intuitive des inscriptions des utilisateurs pour un administrateur.

3.5.7. Fin des inscriptions

M. Dupont passe le statut des inscriptions de **Fermées** à **Terminées**. La figure 3.43 affiche toutes les équipes avec leurs membres :

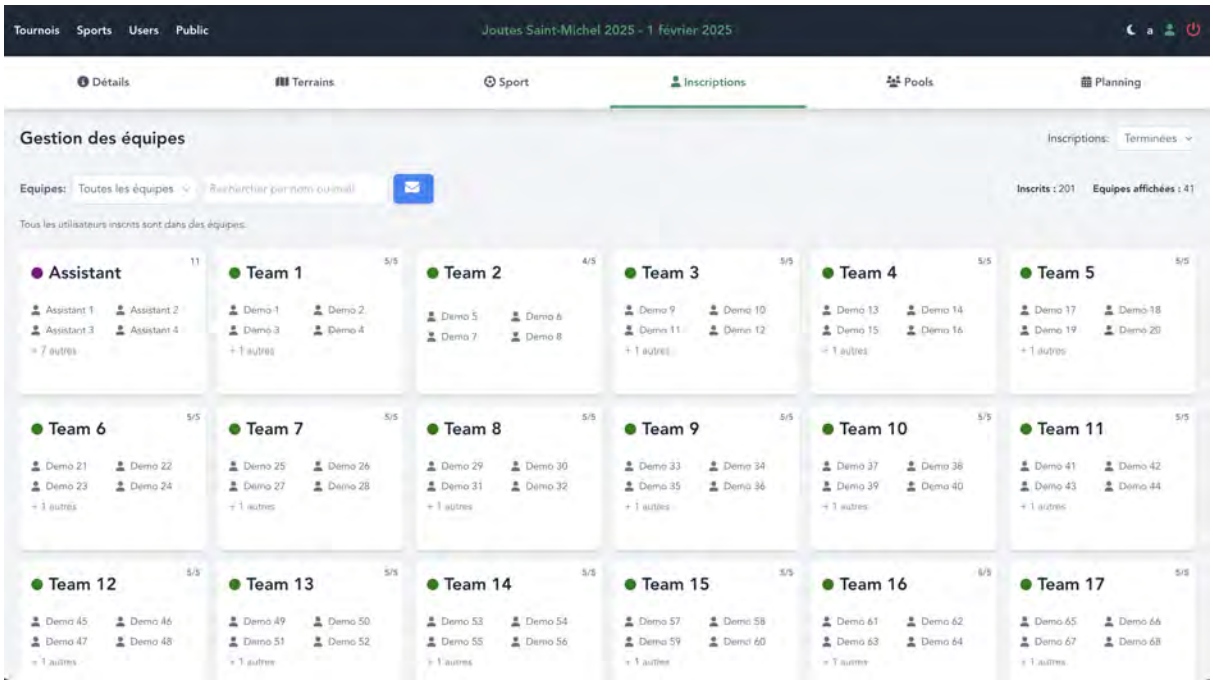


Fig. 3.43.: Inscriptions terminées

Portabilité de l'application

En se rendant à nouveau sur la page *Détails* (Fig. 3.44), M. Dupont constate que toutes les configurations sont terminées.

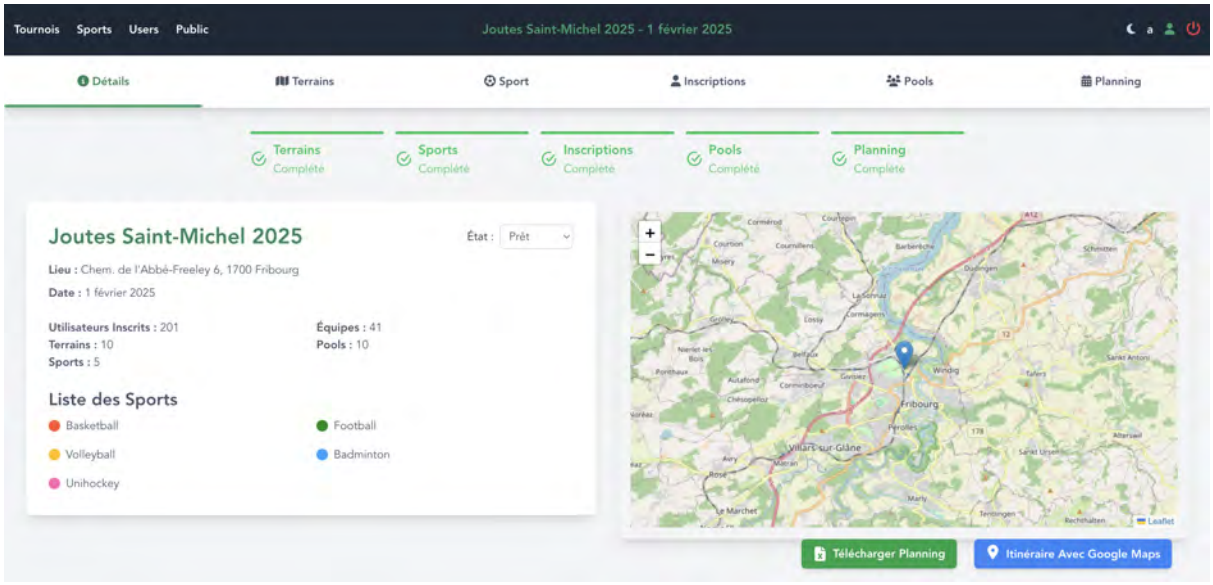


Fig. 3.44.: Statut du tournoi terminé

Important : Un bouton **Télécharger Planning** est apparu. C'est une fonctionnalité très utile car elle permet d'obtenir un fichier Excel du tournoi contenant les plannings, les arbitrages, la liste des utilisateurs, etc. Cela rend l'application portable dans des endroits où il n'y a pas forcément de courant ou de réseau. Je reconnais également qu'il peut

être contraignant de toujours devoir utiliser une application et que parfois, la simplicité fonctionne mieux. Le système d'arbitrage est intéressant mais pas indispensable. La force de cette application réside dans sa gestion automatisée des plannings et la gestion des inscriptions, et le fichier Excel comme support suffit à organiser de manière efficace le tournoi le jour J. Toutefois, j'ai implémenté le système d'arbitrage et de score en temps réel en réponse aux défis technologiques et aux objectifs de mon travail de master.

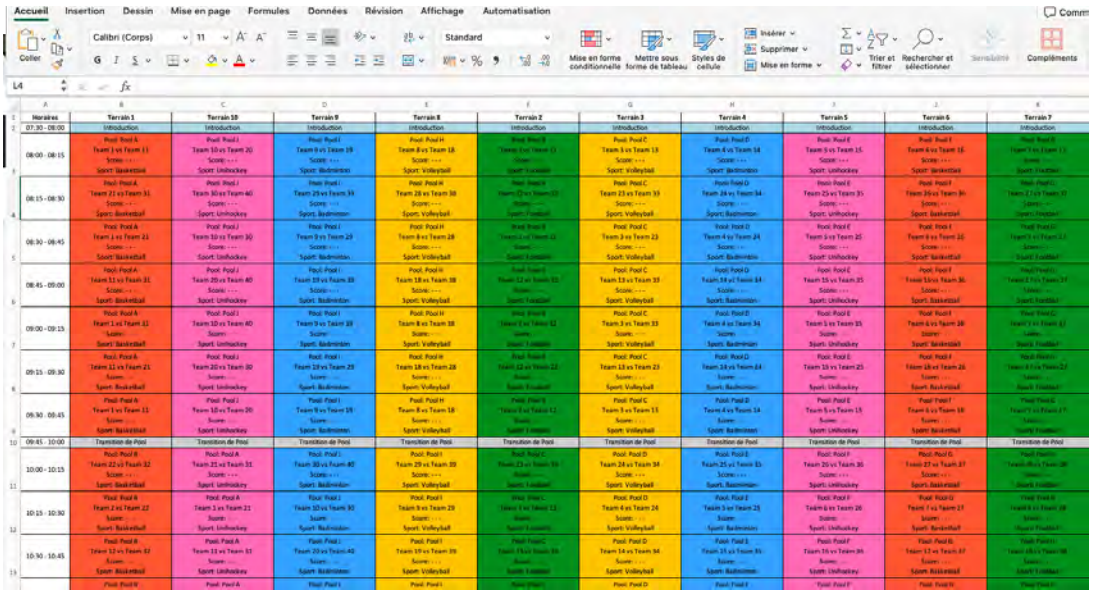
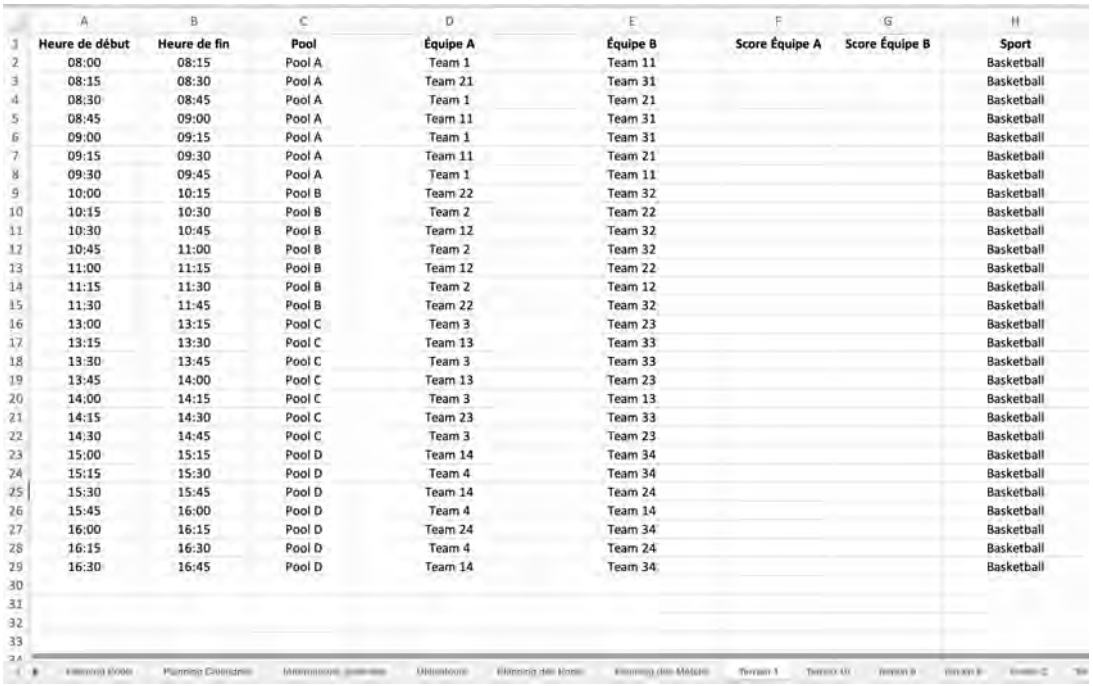


Fig. 3.45.: Excel - Planning des matchs



Heure de début	Heure de fin	Pool	Équipe A	Équipe B	Score Équipe A	Score Équipe B	Sport
08:00	08:15	Pool A	Team 1	Team 11			Basketball
08:15	08:30	Pool A	Team 21	Team 31			Basketball
08:30	08:45	Pool A	Team 1	Team 21			Basketball
08:45	09:00	Pool A	Team 11	Team 31			Basketball
09:00	09:15	Pool A	Team 1	Team 31			Basketball
09:15	09:30	Pool A	Team 11	Team 21			Basketball
09:30	09:45	Pool A	Team 1	Team 11			Basketball
10:00	10:15	Pool B	Team 22	Team 32			Basketball
10:15	10:30	Pool B	Team 2	Team 22			Basketball
10:30	10:45	Pool B	Team 12	Team 32			Basketball
10:45	11:00	Pool B	Team 2	Team 32			Basketball
11:00	11:15	Pool B	Team 12	Team 22			Basketball
11:15	11:30	Pool B	Team 2	Team 12			Basketball
11:30	11:45	Pool B	Team 22	Team 32			Basketball
13:00	13:15	Pool C	Team 3	Team 23			Basketball
13:15	13:30	Pool C	Team 13	Team 33			Basketball
13:30	13:45	Pool C	Team 3	Team 33			Basketball
13:45	14:00	Pool C	Team 13	Team 23			Basketball
14:00	14:15	Pool C	Team 3	Team 13			Basketball
14:15	14:30	Pool C	Team 23	Team 33			Basketball
14:30	14:45	Pool C	Team 3	Team 23			Basketball
15:00	15:15	Pool D	Team 14	Team 34			Basketball
15:15	15:30	Pool D	Team 4	Team 34			Basketball
15:30	15:45	Pool D	Team 14	Team 24			Basketball
15:45	16:00	Pool D	Team 4	Team 14			Basketball
16:00	16:15	Pool D	Team 24	Team 34			Basketball
16:15	16:30	Pool D	Team 4	Team 24			Basketball
16:30	16:45	Pool D	Team 14	Team 34			Basketball

Fig. 3.46.: Excel - Arbitrage des matchs du Terrain 1

La figure 3.46 permet de noter les scores de tous les matchs sur le terrain 1.

3.5.8. Conclusion du Scénario 2

Ce scénario illustre l'intégralité de la gestion des inscriptions. Après la configuration initiale, M. Dupont peut partager un lien d'invitation, suivre les inscriptions et répartir automatiquement les utilisateurs dans les équipes.

Grâce à la portabilité Excel, le tournoi reste gérable même hors ligne, tandis que la force principale d'*Easy Tourney* réside dans sa gestion automatisée des inscriptions et du planning. L'application libère du temps pour l'organisateur et offre un processus d'inscription fluide aux participants.

3.6. Scénario 3 : Jour J - arbitrage et scores

Note : Le tournoi était prévu pour le **1er février** dans le scénario précédent. Pour les besoins de démonstration, la date a été reconfigurée au **9 janvier**, permettant ainsi de tester le système « en direct ».

3.6.1. Introduction et contexte

Le jour J est arrivé. M. Dupont a finalisé toutes les étapes de configuration et de préparation du tournoi (Scénarios 1 et 2). Les élèves sont inscrits et répartis dans leurs équipes, les pools sont définis, et le planning détaillé est en place.

Dans ce **Scénario 3**, nous nous plaçons principalement **du point de vue d'un utilisateur arbitre**, nommé *Demo1*, pour illustrer le déroulement en temps réel du tournoi. Nous aborderons également le point de vue d'un joueur standard, nommé *Demo2*, afin de mettre en avant la consultation de ses prochains matchs.

3.6.2. Connexion de l'arbitre « Demo1 »

Liste des tournois en cours

Le jour du tournoi, l'arbitre *Demo1* se connecte à l'application *Easy Tourney*. Après identification, il accède à son tableau de bord listant les tournois auxquels il participe et constate que le tournoi est passé en statut **En cours** (Fig. 3.47).

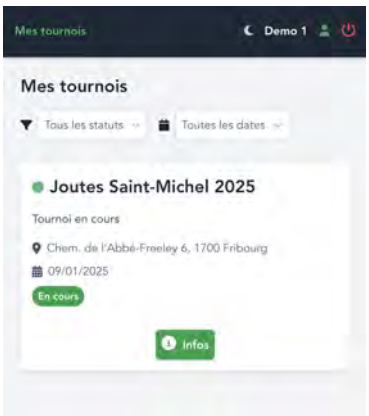


Fig. 3.47.: Tournoi en cours pour l'utilisateur *Demo1*

En cliquant sur la carte du tournoi, *Demo1* accède au contexte du tournoi, qui regroupe la consultation du planning, l'arbitrage des matchs, la visualisation des scores et les informations générales.

3.6.3. Consultation du planning du tournoi

L'interface principale du tournoi propose un **sous-menu** pour naviguer entre :

- **Planning** : Vue des pools et des matchs.
- **Scores** : Classements et résultats en temps réel.
- **Infos** : Informations générales (lieu, contact d'urgence, date, etc.).

Vue planning

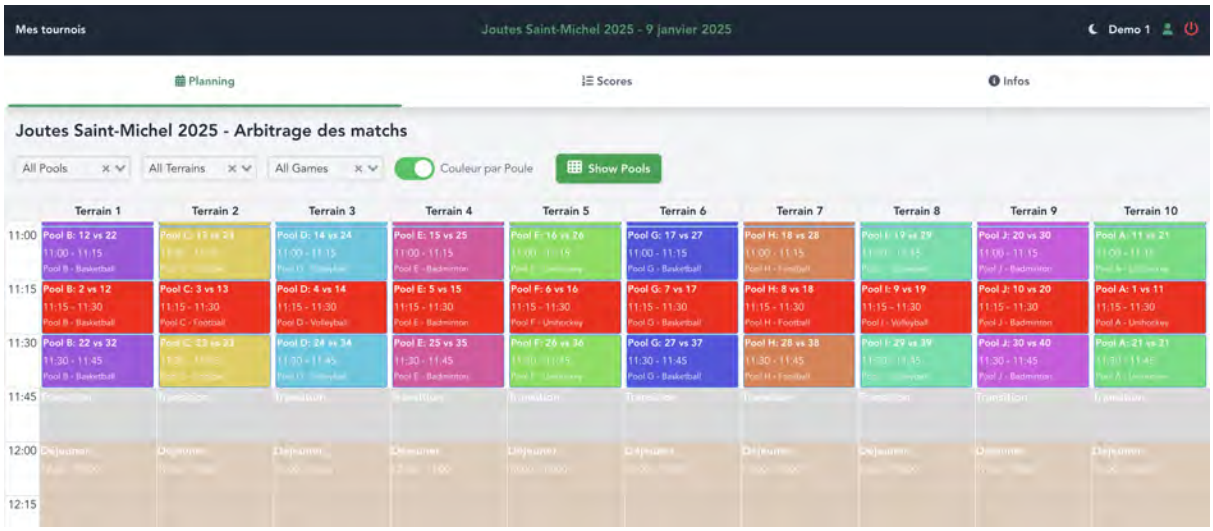


Fig. 3.48.: Planning en mode arbitre

La figure 3.48 présente une **vue globale** du planning, affichant l'ensemble des matchs, des terrains et des sports associés. Cette représentation fournit une **vision d'ensemble** et met en évidence, en rouge, les matchs en cours. Elle permet de visualiser simultanément de nombreuses informations sur une même page.

Pour affiner la recherche, plusieurs **filtres** sont disponibles :

- Filtrer par **pool** (A, B, C, etc.) — pratique pour les joueurs.
- Filtrer par **terrain** — destiné aux arbitres ou assistants.
- Filtrer par **matchID** — utile pour retrouver un match particulier si l'on est arbitre ou administrateur.

Note : Les administrateurs ne peuvent pas, dans l'application, attribuer un arbitre à un terrain de manière fixe. Cette organisation se fait oralement, laissant ainsi la flexibilité nécessaire pour gérer les imprévus, comme l'absence d'un arbitre ou un remplacement de dernière minute. Par exemple, M. Dupont demande à *Demo1* d'arbitrer le terrain 1.

Vue planning par *terrain*

Demo1 applique alors le filtre « Terrain 1 ».

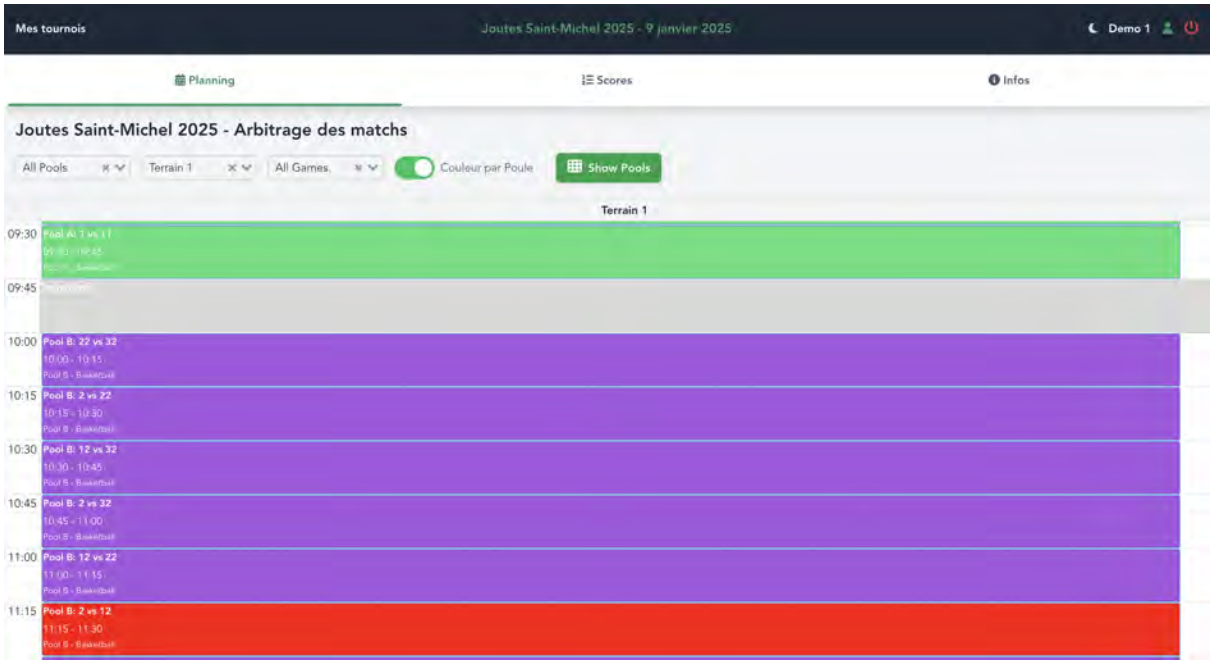


Fig. 3.49.: Planning filtré par terrain

L'arbitre se concentre ainsi uniquement sur ce terrain, sans risquer de gérer un match se déroulant ailleurs.

3.6.4. Arbitrage en temps réel

Demo1 clique sur le match en cours pour l'arbitrer. Il est redirigé vers une **page de détails du match** (Fig. 3.50), utilisant un système de **websocket** pour afficher les scores et les événements en direct, sans besoin de rafraîchir la page.

Fig. 3.50.: Page de Détail du Match pour l'arbitre

Cette interface inclut :

1. En haut à gauche : Un bouton « Retour », l'identité de l'arbitre et le nombre de spectateurs connectés.
2. Au centre : Le titre du match, son *ID*, l'horaire, ainsi que l'affichage du score et du chronomètre pour chaque équipe.
3. En haut à droite : Le statut du match.

Mise à jour de l'état du match

Dans un premier temps, *Demo1* passe le match à l'état « En cours », marquant le début officiel de la rencontre (Fig. 3.51).

Fig. 3.51.: Arbitrage d'un match

Les principales fonctionnalités sont :

1. **Gérer le score en temps réel** : nombre de points.
2. **Enregistrer des événements clés** (p. ex. *but*, *faute*, *carton jaune*, *carton rouge*, etc.).
3. Afficher une **timeline horodatée** retraçant tous les événements dans l'ordre antéchronologique.
4. **Mettre le match en pause**, le **reprendre** ou le **réinitialiser** (ce qui remet à zéro le score, le chronomètre et les événements).
5. **Passer le match à « Terminé »** : enclenche la mise à jour du score final et détermine un vainqueur ou un match nul.

Gestion des événements

L'arbitre *Demo1* peut ajouter, supprimer ou modifier un événement (Fig. 3.52), tandis que tout spectateur connecté peut faire défiler la timeline via une barre de défilement :

Fig. 3.52.: Ajout d'un carton rouge

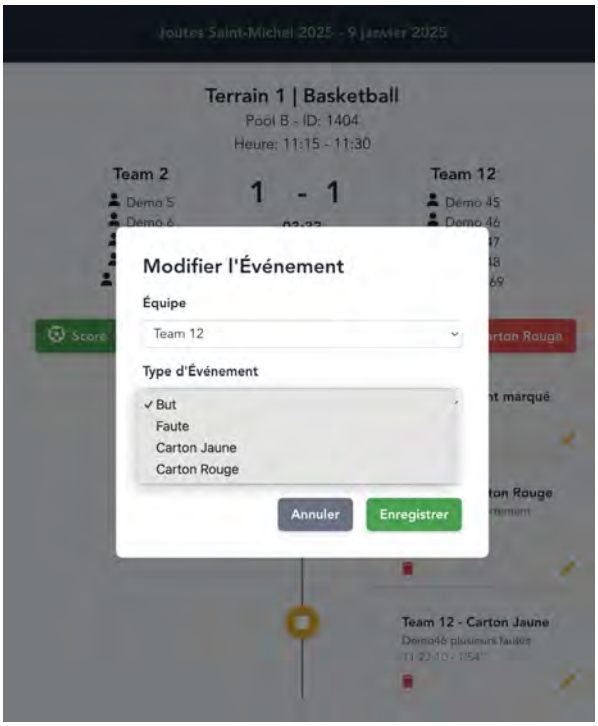


Fig. 3.53.: Modification d'un événement

Fin d'un match et mise à jour en temps réel

Lorsque l'arbitre déclare le match **Terminé**, le score se met automatiquement à jour sur la page dédiée grâce à une synchronisation en temps réel.

La Figure 3.54 illustre, à gauche, l'écran de l'arbitre (*Demo1*) prêt à clôturer un match remporté par la team 31 contre la team 11. À droite, l'écran utilisateur (*Demo2*) montre la team 31 initialement classée 3$^{\text{ème}}$.

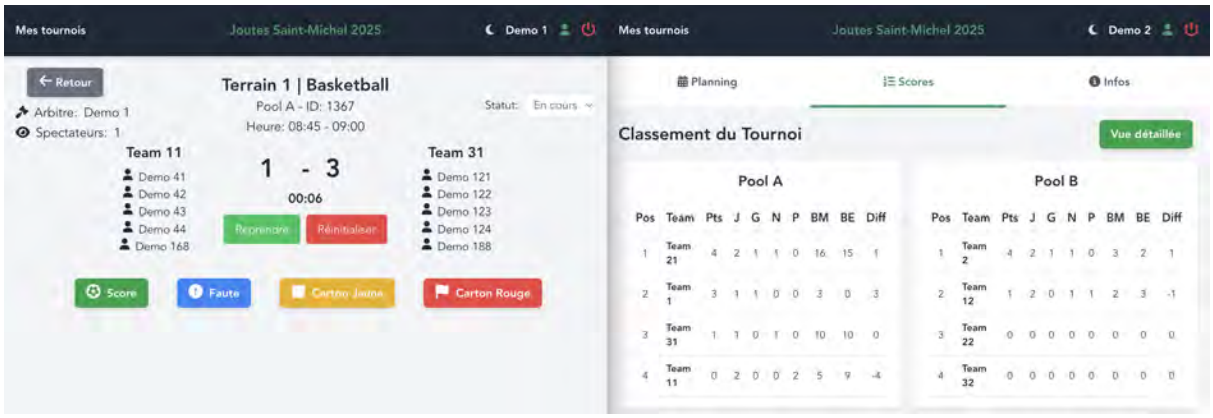


Fig. 3.54.: Score pas encore actualisé (Match "En cours")

Une fois le score validé, la Figure 3.55 illustre le classement mis à jour. La team 31 passe en première position dans le **Pool A**, grâce à une meilleure différence de points, et ce, sans rechargement de la page.

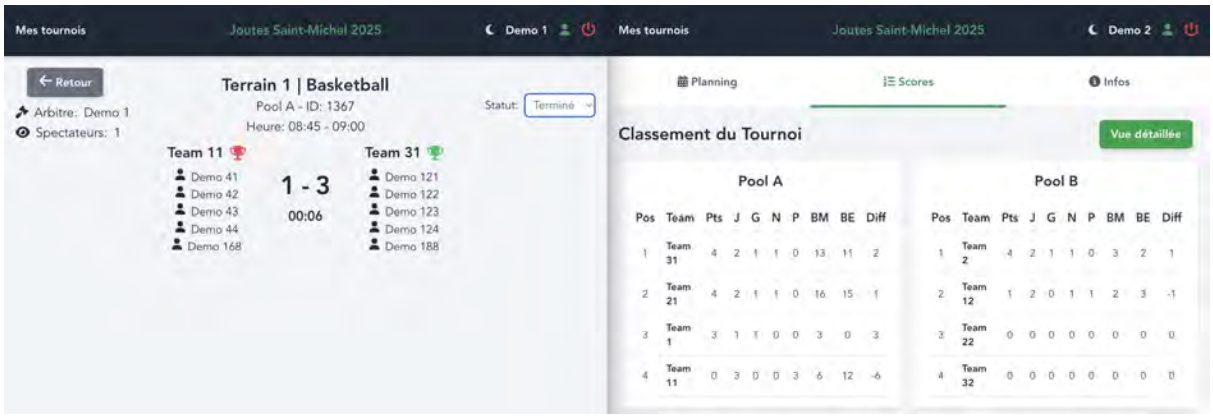


Fig. 3.55.: Classement actualisé après match "Terminé"

3.6.5. Système de scores

La page des scores (Fig. 3.56) offre une vue d'ensemble de **tous les matchs** et de leur évolution en temps réel, ce qui permet de diffuser ces informations sur un grand écran ou un vidéoprojecteur.

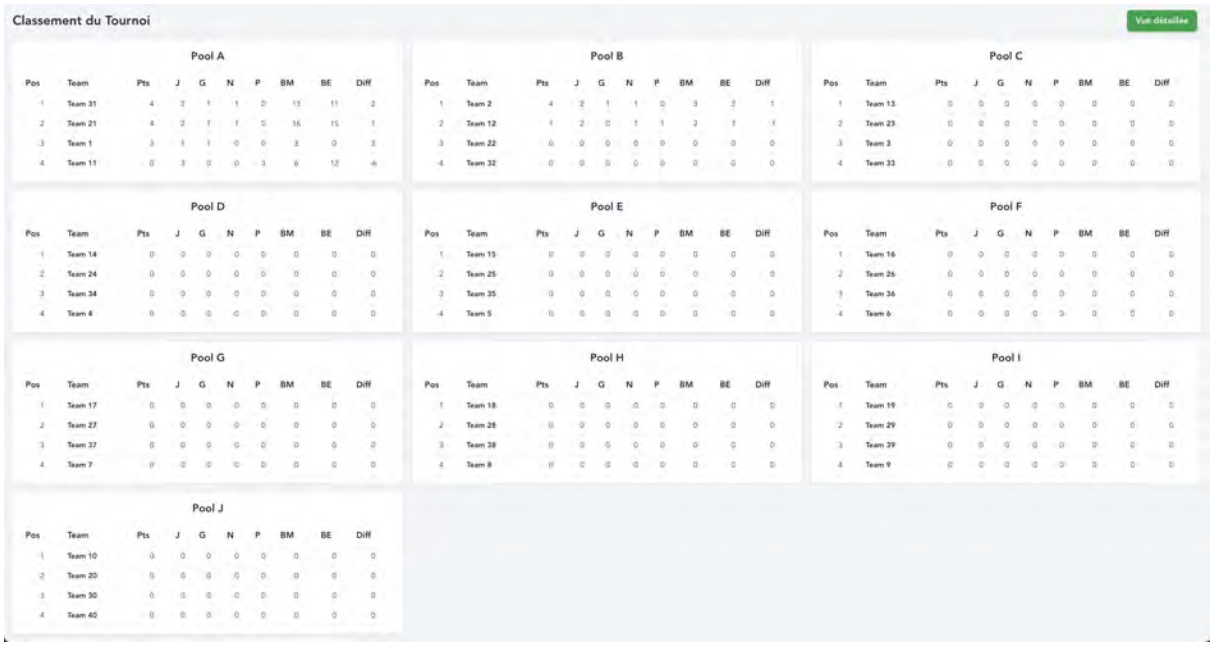


Fig. 3.56.: Vue d'ensemble des scores

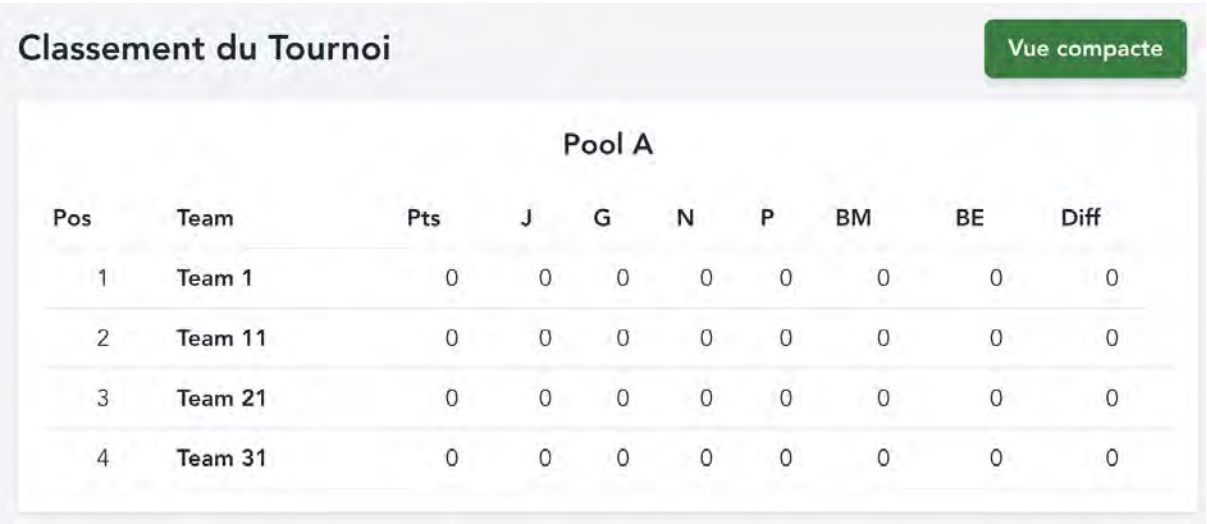


Fig. 3.57.: Zoom sur le système de scores

Nomenclature

- **Pos** : Classement de l'équipe au sein du Pool.
- **Team** : Nom de l'équipe.
- **Pts** : Points acquis (3 pour une victoire, 1 pour un nul, 0 pour une défaite). En cas d'égalité, la différence de points (BM – BE) départage les équipes.
- **J** : Matchs joués.
- **G** : Matchs gagnés.
- **N** : Matchs nuls.
- **P** : Matchs perdus.
- **BM** : Buts marqués.
- **BE** : Buts encaissés.
- **Diff** : Différence (BM – BE).

Les tableaux sont mis à jour **automatiquement** grâce à *Vue.js 3* et aux **websockets**. Une **vue compacte** ou **étendue** est disponible selon la taille de l'écran, que ce soit un grand écran de salle de sport, une tablette ou un smartphone.

Note sur la gestion multisport: Le système actuel unifie tous les sports dans un modèle de points unique. Les sports individuels ou à temps n'ont pas encore été intégrés. Une réflexion approfondie sera nécessaire à l'avenir pour gérer chaque discipline de manière spécifique. Néanmoins, ce modèle prend en charge un large éventail de sports collectifs et constitue une base solide pour de futures évolutions.

3.6.6. Planning - Point de vue Joueur

Nous avons jusqu'ici exploré la perspective d'un arbitre. Voici à présent quelques captures d'écran illustrant l'expérience côté **joueur**.

Indication des prochains matchs à jouer

Les prochains matchs du joueurs sont indiqués en mentionnant l'heure, l'affrontement et le terrain.

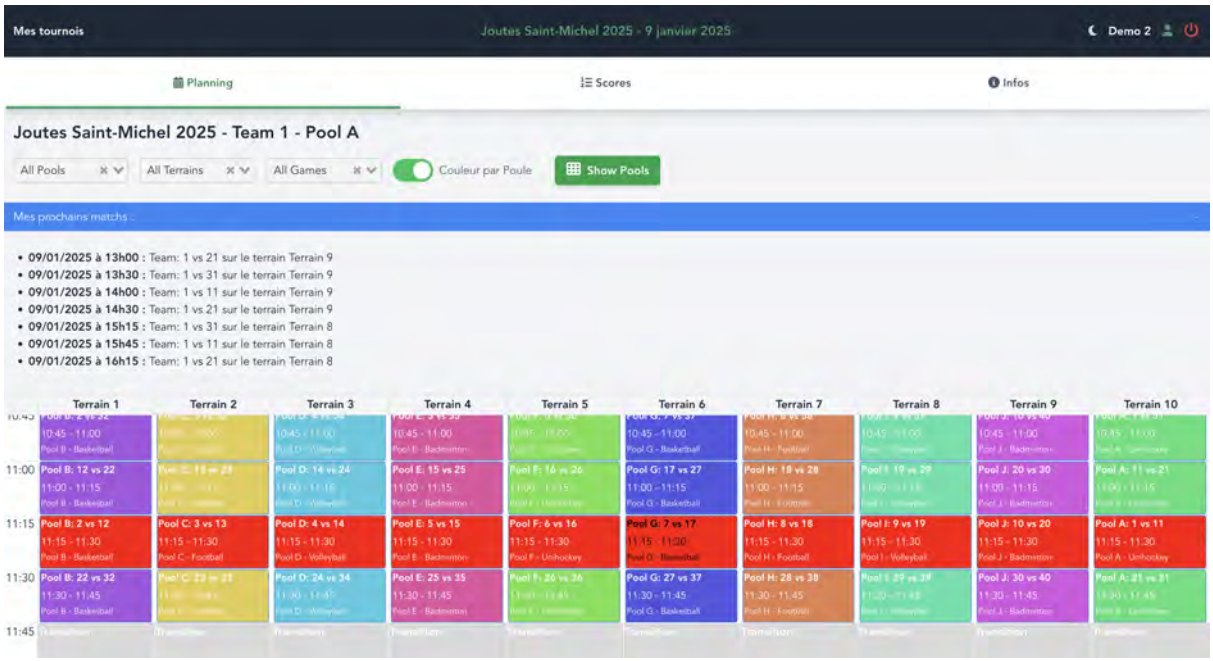


Fig. 3.58.: Prochains matchs d'un joueur

Filtrer par Pool

Un joueur peut préférer visualiser uniquement les matchs de son propre **Pool**, par exemple le **Pool A**:

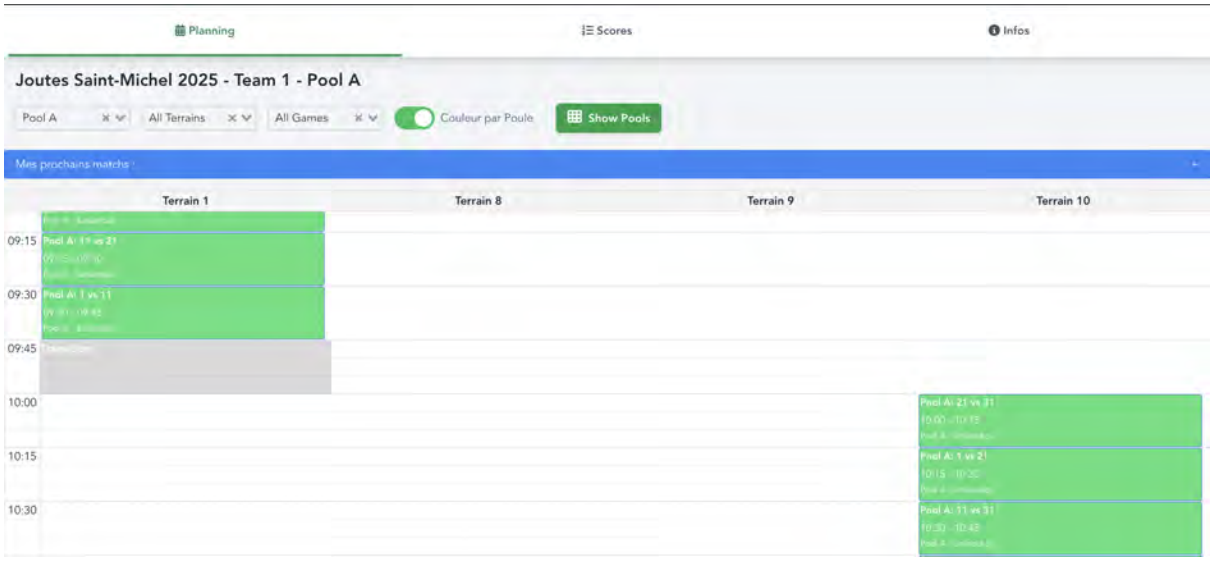


Fig. 3.59.: Vue par Pool

Le calendrier affiche uniquement les terrains liés au Pool. **Note :** Un joueur n'a aucun droit d'édition ou de modification, hormis la gestion de son propre profil. L'interface lui

permet simplement d'identifier ses prochains matchs, de consulter les résultats et d'accéder aux informations du tournoi.

3.6.7. Conclusion du Scénario 3

Ce troisième scénario illustre la **phase finale** et la plus dynamique du tournoi : son déroulement en temps réel. Parmi les éléments clés, on retrouve :

- Une **interface intuitive** pour l'arbitre : démarrage, suivi et clôture du match, enregistrement des événements et mise à jour du score.
- Une **actualisation en direct** via websocket, afin que joueurs et spectateurs puissent suivre l'évolution des scores sans rechargement.
- Une **gestion flexible des rôles** : un administrateur, même s'il est également joueur, peut arbitrer si nécessaire. Un joueur standard ou un invité *(Guest)* ne peut, quant à lui, qu'accéder aux informations.

Easy Tourney propose ainsi un planning clair indiquant à chaque équipe ses prochains matchs, un système d'arbitrage en temps réel et une **visualisation des résultats** fluide et cohérente.

3.6.8. Conclusion des scénarios

Grâce à *Easy Tourney*, M. Dupont a pu configurer et gérer son tournoi en toute simplicité. Les fonctionnalités avancées de l'application — comme l'automatisation des inscriptions et la gestion des plannings — simplifient grandement la logistique. Certains points peuvent encore être améliorés, mais l'interface et l'ergonomie actuelles rendent déjà son usage intuitif et accessible. Notons que la génération d'un fichier Excel permet de continuer à gérer le tournoi hors ligne, tandis que la **Progressive Web App (PWA)**, présentée dans la prochaine section, offre la possibilité de consulter le planning si l'on perd la connexion courante. Nous verrons dans les prochaines pages comment ces **fonctionnalités systèmes** sont mises en place et en quoi elles viennent compléter l'expérience utilisateur.

3.7. Fonctionnalités Systèmes

En complément des scénarios décrits précédemment, *Easy Tourney* intègre plusieurs **fonctionnalités systèmes** visant à améliorer l'expérience utilisateur et à faciliter la gestion du tournoi. Ces fonctionnalités, bien que secondaires par rapport au noyau fonctionnel du logiciel, contribuent à la robustesse et à la praticité de la solution. Nous aborderons dans ce chapitre :

- **Progressive Web App (PWA)** : consultation hors ligne et installation rapide.
- **Gestion des utilisateurs** : rôles (super-admin, admin, utilisateur), suppression, modification et création.
- **Récupération de mot de passe** : système d'email de réinitialisation via *Nodemailer*.
- **Design Responsive et Mode Dark/Light** : adaptation à tous les formats d'écran et personnalisation de l'interface.

3.7.1. Progressive Web App (PWA)

Définition

> « Une Progressive Web App (PWA) est une application construite avec des technologies de la plateforme web, mais qui fournit une expérience utilisateur semblable à celle d'une application spécifique à une plateforme. À l'instar d'un site web, une PWA peut fonctionner sur plusieurs plateformes et appareils à partir d'une seule base de code. Comme une application native, elle peut être installée sur l'appareil, fonctionner hors ligne et en arrière-plan, et s'intégrer avec le dispositif et d'autres applications installées. » [32]

Fonctionnalités et Objectifs

La PWA intégrée dans *Easy Tourney* répond principalement aux objectifs suivants :

- **Lecture du planning hors ligne :** Les arbitres et joueurs peuvent consulter les matchs et horaires même sans connexion réseau.
- **Installation rapide :** La PWA peut être installée sur l'écran d'accueil d'un smartphone ou d'un ordinateur pour un accès direct.
- **Implémentation minimale :** Seule la page de planning est mise en cache pour fonctionner hors ligne. Les autres fonctionnalités, comme les notifications push, ne sont pas disponibles.

Installation

Remarque : L'application a été testée exclusivement sur iOS et macOS, mais elle devrait également fonctionner sur Android, Google Chrome assurant la compatibilité entre les différents systèmes d'exploitation.

Sur macOS (Google Chrome) : Ajout de l'application au Dock en quelques clics. Voir la figure 3.60.

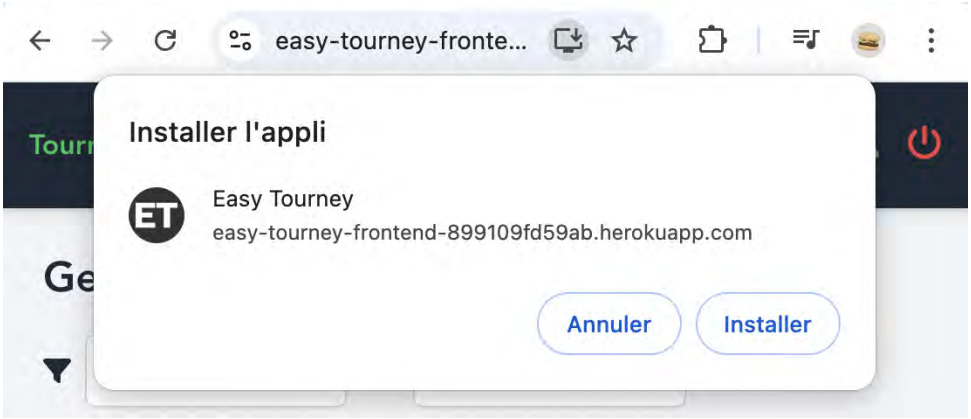


Fig. 3.60.: Installation de la PWA sur macOS.

Sur iPhone : Ajout à l'écran d'accueil via Google Chrome. Les figures 3.61 et 3.62 montrent le processus d'installation.

Fig. 3.61.: Écran d'ajout.

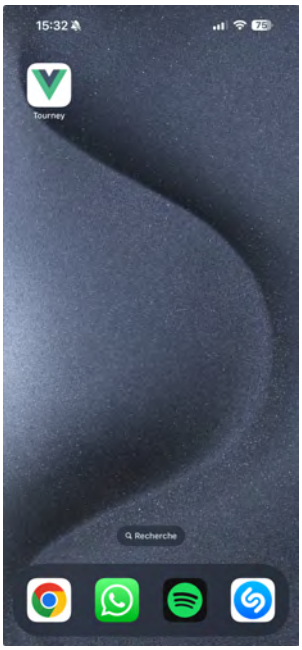


Fig. 3.62.: Icône ajoutée.

Mode Hors Ligne

La PWA permet d'accéder aux informations du planning en cas de coupure réseau. Voici des démonstrations :

- **MacBook :** Le planning reste accessible hors ligne après une première consultation en ligne (figure 3.63).
- **iPhone :** Visualisation du planning hors-ligne (mode avion) depuis l'icône ajoutée à l'écran d'accueil (figure 3.64).

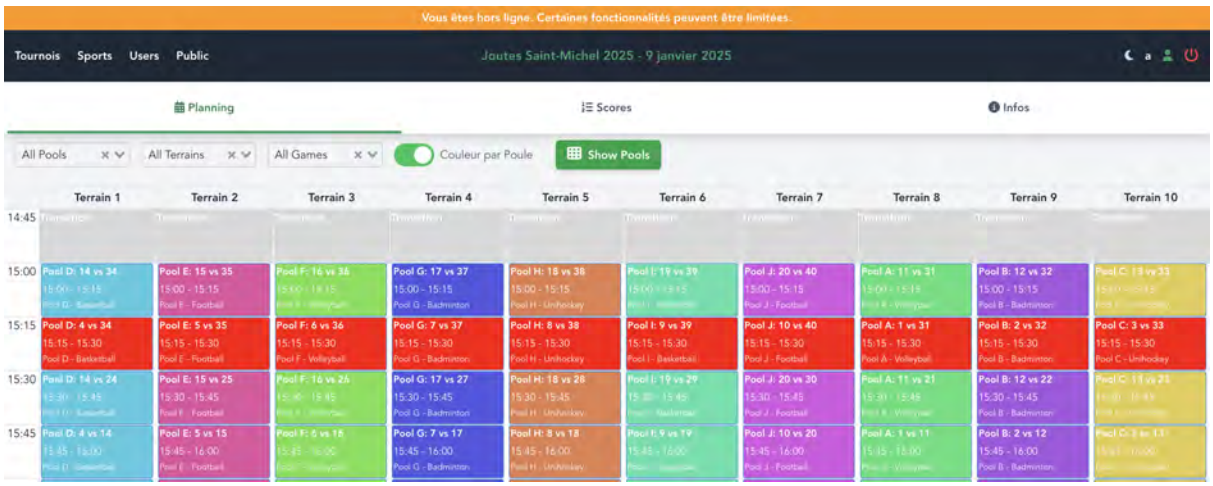


Fig. 3.63.: Planning visible hors ligne sur macOS.

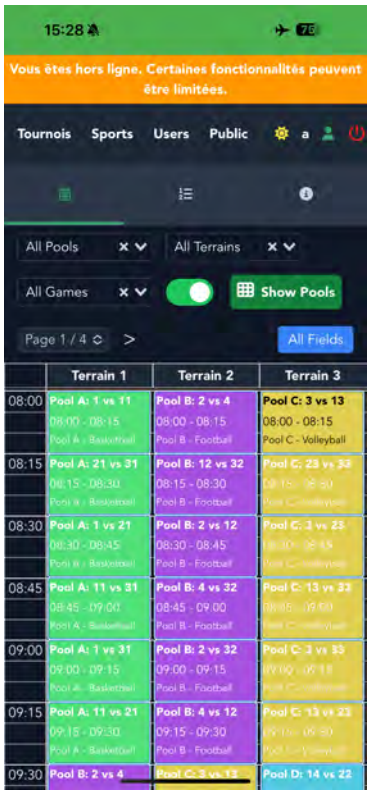


Fig. 3.64.: Planning visible hors ligne sur iPhone.

Un message informatif orange indique que les fonctionnalités de l'application sont limitées en mode hors-ligne.

Limitations

- **Page admin non disponible hors ligne :** Seule la page planning est mise en cache pour l'instant. Les autres pages, comme la page *Infos*, ne fonctionnent pas hors ligne (figure 3.65).
- **Mise à jour requise :** Une connexion internet est nécessaire pour mettre à jour les données.

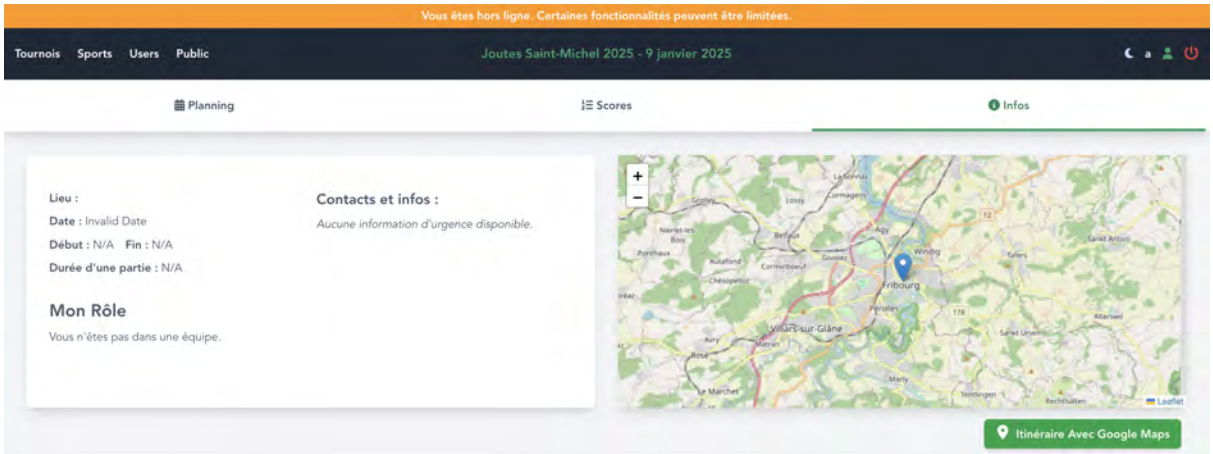


Fig. 3.65.: Message d'erreur hors ligne sur une page non supportée.

Note : L'implémentation d'un mode hors-ligne pour un mode administrateur n'a pas été jugée pertinente car les opérations admin interagissent avec la base de données et nécessitent de ce fait une connexion.

Retour sur la PWA

La PWA offre une solution pratique pour accéder au planning dans des conditions de connectivité limitées. Cependant, cette fonctionnalité nécessite encore des améliorations, notamment :

- Ajout d'autres pages au mode hors ligne.
- Intégration de notifications push pour informer les utilisateurs des changements (prochain match à jouer).

3.7.2. Gestion des utilisateurs

Une gestion efficace des utilisateurs est essentielle pour garantir le bon fonctionnement de l'application et permettre une administration fluide en cas de problème ou de changement. La page *Users* offre aux administrateurs des outils complets pour gérer les utilisateurs de manière centralisée.

Types de rôles et droits associés

- **Super-admin (ID 1)** : Ce rôle dispose des droits les plus étendus, incluant la suppression et la modification des autres administrateurs. Par défaut, le super-admin est un administrateur avec l'identifiant `1` dans la base de données. Une amélioration future envisage l'introduction d'un rôle explicite `super-admin` pour une gestion plus claire.
- **Admin** : Les administrateurs possèdent des droits généraux d'administration, notamment pour créer, modifier ou supprimer des utilisateurs et des tournois. Toutefois, ils ne peuvent ni supprimer ni modifier un super-admin.
- **Utilisateur** : Les utilisateurs n'ont accès qu'à leur propre profil et ne peuvent pas interagir avec la page *Users* ni accéder à ses fonctionnalités.

Fonctionnalités de la page *Users*

La page *Users* (Fig. 3.66) centralise la gestion des utilisateurs et propose plusieurs fonctionnalités :

Vue globale : La vue principale liste tous les utilisateurs avec leur nom, leur email, et les tournois auxquels ils sont associés. Les administrateurs peuvent effectuer diverses actions directement depuis cette vue :

- Supprimer une association d'utilisateur avec un tournoi.
- Ajouter un utilisateur à un tournoi.
- Modifier les informations d'un utilisateur (via un bouton *Edit* qui redirige vers la page de profil).

- Supprimer un utilisateur en cliquant sur l'icône de la poubelle.
- Ajouter un nouvel utilisateur avec la possibilité de spécifier un rôle (*admin* ou *user*).

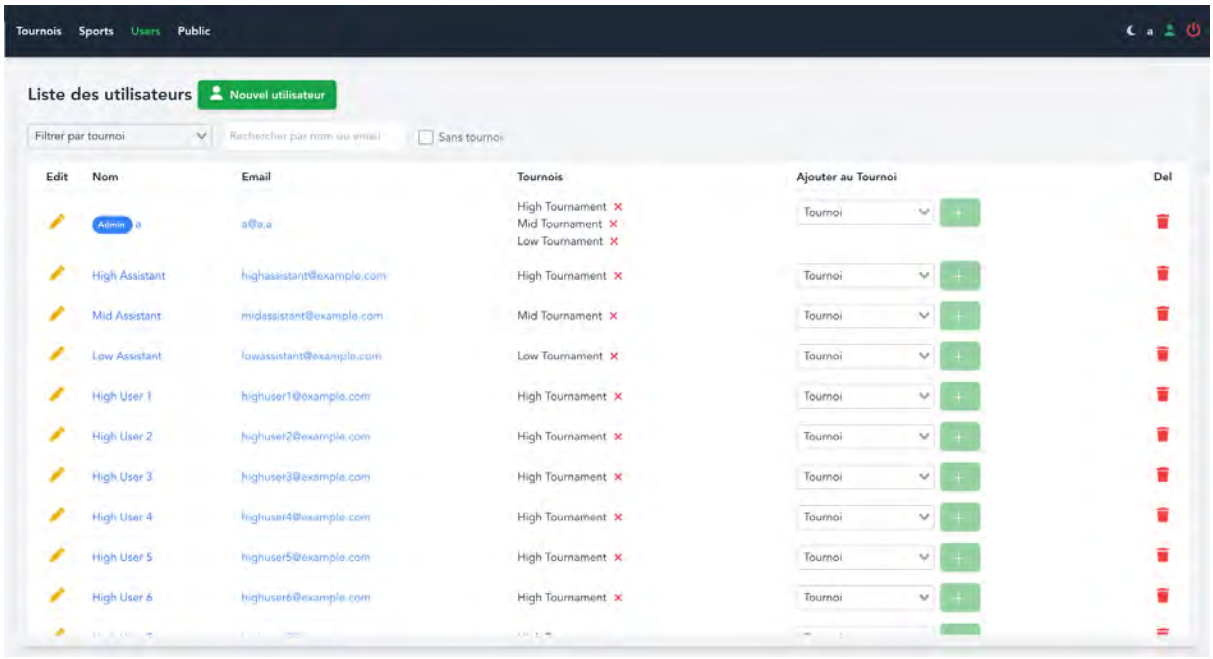


Fig. 3.66.: Vue globale de la page *Users* listant tous les utilisateurs.

Recherche et filtres : La page intègre des fonctionnalités de tri et de recherche pour faciliter la navigation parmi les utilisateurs (Fig. 3.67). Les administrateurs peuvent :

- Rechercher des utilisateurs par nom, email ou tournoi.
- Afficher uniquement les utilisateurs sans tournoi.
- (Amélioration future) Filtrer les utilisateurs par rôle (*admin*, *user*).

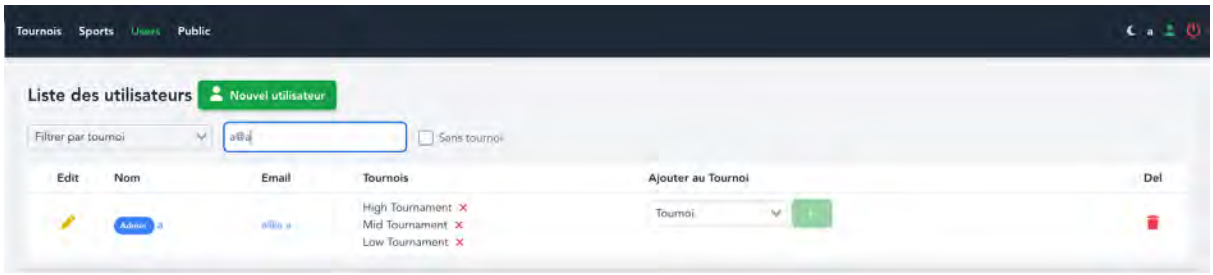


Fig. 3.67.: Exemple de tri par email dans la page *Users*.

Ajout d'un utilisateur : Un nouvel utilisateur peut être ajouté en spécifiant son rôle. La figure 3.68 montre l'ajout d'un utilisateur nommé *New Admin* avec le rôle *admin*.

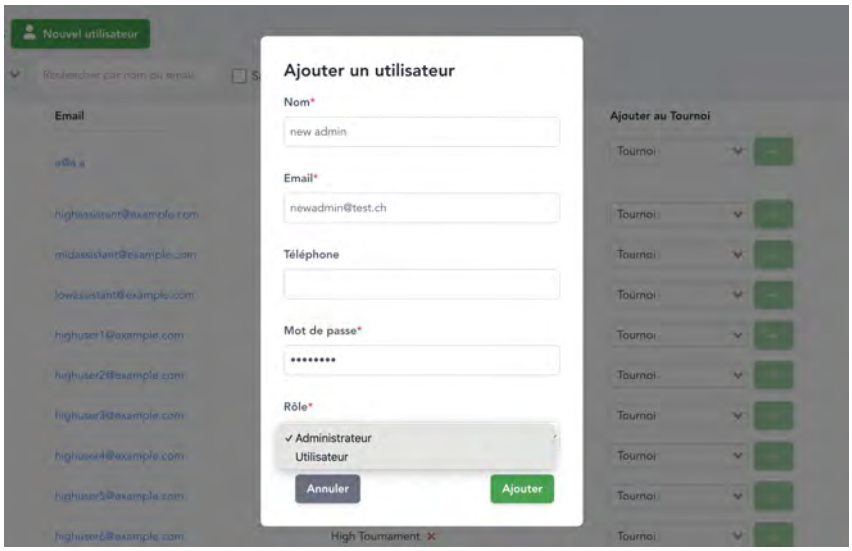


Fig. 3.68.: Ajout d'un nouvel utilisateur *New Admin* avec le rôle *administrateur*.

Gestion des accès et rôles

Exemple d'accès admin : Une fois un utilisateur configuré comme *admin*, il dispose d'un accès aux fonctionnalités administratives comme illustré à la figure 3.69.

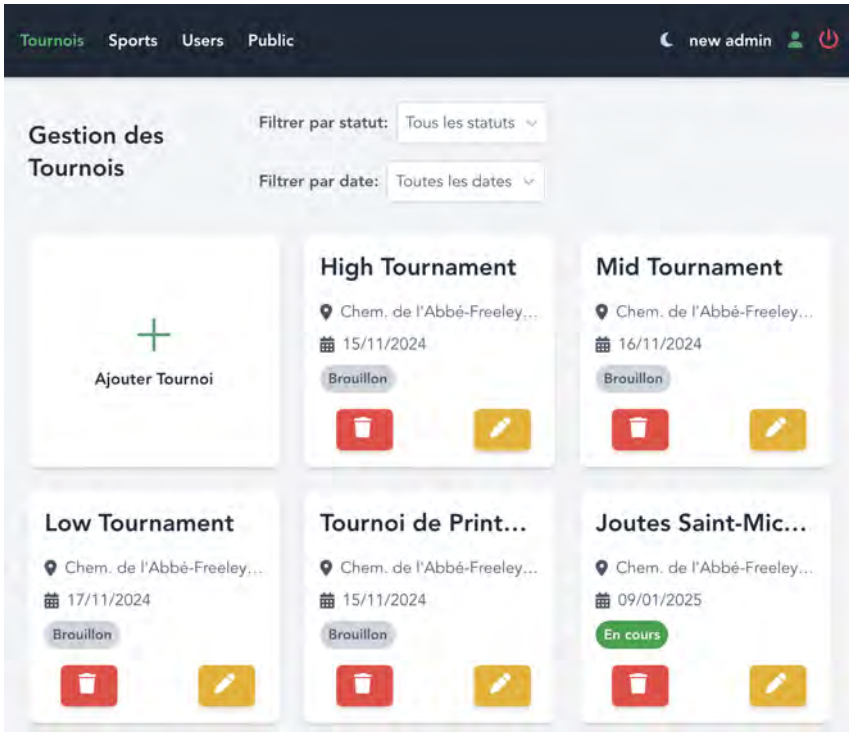


Fig. 3.69.: Accès à la page admin pour l'utilisateur *New Admin*.

Restrictions liées au super-admin : Un admin ne peut pas supprimer ou modifier un super-admin. Lorsqu'une tentative de suppression est effectuée, un message d'erreur est affiché (Fig. 3.70).

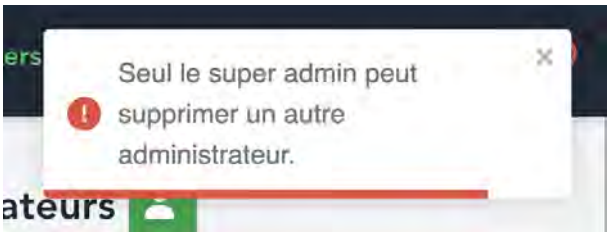


Fig. 3.70.: Message d'erreur indiquant l'impossibilité de supprimer le super-admin.

Révocation des droits : Si un *admin* est rétrogradé au rôle *user*, il perd immédiatement ses accès aux fonctionnalités administratives. Une page d'erreur 403 est affichée lorsqu'il tente d'accéder à une page protégée (Fig. 3.71). Une page d'erreur 404 est également implémentée pour toute URL incorrecte (Fig. 3.72).

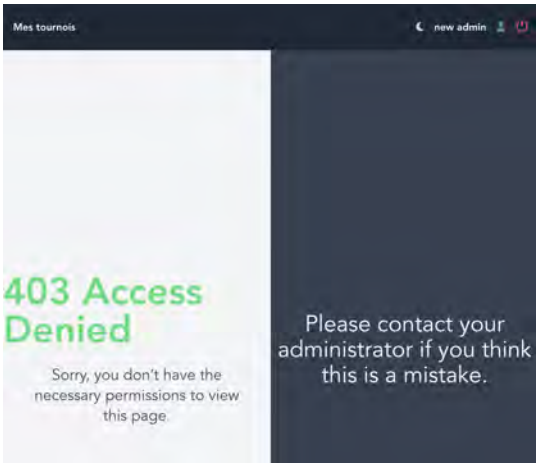


Fig. 3.71.: Page d'erreur 403 pour un utilisateur sans accès.

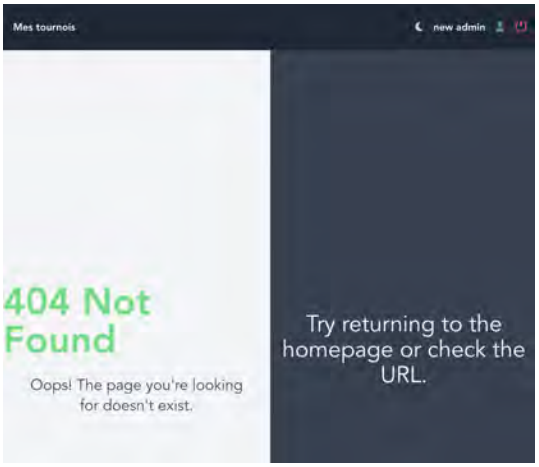


Fig. 3.72.: Page d'erreur 404 pour une URL non trouvée.

Modification des rôles dans un tournoi

- Pour modifier le rôle d'un utilisateur (*joueur*, *arbitre*, etc.) dans un tournoi spécifique, il est nécessaire de se rendre dans la configuration du tournoi concerné.
- Les administrateurs peuvent ainsi gérer efficacement les droits des utilisateurs au sein d'un tournoi sans affecter leur rôle global.

3.7.3. Récupération de mot de passe

L'application *Easy Tourney* intègre un système simple et efficace de réinitialisation de mot de passe par email. Voici le processus illustré étape par étape à l'aide des captures d'écran suivantes :

Étape 1 : Accès au lien de réinitialisation

Depuis la page de connexion, un utilisateur ayant entré un mot de passe incorrect peut cliquer sur *Mot de passe oublié* pour accéder à l'interface de réinitialisation (Fig. 3.73).

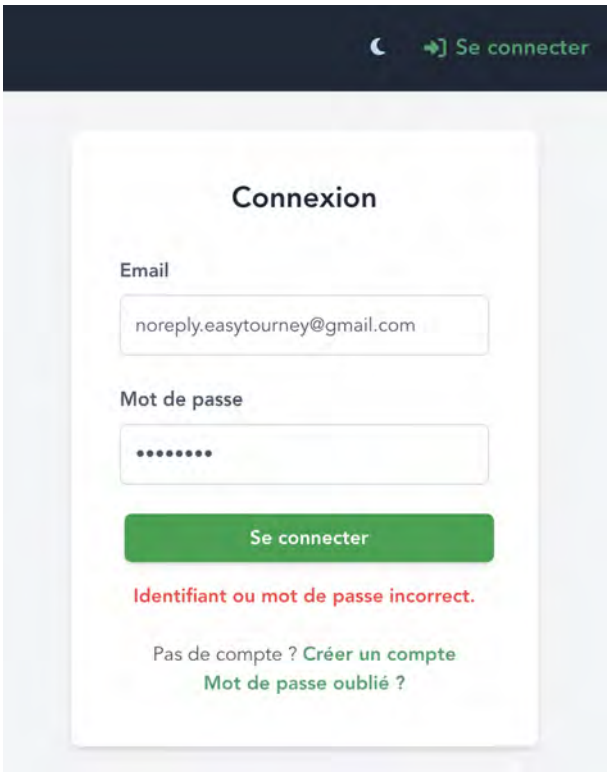


Fig. 3.73.: Lien *Mot de passe oublié* accessible depuis la page de connexion.

Étape 2 : Saisie de l'adresse email

L'utilisateur est redirigé vers une page où il peut entrer son adresse email pour recevoir un lien de réinitialisation (Fig. 3.74).

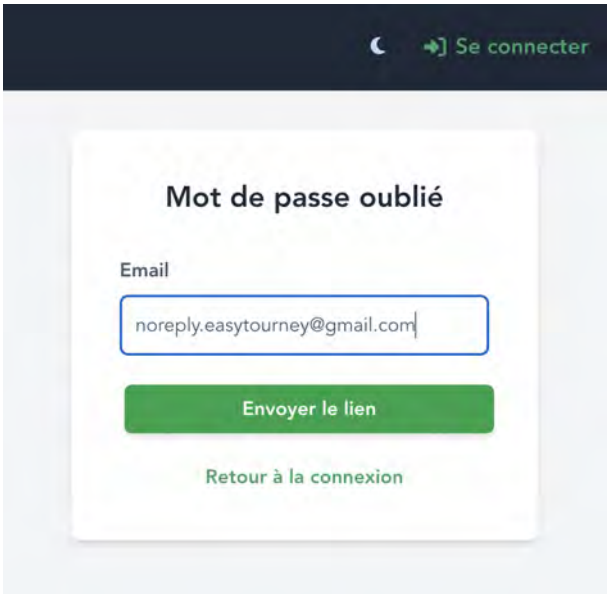


Fig. 3.74.: Page de demande de lien de réinitialisation.

Un toast confirme l'envoi du lien de réinitialisation à l'adresse spécifiée (Fig. 3.75).

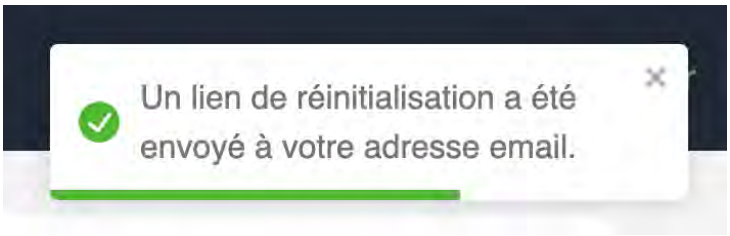


Fig. 3.75.: Confirmation de l'envoi du lien.

Étape 3 : Réception de l'email

L'utilisateur reçoit un email contenant un lien sécurisé pour réinitialiser son mot de passe.

Fig. 3.76.: Email contenant le lien de réinitialisation.

Étape 4 : Définition du nouveau mot de passe

En cliquant sur le lien, l'utilisateur est redirigé vers une page où il peut définir et confirmer un nouveau mot de passe.

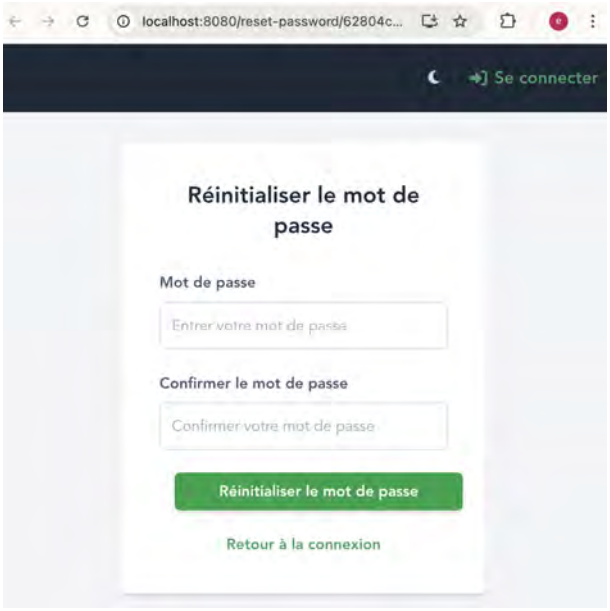


Fig. 3.77.: Page de réinitialisation du mot de passe.

Étape 5 : Connexion réussie

Après validation, l'utilisateur peut se reconnecter avec son nouveau mot de passe. Un pop-up de son navigateur propose d'enregistrer le mot de passe pour une utilisation future.

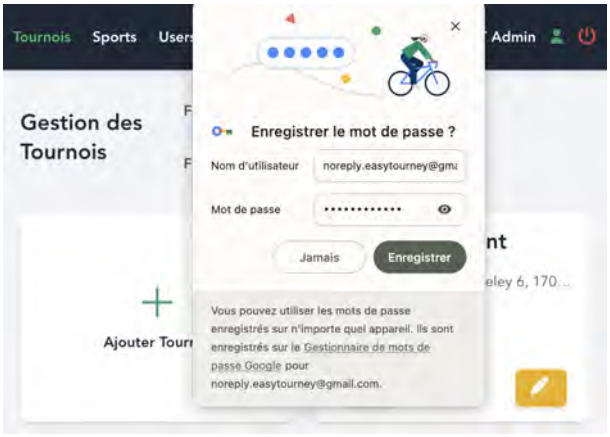


Fig. 3.78.: Connexion réussie avec le nouveau mot de passe.

Conclusion

Ce système de récupération de mot de passe est essentiel pour améliorer l'expérience utilisateur et réduire la charge administrative liée aux demandes d'assistance. Il garantit également une sécurité accrue en évitant les échanges non sécurisés de mots de passe.

3.7.4. Design Responsive et Mode Dark/Light

Cette section présente le mode responsive et les thèmes de couleur à travers diverses captures d'écran, qui ont été prises sur le serveur en production hébergé chez Heroku[1] Elle agit principalement comme un portfolio visuel, sans entrer dans les détails techniques, afin de démontrer le bon fonctionnement du mode responsive et du thème sombre. Sur chaque capture :

1. **Figure gauche** est en thème sombre sur un grand écran (ordinateur portable) via Google Chrome.
2. **Figure centrale** est un thème clair sur un écran moyen (tablette) utilisant la PWA EasyTourney.
3. **Figure droite** est un thème sombre sur un petit écran (smartphone) utilisant la PWA EasyTourney.

Ces fonctionnalités permettent de répondre aux attentes des utilisateurs modernes, en offrant une interface accessible et adaptable quelles que soient leurs préférences ou leurs conditions de travail. L'utilisateur peut changer de mode (sombre/clair) en cliquant sur la petite lune ou soleil, à gauche de son profil utilisateur.

[1] `https://easy-tourney-frontend-899109fd59ab.herokuapp.com/login` : L'accessibilité de ce serveur n'est pas garantie, car il peut ne plus être maintenu ou actif.

Responsive Design

L'objectif principal du design responsive est d'assurer une expérience utilisateur optimale sur tous les types d'appareils:

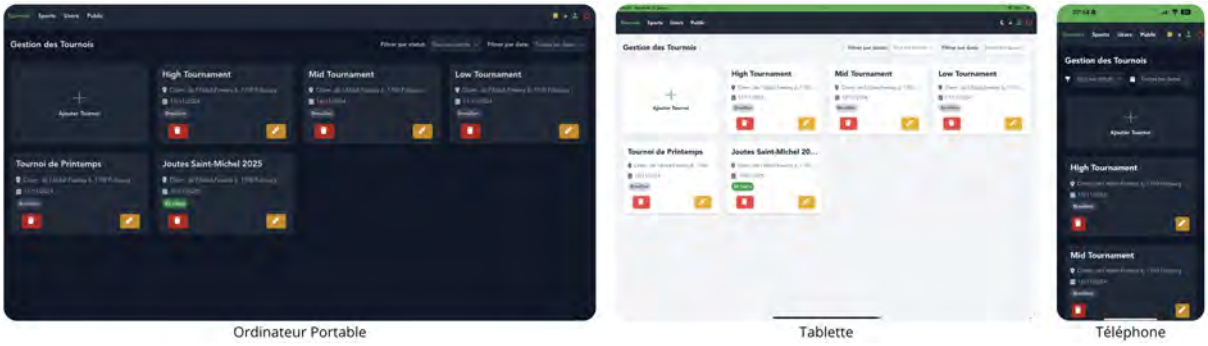


Fig. 3.79.: Page de gestion des tournois en mode administrateur.

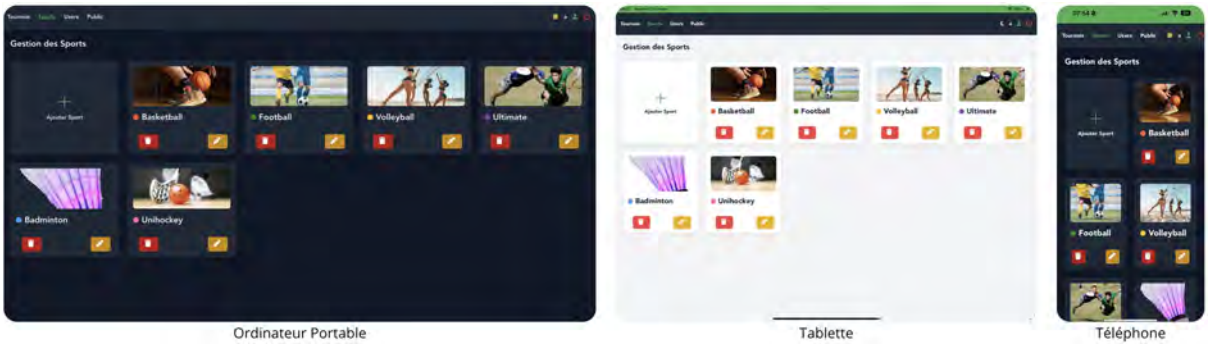


Fig. 3.80.: Page de gestion des sports en mode administrateur.

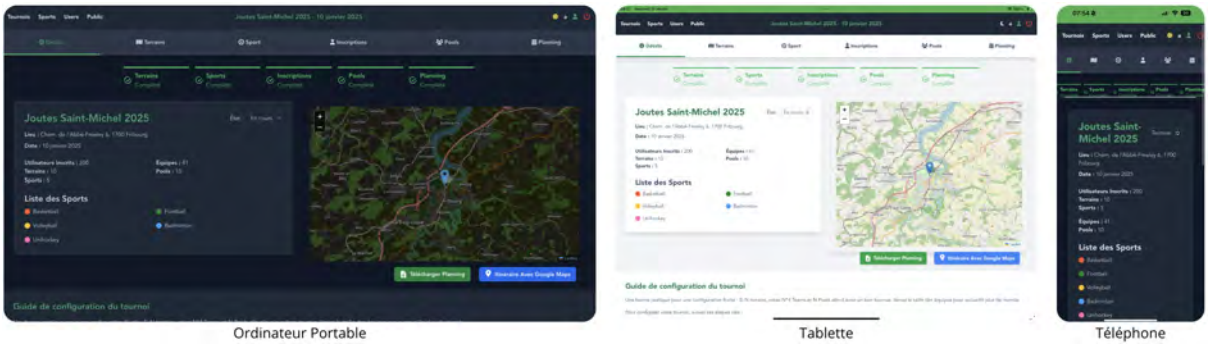


Fig. 3.81.: Page de détails d'un tournoi en mode administrateur. (Note : la carte est affichée sous les informations mais n'est pas visible ici.)

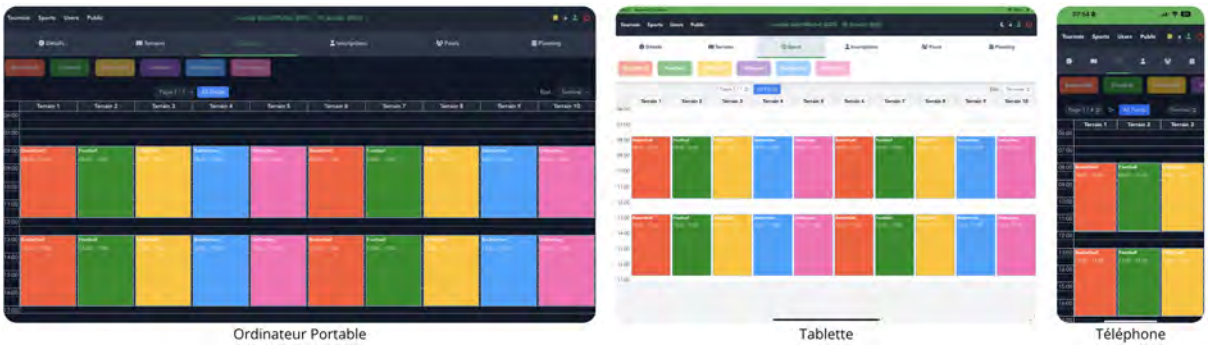


Fig. 3.82.: Page d'association des sports aux terrains.

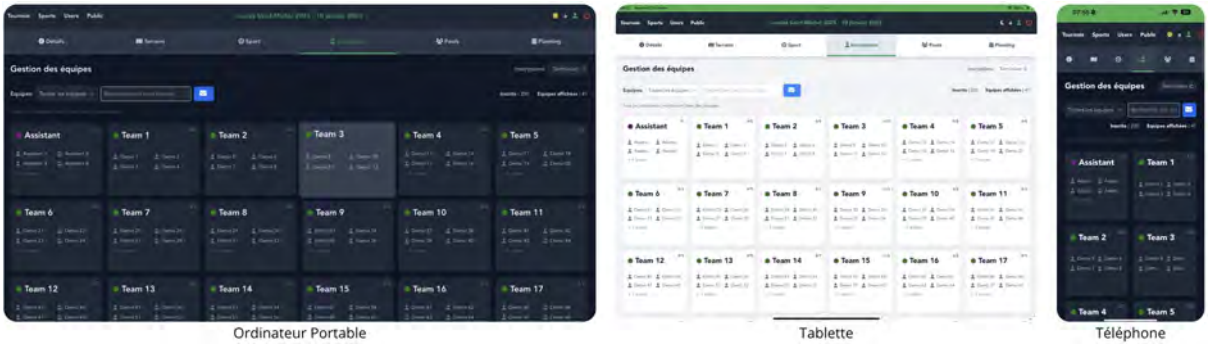


Fig. 3.83.: Page de gestion des inscriptions et des équipes en mode administrateur.

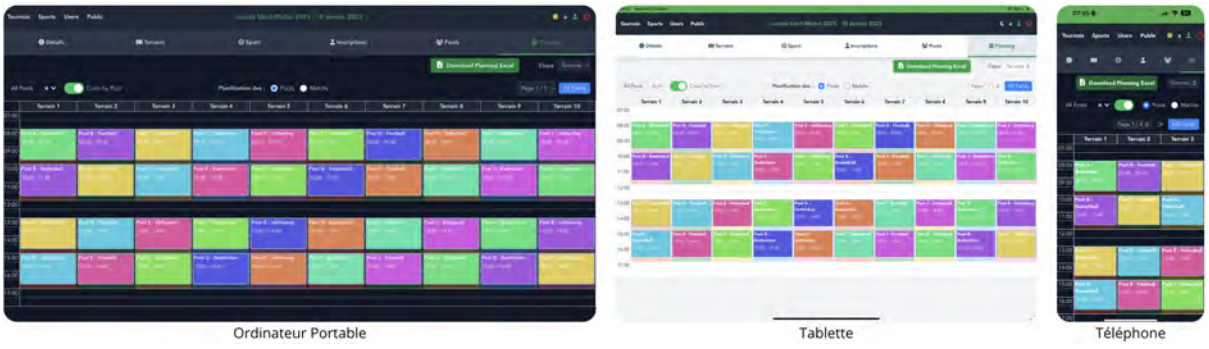


Fig. 3.84.: Page des plannings. L'édition manuelle reste possible sur smartphone.

Vue Arbitre

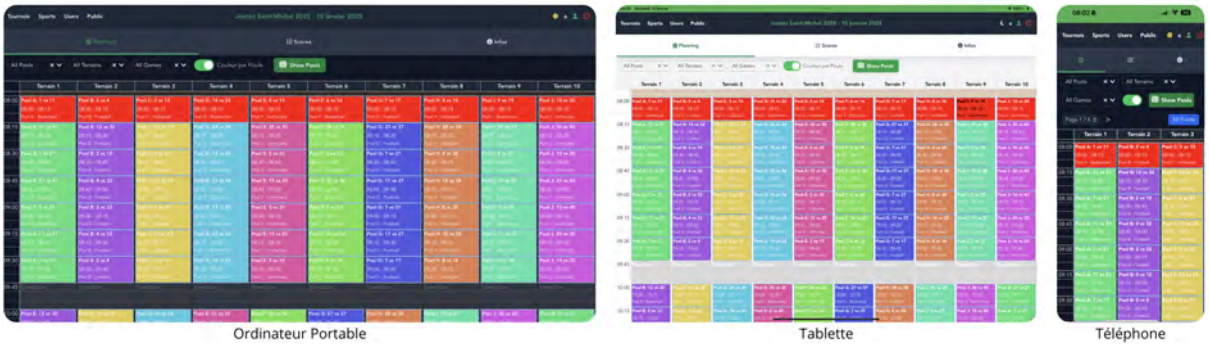


Fig. 3.85.: Page de planning pour un arbitre.

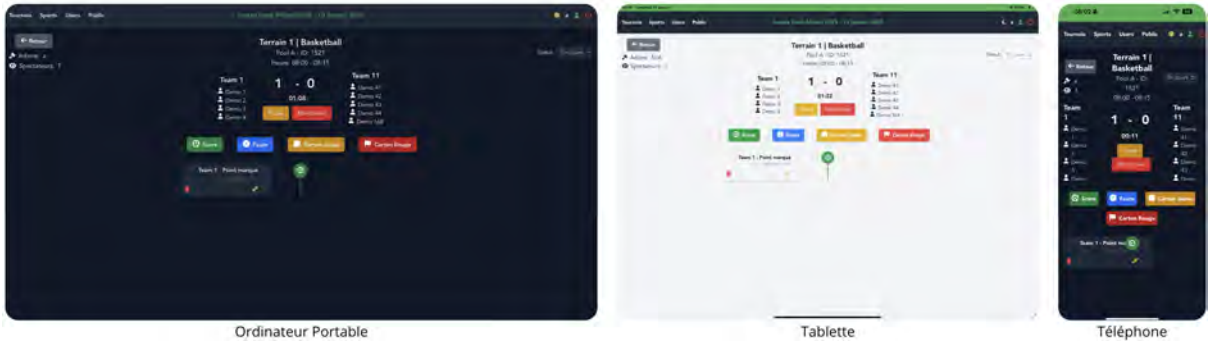


Fig. 3.86.: Page d'arbitrage d'un match. Mode paysage recommandé sur mobile.

Vue Joueur

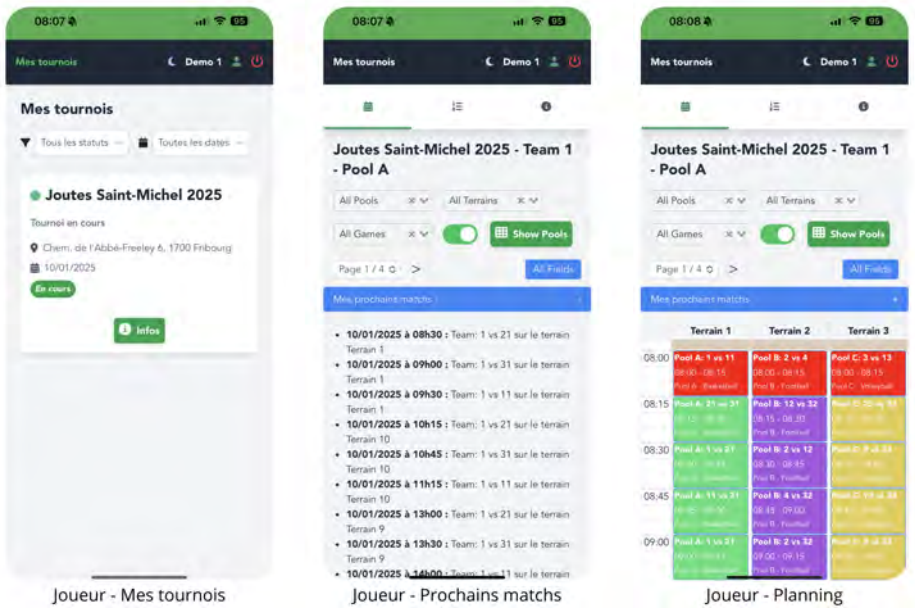


Fig. 3.87.: Vue joueur sur smartphone : liste des tournois, planning des prochains matchs, et aperçu global du planning.

Gestion administrative sur mobile

Bien que l'administration soit plus confortable sur un ordinateur, l'application permet d'effectuer des opérations essentielles, comme le drag & drop des plannings ou l'association des sports aux terrains, directement depuis un smartphone. Cette flexibilité s'avère particulièrement utile pour des ajustements de dernière minute.

Conclusion Design Responsive

Le design responsive et les options de thèmes de couleur permettent à *Easy Tourney* de s'adapter efficacement à divers appareils et préférences utilisateurs. Cette flexibilité assure une expérience fluide, que ce soit sur ordinateur, tablette ou smartphone, et répond aux besoins de confort visuel grâce aux modes Dark et Light.

3.7.5. Conclusion Fonctionnalités Systèmes

Les fonctionnalités systèmes ajoutent une valeur en plus en termes de robustesse et de convivialité. La transformation en Progressive Web App (PWA) permet une utilisation hors ligne du planning, tandis que la gestion des utilisateurs et le système de récupération de mot de passe assurent une administration sécurisée. Le design responsive et les thèmes dark/light complètent ces fonctionnalités, offrant une expérience utilisateur riche.

3.8. Conclusion *Point de vue utilisateur*

À travers les trois scénarios présentés, *Easy Tourney* démontre sa capacité à simplifier l'organisation et la gestion d'un tournoi multisport. De la préparation initiale à la gestion des inscriptions et au déroulement en temps réel, l'application offre des outils intuitifs pour les administrateurs, arbitres et participants. Les fonctionnalités systèmes et le design adaptable renforcent l'expérience utilisateur, rendant *Easy Tourney* une solution complète et pratique pour la gestion de tournois scolaires.

La prochaine section aborde le **point de vue développeur**, où nous plongerons dans l'architecture logicielle, la structure du code, et les algorithmes d'automatisation (planification, répartition des équipes, affectation des matchs, etc.) qui rendent cette solution possible.

4

Point de Vue Développeur

4.1. Introduction

Ce chapitre explore en profondeur les aspects techniques du développement de l'application *Easy Tourney*. Nous commencerons par présenter l'architecture globale (frontend, backend et base de données), puis nous rappellerons les principaux concepts techniques (API REST, WebSocket, JWT, etc.). Pour rendre cette section accessible à tous, ces rappels restent optionnels pour le lecteur habitué aux bonnes pratiques de développement.

Ensuite, nous détaillerons le système d'authentification et de navigation, en mettant en évidence la gestion fine des rôles et droits d'accès. Les fonctionnalités avancées, telles que l'utilisation de *FullCalendar* ou du *token d'invitation*, illustreront la valeur ajoutée du temps réel (via *Socket.IO*) et la flexibilité offerte pour la planification des matchs.

Nous verrons également comment le *Strategy Pattern* a été appliqué pour la gestion des *Pools*, avant d'aborder des fonctionnalités annexes (mode sombre/clair, export Excel, PWA, etc.). Enfin, nous terminerons par un tour d'horizon de la méthode agile et de la mise en place des tests et pipelines de déploiement, suivi d'un bilan critique sur les points d'amélioration.

Structure du chapitre :

1. **Architecture globale et rappels technologiques** : Vue d'ensemble de l'application, modèle relationnel de la base de données, présentation de la structure frontend et backend, et rappel des concepts clés (REST, WebSockets, JWT).
2. **Système d'authentification et gestion des droits** : Détails de l'implémentation du système de connexion suivi de la gestion des droits.
3. **Fonctionnalités avancées** : Explication de FullCalendar, du token d'invitation et de l'algorithme d'assignation des joueurs sans groupe aux équipes.

4. **Strategy Pattern** : Présentation du Strategy Pattern et de l'implémentation du planning des Pools.
5. **Fonctionnalités annexes** : Mode sombre/clair, export Excel, et PWA.
6. **Méthode Agile, tests et CI/CD** : Philosophie de développement, tests, et pipelines de déploiement.
7. **Améliorations et problèmes rencontrés** : bilan critique et solutions futures.

4.2. Architecture Globale et Rappels Technologiques

4.2.1. Vue d'Ensemble Frontend – Backend – Base de Données

L'architecture de l'application repose sur un modèle classique **client-serveur** (Fig. 4.1), avec une séparation claire entre :

- **Frontend (Vue.js)** : Gère l'interface utilisateur (UI) et la logique client. Les composants Vue consomment les données via des requêtes HTTP (`axios` [21]) ou interagissent en temps réel via `Socket.IO` [43].
- **Backend (Node.js + Express)** : Sert d'intermédiaire entre le frontend et la base de données. Il expose une API RESTful et gère également les événements temps réel (WebSockets).
- **Base de Données (MySQL)** : Stocke les données structurées du système (utilisateurs, tournois, équipes, etc.). La couche `Sequelize` (ORM) est utilisée pour abstraire les interactions avec la base de données et faciliter les migrations/seeders.

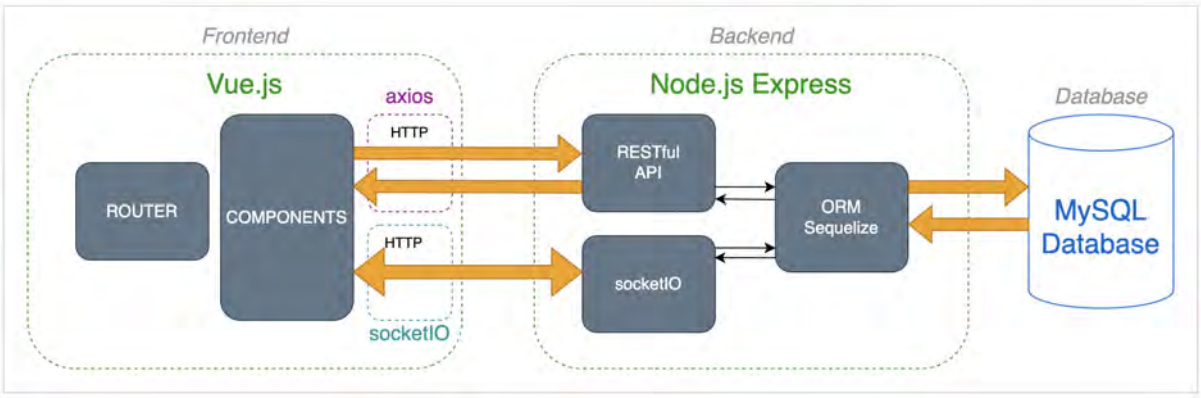


Fig. 4.1.: Architecture globale Frontend – Backend

Cette architecture garantit une séparation des responsabilités en facilitant la scalabilité et la maintenabilité. Les échanges entre le frontend et le backend s'opèrent de deux façons :

- **API REST** (via Axios) pour les opérations CRUD (utilisateurs, tournois, etc.).
- **WebSockets** (via Socket.IO) pour la mise à jour des scores en temps réel et l'arbitrage.

Déf : *Une* ***API RESTful*** *[10] (Representational State Transfer) est un style architectural permettant à des applications de communiquer entre elles via des requêtes HTTP standard (GET, POST, PUT, DELETE). Elle expose des*

points d'accès, appelés endpoints, où des clients peuvent récupérer ou envoyer des données structurées, généralement au format JSON.

Déf : *Un* ***WebSocket*** *[44] est un protocole de communication bidirectionnel en temps réel. Contrairement aux requêtes HTTP classiques, il permet de maintenir une connexion ouverte entre le client et le serveur, facilitant des échanges rapides et continus, par exemple pour des mises à jour en temps réel des scores ou voir le nombre de personnes connectées à la page.*

4.2.2. Détails des Choix Technologiques

Backend : Node, Express et Sequelize

Node.js [8] est une plateforme JavaScript rapide et performante, idéale pour les applications temps réel grâce à son modèle d'Entrée/Sortie non bloquant. Associé à **Express** [9], un framework minimaliste, il simplifie la gestion des routes et middlewares pour le backend.

Sequelize [17] est un ORM (Object-Relational Mapping) [18] qui permet d'interagir avec une base de données relationnelle en écrivant du JavaScript plutôt que du SQL brut. Il gère aussi bien **les migrations** que **les seeders**, simplifiant l'évolution du schéma et le peuplement initial en données de test. Dans des projets antérieurs (notamment en Laravel ou Ruby on Rails), j'avais déjà apprécié les avantages de l'approche ORM, d'où le choix naturel de Sequelize pour Easy Tourney.

Frontend : Vue.js, TailwindCSS et FullCalendar

Vue.js : le framework frontend

Vue.js [4] offre une syntaxe intuitive pour développer des interfaces utilisateur dynamiques. Le choix fait suite à sa popularité alors que je n'avais jamais travaillé avec.
TailwindCSS [37] permet une personnalisation rapide du design via des classes utilitaires. Je l'ai adoptée car j'en ai également souvent entendu parler et je recherchais une librairie CSS légère [35].
FullCalendar [26], quant à lui, fournit un calendrier interactif avec support du *drag-and-drop* pour planifier les événements sportifs (cf. Chapitre 4.6.4). Il a été rapidement choisi suite à des recherches sur internet comme solution numéro 1 pour travailler avec des plannings en javascript [27].

4.2.3. MySQL : Diagramme MLD et Retours

Comme présenté dans le *Chapitre 2.7 (Analyse)*, la base de données d'Easy Tourney est conçue selon un **modèle relationnel**. Cette section revient sur le **choix de MySQL**, tout en intégrant un bilan critique sur la modélisation (MLD) et sur les points à améliorer.

Justification du Choix de MySQL

Le choix de MySQL [12] comme Système de Gestion de Base de Données (SGBD) relationnel s'est imposé pour plusieurs raisons techniques et pratiques. Outre la **familiarité** avec MySQL (projets précédents, cours) et un **écosystème** riche (intégration avec Sequelize,

documentation et communauté importantes), la nature du projet, axée sur l'**intégrité des données**, a favorisé ce choix.

Comparaison entre Bases de Données Relationnelles et Non Relationnelles : Dans le contexte d'un *tournoi multisports*, plusieurs problématiques se posent quant au choix entre une base de données relationnelle (SQL) et une base de données non relationnelle (NoSQL). Voici une analyse détaillée des principales problématiques et des approches respectives SQL et NoSQL [13]:

1. **Modélisation des Données**
   - **SQL:** Utilisation de tables interconnectées par des clés primaires et étrangères. Par exemple, la table *Tourneys* est reliée à *Users* via une table intermédiaire *UsersTourneys*, facilitant la gestion des relations complexes entre entités.
   - **NoSQL:** Les données sont stockées sous forme de documents, de colonnes larges, de graphes ou de paires clé-valeur. Par exemple, les entités `Tourney` et `User` peuvent être stockées dans des documents séparés sans contrainte formelle, ce qui peut compliquer la gestion des relations complexes.
2. **Intégrité des Données**
   - **SQL:** Forte intégrité des données grâce aux contraintes de clé étrangère, aux cascades et aux transactions ACID [14]. Par exemple, la suppression d'un tournoi peut être contrôlée pour éviter la suppression involontaire d'utilisateurs, en configurant des options de cascade comme `ON DELETE CASCADE` ou `SET NULL`.
   - **NoSQL:** Intégrité des données moins stricte en l'absence de schéma rigide, nécessitant une gestion manuelle de la cohérence. Par exemple, la suppression d'un document `Tourney` peut laisser des références orphelines dans d'autres documents.
3. **Requêtes Complexes**
   - **SQL:** Support avancé des requêtes complexes incluant des jointures et des agrégations. Cela permet d'exécuter des requêtes multi-jointures pour extraire l'ensemble des matchs, terrains et équipes en une seule opération.
   - **NoSQL:** Requêtes plus limitées en natif, souvent optimisées pour des cas d'utilisation spécifiques. Par exemple, agréger [15] les matchs par sport et par tournoi peut nécessiter une dénormalisation des données, augmentant le risque de redondance.
4. **Scalabilité:** La scalabilité [31] verticale consiste à ajouter des ressources (CPU, RAM) à un seul serveur pour gérer la charge, tandis que la scalabilité horizontale implique d'ajouter plus de serveurs pour répartir la charge
   - **SQL:** Scalabilité principalement verticale par défaut.
   - **NoSQL:** Scalabilité horizontale native, mieux adaptée aux très gros volumes de données avec une flexibilité de schéma accrue.
5. **Cas d'Utilisation**
   - **SQL:** Idéal pour les données à fortes relations et les transactions critiques, comme la gestion des paiements, de la billetterie ou des droits d'accès.

- **NoSQL:** Adapté aux grands volumes de données nécessitant une grande flexibilité, tels que les réseaux sociaux, les logs massifs ou le streaming.

Conclusion de la comparaison : Après une analyse des différentes problématiques liées à la gestion des données pour Easy Tourney, il apparaît que la nature du projet tend vers une base de données relationnelle pour les raisons suivantes:

- **Gestion des Relations Complexes:** La structure des tournois multisports implique de nombreuses relations entre utilisateurs, matchs, terrains, etc., ce qui est naturellement supporté par un modèle relationnel.
- **Intégrité et Cohérence des Données:** La nécessité de maintenir une forte intégrité des données est cruciale pour éviter les incohérences, particulièrement lors des opérations de suppression ou de mises à jour multiples (génération planning).
- **Écosystème et Outils Disponibles:** L'intégration avec des outils tels que Sequelize et le support d'une communauté active facilitent le développement et la maintenance de la base de données.

Cependant, il est important de reconnaître que dans certains cas spécifiques, une base de données NoSQL pourrait être un bon choix:

- **Gestion des Matchs et Événements en Temps Réel:** Pour des fonctionnalités nécessitant une gestion en temps réel, telles que le suivi des scores en direct ou les événements dynamiques, une base de données NoSQL comme MongoDB ou Redis pourrait offrir des performances et une flexibilité supérieures.
- **Stockage de Données Non Structurées:** Si le projet devait intégrer des données non structurées ou semi-structurées, telles que des logs d'événements, des flux de données ou des contenus multimédias, une solution NoSQL serait plus adaptée.
- **Scénarios de Scalabilité Extrême:** Pour des projets nécessitant une scalabilité horizontale très élevée dès le départ, notamment avec des utilisateurs ou des données à grande échelle, NoSQL peut offrir une meilleure adaptabilité. Cependant dans un contexte scolaire, ce principe ne sera pas appliqué.

En conclusion, MySQL répond parfaitement aux besoins actuels d'Easy Tourney en assurant une gestion efficace des relations entre les différentes entités, adaptée à un projet de nature scolaire.

Modélisation ERM et Diagramme MLD

Comme présenté dans le *Chapitre 2.7 (Analyse)*, la base de données d'Easy Tourney repose sur un **modèle relationnel** solide. Cette section se concentre sur le **Modèle Logique de Données (MLD)** implémenté, en détaillant les choix de modélisation, les défis rencontrés et les améliorations possibles. La figure 4.2 représente le modèle logique de données.

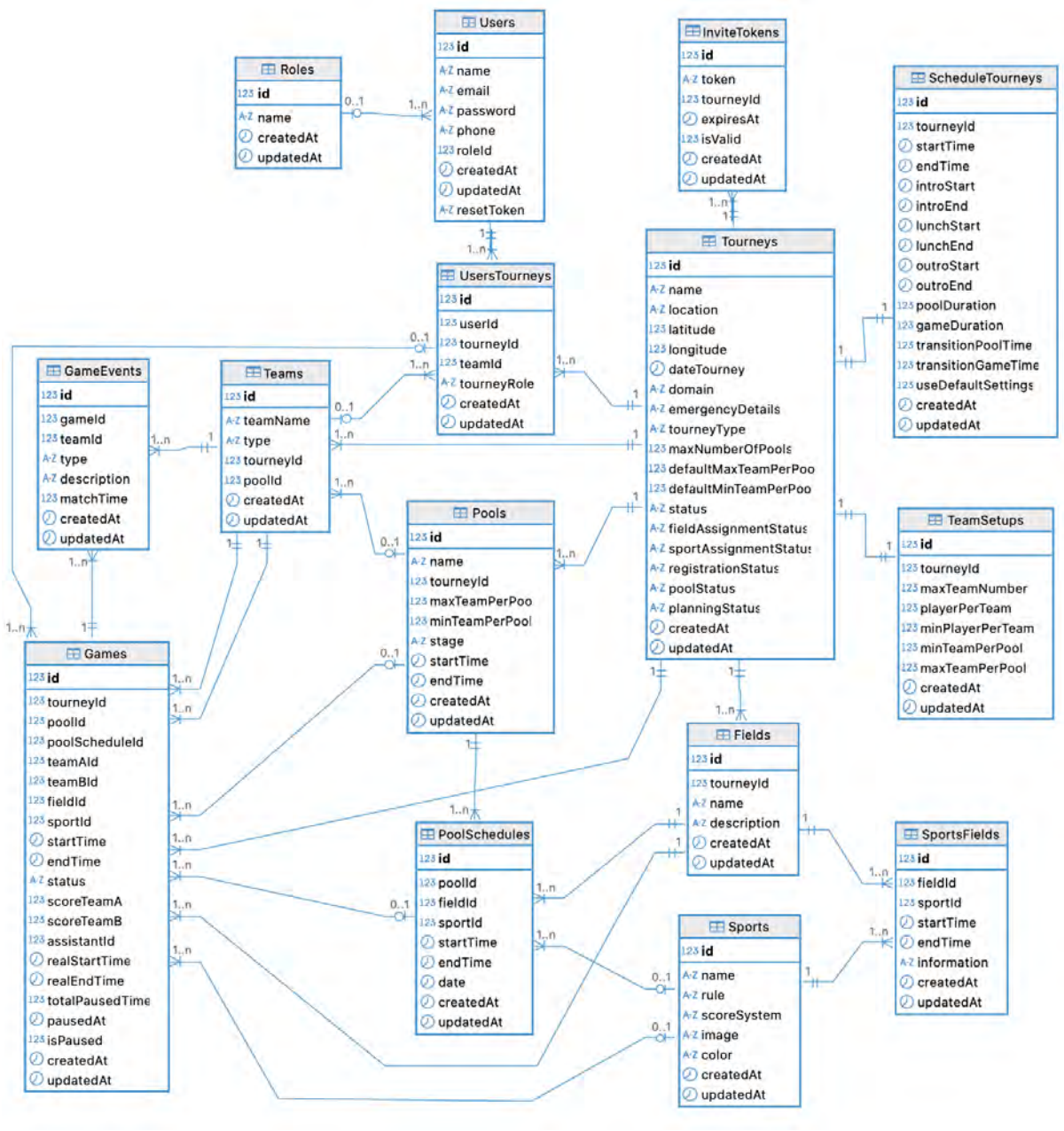


Fig. 4.2.: Modèle Logique de Données (MLD) - Easy Tourney

Analyse des Choix de Modélisation : Pour une compréhension détaillée des entités et de leurs relations, veuillez vous référer à la section 2.7.4 du *Chapitre 2*. Ici, nous discuterons des décisions spécifiques prises lors de l'implémentation du MLD et des implications de ces choix.

- **Système des scores** : Afin d'optimiser les performances et de simplifier les requêtes, les scores des équipes A et B sont stockés directement dans la table `Games` [23]. Cette méthode permet un accès rapide aux résultats des matchs sans la nécessité de gérer une table supplémentaire pour les leaderboards, réduisant ainsi la complexité et améliorant l'efficacité des opérations de lecture et d'écriture.
- **Relations Doublées** : Les relations `Games-Teams` sont représentées par les champs `teamAId` et `teamBId` dans la table `Games`. Cette approche permet de distinguer

clairement les deux équipes participant à un match, simplifiant les requêtes pour extraire les informations des matchs.

- **Gestion des Terrains** : La relation entre `Games` et `Pools` a été établie de manière directe, ce qui peut entraîner des redondances. Une liaison unique via `PoolSchedules` pourrait réduire ces redondances et améliorer l'intégrité des données.
- **Stockage des Dates en UTC** : Initialement, les dates étaient stockées en fonction du fuseau horaire local, ce qui a causé des écarts lors de déploiements sur des plateformes distantes comme Heroku. La solution idéale aurait été de stocker toutes les dates en UTC pour éviter ces problèmes.
- **Optimisation des Requêtes avec Sequelize** : L'utilisation initiale des inclusions non optimales dans Sequelize a conduit à des problèmes de performance, notamment le fameux problème N+1. Une optimisation des requêtes a permis de charger les relations en une seule requête SQL, améliorant considérablement les performances.

Bilan Critique sur la Base de Données

Forces

- **Intégrité des Données** : La gestion rigoureuse des clés primaires/étrangères et l'utilisation des transactions ACID assurent une cohérence et une fiabilité des données, excepté pour 2-3 liens effectués pour des "raccourcis".
- **Flexibilité des Rôles** : La possibilité pour un utilisateur d'avoir des rôles globaux et spécifiques à un tournoi offre une grande flexibilité dans la gestion des permissions et des accès.
- **Abstraction via ORM** : L'utilisation de Sequelize permet de simplifier les interactions avec la base de données, réduisant la dépendance aux requêtes SQL brutes et accélérant le développement.

Limites et Améliorations Potentielles

- **Redondance de Liaisons** : Les liens directs entre `Games` et `Pools`, ainsi qu'entre `Games` et `PoolSchedules`, introduisent des redondances qui peuvent compliquer la maintenance et augmenter le risque d'incohérences.
- **Relation Games – SportsFields** : Lier directement `Games` à `Fields` et `Sports` pourrait entraîner des données orphelines en cas de suppression d'un terrain ou d'un sport. Lier à la table intermédiaire `SportsFields` résout ce problème en maintenant la cohérence des relations, mais l'administrateur pourrait perdre en flexibilité sur des créations de games.
- **Gestion des Fuseaux Horaires** : Le stockage des dates en fonction du fuseau horaire local a causé des problèmes lors du déploiement sur des plateformes distantes. Une gestion uniforme des dates en UTC serait une amélioration souhaitable.
- **Problème N+1 avec Sequelize** : L'initialisation des requêtes avec des inclusions non optimisées a provoqué des ralentissements importants. Bien que cela ait été résolu par une optimisation des requêtes, cela souligne la nécessité d'une maîtrise approfondie de l'ORM utilisé.

Explication du problème N+1 : L'utilisation d'un ORM comme Sequelize facilite la gestion des données sous forme d'objets, mais cette abstraction peut introduire des problèmes de performance, notamment le **problème N+1** [41]. Ce problème survient lorsque l'ORM exécute une requête initiale pour récupérer une collection d'objets (N objets), suivie de requêtes distinctes pour chaque objet afin de charger ses relations. Par exemple, récupérer une liste de voitures et leurs roues pourrait générer une requête pour les voitures, puis une requête pour chaque voiture pour obtenir ses roues, soit N+1 requêtes au total.

Dans le projet Easy Tourney, ce problème s'est produit lors de la récupération des utilisateurs avec leurs tournois associés, entraînant un temps d'exécution initial d'environ 5 secondes, ce qui était inacceptable pour l'expérience utilisateur. Le problème provenait d'une configuration incorrecte des relations dans Sequelize, où les alias (`as`) et les clés étrangères (`foreignKey`) n'étaient pas correctement définis.

Pour résoudre ce problème, j'ai ajusté les relations en définissant explicitement les alias et les clés étrangères, tout en utilisant l'option `through` pour optimiser les relations N-N.

```
exports.getAllUsersWithDetails = async (req, res) => {
  console.time('Execution time for getAllUsersWithDetails'); // Enregistrer temps
  const users = await User.findAll({
    include: [
      { model: Tourney, through: UsersTourneys }, // Chargement des tournois via la table intermédiaire
      { model: Role }, // Chargement des rôles associés
    ],
  });
  console.timeEnd('Execution time for getAllUsersWithDetails'); // Afficher temps
  return res.json(users);
};
```

List. 4.1: Calcul du temps d'une requête avec Sequelize

Grâce à cette optimisation, le temps d'exécution a été réduit à environ 30 à 100 millisecondes, une amélioration significative par rapport aux 5 secondes initiales. Toutefois, bien que le backend soit optimisé, le frontend rencontre encore des lenteurs lors du rendu des données, notamment en raison du volume important d'utilisateurs à afficher (environ 3 secondes pour 400 utilisateurs). Pour résoudre ce problème, un système de pagination est recommandé pour afficher les données par blocs et améliorer l'expérience utilisateur.

Conclusion sur la Base de Données : Dans l'ensemble, MySQL offre une base solide pour gérer les données complexes d'Easy Tourney. Les retours d'expérience sur l'ORM (Sequelize) et le besoin d'une structuration claire (par exemple, clarifier le lien *Games–PoolSchedules*) constituent des pistes d'amélioration. Au final, le modèle relationnel s'avère tout à fait pertinent pour ce type d'application.

Note - Développement Individuel et Refactorings : Travailler seul, parfois sous la fatigue, a conduit à des erreurs ou des choix hâtifs, nécessitant par la suite plusieurs refactorings (conventions de nommage, organisation du backend, réécriture partielle du

MLD). Ces erreurs, comme des problèmes de fuseaux horaires (UTC) ou des relations mal configurées, n'ont parfois été détectées que plusieurs jours, voire semaines, après leur introduction. Une solution efficace aurait été de travailler en binôme, avec une revue systématique des *pull requests* ou *merge requests*, afin de réduire ces erreurs en amont.

Malgré cela, l'utilisation d'un système de versionnement sur GitHub s'est révélée salvatrice à plusieurs reprises, permettant de revenir facilement à une version stable ou de retracer l'origine de certains bugs. Cela a considérablement aidé à maintenir la continuité et la qualité du développement.

Malgré tout, toutes les fonctionnalités clés ont été implémentées. Les difficultés rencontrées (relation `Games-Pools`, fuseau horaire Heroku, problème N+1) ont été partiellement résolues et ont renforcé mes compétences en conception de bases de données et en optimisation backend.

Suggestions pour des Améliorations Futures

Afin d'améliorer la structure actuelle de la base de données, plusieurs pistes peuvent être envisagées:

- **Réduction des Redondances** : Simplifier les relations entre `Games`, `Pools` et `PoolSchedules` en établissant des liaisons uniques, ce qui réduirait les incohérences et faciliterait la maintenance.
- **Gestion Optimisée des Dates** : Standardiser le stockage des dates en UTC pour éviter les problèmes de fuseaux horaires lors des déploiements sur des plateformes distantes.

En implémentant ces améliorations, Easy Tourney pourrait gagner en robustesse, répondant ainsi de manière encore plus précise aux exigences croissantes de la gestion de tournois multisports.

4.3. Structure Frontend et Backend

Dans la section précédente, nous avons présenté l'**architecture physique** d'*Easy Tourney* (figure 4.1). Dans ce chapitre, nous nous concentrons sur les éléments clés de la **structure logicielle** du frontend et du backend pour fournir au lecteur une compréhension claire avant d'explorer les fonctionnalités implémentées.
L'objectif est d'expliquer les concepts essentiels qui auront un impact direct sur la suite du rapport.
Note : La structure complète de l'application est disponible sur Github : `https://github.com/JulienRichoz/easy-tourney`

4.3.1. Structure Frontend

L'application **Easy Tourney** suit le paradigme des **SPA** [42]. Ce modèle offre une navigation rapide en chargeant une seule fois les ressources nécessaires et en interagissant dynamiquement avec le backend via des API REST ou des WebSockets.
La structure simplifiée du répertoire `src` est la suivante:

```
frontend/
|-- components/
|   |-- AuthComponent.vue
|   |-- ButtonComponent.vue
|   |-- CardAddComponent.vue
|   |-- ...
|-- views/
|   |-- App.vue
|   |-- admin/
|   |-- user/
|   |-- ...
|-- router/
|   |-- index.js
|   |-- adminRoutes.js
|   |-- guards/
|   |-- ...
|-- store/
|   |-- modules/
|   |   |-- tourney.js
|   |   |-- userTourney.js
|   |   |-- ...
|   |-- index.js
|-- services/
|   |-- apiService.js
|   |-- authService.js
|   |-- socketService.js
|   |-- ...
|-- ...
```

Les dossiers principaux jouent les rôles suivants:

- /**components** : Les composants [28] Vue.js constituent les blocs de construction de l'interface utilisateur. Chaque composant est autonome, réutilisable, et améliore la modularité du code. Par exemple, un composant comme `ButtonComponent.vue` encapsule l'apparence et le comportement d'un bouton, évitant les répétitions dans le code.
- /**store** : Le store [25] centralise la gestion de l'état global de l'application. Il permet de partager et synchroniser des données globalement entre les composants, telles que les informations liées aux utilisateurs ou tournois via des actions et mutations. Une explication détaillée du fonctionnement du store est présente dans l'appendice. Il faut différencier la mémoire du `store` qui est volatile et le `localStorage` qui est persistant.
- /**views** : Représente les pages principales (login, planning, gestion des équipes, etc.).
- /**router** : Configure les routes de navigation (par exemple, `/admin`, `/tourney/:id`). Les guards y sont définis pour gérer l'authentification ou les permissions.
- /**services** : Contient les services d'API (*apiService.js*), d'authentification (*authService.js*) et de WebSocket (*socketService.js*).

4.3.2. Structure Backend

Le backend utilise une architecture **MVC** (Modèle-Vue-Contrôleur) pour gérer la logique métier et les données, laissant au frontend la responsabilité de leur affichage dynamique. Ce choix a été motivé par mon expérience positive avec le framework Laravel, qui implémente également le pattern MVC et offre une structure claire et maintenable.

Définitions : Le modèle MVC [20] est une architecture logicielle qui sépare les responsabilités en trois composants principaux :

- **Modèle** : Gère les données et la logique métier.
- **Vue** : S'occupe de l'interface utilisateur.
- **Contrôleur** : Sert d'intermédiaire en traitant les requêtes et en mettant à jour le modèle ou la vue.

Note : Le client étant construit en SPA, les routes du backend fournissent principalement des données en JSON plutôt que des vues HTML complètes, ce qui associe le rôle des routes à celui de la Vue côté client.

Définition Migrations et Seeders :

- **Migration** : Versionne et automatise les modifications de la structure de la base de données, assurant la synchronisation du schéma entre les développeurs. Permet de créer les tables de l'application dans la base de données.
- **Seeder** : Insère automatiquement des données initiales ou de test, facilitant le préremplissage de la base avec des données fictives telles que des utilisateurs, équipes et tournois.

```
backend/
|-- config/
|-- controllers/
|   |-- authController.js
|   |-- tourneyController.js
|   `-- ...
|-- models/
|   |-- User.js
|   |-- Tourney.js
|   |-- Team.js
|   `-- ...
|-- routes/
|   |-- auth.js
|   |-- tourney.js
|   |-- team.js
|   `-- ...
|-- middlewares/
|   |-- authenticateToken.js
|   |-- authorizeTourneyRole.js
|   `-- ...
|-- services/
|   |-- authService.js
|   |-- planningStrategies.js
|   |-- mailerService.js
|   `-- ...
|-- migrations/
|-- seeders/
`-- ...
```

Les dossiers principaux sont définis comme suit :

- /**models** : Contient les entités Sequelize (`User.js`, `Tourney.js`, etc.), qui décrivent les tables et leurs relations.
- /**controllers** : Implémente la logique métier (authentification, gestion des tournois, etc.).
- /**routes** : Définit les endpoints REST, par exemple `POST /api/tourneys`.
- /**middlewares** : Gestion des permissions (`authorizeTourneyRole.js`) et validation des tokens JWT (`authenticateToken.js`).
- /**services** : Logique réutilisable comme la gestion des mots de passe, l'authentification, et les algorithmes de planification (*Round Robin*, etc.).
- /**migrations et /seeders** : Scripts Sequelize pour évoluer la structure des bases de données et insérer des données initiales.

Cette architecture distribue les responsabilités entre le backend, qui gère les données et la logique métier, et le frontend, qui se concentre sur l'affichage et les interactions utilisateur. Ce choix permet de construire une application maintenable, évolutive, et performante, tout en exploitant les avantages des SPA modernes.

4.4. Système d'Authentification

J'ai choisi de développer un système d'authentification from scratch pour comprendre en profondeur les mécanismes de la gestion des utilisateurs, des mots de passe, et des tokens. Bien que la sécurité soit un domaine complexe et mieux géré par des solutions tierces (comme OAuth2 ou Google Auth) [33], ce choix pédagogique m'a permis de mieux comprendre les principes clés de l'authentification.

Le système implémenté repose sur l'utilisation de JSON Web Tokens (JWT) pour la gestion des sessions utilisateurs. Ce chapitre présente la conception et le fonctionnement de ce système, structuré en deux grandes parties :

1. La création et la gestion des utilisateurs au niveau du backend, incluant le stockage sécurisé des mots de passe et la génération des tokens.
2. L'authentification des utilisateurs, illustrant le processus de validation des tokens et leur utilisation côté frontend avec Vuex.

Ces sections explorent également les principes de sécurité, tels que la limitation des tentatives de connexion (*rate limiting*) et l'expiration des sessions, pour protéger l'application contre des menaces courantes.

4.4.1. Stockage sécurisé des mots de passe

Toutes les fonctionnalités liées à l'authentification (hachage, comparaison de mot de passe, génération de token, etc.) sont centralisées dans le fichier *server/services/authService.js*[1]. Cette organisation permet de gérer de manière uniforme et évolutive les aspects critiques de la sécurité. Lors de l'inscription, les mots de passe sont hachés avant d'être enregistrés dans la base de données, éliminant ainsi tout stockage en clair, conformément aux meilleures pratiques.

```
// Fonction pour hasher un mot de passe
exports.hashPassword = async (plainPassword) => {
  return await argon2.hash(plainPassword);
};

// Fonction pour comparer les mots de passe
exports.comparePassword = async (plainPassword, hashedPassword) => {
  try {
    return await argon2.verify(hashedPassword, plainPassword);
  } catch (error) {
    console.error('Erreur lors de la verification du mot de passe:', error);
    return false; // Retourne 'false' en cas d'echec
  }
};
```

List. 4.2: Hachage et comparaison des mots de passe avec Argon2

Le fonctionnement est le suivant :

[1]Code source-authService: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/services/authService.js`

- **Hachage de mot de passe** : Lorsqu'un utilisateur s'inscrit, la fonction `hashPassword` applique Argon2 pour générer un hachage sécurisé de son mot de passe. Ce hachage inclut des *salts* et des paramètres spécifiques (mémoire, temps, parallélisme), augmentant la sécurité.
- **Vérification de mot de passe** : Lors de la connexion, `comparePassword` compare le mot de passe fourni avec le hachage stocké. Si la vérification échoue (par exemple, mot de passe incorrect ou hachage invalide), la fonction retourne `false`.

Ces mécanismes garantissent que seuls des hachages sécurisés transitent dans le système, minimisant les risques en cas de fuite de données.
Lorsqu'un utilisateur créé un compte, son mot de passe est stocké hashé:

```
// server/controllers/authController.js (register)
const newUser = await User.create({
      name,
      email,
      phone,
      password: hashedPassword,
      roleId: userRole.id,
});
```

List. 4.3: Stockage du mot de passe haché dans la DB.

Sequelize a créé un utilisateur et nous pouvons voir avec la figure 4.3 que le mot de passe est bien hashé dans la base de données.

id	name	email	password	phone	roleId
67	High User 63	highuser63@example.com	$argon2id$v=19$m=65536,t=3,p=4$SrHL8N0Tq00LPPuS7VmpZA$uHpePoKxEL...	NULL	2

Fig. 4.3.: Mot de passe hashé

4.4.2. Utilisation des JSON Web Tokens (JWT)

Les **JWT** [34] sont des tokens sécurisés utilisés pour gérer les sessions utilisateur de manière stateless [52]. Chaque token contient une charge utile (*payload*) encodée, comme l'`id`, le `roleId` et le `name` de l'utilisateur, permettant au serveur de valider les requêtes sans conserver d'état côté backend[2].

```
// Fonction pour generer un token JWT avec une duree d'expiration variable
exports.generateToken = (user) => {
  return jwt.sign(
    { id: user.id, roleId: user.roleId, name: user.name },
    process.env.JWT_SECRET,
    { expiresIn: process.env.JWT_EXPIRES_IN || '1h' } // Utilisation de la duree
        d'expiration de l'environnement
  );
};
```

List. 4.4: Génération d'un JWT dans le backend

[2]Les JWT améliorent la scalabilité des systèmes en éliminant le besoin de stocker les sessions côté serveur.

On remarque que des variables d'environnement sont utilisées, gardées secrètement dans un fichier `.env`. Le token a également un paramètre d'expiration pour des raisons de sécurité (quelqu'un en possession d'un token sans expiration aurait constamment accès au profil utilisateur par exemple).

4.4.3. Prévention des abus : limitations et contraintes

En plus d'une durée d'expiration, d'autres mesures ont été prises pour limiter les abus, comme les attaques par force brute. Un *rate limiter*[3] est configuré avec `express-rate-limit`, limitant le nombre de requêtes autorisées par IP dans une période donnée.

```
const limiter = rateLimit({
  windowMs: 1 * 60 * 1000, // 1 minute
  max: 100, // Limite : 100 requetes par IP
  message: "Trop de requetes, veuillez reessayer plus tard."
});
```

List. 4.5: Configuration d'un *rate limiter*

Les mots de passe doivent également respecter des contraintes de complexité (au moins 8 caractères et une majuscule), garantissant un niveau minimal de sécurité :

```
const passwordRegex = /^(?=.*[A-Z]).{8,}$/;
if (!passwordRegex.test(password)) {
  return res.status(400).json({
    message: "Le mot de passe doit contenir au moins 8 caracteres et une
        majuscule."
  });
}
```

List. 4.6: Validation de la complexité des mots de passe

4.4.4. Résumé

Ce système d'authentification repose sur des principes éprouvés pour garantir la sécurité :

- Utilisation d'Argon2 pour le hachage des mots de passe, assurant une résistance aux attaques matérielles.
- Gestion des sessions utilisateur via des JWT, offrant une scalabilité accrue.
- Prévention des abus grâce à des limitations de requêtes et des règles de validation strictes.

4.4.5. Traitement du Token côté Vuex et Router Vue

Lors d'une requête de connexion, le contrôleur vérifie les identifiants et s'ils sont valides, retourne un token avec diverses informations que l'utilisateur enregistera dans le store :

[3]Code source-rateLimiter: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/middlewares/index.js`

Fig. 4.4.: Réponse JSON du token JWT

La figure 4.4 représente le JSON retourné par le serveur vers le client. La capture d'écran provient de Postman [22], une application permettant de tester les API que j'ai énormément utilisé pour mes tests d'intégration.

Le frontend gère l'état de l'authentification via le *store Vuex*. Les tokens JWT sont stockés localement et leur validité est vérifiée avant chaque action.

Router Vue - contrôle à chaque navigation : Dans `frontend/router/index.js`[4], un `beforeEach` intercepte la navigation. Il vérifie :

- Si l'utilisateur est authentifié.
- Si le token n'est pas expiré, grâce à une fonction de validation `isTokenExpired()`.
- Les permissions requises par la route (`to.meta.permission`).

Si une condition échoue, l'utilisateur est redirigé, soit vers la page de connexion, soit vers une page d'accès interdit. Voici un extrait du `beforeEach` :

```
router.beforeEach(async (to, from, next) => {
  const token = localStorage.getItem('token');
  const isAuthenticated = !!token;

  if (to.meta.requiresAuth && (!isAuthenticated || isTokenExpired())) {
    return next('/login'); // Redirige si l'utilisateur n'est pas authentifie
  }

  if (to.meta.permission) {
    const decoded = jwtDecode(token);
    if (!hasPermission(decoded.roleId, to.meta.permission)) {
      return next('/access-denied');
    }
  }

  next();
});
```

List. 4.7: Middleware `beforeEach` pour la navigation dans Vue Router

Le router.beforeEach est appelé avant chaque requête et vérifiera en permanence si l'utilisateur a les droits ou non d'accéder à une page.

[4]Code source-router.beforeEach: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/frontend/src/router/index.js`

4.4.6. Gestion des Tokens Expirés

Pour gérer les tokens expirés, le frontend utilise un mécanisme de `refreshToken()`. Lorsqu'un token est sur le point d'expirer, une requête est envoyée au backend pour obtenir un nouveau token, qui est ensuite sauvegardé. Voici comment cela fonctionne :

```
export async function refreshToken() {
  const response = await apiService.post('/auth/refresh-token');
  const newToken = response.data.token;
  localStorage.setItem('token', newToken);
  apiService.defaults.headers.common['Authorization'] = `Bearer ${newToken}`;
}
```

List. 4.8: Rafraîchissement automatique des tokens JWT

Ce mécanisme assure une continuité de session sans nécessiter de nouvelle connexion pour l'utilisateur.

4.4.7. Exemple de flux utilisateur :

1. **Connexion :** L'utilisateur se connecte, reçoit un token, et celui-ci est stocké dans le frontend.
2. **Navigation :** Avant chaque changement de page, le frontend vérifie si le token est valide.
3. **Requêtes backend :** Le token est envoyé avec chaque requête pour que le backend puisse identifier l'utilisateur.
4. **Renouvellement token :** A chaque requête serveur, le token est renouvelé.

4.4.8. Retour d'Expérience: Stockage dans `localStorage` et Projet d'Évolution vers les Cookies

Après avoir implémenté mon système d'authentification avec des JWT, j'ai initialement choisi de **stocker** le token dans le `localStorage` du navigateur. Cela m'a paru simple à l'époque et fonctionnel dans une grande partie des cas. Cependant, au fil de discussions avec des collègues et en consultant des ressources (ex. Reddit [55], j'ai découvert que ce choix était **vulnérable** en raison des attaques XSS (Cross-Site Scripting [56].

Mes limites actuelles : `localStorage` toujours en place

Sécurité et gestion du token côté frontend : Comme détaillé précédemment, stocker le JWT dans le `localStorage` implique que:

- N'importe quel script malveillant (injection XSS) peut potentiellement lire le `localStorage` et voler le token.
- Le frontend doit gérer la validité du token (expiration, *refresh*, etc.) et injecter manuellement le token dans l'en-tête `Authorization` à chaque requête.

Je me suis rendu compte que cela rajoute une **couche de complexité** non négligeable côté frontend, au lieu de laisser le serveur se charger des contrôles (dates d'expiration, droits, etc.).

Début de migration vers les cookies : Suite à cette prise de conscience, j'ai entamé la création d'une *branche Git* où je teste le stockage du JWT dans un `cookie httpOnly`. Cette approche devrait :

- Rendre le token inaccessible au JavaScript, neutralisant la menace XSS sur ce point particulier.
- Simplifier la logique frontend, puisque le navigateur enverra automatiquement le cookie au serveur, qui validera l'authentification et renverra un code d'erreur `401` ou `403` en cas de problème.

Toutefois, la mise en place d'un cookie `httpOnly` introduit d'autres considérations comme la gestion **CSRF**(Cross-Site Request Forgery) [57], où des requêtes malveillantes pourraient faire usage du cookie si des contre-mesures ne sont pas prises (par ex. `sameSite`, `CSRF token`, etc.).

Avantages et inconvénients comparés

Aspect	**Stockage dans** `localStorage`	**Stockage dans un cookie** `httpOnly`
Vulnérabilité XSS	Token accessible en clair, plus à risque.	Token masqué en JS, largement protégé.
Gestion token	Le frontend doit tout gérer (expiration, refresh, ...).	Le serveur gère la validité, le front se base sur les codes d'erreur.
CSRF	Moins critique (le token n'est pas envoyé automatiquement).	Doit mettre en place un anti-CSRF (ex: `sameSite=strict` ou `CSRF token`).

Tab. 4.1.: Comparaison `localStorage` vs cookie `httpOnly`

Mon bilan: apprentissage par l'erreur

Aujourd'hui, je maintiens encore la version `localStorage` pour ne pas tout casser, mais je prévois de finaliser la branche github *cookie-jwt* dès que possible. J'admets avoir découvert ces problématiques de sécurité un peu tard, mais cela m'a fait énormément progresser sur la compréhension des systèmes d'authentification et de risques XSS/CSRF.
Comme le disait *Mark Twain* :

'Good decisions come from experience, but experience comes from making bad decisions.'

Ce projet illustre parfaitement que mes erreurs m'ont contraint à me plonger dans les subtilités de la sécurité web, une démarche que je n'aurais pas entreprise en utilisant systématiquement des systèmes d'authentification préfabriqués ou des frameworks. En

effet, en choisissant de développer un système from scratch, j'ai fait face à des défis tel que la gestion manuelle de token frontend, qui m'ont obligé à approfondir mes connaissances et à comprendre en détail les principes sous-jacents de l'authentification sécurisée. D'un point de vue pédagogique, cette approche a été extrêmement bénéfique : elle m'a permis d'acquérir une vision bien plus claire des enjeux réels de la protection des tokens, de la gestion des statuts HTTP (`401`, `403`, `404`, etc.), et du rôle crucial du serveur dans la validation de chaque requête. Ainsi, ce qui aurait pu être perçu comme un échec initial s'est révélé être (du moins personnellement) une véritable réussite en termes d'apprentissage et de compréhension des concepts de sécurité web.

En résumé

- **LocalStorage** : demeure fonctionnel mais moins sûr face à la XSS, et alourdit la logique côté frontend.
- **Cookie** `httpOnly` : nécessite une bonne configuration (CSRF, `secure`, `sameSite`), mais retire un grand poids du frontend et augmente la sécurité globale.

Ce sera une de mes priorités si je poursuis la maintenance de ce projet, dans le but d'offrir un système plus robuste et securisé.

4.5. Diagramme de séquence détaillé de la création d'un terrain

Dans les chapitres précédents, nous avons exploré le système d'authentification de l'application ainsi que la manière dont le frontend gère les tokens pour la navigation via le middleware Vue Router `beforeEach`. Nous allons à présent examiner la gestion des middlewares côté backend et leur interaction avec le modèle MVC lors de la création d'un terrain dans un tournoi.

Ce chapitre dévoile, à travers un diagramme de séquence personnalisé, le cheminement complet d'une requête utilisateur, de l'action de cliquer sur "Créer un terrain" jusqu'à son insertion dans la base de données. Nous présenterons ensuite les avantages de cette architecture modulaire et un exemple rapide de la gestion des rôles au niveau des middlewares.

Diagramme de séquence

Note : Le diagramme de la figure 4.5 peut être appelé un diagramme de séquence, bien qu'il s'éloigne des conventions strictes de l'UML en représentant des fichiers ou composants comme entités. Il illustre le flux temporel des interactions pour la création d'un terrain entre le frontend et le backend, ce qui est son objectif principal.

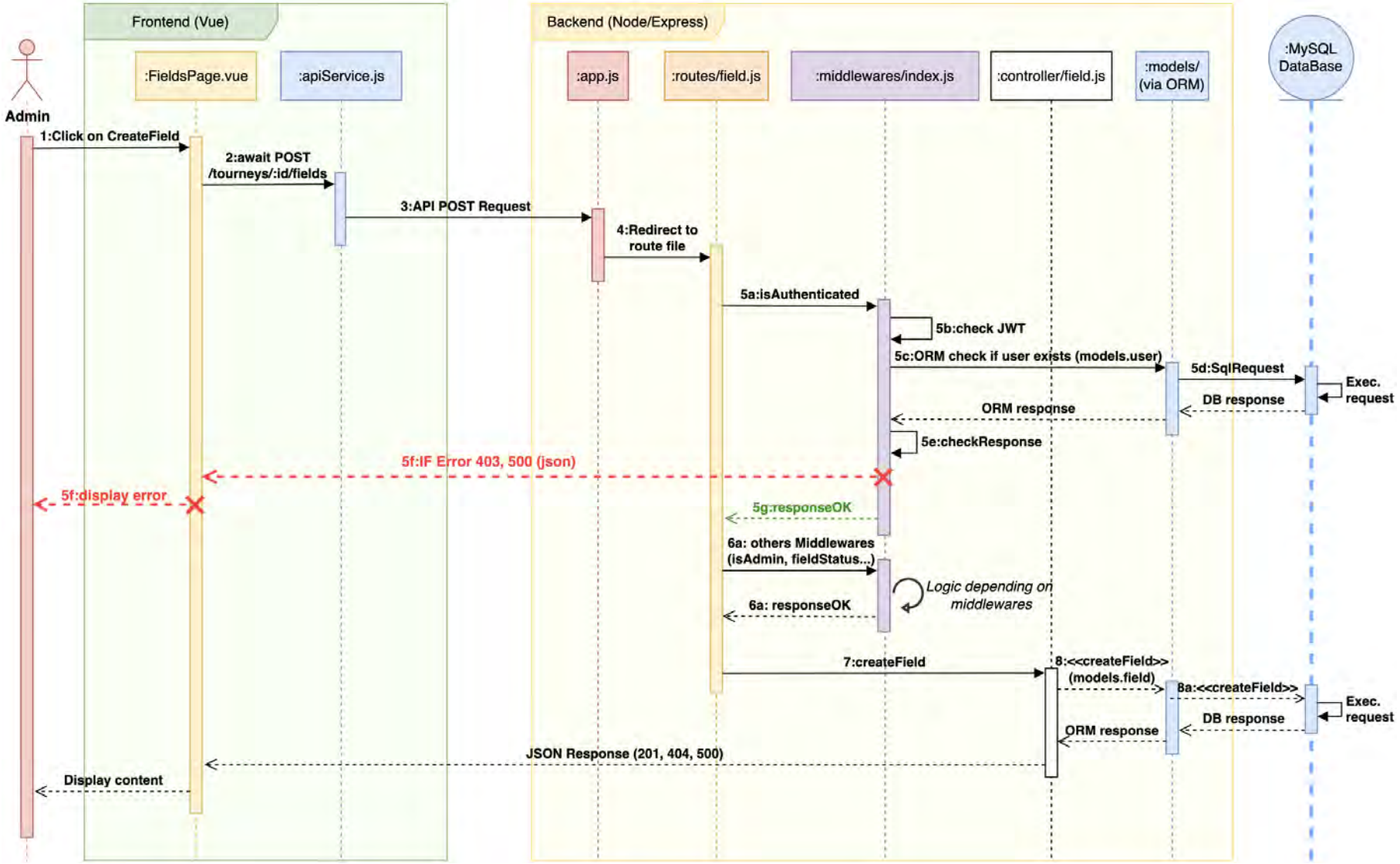


Fig. 4.5.: Diagramme de séquence personnalisé pour la création d'un terrain

4.5.1. Déroulement des étapes

Voici les différentes étapes du processus illustré dans le diagramme, avec une explication et des extraits de code pertinents.

1. et 2. Action de l'utilisateur : clic sur `CreateField`

L'utilisateur (administrateur) clique sur le bouton `Créer un terrain` depuis la page Vue `FieldsPage.vue`[5], ce qui déclenche une requête POST via l'appel de `apiService`.

```
//frontend/src/views/admin/tourneys/TourneyFieldsManagement.vue
async saveField() {
    await apiService.post(
        `/tourneys/${this.tourneyId}/fields`,
        this.newField
    );
    toast.success('Nouveau terrain ajoute avec succes!');
    }
}
```

List. 4.9: Admin clique sur "Créer un terrain"

3. Requête envoyée par `apiService.js`

Le fichier `apiService.js`[6] utilise Axios pour envoyer une requête POST vers le backend. Un token JWT est inclus dans les en-têtes pour authentifier l'utilisateur. L'adresse de l'API est accessible via le fichier `.env`.

```
//frontend/src/services/apiService.js

// Ajoute le token si disponible
apiService.interceptors.request.use((config) => {
  const token = localStorage.getItem('token'); // Recuperation du token JWT
  if (token) {
    config.headers['Authorization'] = `Bearer ${token}`;
  }
  return config;
});
```

List. 4.10: Axios Requête POST vers l'API

4. Redirection dans `app.js` (Backend)

La requête arrive sur le serveur backend au point d'entrée `app.js` et est redirigée vers le fichier de routes correspondant `routes/field.js`[7].

[5]Code source-FieldsPage: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/frontend/src/views/admin/tourneys/TourneyFieldsManagement.vue`

[6]Code source-apiService: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/frontend/src/services/apiService.js`

[7]Code source-FieldRoute: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/routes/field.js`

```
//server/app.js
app.use('/api/tourneys/:tourneyId/fields', fieldRoutes);
```

List. 4.11: Point d'entrée de l'application serveur

5. Gestion dans `routes/field.js`

Le fichier de routes appelle une chaîne de middlewares[8] pour vérifier les permissions de l'utilisateur et le statut du tournoi, pour appeler la méthode du contrôleur `createField` si les middlewares ont réussi.

```
//server/routes/field.js
router.post('/', isAuthenticated, isAdmin, verifyFieldAssignmentStatus,
    createField);
```

List. 4.12: Middlewares dans la route field

5a. Middleware `isAuthenticated`

Ce middleware valide le token JWT et vérifie que l'utilisateur est toujours actif tout en récupérant son **rôle** dans la base de données via le modèle `User`. La récupération du rôle est cruciale si jamais un administrateur se fait rétrograder alors qu'il possède encore un token admin valide. **Il y a donc une vérification sur le token et dans la base de données**.

```
//server/middlewares/index.js
// Verifier le token
try {
const decoded = jwt.verify(token, process.env.JWT_SECRET);
req.user = decoded;

// Verifier si l'utilisateur existe toujours dans la base de donnees
const user = await User.findByPk(req.user.id);
if (!user) {
  return res
    .status(401)
    .json({
      message: "Utilisateur non trouve. Veuillez vous reconnecter.",
    });
}
//Recuperer le role de la requete User
req.user.isAdmin = req.user.roleId === roles.ADMIN;
next();
```

List. 4.13: Middleware 'isAuthenticated'

On aperçoit, conformément au diagramme de séquence de la figure 4.5, que le middleware va interroger le modèle `User` qui effectuera une requête vers la base de données MySql, le tout avec l'ORM Sequelize.

[8]Code source-Middlewares: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/middlewares/index.js`

6a. Middlewares `isAdmin` et `verifyFieldAssignmentStatus`

Les middlewares suivants sont executés dans l'ordre. Si un échoue, l'opération est arrêtée.
- Le middleware `isAdmin` vérifie le rôle d'utilisateur

```
//server/middlewares/index.js
const isAdmin = (req, res, next) => {
  const user = req.user; // Utilisateur doit etre defini apres la verification
      du token
  if (user && user.roleId === roles.ADMIN) {
    return next();
  }
};
```

List. 4.14: Middleware 'isAdmin'

- Le middleware `verifyFieldAssignmentStatus` vérifie si le terrain est en mode *Edition* ou *Terminé*, via le modèle `Tourney`.

```
//server/middlewares/statusMiddleware.js
const verifyFieldAssignmentStatus = async (req, res, next) => {
  const { tourneyId } = req.params;
  const tourney = await Tourney.findByPk(tourneyId); // Recuperation du tournoi
      associe

  if (tourney.fieldAssignmentStatus === "completed") {
      return res
        .status(403)
        .json({
          message:
            "L'action requiert que la page Terrain soit en mode edition.",
        });
    }

    next(); // Passer a la prochaine etape si le statut est correct
```

List. 4.15: Middleware 'verifyFieldAssignmentStatus'

Note : Lors d'une requête POST, des paramètres peuvent être fournis soit dans l'URL (comme le tourneyId), soit dans le payload de la requête HTTP (comme le nom du terrain).

7. Contrôleur `createField`

Les middlewares validés, la méthode du contrôleur[9] est appelée pour créer un nouveau terrain dans la base de données via l'ORM Sequelize et retourner une réponse JSON.

```
//server/controllers/fieldController.js
const { name, description } = req.body;
const { tourneyId } = req.params;

```

[9]Code source-CreateField: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/controllers/fieldController.js`

```
exports.createField = async (req, res) => {
    const field = await Field.create({ name, description, tourneyId });
    res.status(201).json(field);
};
```

List. 4.16: Méthode 'createField' du controller 'field'

Exemple de requête POST avec Postman : La figure 4.6 illustre une requête POST effectuée via Postman pour créer un terrain. L'URL inclut le paramètre `tourneyId`, ici fixé à 1, et le body de la requête contient les champs `name` et `description`. Un token admin a été inclut dans l'en-tête de la requête.

À droite (Fig. 4.7), on observe la réponse JSON retournée par le serveur, confirmant que le terrain a été créé avec succès. Le code HTTP 201 indique que la création a été effectuée sans erreur. Notez que le serveur ne retourne que les informations nécessaires pour éviter toute fuite de données sensibles (e.g., mots de passe).

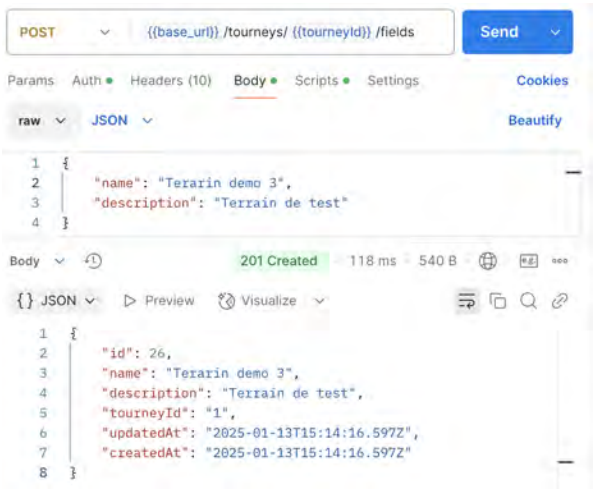


Fig. 4.6.: Requête POST dans Postman pour créer un terrain

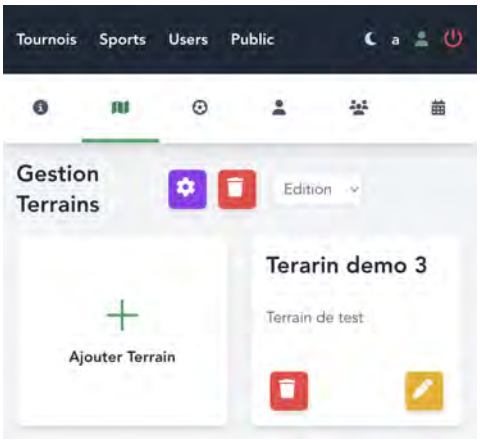


Fig. 4.7.: Terrain créé dans la vue réactivement

Fig. 4.8.: Création d'un terrain via une requête POST avec Postman.

8. Interaction avec l'ORM et la Base de données

Le modèle Sequelize `Field`[10] gère l'insertion dans la base de données. Voici un aperçu du modèle :

```
const { Model } = require('sequelize');

class Field extends Model {
  static associate(models) {
    Field.belongsTo(models.Tourney, {
      as: 'tourney',
```

[10]Code source-FieldModel: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/models/field.js`

```
      foreignKey: 'tourneyId',
      onDelete: 'CASCADE',
    });
  }
}

Field.init({
  name: DataTypes.STRING,
  description: DataTypes.TEXT,
  tourneyId: DataTypes.INTEGER,
});

return Field;
```

List. 4.17: Présentation du Modèle Field

Affichage réactif des terrains avec Vue.js

Le frontend reçoit maintenant les données du terrain en JSON (figure 4.6). Vue.js permet un affichage réactif grâce à son système de liaison entre les données et l'interface utilisateur. Dans le cas des terrains, l'état réactif est géré via la propriété fields et les directives de Vue.
Les prochains extraits de codes concernent la page ***TourneyFieldsManagement.vue***[11].

Déclaration des données réactives : La propriété fields est définie dans la section data du composant, ce qui en fait une donnée réactive :

```
data()
{
    return {
        fields: [], // Tableau des terrains
        ..., // autres donnees
        };
},
```

List. 4.18: Déclaration des données réactives

Toute modification de ce tableau est automatiquement reflétée dans l'interface utilisateur.

Rendu conditionnel avec v-for : Les terrains sont affichés dynamiquement dans une grille en utilisant la directive v-for, qui parcourt le tableau fields:

```
 // <!-- Cartes des terrains existants -->
<CardEditComponent
  v-for="field in fields"
  :key="field.id"
  :title="field.name"
  :description="field.description"
```

[11] Code source-FieldsManagement: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/frontend/src/views/admin/tourneys/TourneyFieldsManagement.vue`

```
  :showDeleteButton="isEditable"
  :showEditButton="true"
  :isEditable="isEditable"
  @delete="confirmDeleteField(field.id)"
  @edit="editField(field)"
  @click="editField(field)"
/>
</div>
```

List. 4.19: Rendu dynamique avec v-for

Lorsque le tableau fields est mis à jour, Vue ajuste automatiquement le DOM en fonction des changements.

Note : Le composant `CardEditComponent` est un composant personnalisé qui est réutilisé dans de nombreuses pages de l'application. On peut voir de nombreuses propriétés (props) qui permettent de rendre ce composant modulable et réutilisable selon les données à afficher. Il est par exemple utilisé pour lister les sports, les équipes, les pools, les tournois ou les terrains en garantissant un design commun.

Réaction au montage du composant : Lors du montage initial du composant, les terrains sont récupérés depuis le backend pour être affichés :

```
mounted(){
    this.fetchFieldDetails(); // Recupere les terrains au chargement de la page
}
```

List. 4.20: Chargement initial des terrains

Grâce à la réactivité de Vue, toute modification des données (via des méthodes ou des API) est immédiatement reflétée dans l'interface utilisateur, sans nécessiter un rechargement de la page. Ce mécanisme permet une expérience utilisateur fluide et moderne.

Conclusion

Ce diagramme de séquence personnalisé met en évidence les avantages du modèle MVC et des middlewares. Ceux-ci permettent une validation modulaire, laissant le contrôleur focalisé sur la logique métier. L'exemple présenté offre une vue d'ensemble complète des interactions entre le frontend, les middlewares, le contrôleur, base de données et l'affichage réactif par Vue.js.

Une logique similaire est implémentée pour gérer les rôles liés aux tournois (e.g., arbitrage), en ajoutant simplement un middleware supplémentaire pour vérifier les permissions de l'utilisateur au tournoi. Ce système permet donc de gérer les droits globaux et liés aux tournois de manière très propre et logique.

4.6. Fonctionnalités Avancées

La structure de navigation et l'architecture globale présentée, nous pouvons à présent nous concentrer sur les implémentations techniques de l'application.

4.6.1. Rôles Globaux vs. Rôles liés au Tournoi

Dans *Easy Tourney*, chaque **utilisateur** possède un **rôle global** (`Admin` ou `User`) *et* un **rôle spécifique à chaque tournoi** (`Assistant`, `Player`, `Guest`). Cela permet à un utilisateur d'avoir des responsabilités et des autorisations différentes selon le contexte. Par exemple, un utilisateur globalement classé comme `User` peut occuper le rôle d'`Assistant` dans un tournoi particulier ce qui lui confère des droits étendus localement (comme la gestion des scores ou des événements du match), tout comme un rôle `Player` dans un autre tournoi.

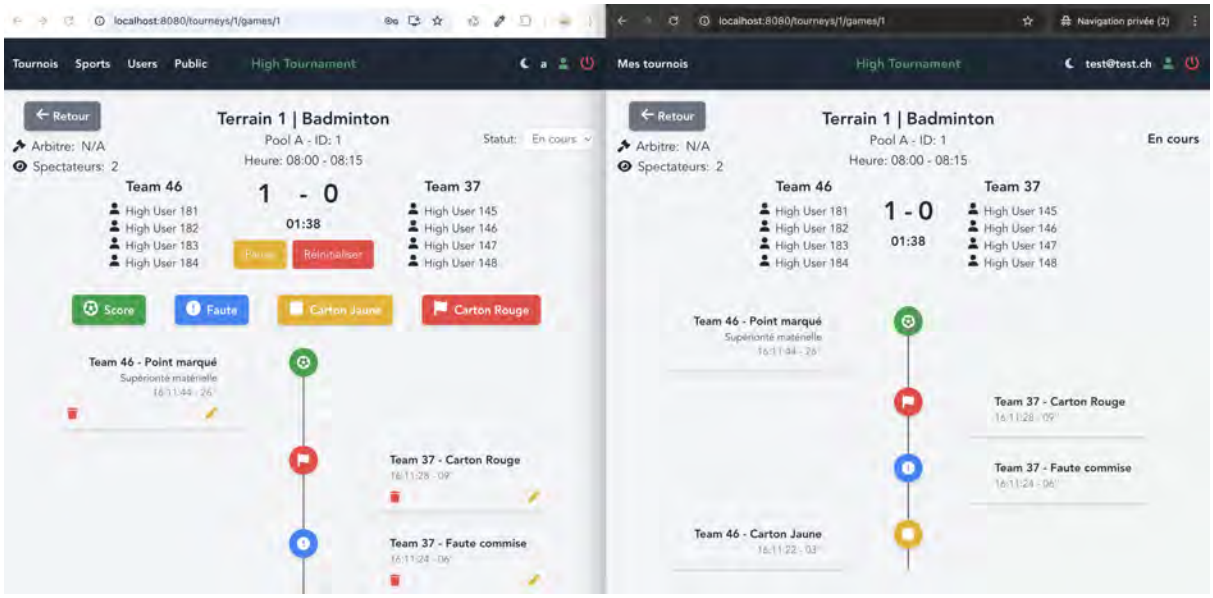


Fig. 4.9.: Comparaison des vues entre un rôle **Admin** et **Player**

La figure 4.9 illustre cette distinction. À gauche, un utilisateur possédant le rôle global `Admin` (ici, avec des fonctionnalités équivalentes à celles d'un rôle Assistant) dispose d'un accès à davantage d'options que celui qui a le rôle spécifique `Player` (à droite). Bien que les deux utilisateurs soient sur la même vue (même URL), les fonctionnalités affichées diffèrent en fonction de leurs rôles.
C'est ce que nous allons démontrer dans la suite de ce chapitre, en prenant le scénario de mise à jour d'un événement de match (un but) pour un `Assistant`. La route **gameEvents** illustre ce fonctionnement :

```
router.put('/:eventId',
  isAuthenticated, //middleware d'authentification
  authorizeTournamentAccess, // middleware verifiant acces au tournoi
  authorizeTourneyRoles([tourneyRoles.ASSISTANT]), //middleware pour assistant
  updateGameEvent // methode du controlleur pour mettre a jour un event du match
);
```

List. 4.21: Exemple de route pour mettre à jour un événement de match

Ici, `authorizeTourneyRoles([tourneyRoles.ASSISTANT])`[12] vérifie que l'utilisateur a, au minimum, le rôle `ASSISTANT` dans ce tournoi. Le code du middleware est présenté ci-dessous :

```
exports.authorizeTourneyRoles = (allowedRoles) => {
  return async (req, res, next) => {
    try {
      const userId = req.user.id;
      const tourneyId = parseInt(req.params.tourneyId, 10);

      // 1. Si l'utilisateur est Admin global, acces immediat
      if (req.user.roleId === roles.ADMIN) {
        return next();
      }

      // 2. Recuperer le role local de l'utilisateur dans ce tournoi
      const userTourney = await UsersTourneys.findOne({
        where: { userId, tourneyId },
      });
      if (!userTourney) {
        return res.status(403).json({ message: "Acces interdit au tournoi." });
      }

      // 3. Verifier si le role du tournoi fait partie des roles autorises
      const userTourneyRole = userTourney.tourneyRole;
      if (!allowedRoles.includes(userTourneyRole)) {
        return res.status(403).json({ message: "Role insuffisant pour cette
            action." });
      }

      // 4. Conserver le role local pour d'autres traitements et continuer
      req.user.tourneyRole = userTourneyRole;
      next();
    } catch (error) {
      console.error('Erreur role tournoi:', error);
      res.status(500).json({ message: "Erreur serveur" });
    }
  };
};
```

List. 4.22: Middleware `authorizeTourneyRoles` pour filtrer les accès

Résumé de la logique :

1. Si l'utilisateur est `Admin` global, il est automatiquement autorisé.
2. Sinon, on récupère le rôle local (p. ex. `Assistant`) via la table `UsersTourneys`.
3. On compare ce rôle local à la liste des rôles autorisés dans la route.
4. S'il correspond, la requête continue, sinon le serveur renvoie un code `403` "*Access Denied*".

[12] Code source-TourneyMiddleware: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/middlewares/authorizeTourneyRole.js`

Contrôle d'affichage côté Frontend

Pour éviter d'afficher des éléments non autorisés, le frontend (Vue.js) contrôle également le rôle de l'utilisateur. Par exemple, dans `TourneyGameDetails.vue`[13] :

```
// Affiche le champ de score editable pour un assistant (ou admin)
<div v-if="canEdit && match.status === 'in_progress'">
  <input
    v-model.number="scoreTeamA"
    type="number"
    class="w-16 text-center text-4xl font-bold bg-transparent border-none"
    @input="updateScores"
  />
</div>
```

List. 4.23: Condition d'affichage pour la modification du score

```
// Détermine si l'utilisateur actuel peut modifier ce match.
canEdit() {
  const isAdmin = this.$store.state.user?.roleId === 1;
  const isAssistant = this.$store.getters['userTourney/isAssistant'];
  const isTournamentActive = (this.tourney && this.tourney.status === 'active');

  if (isAdmin) return true;
  if (isAssistant && isTournamentActive) return true;
  return false;
},
```

List. 4.24: Vérification du droit d'édition dans Vue

Dans cet exemple, on vérifie à la fois le rôle global (`roleId === 1` pour `Admin`) et le rôle local (`isAssistant`). Seuls les utilisateurs possédant au moins l'un de ces statuts, et si le tournoi est `active`, peuvent mettre à jour les scores. De cette façon, l'**interface graphique** s'adapte aux privilèges de chaque utilisateur, évitant de proposer des actions qu'il ne pourrait pas exécuter côté serveur.

4.6.2. Gestion du Token d'Invitation

Le **token d'invitation** permet à un administrateur de partager un lien unique afin que des utilisateurs puissent rejoindre un tournoi.

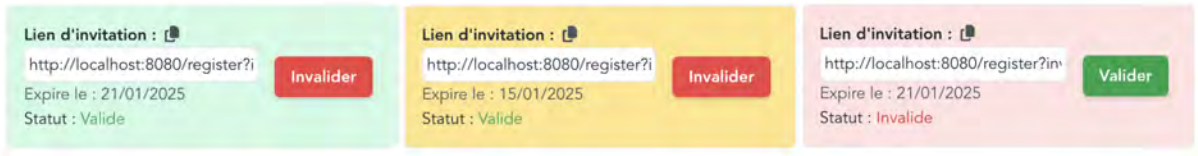


Fig. 4.10.: Gestion des tokens d'invitations

[13] Code source-Vue GameDetails: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/frontend/src/views/user/tourneys/TourneyGameDetails.vue`

Principe Général

1. **Génération du token par un administrateur** : l'admin crée un token en indiquant, par exemple, sa durée de validité. Le serveur enregistre ce token dans la table `InviteTokens` et le renvoie au frontend.
2. **Partage du lien** : l'admin communique l'URL d'invitation, qui inclut le token, aux futurs participants (exemple: `https://easy-tourney/register?inviteToken=XYZ`).
3. **Redirection et stockage du token** : lorsque l'utilisateur clique sur ce lien, le frontend *stocke* temporairement le token (dans `localStorage`) et lui propose de se connecter ou de créer un compte.
4. **Validation du token et inscription au tournoi** : une fois l'utilisateur authentifié, le store Vuex envoie le token au serveur pour qu'il l'associe au tournoi, créant un nouvel enregistrement dans `UsersTourneys` (souvent avec un rôle initial `Guest`).
5. **Gestion dynamique du token** : l'admin peut invalider, valider ou prolonger un token via le backend. Si le token est expiré ou rendu `isValid = false`, la route de jonction renvoie une erreur.

Création du Token d'Invitation (Backend)

Le contrôleur `inviteTokenController.js`[14] fournit une route pour *POST (:tourneyId/invite-token)*, qu'un admin peut appeler pour générer un token. La fonction `generateInviteToken` crée un `JWT` marqué avec `type: 'invite'` et l'enregistre dans la base de données :

```
exports.generateInviteToken = async (req, res) => {
  const { tourneyId } = req.params;
  const { expiresInDays } = req.body || 7; // 7 jours par defaut

  // 1. Verifier l'existence du tournoi
  const tourney = await Tourney.findByPk(tourneyId);
  if (!tourney) {
    return res.status(404).json({ message: "Tournoi non trouve." });
  }

  // 2. Creer et signer un token JWT, type 'invite'
  const token = authService.generateInviteToken(tourneyId);

  // 3. Determiner sa date d'expiration
  const expiresAt = new Date();
  expiresAt.setDate(expiresAt.getDate() + expiresInDays);

  // 4. Enregistrer en base de donnees
  const newInviteToken = await InviteToken.create({
    token,
    tourneyId,
    expiresAt,
    isValid: true,
  });
```

[14] Code source-Contrôleur token invitation: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/controllers/inviteTokenController.js`

```
25
26   res.status(200).json({
27     token: newInviteToken.token,
28     expiresAt: newInviteToken.expiresAt,
29   });
30 };
```

List. 4.25: Génération et enregistrement d'un token d'invitation

La table `InviteTokens` stocke le token, sa validité et sa date d'expiration.

Partage et Utilisation du Lien d'Invitation (Frontend)

Lorsque l'administrateur clique sur "*Générer un token d'invitation*" dans l'interface, l'API retourne le `token`. L'admin peut alors transmettre un **lien** aux participants, du type :

`https://easy-tourney/register?inviteToken=${token}`

Exemple d'appel côté Vue, où l'admin obtient et affiche le lien :

```
1  async handleGenerateInviteToken() {
2    try {
3      const response = await apiService.post(
4        `/tourneys/${this.tourneyId}/invite-token`,
5        { expiresInDays: this.inviteTokenForm.expiresInDays }
6      );
7      // Construire l'URL (BASE_URL est defini dans la config frontend)
8      this.inviteLink = `${BASE_URL}/register?inviteToken=${response.data.token}`;
9      toast.success('Lien d'invitation genere avec succes !');
10     this.fetchInviteTokens();
11   } catch (error) {
12     toast.error('Erreur lors de la generation du token.');
13   }
14 }
```

List. 4.26: Génération du lien d'invitation dans la vue Admin

Inscription et Association au Tournoi

1. Détection du token : Côté client, lorsque l'utilisateur atterrit sur la page, le **frontend** extrait le paramètre `inviteToken` de l'URL et le sauvegarde dans le `store Vuex` (mutation `SET_INVITE_TOKEN`). Puis l'utilisateur est invité à créer un compte ou se connecter.

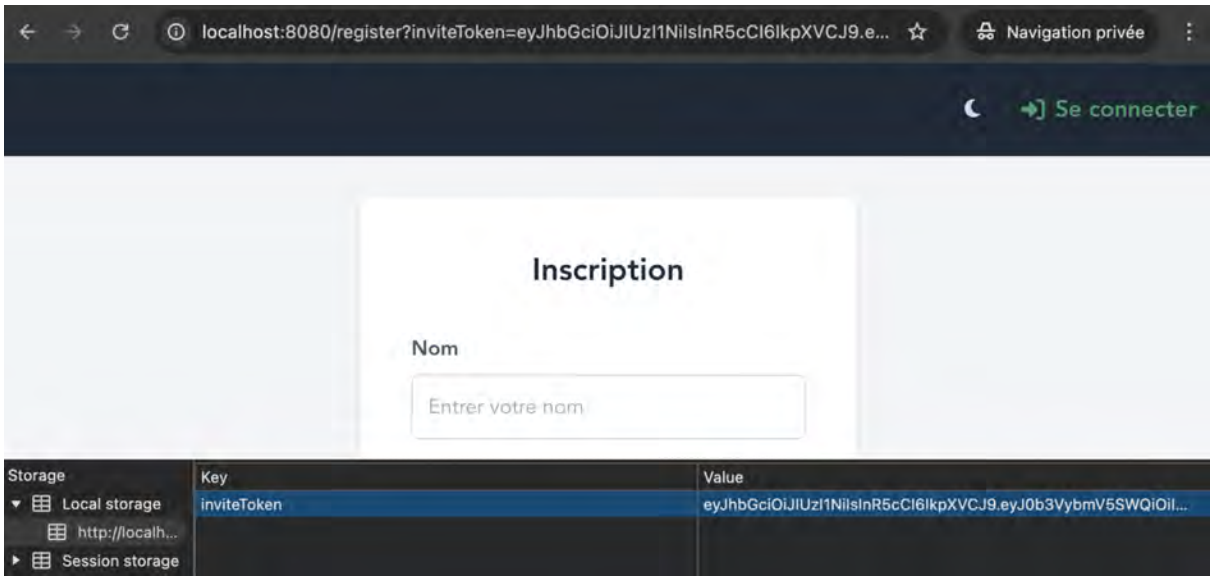


Fig. 4.11.: Token d'invitation dans le 'Local storage'

La figure 4.11 montre que le token `inviteToken` est bien enregistré dans le local storage, suite à avoir cliqué sur un lien d'invitation comme le montre l'URL.

2. Passage du token au backend après login : Au moment du **login** ou de l'enregistrement (`RegisterPage.vue`), le code vérifie si un `inviteToken` est présent dans le store, et appelle alors `POST /tourneys/join` pour inscrire l'utilisateur :

```
const inviteToken = this.$store.state.inviteToken;
if (inviteToken) {
  try {
    await apiService.post(‘/tourneys/join‘, { token: inviteToken });
    // L’utilisateur est désormais ajouté àla table UsersTourneys
  } catch (err) {
    toast.error(’Impossible de rejoindre le tournoi.’);
  } finally {
    this.$store.dispatch(’clearInviteToken’);
  }
}
```

List. 4.27: Extrait de code lors du login

3. Vérification du token côté serveur : Le contrôleur `joinTourneyWithToken` (*tourneyController.js*) *décode* le token pour retrouver `tourneyId` et `type`. Il vérifie également que le token n'est ni expiré ni invalidé dans la table `InviteToken`, et que le tournoi accepte encore les inscriptions. Si tout est correct :

- Un enregistrement est créé dans `UsersTourneys`, associant l'utilisateur (`userId`) et le tournoi (`tourneyId`) avec le rôle `guest` par défaut.
- L'utilisateur est officiellement inscrit et pourra rejoindre une équipe ultérieurement.

Gestion du Cycle de Vie du Token d'Invitation

Pour administrer les tokens, les routes suivantes existent :

- `PATCH /tourneys/:tourneyId/invite-token/:tokenId/invalidate` : *Invalider* un token.
- `PATCH /tourneys/:tourneyId/invite-token/:tokenId/validate` : *Rendre valide* un token précédemment invalide.
- `PUT /tourneys/:tourneyId/invite-token/invalidate-all` : Invalider d'un coup tous les tokens.

Ainsi, l'**administrateur** peut maintenir la liste des tokens à jour, désactivant les anciens liens pour empêcher de nouvelles inscriptions.

Assignation à une Équipe et Rôle Final

L'utilisateur inscrit, il apparaît en tant que `guest` dans `UsersTourneys`. L'administrateur ou l'utilisateur lui-même peut alors rejoindre une `Team`, déclenchant la mise à jour de `tourneyRole` en fonction du type d'équipe. Par exemple :

```
1 userTourney.teamId = teamId;
2 userTourney.tourneyRole = getRoleByTeamType(team.type); //'player','assistant'
3 await userTourney.save();
```

List. 4.28: Extrait de code pour assigner un utilisateur à une équipe

De cette façon, **le cycle d'invitation** est complet : *du lien envoyé par l'admin* jusqu'à l'inscription de l'utilisateur et l'obtention d'un rôle défini par l'équipe qu'il a rejointe. L'expérience reste fluide tout en respectant les contraintes de validité des tokens (dates d'expiration, statuts d'inscription, etc.).

4.6.3. Algorithme d'Assignation Automatique de Joueurs Sans Groupe

Le chapitre précédent expliquait le processus d'invitation des édudiants à un groupe. Nous allons maintenant expliquer la fonctionnalité côté **Admin**, permettant d'assigner automatiquement les joueurs sans groupe aux équipes. Cette section décrit en détail le **fonctionnement** de cet algorithme, son **implémentation** sur le frontend avec Vue.js, ainsi que la gestion des **transactions** côté backend pour assurer l'intégrité des données.

Rappel de l'Interface Utilisateur

L'interface dédiée à l'assignation des joueurs présente une table listant tous les utilisateurs sans groupe (`tourneyRole`=Guest)(Fig. 4.12). L'administrateur dispose de deux options principales:

- **Assignation Manuelle** : Permet d'assigner chaque joueur individuellement à une équipe spécifique.
- **Assignation Automatique** : Lance un algorithme qui répartit les joueurs de manière équilibrée entre les équipes disponibles.

Un aperçu des composants clés de cette interface est illustré ci-dessous.

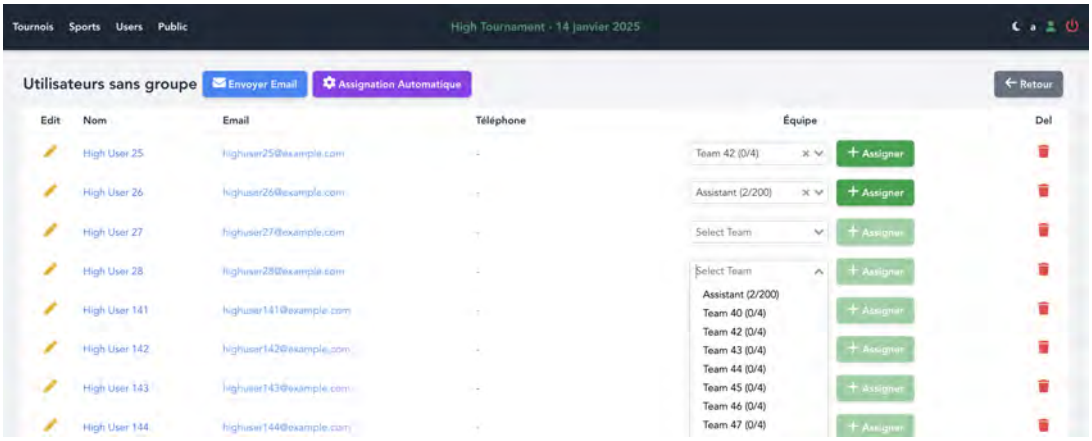


Fig. 4.12.: Interface d'assignation automatique des joueurs

Algorithme d'Assignation Automatique

L'algorithme agit en deux temps afin d'améliorer l'expérience utilisateur lorsqu'on clique sur *Assignation Automatique*.

1. **Prévisualisation des assignations :** les dropdowns sous la colonne `Equipe` affichent dans quelle équipe l'utilisateur sera assigné. Il est possible d'y effectuer des changements à ce moment ou d'annuler l'assignation manuellement sur chaque utilisateur
2. **Validation :** Le bouton change en *Valider*, et la requête va être envoyé au backend pour assigner les utilisateurs aux équipes correspondantes.

L'algorithme d'assignation a toute sa logique côté frontend. Il est conçu pour répartir les joueurs sans groupe de manière équilibrée entre les équipes disponibles, en respectant les contraintes définies dans *teamSetup* (nombre minimum et maximum de joueurs par équipe). Voici les étapes détaillées de cet algorithme, avec le **code légèrement modifié en pseudo-code** pour un affichage plus compréhensible.

1. **Initialisation :**
   - Stocker l'état initial des affectations pour permettre une annulation si nécessaire.
   - Séparer les équipes en deux catégories : *player* et *assistant*.
2. **Remplissage des Équipes Partiellement Remplies :**
   - Identifier les équipes qui ont déjà des joueurs mais qui n'ont pas atteint le nombre minimum requis.
   - Assigner des joueurs non assignés à ces équipes jusqu'à atteindre le seuil minimal.
3. **Remplissage des Équipes Vides :**
   - Calculer combien d'équipes vides peuvent être entièrement remplies avec les joueurs restants.
   - Assigner les joueurs à ces équipes en respectant le nombre minimum par équipe.
4. **Distribution des Joueurs Restants :**

- Distribuer les joueurs restants aux équipes déjà partiellement remplies, sans dépasser le nombre maximum par équipe.

5. **Assignation des Joueurs Restants à l'Équipe 'Assistant' :**
   - Si des joueurs restent après les étapes précédentes, les assigner à l'équipe 'assistant'.

Implémentation de l'Algorithme

L'algorithme est entièrement implémenté sur le frontend, dans la méthode `autoFillGroups` de la vue `TourneyUnassignedUsers.vue`[15]. Voici un extrait de cette méthode, légèrement modifié en pseudocode, pour illustrer son fonctionnement :

```
autoFillGroups() {
    // 1. Stocker l'état initial
    initialSelectedTeamIds = copy(selectedTeamIds)
    unassignedUsers = copy(unassignedUsers)

    // 2. Séparer les équipes
    assistantTeam = null
    teams = []
    for each team in teams:
        if team.type == 'assistant':
            assistantTeam = team
        else:
            teams.append(team)

    // 3. Fonction d'assignation
    function assignUsersToTeam(team, number) {
        usersToAssign = unassignedUsers.splice(0, number)
        team.assignedUsers.push(...usersToAssign)
        for each user in usersToAssign:
            selectedTeamIds[user.id] = team.id
    }

    // a) Remplir les équipes partiellement remplies
    for each team in teams where team.assignedUsers.length < minPlayerPerTeam:
        needed = minPlayerPerTeam - team.assignedUsers.length
        assignUsersToTeam(team, min(needed, unassignedUsers.length))

    // b) Remplir les équipes vides
    emptyTeams = teams where team.assignedUsers.length == 0
    maxFillable = floor(unassignedUsers.length / minPlayerPerTeam)
    teamsToFill = emptyTeams.slice(0, maxFillable)
    for each team in teamsToFill:
        assignUsersToTeam(team, minPlayerPerTeam)

    // c) Distribuer les joueurs restants
    while unassignedUsers.length > 0:
```

[15] Code source-Assignation Automatique: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/frontend/src/views/admin/tourneys/teams/TourneyUnassignedUsers.vue`

```
        for each team in teams:
            if team.assignedUsers.length < playerPerTeam and unassignedUsers.
                length > 0:
                assignUsersToTeam(team, 1)

    // d) Assignation a l'équipe 'assistant'
    if unassignedUsers.length > 0 and assistantTeam:
        assignUsersToTeam(assistantTeam, unassignedUsers.length)

    isAutoFilled = true
}
```

List. 4.29: Pseudocode de l'algorithme d'assignation automatique

Résumé de la Méthode `autoFillGroups` : Cette méthode commence par sauvegarder l'état initial des assignations existantes et crée des copies des utilisateurs non assignés et des équipes disponibles. Ensuite, elle sépare les équipes en deux catégories : les équipes de type 'assistant' et les équipes de type 'player'. Une fonction interne, `assignUsersToTeam`, est définie pour faciliter l'assignation de joueurs à une équipe spécifique. L'algorithme suit ensuite plusieurs étapes logiques :

1. **Remplir les Équipes Partiellement Remplies :** Identifie les équipes qui n'ont pas encore atteint le nombre minimum de joueurs requis et leur assigne les joueurs nécessaires.
2. **Remplir les Équipes Vides :** Calcule combien d'équipes vides peuvent être entièrement remplies avec les joueurs restants et procède à ces assignations.
3. **Distribuer les Joueurs Restants :** Répartit les joueurs restants entre les équipes déjà partiellement remplies sans dépasser le nombre maximum de joueurs par équipe.
4. **Assignation à l'Équipe 'Assistant' :** Si des joueurs restent après les étapes précédentes, ils sont assignés à une équipe spéciale de type 'assistant'.

Cette approche garantit une répartition équilibrée des joueurs tout en respectant les contraintes de capacité des équipes.

Prévisualisation et Modification des Assignations

Une des fonctionnalités clés de cet algorithme est la possibilité pour l'administrateur de prévisualiser les assignations automatiques avant de les valider. Lorsqu'il clique sur le bouton d'assignation automatique, les affectations proposées sont affichées avant d'être envoyer au server, permettant à l'administrateur de :

- **Visualiser** où chaque joueur sera assigné.
- **Modifier** les assignations en déplaçant des joueurs vers d'autres équipes si nécessaire.
- **Annuler** l'assignation automatique avant validation, revenant ainsi à l'état initial.

Cette flexibilité assure que les assignations répondent aux besoins spécifiques du tournoi et permettent d'ajuster manuellement les répartitions automatiques si besoin.

Validation des Assignations

Une fois satisfait des assignations proposées, l'administrateur peut valider les assignations. Cette action déclenche la méthode `handleValidateAssignments`, qui envoie les données au backend pour une persistance sécurisée.

```
async handleValidateAssignments() {
    // exclue les entrées sans teamId, puis transforme chaque paire (userId,
       teamId) en objets contenant userId et teamId.
    const assignments = Object.entries(this.selectedTeamIds)
        .filter(([, teamId]) => teamId != null && teamId != undefined)
        .map(([userId, teamId]) => ({
            userId: Number(userId),
            teamId,
        }))

    const tourneyId = this.$route.params.tourneyId;
    try {
        // Envoyer les affectations au backend
        await apiService.post(`/tourneys/${tourneyId}/teams/auto-fill`, {
            assignments,
        });

        // Rafraîchir les données après la validation
        await this.fetchData();

        // Message de validation ou error (catch)...
}
```

List. 4.30: Méthode de validation des assignations dans Vue.js

Résumé de la Méthode `handleValidateAssignments` : Cette méthode commence par structurer les assignations en objets contenant les identifiants des utilisateurs et des équipes (`userId`, `teamId`). Elle envoie ensuite ces données au backend via une requête POST vers l'endpoint `/teams/auto-fill`. Après une validation réussie, les données du composant sont rafraîchies pour refléter les nouvelles assignations, et une notification de succès est affichée à l'utilisateur. En cas d'erreur, une notification d'échec est affichée et les erreurs sont consignées dans la console pour le débogage.

Gestion des Transactions Côté Backend

Pour garantir l'intégrité des données lors de l'assignation de plusieurs utilisateurs à des équipes, le backend utilise des transactions. Une transaction assure que toutes les opérations d'assignation sont traitées de manière atomique : soit toutes les assignations sont effectuées avec succès, soit aucune n'est appliquée en cas d'erreur. Cela empêche des états incohérents dans la base de données.

Voici un extrait du contrôleur *teamController*[16] gérant cette fonctionnalité :

[16] Code source-AutoFillTeams: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/controllers/teamController.js`

```
exports.autoFillTeams = async (req, res) => {
    const { tourneyId } = req.params;
    const { assignments } = req.body;

    // Démarrer la transaction
    const transaction = await sequelize.transaction();

    // Assigner chaque utilisateur àson équipe
    try {
        for (const assignment of assignments) {
            const { userId, teamId } = assignment;

            // ... code vérifiant si les données existent dans la DB ...

            // Assigner l'utilisateur àsa team et update son role
            userTourney.teamId = teamId;
            userTourney.tourneyRole = getRoleByTeamType(team.type);
            await userTourney.save({ transaction });
        }

        // Valider la transaction
        await transaction.commit();
        res.status(200).json({ message: "Affectations enregistrees avec succes."
            });
    } catch (error) {

        // En cas d'erreur, revert la transaction
        await transaction.rollback();
        console.error('Erreur lors des affectations en lot:', error);
        res.status(500).json({
            message: 'Erreur lors des affectations en lot.',
            error: error.message,
        });
    }
};
```

List. 4.31: Contrôleur pour l'assignation automatique des équipes

Résumé du Contrôleur `autoFillTeams` **:** Ce contrôleur reçoit les assignations automatiques envoyées depuis le frontend. Il utilise une transaction pour s'assurer que toutes les assignations sont effectuées de manière cohérente. Pour chaque assignation, il vérifie l'existence de l'équipe et de la participation de l'utilisateur au tournoi. Si toutes les vérifications sont réussies, il assigne l'utilisateur à l'équipe correspondante et met à jour son rôle dans le tournoi. Si une erreur survient à n'importe quelle étape, la transaction est annulée, garantissant ainsi que la base de données reste dans un état cohérent.

Note : Une **transaction** [39] est une séquence d'opérations qui sont traitées comme une unité unique. Si une des opérations échoue, toutes les modifications effectuées dans le cadre de la transaction sont annulées, assurant ainsi la cohérence des données.

Défis et Solutions

L'implémentation de cet algorithme a présenté plusieurs défis :

- **Prévisualisation des Assignations :** Ajouter une prévisualisation des assignations automatiques n'était pas trivial. Il a fallu concevoir une interface permettant à l'administrateur de voir les affectations proposées et de les modifier avant validation de manière réactive.
- **Gestion des Assignations Dynamiques :** Permettre à l'administrateur de changer les assignations après la pré-assignation nécessite une gestion réactive des données et une mise à jour efficace de l'interface utilisateur.
- **Intégrité des Données :** Assurer que les assignations soient cohérentes et ne créent pas de conflits dans la base de données a nécessité l'utilisation de transactions côté backend.

Malgré ces défis, l'algorithme offre une solution robuste et flexible pour gérer l'assignation des joueurs, améliorant ainsi l'efficacité de l'organisation des tournois.

Résumé

L'algorithme d'assignation automatique développé pour ce projet permet une répartition équilibrée des joueurs sans groupe entre les équipes disponibles. Implémenté entièrement sur le frontend avec Vue.js, il offre une prévisualisation et une flexibilité dans les assignations avant validation. Côté backend, l'utilisation de transactions garantit l'intégrité des données lors de l'enregistrement des affectations. Cette approche unifiée assure une expérience utilisateur fluide et fiable, facilitant la gestion des tournois.

4.6.4. FullCalendar et Drag & Drop

La gestion des plannings, souvent perçue comme complexe, devient intuitive grâce au composant **FullCalendar** [26], offrant une solution puissante et flexible (Fig. 4.13).

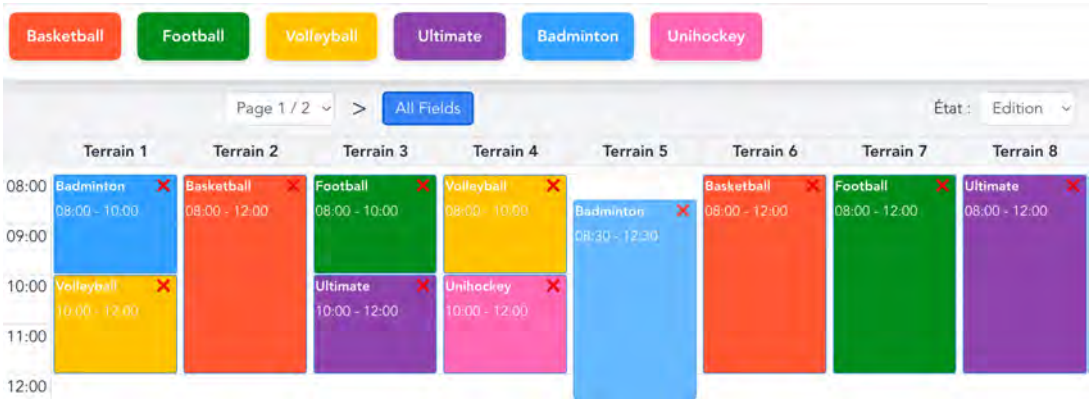


Fig. 4.13.: Calendrier pour assigner des sports aux terrains

FullCalendar est utilisé sur toutes les pages avec une gestion d'horaire visuelle. Par exemple à la page pour **assigner des sports à des terrains** via le *glisser-déposer*. Chaque *"ressource"* correspond à un **terrain**, et les *événements* représentent des créneaux

horaires de pratique sportive. Grâce au **plugin** `resourceTimeGrid`, la vue calendaire est segmentée par ressource, permettant de :

- **Déplacer** un événement pour changer de terrain ou de plage horaire (`eventDrop`).
- **Redimensionner** un événement pour ajuster sa durée (`eventResize`).
- **Glisser un sport** depuis une liste externe pour créer un événement (`eventReceive`).

Note : Le plugin `resourceTimeGridDay` [36] est disponible sous licence non commerciale[17].

```
calendarOptions() {
  // ... nombreuses options
  plugins: [timeGridPlugin, interactionPlugin, resourceTimeGridPlugin],
  schedulerLicenseKey: 'CC-Attribution-NonCommercial-NoDerivatives', // Usage non-commercial
  initialView: 'resourceTimeGridDay',
}
```

List. 4.32: Licence non commerciale FullCalendar

Ce système offre une base solide et extensible pour gérer des plannings complexes. Par exemple, des pages avancées comme la gestion des plannings de `games` s'appuient sur ce composant en intégrant des fonctionnalités supplémentaires telles que des arrière-plans interactifs pour créer des matchs. Pour simplifier la compréhension, une page moins complexe peut être présentée pour expliquer les principes de base du composant.

Structure Vue.js : data, computed, mounted, watch

Le fichier `TourneySportsFields.vue`[18] illustre la **structure standard** d'un composant Vue 3 :

- `data()` : propriétés réactives locales (*fields*, *sports*, *currentPage*).
- `computed` : fonctions calculées dépendant de l'état (*calendarOptions*, *paginatedFields*), avec réactivité automatique.
- `mounted()` : *hook* invoqué après le montage, idéal pour appeler l'API initiale (*fetchTourneySportsFields*) et initialiser les fonctionnalités (*initializeExternalEvents*).
- `watch` : surveille des variables (*currentPage*) pour déclencher une logique spécifique (ici, *refetchResources()* dans FullCalendar).

`CalendarOptions` est une propriété *computed* recalculée à chaque modification de `fields` ou `tourney.dateTourney`, rafraîchissant ainsi la configuration du calendrier.

Le calendrier est divisé en deux sections :

1. **Liste des sports draggable** : représente tous les sports de la base de données, avec défilement horizontal si nécessaire.
2. **Calendrier** : reçoit et gère les assignations de sports aux terrains sur des créneaux horaires.

[17] `https://fullcalendar.io/license`

[18] Code source-Vue Sports-Terrains: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/frontend/src/views/admin/tourneys/TourneySportsFields.vue`

Initialisation du Drag & Drop

Une des principales difficultés a été de rendre les sports "draggables" sur smartphone et desktop, et de les faire interagir avec le calendrier. Après plusieurs essais infructueux avec diverses librairies, la solution a été d'utiliser `Draggable` [30].

```
initializeExternalEvents() {
  // Détruire l'instance précédente si elle existe
  if (this.externalDraggableInstance) {
    this.externalDraggableInstance.destroy();
  }
  const containerEl = document.getElementById('external-events');
  // Instancie un nouveau "Draggable" sur la zone des sports
  this.externalDraggableInstance = new Draggable(containerEl, {
    itemSelector: '.external-event', // Classes drag
    eventData(eventEl) {
      // Infos pour FullCalendar : titre, couleur, etc.
      return {
        title: eventEl.innerText.trim(),
        backgroundColor: eventEl.style.backgroundColor,
        duration: '04:00',
        extendedProps: {
          sportId: eventEl.getAttribute('data-id'),
        },
      };
    },
  });
}
```

List. 4.33: Extrait d'initialisation du drag & drop

Dans ce code :

1. **containerEl** pointe vers la `div` listant les sports (`#external-events`).
2. `itemSelector` restreint le drag aux éléments de la classe `.external-event`.
3. **eventData** génère l'objet événement compris par FullCalendar (incluant `sportId`, `backgroundColor`, etc.).
4. La manipulation directe du DOM a parfois été nécessaire (ex. *document.createElement*) pour ajouter un bouton de suppression ou gérer `touchstart` sur mobile.

Cette solution, bien qu'efficace, s'est avérée **complexe** à mettre en place, notamment pour gérer les événements `touch` (*smartphone/tablette*) ou `double-click` (suppression). De plus, l'utilisation de deux systèmes de Drag distincts a ajouté à la complexité :

- `Draggable` gère la liste des sports en externe.
- `interaction` de FullCalendar permet de modifier les entités au sein du calendrier.

Configuration du Calendrier et Scénario d'Interaction

Grâce à `resourceTimeGridPlugin`, le calendrier se découpe en plusieurs lignes (ressources), chacune représentant un *terrain*. Notre `calendarOptions` (voir list 4.34) définit :

- `resources` : les **terrains paginés** pour éviter la surcharge de l'écran.

- `events` : la liste `sportsFields` (sports assignés).
- `eventReceive`, `eventDrop`, `eventResize` : **callbacks** pour synchroniser le drag & drop avec l'API backend.

```
calendarOptions() {
  return {
    plugins: [timeGridPlugin, interactionPlugin, resourceTimeGridPlugin],
    initialView: 'resourceTimeGridDay', // Calendrier par section sur un jour
    editable: this.isEditable, // Autorise le drag
    droppable: true, // Autorise le drop externe
    resources: this.paginatedFields.map((field) => ({
      id: field.id.toString(),
      title: field.name,
    })),
    // Events réactifs aux interactions utilisateurs
    events: this.buildEvents(), // Sports déjà assignés
    eventReceive: this.handleEventReceive,
    eventDrop: this.handleEventDrop,
    eventResize: this.handleEventResize,
    eventContent: this.renderEventContent,
  };
}
```

List. 4.34: Extrait de `calendarOptions` qui configure FullCalendar

Scénario détaillé d'assignation d'un sport à un terrain

Étape 1 - Glisser un sport dans le calendrier : L'utilisateur sélectionne un sport dans la liste (`#external-events`) et le glisse sur une plage horaire du calendrier, déclenchant ainsi l'événement `eventReceive` de FullCalendar.

```
async handleEventReceive(info) {
    const event = info.event;
    try {
        // 1. Récupère sport et terrain via l'event
        const sportId = event.extendedProps.sportId;
        const newFieldId = event.getResources()[0]?.id;

        // 2. Si erreur, annule l'opération
        if (!newFieldId || !sportId) {
          console.error('ID invalide');
          info.revert(); // Annule
          return;
        }

        // 3. Prépare les données pour le backend
        const data = {
          fieldId: newFieldId,
          sportId: sportId,
          startTime: this.formatTime(event.start),
          endTime: this.formatTime(event.end),
```

```
        };

        // 4. Requête POST création sport-terrain
        const response = await apiService.post(
          `/tourneys/${this.tourneyId}/sports-fields`,
          data
        );
         event.setProp('id', response.data.id); // Associe ID backend->event
    }
    catch (error) {
        console.error('Erreur lors de la reception de evenement :', error);
        info.revert();
    }
}
```

List. 4.35: Gestion de `eventReceive` côté Vue.js

Étape 2 - Création de l'association sport-terrain côté backend : Le backend reçoit une requête `POST /sports-fields` pour créer l'association.

```
router.post('/', isAuthenticated, isAdmin, verifySportAssignmentStatus,
    createSportsFields); // Associer sport àterrain
```

List. 4.36: Route 'sportFields' pour créer une association sport-terrain

La méthode `createSportsFields` vérifie les données, confirme l'existence du sport et du terrain, puis crée l'association.

```
// Vérification si les données existent
const field = await Field.findByPk(fieldId);
const sport = await Sport.findByPk(sportId);

// Création via ORM dans la DB
const sportsFields = await SportsFields.create({ fieldId, sportId, startTime,
    endTime });
```

List. 4.37: Creation d'une association sport-terrain

Étape 3 - Déplacement ou redimensionnement d'un événement : Ces actions appellent des fonctions similaires (mais avec l'`eventResize`) qui effectuent des appels API au backend pour synchroniser les changements.

Étape 4 - Suppression d'un événement : L'utilisateur peut supprimer un événement en cliquant sur une icône "×". Cela est géré via le DOM dans `renderEventContent`, où un bouton de suppression est ajouté manuellement :

```
renderEventContent(arg) {
    const deleteIcon = document.createElement('span');
    deleteIcon.innerHTML = 'X'; // icone croix
```

```
4    deleteIcon.addEventListener('dblclick', () => this.deleteEvent(arg.event));
```

List. 4.38: Ajout d'un bouton de suppression pour chaque événement

L'action déclenche ensuite une requête API DELETE pour supprimer l'assignation.

Grâce à ces interactions, le frontend et le backend collaborent pour maintenir une synchronisation parfaite entre l'interface utilisateur et les données du serveur.

Difficultés Rencontrées

- **Choix de la librairie *draggable*** : Plusieurs essais ont échoué avant de découvrir `Draggable`, garantissant la compatibilité *drag-to-calendar*.
- **Manipulation directe du DOM** : `renderEventContent` injecte manuellement des `span` et gère les événements `touchstart`, `dblclick` pour la suppression, augmentant la complexité tout en personnalisant l'UI.
- **Responsivité et pagination** : Assurer un affichage correct et des interactions fonctionnelles sur smartphone malgré de nombreuses données.
- **Deux librairies de Drag distinctes** : Complexité accrue pour intégrer des éléments externes (sports) au calendrier, nécessitant beaucoup de code personnalisé et de manipulation du DOM.

Résumé

Le composant `TourneySportsFields.vue` démontre comment **Vue.js** et **FullCalendar** s'intègrent pour offrir :

- Un **drag-and-drop** fluide (grâce à `@fullcalendar/interaction`).
- Un **calendrier multi-ressources** (terrains) avec *drag*, *resize*, *drop*.
- Une **mise à jour réactive** : chaque action notifie le backend (*SportsFieldsController*) et modifie immédiatement la base SQL.

D'autres pages du projet (ex. planification de matchs) utilisent des logiques similaires, s'appuyant sur FullCalendar avec des **options élargies** (gestion des pools, événements en arrière-plan, etc.). Cette unification assure une expérience cohérente pour l'utilisateur et facilite l'évolution du code.

4.6.5. WebSockets & Gestion Temps Réel

Dans l'application *Easy Tourney*, les **WebSockets** [46] permettent une gestion en temps réel des matchs. Cette technologie maintient une connexion bidirectionnelle persistante entre le client et le serveur, offrant ainsi une communication immédiate et efficace. Contrairement au modèle de requêtes HTTP, les WebSockets permettent au serveur d'envoyer des données de manière proactive dès qu'un événement survient, éliminant ainsi la nécessité de recharger la page pour synchroniser les informations.

Fonctionnement dans Easy Tourney

Dans *Easy Tourney*, les WebSockets assurent la mise à jour en temps réel des scores, du chronomètre, et des événements de match. Chaque client (joueur, spectateur ou arbitre) établit une connexion sécurisée avec le serveur via un jeton d'authentification `JWT`. Une fois connecté, le client peut rejoindre une "salle" spécifique à un match ou à un tournoi, où il recevra instantanément toutes les mises à jour pertinentes.

La sécurité est assurée grâce à un middleware d'authentification côté serveur[19], qui vérifie la validité des jetons avant d'autoriser la connexion. Les reconnections automatiques et les notifications en cas d'erreurs de connexion sont également prises en charge pour garantir une expérience utilisateur fluide. Cette architecture garantit une synchronisation efficace et une interactivité renforcée dans l'application.

Scénario de Communication

Considérons un scénario où un arbitre et deux spectateurs sont connectés à un match spécifique (par exemple, `game_4`). La figure 4.14 illustre comment la communication se déroule grâce aux WebSockets[20] :

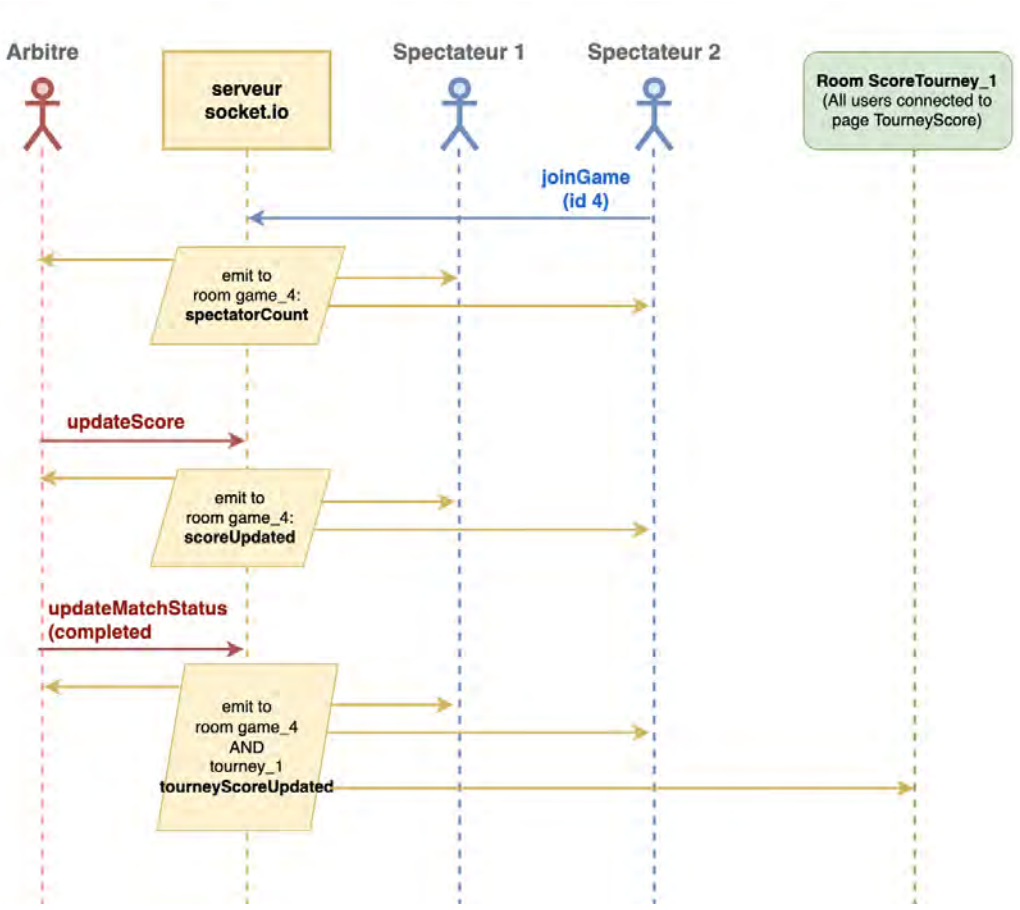


Fig. 4.14.: Flux de Communication en Temps Réel dans *Easy Tourney*

[19] Code source-authenticateSocket: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/socketHandlers/authenticateSocket.js`

[20] Code source-gameSocket server: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/socketHandlers/gameSockets.js`

Étape 1 : Connexion des Clients

1. **Rejoindre une Salle :**
   - Le spectateur2 rejoint une salle où un match est en cours. Son client Vue va d'abord récupérer les informations en cours du match puis envoyer un événement `joinGame` pour avertir qu'il a rejoint la salle `game_4` au server.
   - Le **Serveur** ajoute le client à la salle correspondante et émet un événement `spectatorCount` à tous les membres de la salle pour mettre à jour le nombre de spectateurs.

Étape 2 : Mise à Jour du Score

1. **Mise à Jour par l'Arbitre :**
   - L'Arbitre émet un événement `updateScore` avec les nouvelles valeurs de score.
   - Le **Serveur** traite cette mise à jour et diffuse un événement `scoreUpdated` à tous les membres de la salle `game_4`, incluant l'Arbitre et les Spectateurs.

Étape 3 : Terminer le Match

1. **Terminer par l'Arbitre :**
   - L'Arbitre émet un événement `updateMatchStatus` pour marquer le match comme terminé.
   - Le **Serveur** met à jour le statut du match et diffuse un événement `matchStatusUpdated` à tous les membres de la salle `game_4`. De plus, s'il s'agit de la fin du tournoi, un événement `tourneyScoresUpdated` est émis à la salle de scores (page Score) pour mettre à jour les scores.

Explication du Scénario avec Extraits de Code

1. Rejoindre une Salle

Côté Serveur (`server/socketHandlers/gameSockets.js`) :

```
// Un user a rejoint la room et envoi du nombre de spectateurs aux autres users
socket.on('joinGame', async (gameId) => {
    socket.join(`game_${gameId}`);
    console.log(`Socket ${socket.id} a rejoint la salle game_${gameId}`);

    const room = io.sockets.adapter.rooms.get(`game_${gameId}`);
    const spectatorCount = room ? room.size : 0; // Récupérer le nombre de spec.

    io.to(`game_${gameId}`).emit('spectatorCount', { count: spectatorCount });
});
```

List. 4.39: Gestion de l'événement 'joinGame'

Côté Frontend (`gameDetails.vue`) :

```
methods: {
    setupSocket() {
        const socket = getSocket();
```

```
        if (socket) {
            socket.emit('joinGame', this.match.id);

            socket.on('spectatorCount', (data) => {
                this.spectatorCount = data.count; // Update réactivemetn
                    spectateurs
            });
        }
    }
}
```

List. 4.40: Rejoindre une salle via Socket.IO

2. Mise à Jour du Score

Côté Serveur (server/socketHandlers/gameSockets.js) :

```
// Arbitre met a jur le score
socket.on('updateScore', async (data) => {
    const game = await Game.findByPk(gameId);
    if (game) {
        game.scoreTeamA = scoreTeamA;
        game.scoreTeamB = scoreTeamB;
        await game.save(); // Score sauvegarder via sequelize

        // Nouveau score envoyé aux users de la room
        io.to(`game_${gameId}`).emit('scoreUpdated', { scoreTeamA, scoreTeamB });

    }
});
```

List. 4.41: Gestion de l'événement 'updateScore'

Côté Frontend (gameDetails.vue) :

```
// couter les mises àjour du score
socket.on('scoreUpdated', (data) => {
    this.scoreTeamA = data.scoreTeamA;
    this.scoreTeamB = data.scoreTeamB;
}
```

List. 4.42: Écoute des mises à jour de score

3. Terminer le Match

Côté Serveur (server/socketHandlers/gameSockets.js) :

```
socket.on('updateMatchStatus', async (data) => {
    try {
        // Si le match est terminé, store le temps de fin dans la DB.
        game.status = status;
        if (status === 'completed') {
            game.realEndTime = new Date();
```

```
        }
        await game.save();

        // Envoyer aux users de la room game l'info que la game est terminée.
        io.to(`game_${matchId}`).emit('matchStatusUpdated', {
            matchId,
            status,
            isPaused: game.isPaused,
            pausedAt: game.pausedAt ? game.pausedAt.toISOString() : null,
            totalPausedTime: game.totalPausedTime
        });

        // Envoyer aux users de la room 'score' le nouveau score.
        if (status === 'completed') {
            io.to(`tourney_${tourneyId}`).emit('tourneyScoresUpdated');
        }
    } catch (error) {
        console.error('Erreur lors de la mise àjour du statut du match :', error)
            ;
        socket.emit('error', 'Erreur lors de la mise àjour du statut du match.');

    }
});
```

List. 4.43: Gestion de l'événement 'updateMatchStatus'

Côté Frontend (gameDetails.vue) :

```
// couter les mises àjour du statut du match
socket.on('matchStatusUpdated', (data) => {
    if (data.matchId === this.match.id) {
        this.match.status = data.status;
        // Mettre àjour l'interface utilisateur en conséquence
    }
});
```

List. 4.44: Écoute des mises à jour du statut du match

Difficultés Rencontrées

Synchronisation du Temps en Temps Réel : L'un des principaux défis a été de garantir la synchronisation précise du chronomètre entre le serveur et les clients. Des désynchronisations de quelques millisecondes peuvent entraîner des comportements innatendus, comme des retours en arrière du temps écoulé si l'on appuyait sans cesse entre le bouton 'pause' et 'reprendre'. Pour éviter ce problème de manière simple, l'implémentation d'un délai d'utilisation sur les boutons de commande est à privilégier.

Gestion de Multiples Événements : Avec la multiplicité des événements possibles (mise à jour des scores, gestion du chronomètre, création d'événements de match, etc.), maintenir la cohérence de l'état de l'application côté frontend devient complexe.

Conclusion

L'intégration des **WebSockets** dans *Easy Tourney* est fonctonnel et améliore l'expérience utilisateur en permettant une gestion dynamique et en temps réel des matchs. Malgré les défis techniques liés à la synchronisation du temps et à la gestion des multiples événements, l'utilisation de la bibliothèque **Socket.IO** a grandement simplifié l'implémentation, offrant une communication bidirectionnelle efficace et fiable. Cette approche assure que toutes les actions effectuées par l'Arbitre sont immédiatement visibles par les Spectateurs, renforçant ainsi l'interactivité et la réactivité de l'application.

4.7. Strategy Pattern

Note : L'application *EasyTourney* a été initialement conçue pour gérer un type unique de tournoi, le *CustomRoundRobin* (chapitre 3.2), en accord avec les exigences de la thèse. Cependant, l'intérêt manifesté pour l'application par un professeur de sport de l'Université de Fribourg a motivé une réflexion sur l'intégration de futurs types de tournois dès les premières phases de conception. Pour répondre à ce besoin d'extensibilité et de modularité, le **Strategy Pattern** a été adopté.

Le **Strategy Pattern** [40] est un design pattern comportemental permettant de définir et encapsuler une famille d'algorithmes interchangeables. Ce pattern facilite l'ajout de nouveaux algorithmes sans modifier le code existant, ce qui améliore la maintenabilité et la flexibilité du système.

En implémentant ce pattern, l'application peut intégrer de nouveaux types de tournois simplement en ajoutant une nouvelle stratégie, sélectionnée dynamiquement en fonction du contexte. Cette approche garantit une évolution structurée de l'application sans compromettre la qualité du code existant.

Ce design pattern, étudié dans le cadre du cours de *Génie Logiciel* dispensé par le **Prof. Dr. Jacques Pasquier-Rocha**, responsable de cette travail de master, a été une opportunité intéressante pour appliquer un concept théorique dans un cas pratique précis. Cette expérience illustre parfaitement l'utilité des principes de modularité et d'extensibilité enseignés.

4.7.1. Application du Strategy Pattern dans l'Application

Plusieurs Strategy Pattern ont été intégrées dans l'application *EasyTourney* pour répondre aux différents besoins des types de tournois. Ces implémentations permettent de gérer divers aspects de la planification des tournois, notamment la création des pools, la génération des plannings des pools, et la génération des matchs. Voici les principaux domaines où le Strategy Pattern a été appliqué :

1. **Création de Pools** : Permettrait de gérer des types de tournois avec ou sans pools, ou avec des pools à élimination directe.
2. **Génération de Planning - Pools** : Gère l'assignation des pools aux créneaux horaires en fonction du type de tournoi.

3. **Génération de Planning - Matchs** : Organise les matchs au sein des pools selon les règles spécifiques de chaque type de tournoi.

Cette approche permet d'ajouter facilement de nouveaux types de tournois en fournissant simplement une nouvelle implémentation de l'algorithme de planification via le contrôleur pour utiliser cette stratégie en fonction du type de tournoi sélectionné.

4.7.2. Architecture logicielle

L'architecture adoptée suit le diagramme de classes présenté dans la Figure 4.15, où chaque entité joue un rôle précis :

1. **PlanningController (Client)** : Reçoit les requêtes pour générer ou valider les plannings et délègue les opérations au `PlanningStrategyManager`.
2. **PlanningStrategyManager (Context)** : Gère la sélection et l'exécution des stratégies. Il maintient une référence à une stratégie concrète et délègue l'appel des méthodes.
3. **PlanningStrategy (Interface)** : Définit un contrat aux méthodes *generatePlanning()* et *validatePlanning()* que toutes les stratégies concrètes doivent implémenter.
4. **CustomRoundRobinPoolPlanning (Concrete Strategy)** : Implémente l'interface pour fournir une version spécifique des algorithmes de planification adaptée au tournoi de type *Custom Round Robin*.

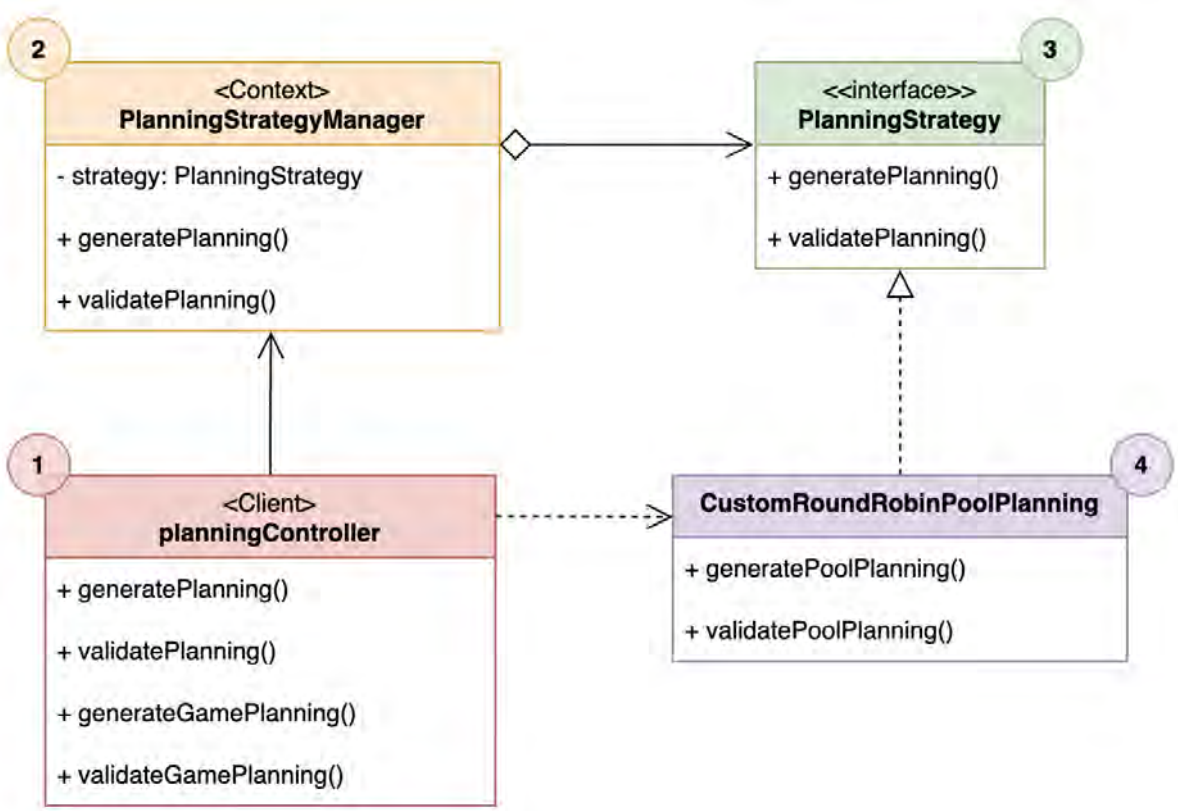


Fig. 4.15.: Architecture de l'application avec le Strategy Pattern

Voici une explication détaillée de chaque composant :

1. PlanningController (Client) : Le `PlanningController` est responsable de la réception des requêtes HTTP relatives aux plannings. Par exemple, il reçoit une requête pour générer le planning des pools et instancie le `PlanningStrategyManager` avec le type de stratégie à utiliser, comme montré ci-dessous :

```
const strategy = tourney.tourneyType;
// Instanciation du contexte
const planningStrategyManager = new PlanningStrategyManager(
  tourneyId,
  strategy,
  { randomMode: randomMode === 'true' }
);

// Execution de l'algorithme
await planningStrategyManager.generatePlanning();
};
```

List. 4.45: Extrait du PlanningController

2. PlanningStrategyManager (Context) : Le `PlanningStrategyManager` agit comme un intermédiaire. Il maintient une référence à une instance d'une classe concrète qui implémente l'interface `PlanningStrategy`.Il communique avec l'algorithme concret uniquement via cette interface.

```
class PlanningStrategyManager {
    constructor(tourneyId, strategyName, options = {}) {
        this.tourneyId = tourneyId;
        this.strategy = this.getStrategy(strategyName, options);
    }

    // Sélectionne et instance une stratégie en fonction du nom
    getStrategy(strategyName, options) {
        switch (strategyName) {
            case 'customRoundRobin':
            default:
                return new CustomRoundRobinPoolPlanning(this.tourneyId, options);
            // Ajout des autres stratégies...
        }
    }

    // Génère le planning en utilisant la stratégie sélectionnée.
    async generatePlanning() {
        return this.strategy.generatePlanning();
    }
    // Valide le planning généré.
    async validatePlanning() {
        return this.strategy.validatePlanning();
    }
}
```

List. 4.46: Extrait du PlanningStrategyManager

3. PlanningStrategy (Interface) : L'interface `PlanningStrategy` définit les méthodes `generatePlanning()` et `validatePlanning()`. Toute classe implémentant cette interface devra implémenter ses propres versions..

```
class PlanningStrategy {
  constructor(tourneyId) {
    this.tourneyId = tourneyId;
  }

  // Méthode abstraite pour générer un planning.
  async generatePlanning() {
    throw new Error('generatePlanning() doit être implémenté par la stratégie
        concrète.');
  }

  // Méthode abstraite pour valider un planning.
  async validatePlanning() {
    throw new Error('validatePlanning() doit être implémenté par la stratégie
        concrète.');
  }
}
```

List. 4.47: Interface PlanningStrategy

4. CustomRoundRobinPoolPlanning (Concrete Strategy) : Cette classe implémente les méthodes définies par `PlanningStrategy`. Elle contient la logique spécifique pour un tournoi de type *Custom Round Robin*, et sera expliquée en détail dans la section suivante :

```
const PlanningStrategy = require('./planningStrategy');

// Stratégie concrète implémente l'interface et ses fonctions
class CustomRoundRobinPoolPlanning extends PlanningStrategy {
    constructor(tourneyId, options = {}) {
        super(tourneyId);
        this.randomMode = options.randomMode || false;
    }

    async generatePlanning() {
        // Implémentation spécifique pour Custom Round Robin
    }

    async validatePlanning() {
        // Validation spécifique pour Custom Round Robin
    }
}
```

List. 4.48: CustomRoundRobinPoolPlanning

Lien avec la théorie : Comme expliqué dans la documentation [40], le *Context* dans le Strategy Pattern appelle dynamiquement les méthodes définies par une stratégie

sans connaître son type spécifique. Dans le cas de *EasyTourney*, cette approche est respectée: le `PlanningStrategyManager` agit comme un intermédiaire neutre, ne manipulant que l'interface `PlanningStrategy`, tandis que les stratégies concrètes (*CustomRoundRobinPoolPlanning*) fournissent leur propre logique pour `generatePlanning()` et `validatePlanning()`.

Ainsi, le choix du **Strategy Pattern** garantit une modularité maximale, permettant d'ajouter ou de modifier des stratégies sans altérer la structure globale de l'application. Ce design répond aux besoins de flexibilité identifiés, tout en appliquant les principes du génie logiciel.

4.7.3. Implémentation de l'Algorithme de Génération de Pools

Cet algorithme[21] est conçu pour assigner efficacement les équipes aux créneaux horaires et terrains disponibles, tout en respectant diverses contraintes pour assurer l'équilibrage et la diversité des matchs.

Présentation Générale de l'Algorithme

L'algorithme de génération de pools repose sur une approche itérative visant à assigner les équipes (pools) aux créneaux horaires disponibles sur les différents terrains et sports. L'objectif principal est de :

- Éviter les conflits d'horaires entre les pools et les terrains.
- Assurer que chaque pool participe au même nombre de sessions.
- Maximiser la diversité des sports joués par chaque pool.
- Équilibrer les assignations pour éviter que les pools ne jouent trop souvent le même sport.

Contraintes et Définition des Contraintes

Pour atteindre ces objectifs, plusieurs contraintes sont définies dans l'algorithme :

1. **Disponibilité des Terrains :** Chaque terrain dispose de créneaux horaires spécifiques pendant lesquels il est disponible pour les matchs.
2. **Durée des Sessions :** Chaque session de pool a une durée définie, incluant une période de transition entre les matchs.
3. **Pauses Obligatoires :** Des pauses telles que l'introduction, le déjeuner et la conclusion doivent être respectées, empêchant toute assignation de pools pendant ces périodes.
4. **Équilibrage des Matchs :** Chaque pool doit jouer un nombre équivalent de matchs, et la diversité des sports doit être maintenue pour éviter une répétition excessive.

[21] Code source-customRoundRobin: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/services/planningStrategies/pool/customRoundRobinPoolPlanning.js`

Diagramme de flux de l'Algorithme customRoundRobinPoolPlanning

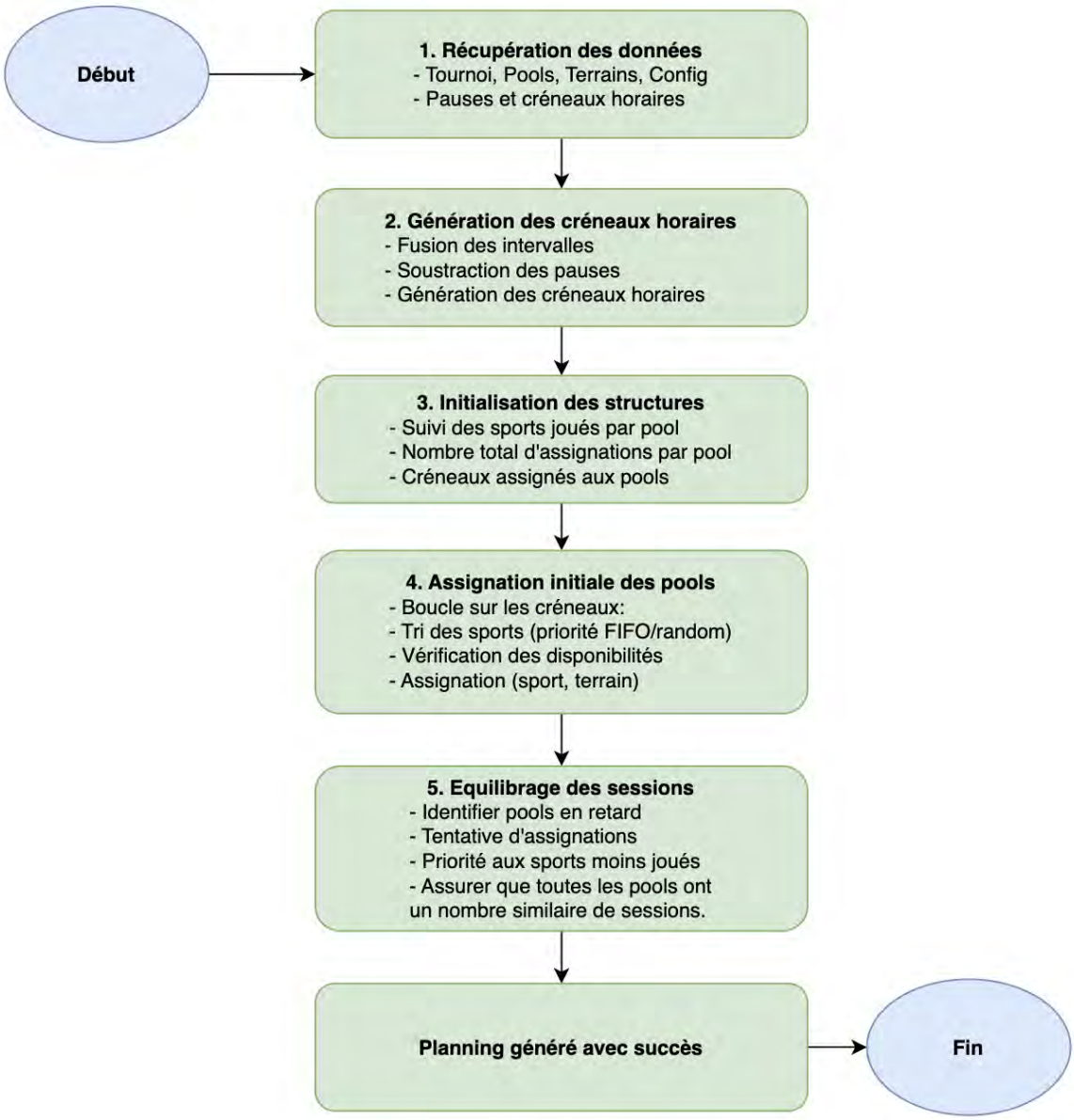


Fig. 4.16.: Diagramme de flux pour générer un planning de pools

L'algorithme se compose de plusieurs étapes clés, chacune correspondant à une phase spécifique du processus de génération de planning (Fig. 4.16):

1. Récupération des Données : L'algorithme commence par la récupération des données nécessaires, incluant les pools, les terrains disponibles, les configurations de planning, et les sports associés. Ces informations sont extraites de la base de données à l'aide des modèles définis dans l'application.

2. Génération des Créneaux Horaires : À partir des horaires de début et de fin du tournoi, ainsi que des pauses définies, l'algorithme calcule les créneaux horaires disponibles. Cette étape implique :

1. **Conversion des Heures en Minutes :** Facilite les calculs ultérieurs en travaillant avec des valeurs numériques.
2. **Fusion des Intervalles de Disponibilité :** Combine les disponibilités des différents terrains pour identifier les plages horaires communes.
3. **Soustraction des Périodes de Pauses :** Exclut les périodes de pauses des créneaux disponibles pour garantir qu'aucune pool n'est assigné pendant ces moments.

3. Assignation Initiale des Pools : Les pools sont assignées aux créneaux horaires en priorisant celles ayant le moins d'assignations totales. L'algorithme parcourt les créneaux disponibles et assigne chaque pool à un terrain et un sport disponibles, en tenant compte des contraintes d'équilibrage et de diversité.

4. Équilibrage des Sessions : Après l'assignation initiale, l'algorithme vérifie l'équilibrage des sessions entre les pools. Il identifie les pools nécessitant des sessions supplémentaires et tente de les assigner à des créneaux restants, en favorisant les sports les moins joués par ces pools.

5. Validation des Contraintes : Une fois le planning généré, l'algorithme procède à une validation rigoureuse pour s'assurer que toutes les contraintes sont respectées. Cela inclut la vérification des conflits d'horaires, le respect des pauses, et l'équilibrage des sessions entre les pools.

Une image illustrant les contraintes de temps entre deux configuration différentes (Durée d'un pool et transition à 60/60 min. pour la figure 4.17 et 120/0 min. pour la figure 4.18.

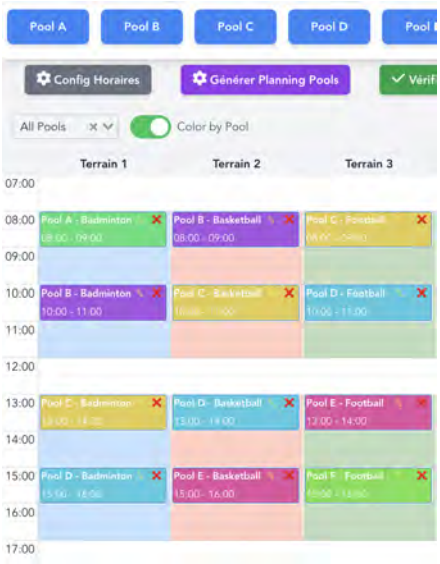


Fig. 4.17.: Avec Pause (1h)

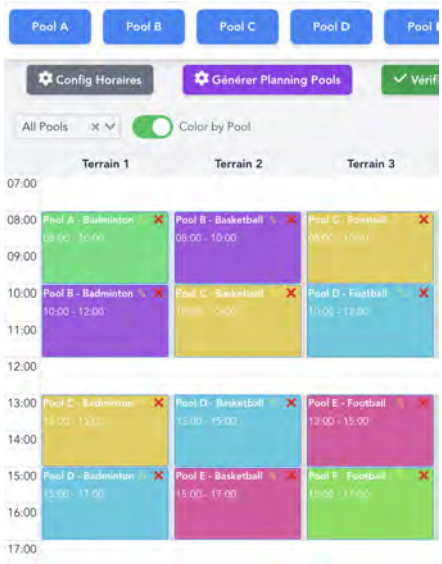


Fig. 4.18.: Sans Pause

Fig. 4.19.: Comparaison Planning avec deux configuration différentes.

On aperçoit que le planning génère correctement les plannings en fonction des contraintes de pauses de transition (trous entre les pools ou pas).

Utilisation de ChatGPT dans le Développement de l'Algorithme

Le développement de cet algorithme a été un défi considérable, nécessitant de gérer de multiples contraintes tout en assurant un équilibre optimal dans les assignations. Reconnaissant mes limites initiales dans ce domaine, j'ai choisi d'utiliser *ChatGPT* comme outil d'assistance pour itérer et affiner la solution. Avant d'avoir accès à un tel outil, je n'aurais probablement pas osé aborder un problème aussi complexe.

Contrairement à ce que l'on pourrait penser, cela n'a pas réduit la charge de travail, mais l'a transformée : j'ai dû analyser chaque proposition, comprendre le code produit, détecter les failles, et reformuler les problèmes pour obtenir des réponses adaptées.

Ce processus a également mis en lumière une tendance que l'on observe dans l'industrie. L'essor des outils d'intelligence artificielle ne remplace pas le travail humain, mais en accroît la productivité. Cela s'est vérifié ici, où l'itération et la reformulation des questions ont été aussi importantes que l'implémentation technique elle-même.

Enfin, ce projet m'a permis de comprendre l'importance d'un environnement de test rapide et robuste. Tester les algorithmes sur des cas limites (par exemple, des pools avec peu d'équipes ou des créneaux horaires restreints) a été essentiel pour garantir la fiabilité et l'efficacité de la solution. Bien que cela m'ait pris plus d'une semaine de travail intensif, ce processus m'a permis d'acquérir une meilleure compréhension de la planification algorithmique et des outils d'intelligence artificielle.

Défis et Résolutions

Plusieurs défis ont été rencontrés lors de la conception de cet algorithme :

- **Détermination de la Faisabilité :** Il est souvent difficile de déterminer à l'avance si un planning est faisable ou non. Plutôt que d'utiliser des formules mathématiques complexes pour vérifier la faisabilité, l'algorithme utilise une approche basée sur des tableaux, les peuplant progressivement, ce qui permet de générer un planning même si certaines contraintes ne peuvent pas être entièrement satisfaites.
- **Définition d'un Bon vs Mauvais Planning :** Un bon planning doit respecter l'équilibrage des sessions, la diversité des sports et éviter les conflits d'horaires. Certaines situations peuvent entraîner des plannings imparfaits. L'algorithme est conçu pour minimiser ces imperfections, mais il est parfois nécessaire d'accepter des petites incohérences, telles qu'un match supplémentaire pour un pool, sans compromettre l'équilibrage global.
- **Gestion des Pools de Petite Taille :** Une difficulté majeure a été de gérer les pools avec peu d'équipes. L'algorithme initial ne fonctionnait pas correctement pour ces configurations, entraînant des assignations déséquilibrées. Pour résoudre ce problème, une fonction spécifique a été implémentée pour les petits pools, tandis qu'une autre fonction gère les pools de taille standard en garantissant un équilibrage optimal, quel que soit le nombre d'équipes.

Conclusion Implémentation CustomRoundRobin

L'algorithme de génération de pools en `CustomRoundRobin` intégré dans *EasyTourney* démontre une approche équilibrée et flexible pour la planification des tournois. L'utilisation du **Strategy Pattern** a facilité l'intégration de cette fonctionnalité complexe, en assurant une modularité et une extensibilité optimales. Malgré les défis rencontrés, notamment en termes de faisabilité et de gestion des pools de petite taille, l'approche adoptée permet de générer des plannings efficaces tout en respectant les contraintes définies.

4.7.4. Conclusion Strategy Pattern

L'intégration du **Strategy Pattern** dans *EasyTourney* a permis de structurer le code de manière extensible et maintenable. Grâce à l'encapsulation des algorithmes dans des stratégies distinctes, l'application est capable d'ajouter facilement de nouveaux types de tournois, répondant ainsi à des besoins évolutifs tout en minimisant les risques d'introduire des bugs dans le code existant. Cette approche prépare également l'application à évoluer vers des formats de tournoi plus complexes tout en maintenant une clarté architecturale.

4.8. Fonctionnalités Annexes

Les fonctionnalités annexes seront rapidement parcourues car non essentielles ni demandées dans les objectifs du travail de master.

4.8.1. Planning Excel

Pour générer un fichier Excel, le frontend envoie toutes les informations nécessaires du planning et des matchs au backend. Ce dernier utilise la librairie **ExcelJS** [48] pour générer un fichier `Excel`[22].

Les étapes consistent à créer un classeur regroupant plusieurs feuilles (sheets), chacune dédiée à une partie spécifique des données du tournoi. Chaque feuille suit un algorithme unique pour générer ses contenus. Le processus commence par la création des colonnes et en-têtes de lignes pour structurer les données. Ensuite, les cellules sont parcourues comme dans un tableau pour insérer les informations nécessaires, telles que les horaires, équipes, terrains ou scores.

Les styles sont appliqués pour améliorer la lisibilité, comme des couleurs, bordures ou alignements. Une fois toutes les feuilles complétées, le fichier Excel est sauvegardé et renvoyé pour être utilisé ou téléchargé.

```
1 const workbook = new ExcelJS.Workbook(); // Création du classeur
2 const sheet = workbook.addWorksheet('Planning'); // Création de la sheet '
    Planning'
3 sheet.columns({header:"quipe A",key:"teamA",width: 20}, // Colonnes en-têtes
4 sheet.getRow(1).font = { bold: true }; // Stylisation
5 sheet.addRow(['08:00 - 09:00', 'Match 1', 'Match 2']); // Contenu cellule
```

[22] Code source-ExcelJS: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/server/utils/exportExcel.js`

```
6 await workbook.xlsx.writeFile('tournament.xlsx'); // Création du fichier
```

List. 4.49: Extrait représentatif de ExcelJS

Cette fonctionnalité renforce la portabilité de l'application en offrant un format facilement partageable. Ainsi, elle libère les utilisateurs de la contrainte d'utiliser l'application directement, tout en restant adaptée aux besoins d'un cadre sportif.

4.8.2. Mode Clair et Mode Sombre

L'implémentation du mode clair et sombre a été réalisée grâce à la configuration **darkMode** de **Tailwind CSS** [38]. Cette approche permet d'appliquer des styles différents en fonction d'une classe CSS (par exemple `dark`) ajoutée au **HTML**. Cela offre une grande flexibilité et une mise en place rapide, puisque les styles sont définis globalement et réutilisables dans tous les composants.

Avantages

- **Facilité de maintenance** : Les couleurs et styles sont centralisés dans le fichier de configuration Tailwind. Modifier une palette dans ce fichier met à jour instantanément l'apparence de l'application entière.
- **Portabilité** : Cette méthode peut être facilement adaptée pour des besoins spécifiques, comme ajuster le thème aux couleurs d'une école ou d'un événement.
- **Efficacité** : Les classes sont directement appliquées aux composants, ce qui réduit la duplication du style dans le code.

Limitation : Une contrainte est que l'utilisation de Tailwind CSS génère des **classes très longues et parfois peu lisibles** dans le HTML, ce qui peut compliquer la relecture du code.

```
1 <nav class="navbar bg-light-menu dark:bg-dark-menu p-4 shadow-md flex items-
    center justify-between">
2   <!-- Contenu du menu -->
3 </nav>
```

List. 4.50: Menu Navigation en Mode Clair et Sombre

Personnalisation : Changer les couleurs dans la configuration de Tailwind[23] permet d'adapter l'apparence de l'application à d'autres thèmes :

```
1 darkMode: 'class',
2 theme: {
3     colors: {
4       light: { menu: '#ffffff' },
5       dark: { menu: '#1f2937' },
```

List. 4.51: Extrait de Configuration Tailwind CSS

[23] Code source-Dark Mode: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/frontend/tailwind.config.js`

En ajustant ces couleurs, on peut rapidement transformer le site pour refléter un autre thème, par exemple celui d'une autre école ou d'un événement particulier. Voici par exemple un code couleur appliquant un style vintage avec des couleurs jaunes et violets:

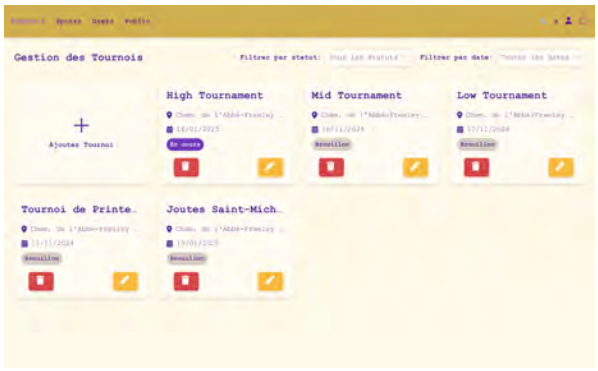


Fig. 4.20.: Vintage Look - Light Mode

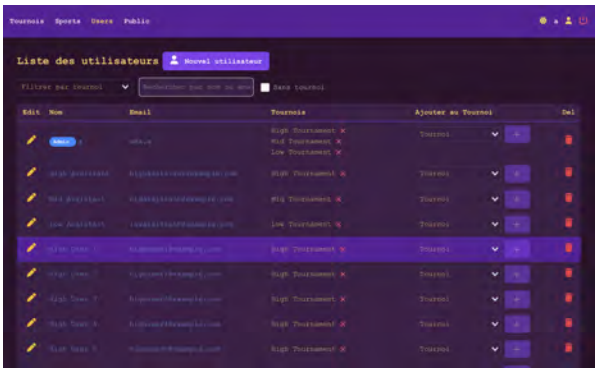


Fig. 4.21.: Vintage Look - Dark Mode

Fig. 4.22.: Modification du code couleur dans `tailwind.config.js`

Ce résultat reflète l'effort porté sur le design UI, la qualité du code et sa réutilisabilité, avec une attention particulière à la cohérence des styles. L'objectif était de démontrer la réutilisabilité des composants en centralisant la logique de style dans un fichier, tout en exploitant la puissance de TailwindCSS pour garantir simplicité et efficacité.

4.8.3. Mode Hors Ligne et PWA

L'implémentation de la Progressive Web App (PWA) repose sur les **service workers** [49] pour gérer le cache des ressources et le fonctionnement hors ligne.

1. **Service worker :** Le fichier `registerServiceWorker.js` enregistre un service worker chargé en arrière-plan pour gérer les ressources. Il met en cache les fichiers statiques et fournit une réponse locale si l'application est hors ligne.
2. **Vue.js et mode hors ligne :** L'application détecte l'état en ligne ou hors ligne via `navigator.onLine`. En ligne, les données sont récupérées depuis l'API ; hors ligne, elles sont chargées depuis le `localStorage`.
3. **Manifest :** Ce fichier personnalise l'application avec des icônes, des couleurs, et un mode *standalone* pour un comportement similaire à une application native.

```
if (navigator.onLine) {
    await this.fetchAllData(); // En ligne : récupérer les données du serveur
} else {
    this.loadAllFromLocalStorage(); // Hors ligne : charger les données locales
    console.warn('Vous êtes hors ligne. Chargement des données locales.');
}
```

List. 4.52: Gestion Hors Ligne Vue Planning Joueur

Cette fonctionnalité, bien que prototype, répond à l'objectif principal : permettre la consultation du planning hors ligne. Cependant, une amélioration notable serait d'automatiser

davantage la gestion de l'état en ligne et hors ligne, par exemple en écoutant les événements `online` et `offline` pour ajuster dynamiquement le comportement.

4.9. Tests et Intégration Continue

Cette section présente les différents processus de tests et de déploiement mis en place pour assurer la qualité et la stabilité de l'application avec **Postman**, en passant par des tests d'intégration continue (CI), une automatisation des pipelines sur **GitHub Actions**, et le déploiement sur **Heroku**.

Note : Le CI/CD (Continuous Integration/Continuous Deployment) désigne un ensemble de pratiques visant à intégrer et déployer les modifications de code de manière continue, automatisée et fiable. Dans le cas de notre application, seule la partie CI est partiellement implémentée : les tests sont automatisés via GitHub Actions, mais le déploiement reste manuel. Une explication détaillée de cette limitation sera fournie dans la conclusion.

4.9.1. Migrations et Seeders pour Accélérer le Développement

Un des piliers de la fiabilité et de la reproductibilité de l'application repose sur l'utilisation de **migrations**[24] et de **seeders**[25]. En effet, pour chaque nouvelle fonctionnalité, il est essentiel de pouvoir reconstruire rapidement l'état initial de la base de données, y injecter des données de test, puis exécuter l'application dans les mêmes conditions qu'en production.

Scripts personnalisés : Afin d'automatiser ces étapes, deux scripts ont été introduits dans le `server/package.json`. Ils accélèrent la mise en place d'un environnement de test ou de développement à l'aide du client sequelize:

```
{
  "scripts": {
    "db:reset": "npx sequelize-cli db:migrate:undo:all && \
                 npx sequelize-cli db:migrate && \
                 npx sequelize-cli db:seed:all && \
                 node utils/cleanupUploads.js && \
                 node app.js",
    "db:hard-reset": "npx sequelize-cli db:drop && \
                      npx sequelize-cli db:create && \
                      npx sequelize-cli db:migrate && \
                      npx sequelize-cli db:seed:all && \
                      node utils/cleanupUploads.js && \
                      node app.js"
  }
}
```

List. 4.53: Extrait du `package.json` montrant les scripts de migrations et seeders

[24] Code source-Migrations: `https://github.com/JulienRichoz/easy-tourney/tree/main/server/migrations`

[25] Code source-Seeder: `https://github.com/JulienRichoz/easy-tourney/tree/main/server/seeders`

- `db:reset`: Supprime et recrée l'historique des migrations, applique toutes les migrations, exécute les **seeders** pour insérer des données initiales (fictives ou réelles), nettoie le dossier `uploads` et lance le serveur.
- `db:hard-reset`: Opère une remise à zéro plus radicale de la base (drop puis create), avant de réappliquer les migrations et d'insérer de nouveau les données de test.

Ces scripts permettent d'accélérer grandement le développement, en testant rapidement et à grande échelle les implémentations.

4.9.2. Méthodologie et scénario de développement

La gestion du code est assurée par **GitHub**, via un workflow [58] de branches adapté à l'ajout fréquent de nouvelles fonctionnalités.

Méthodolgie de travail et tests : Voici un rapide aperçu de ma méthodologie de développement à travers un scénario de création de l'entité sport:

1. **Création d'une branche dédiée**: Pour chaque nouvelle fonctionnalité (*feature*), création d'une branche spécifique (ex. `feature/sport`).
2. **Implémentation du backend**: Définition des **migrations**, **modèles** et **seeders** pour la structure de la BDD, création ou mise à jour du contrôleur et des routes.
3. **Tests en local**:
   - Lancement de `db:reset` ou `db:hard-reset` si nécessaire.
   - Vérification du bon fonctionnement des endpoints via **Postman** pour s'assurer de la validité des retours JSON. En profiter pour écrire les tests afin de maintenir une couverte élevée.
4. **Implémentation du frontend**: Création d'une vue ou d'un composant dédié.
5. **Tests de cohérence en local**: Contrôle rapide de bout en bout (*end-to-end* local) pour confirmer qu'aucune régression n'a été introduite.
6. **Push sur GitHub & Pull Request (PR)**: La PR est alors soumise à la pipeline d'intégration continue (CI) qui déclenche les tests d'intégration automatisés. Si la pipeline échoue, la PR est refusée jusqu'à correction du code. **Versionning et tags**: Chaque étape majeure de l'application a été sauvegardée via un système de versionnement [19]. Par exemple, le tag `0.8.1` correspond à l'état de l'application utilisé pour cette thèse, permettant un retour en arrière en cas de problème critique non détecté.

Cette méthodologie repose sur une approche itérative inspirée des principes de la méthode agile [59].

Une méthodologie itérative et agile : La méthodologie agile repose sur des cycles courts et répétitifs (**itérations**), durant lesquels chaque fonctionnalité est conçue, testée et intégrée au projet. Cette approche favorise la flexibilité, la collaboration (si l'on est plusieurs) et l'amélioration continue. Dans le cadre de ce projet, ce modèle est appliqué à travers un cycle structuré : **implémentation, test, versionnement, et validation**. Chaque fonctionnalité passe par une phase d'ajustements successifs jusqu'à ce qu'elle soit jugée stable et prête pour une intégration finale.

Ce processus offre plusieurs avantages :

- Détection rapide des erreurs grâce aux tests réguliers et aux pipelines CI.
- Réduction des risques de régressions grâce à une gestion rigoureuse des versions.
- Suivi clair des évolutions et possibilités de retour en arrière en cas de besoin.

En conclusion, cette méthodologie garantit un développement structuré et adaptable, tout en permettant d'itérer efficacement sur les fonctionnalités pour répondre aux besoins du projet. Elle illustre bien l'importance d'un workflow agile dans des projets où les ajustements fréquents sont essentiels.

Dans la suite de ce rapport, nous nous concentrerons principalement sur **Postman**[22] et **GitHub Actions** [53] pour le CI/CD [54].

4.9.3. Tests d'Intégration avec Postman

Les tests d'intégration sont essentiels pour valider l'interaction entre les différents composants de l'application. Ils permettent de s'assurer que les endpoints de l'API fonctionnent correctement et que les différentes parties de l'application interagissent comme prévu. **Postman** est un outil puissant, largement utilisé pour concevoir, tester et documenter des APIs de manière manuelle ou automatisée.

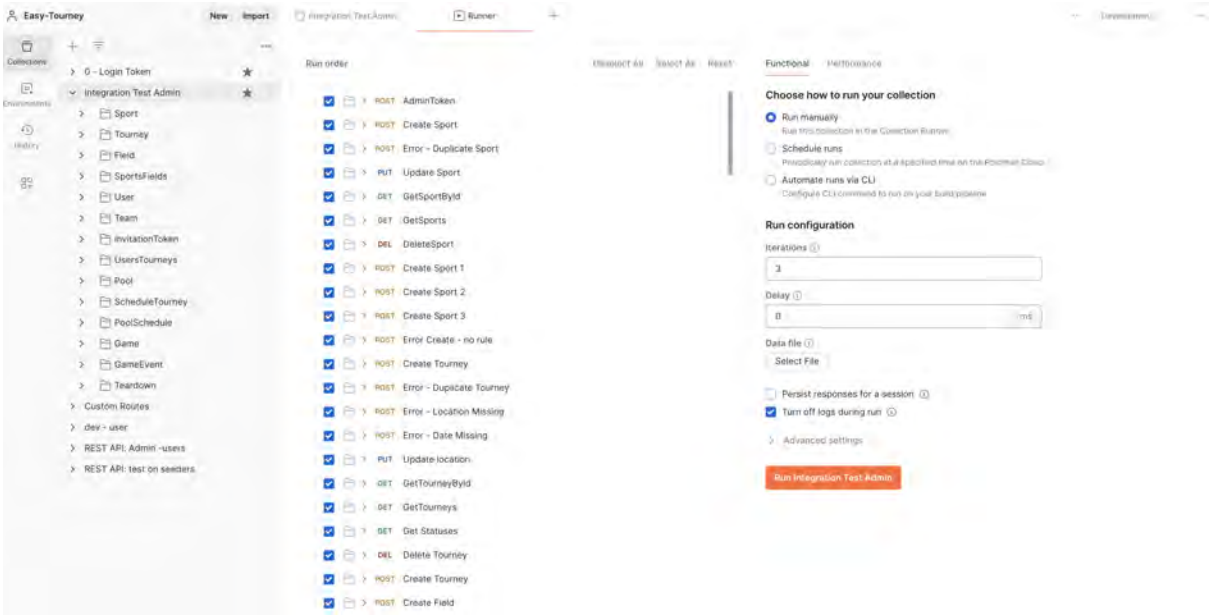


Fig. 4.23.: Collections de tests avec Postman couvrant 80%-90% des API endpoints

Création de scénarios de test : Au cours du développement, des centaines de tests ont été conçus via Postman, comme le montre la figure 4.23, afin de couvrir tous les cas d'utilisation importants. Par exemple, pour tester la création d'une entité `sport`, voici la configuration de la requête `POST` :

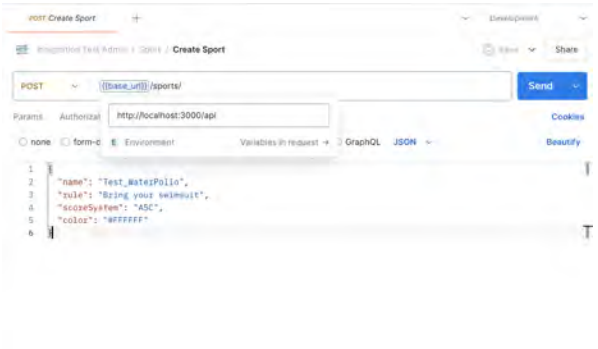


Fig. 4.24.: Body pour créer un sport

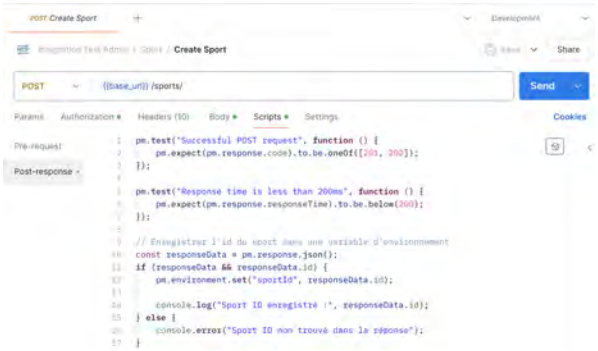


Fig. 4.25.: Scripts et variables

Fig. 4.26.: Requête POST avec Postman pour créer un sport

La figure 4.24 montre l'interface Postman configurée avec le corps de la requête (nom du sport, description, etc.) et un *header*, non visible sur la capture, incluant un token JWT pour l'authentification. Dans la figure 4.25, on observe l'utilisation de scripts Postman : après la requête, l'ID du sport est stocké dans la variable `sportId`, facilitant l'enchaînement de scénarios de test plus complexes (mise à jour, suppression, etc.).

Organisation des collections et exécution : Les tests sont regroupés dans des collections Postman (figure 4.23) et se terminent généralement par un `teardown` qui réinitialise l'état de la base de données. Cette approche permet un environnement de test propre et reproductible, évitant notamment les conflits de données (ex. doublons sur des sports ou des tournois).

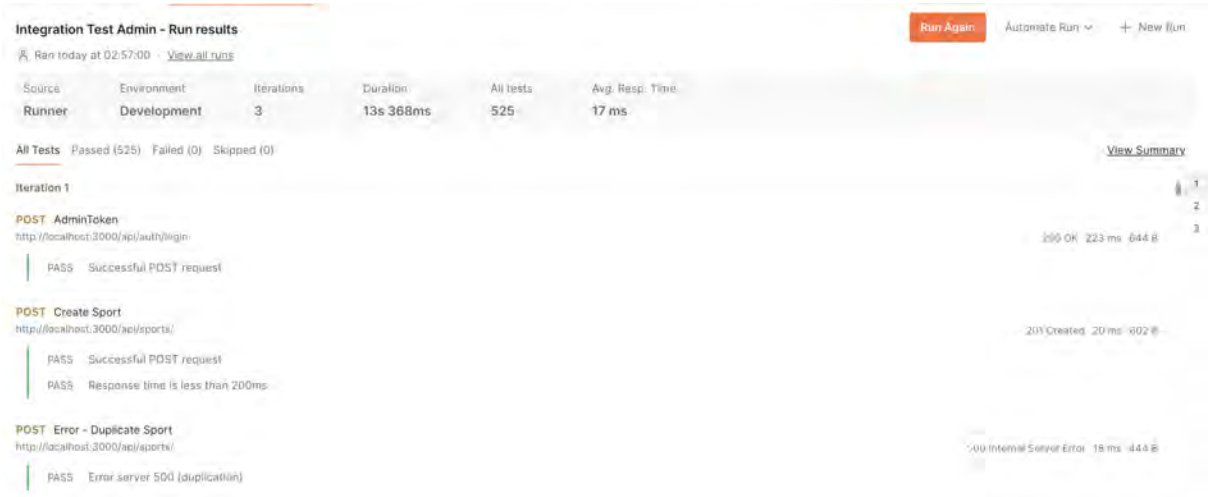


Fig. 4.27.: Exécution d'une collection Postman avec 3 itérations de tests

Sur la figure 4.27, on note par exemple un test spécifique, `Error - Duplicate Sport`, qui vérifie la gestion des doublons lors de la création d'un sport. Un autre test `"Response time is less than 200ms"` est présent: cette vérification de performance est appliquée à chaque requête pour garantir un temps de réponse optimal et éviter des requêtes SQL inefficaces susceptibles de rallonger inutilement le traitement.

Problématique : Les tests manuels sur Postman deviennent rapidement chronophages et laissent la porte ouverte à des régressions. Le passage vers l'automatisation via GitHub Actions permettra de :

- S'assurer de l'exécution systématique des tests à chaque mise à jour du code.
- Détecter les anomalies plus tôt.
- Éviter la répétition et la complexité croissante de tests manuels au fur et à mesure que l'application gagne en fonctionnalités.

Génération des fichiers Postman en JSON pour GitHub Actions : Pour ce faire, Postman permet d'exporter les tests en JSON en gardant toutes les configurations des scripts, qui seront stockés dans le dossier `server/tests` et utilisés dans les tests d'intégration continue avec GitHub Actions.

Résumé des pratiques de test avec Postman

- **Collections de requêtes**: Chaque endpoint de l'API (`/auth/login`, `/tourneys`, `/sport`, etc.) est testé via une requête Postman organisée dans des collections.
- **Assertions et scripts Postman**: Les codes HTTP et les corps de réponses JSON sont vérifiés via des scripts, qui peuvent aussi stocker des variables pour les tests successifs.
- **Exportation en JSON**: Les collections Postman sont exportées au format JSON et intégrées au dépôt `GitHub`, prêtes à être exécutées de façon automatisée par la suite.

4.9.4. Pipeline GitHub Actions pour les Tests Automatisés

GitHub Actions [53] est l'outil d'intégration continue (CI) et de déploiement continu (CD) utilisé dans ce projet. Il permet d'automatiser l'exécution des tests à chaque modification du code, comme un *push* ou une *pull request*. Cela garantit que les nouvelles modifications n'introduisent pas de régressions ou de dysfonctionnements.

Description du pipeline : Le fichier YAML `postman_tests.yml`[26] configure un pipeline qui s'exécute automatiquement lorsqu'un événement (comme un `push` ou une `pull request`) est détecté sur la branche `main`. Ce pipeline effectue les étapes suivantes :

1. Configuration d'un environnement Ubuntu.
2. Installation des dépendances nécessaires, notamment Node.js et MySQL.
3. Exécution des migrations et des seeders pour préparer la base de données.
4. Lancement des tests Postman exportés, à l'aide de l'outil **Newman** [47].

[26] Code source-YAML tests: `https://github.com/JulienRichoz/easy-tourney/blob/0.8.1-thesis/.github/workflows/postman_tests.yml`

Pourquoi GitHub Actions ? GitHub Actions est intégré à GitHub, ce qui en fait un choix naturel pour ce projet. Il permet une gestion centralisée du code et des workflows CI/CD sans nécessiter d'outils tiers. La configuration YAML rend les pipelines facilement reproductibles et modifiables.

```
// Execution des tests a chaque push ou PR sur main
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
// ... suite: jobs pour installer le serveur
```

List. 4.54: Déclenchement des tests lors d'action sur main.

Le serveur prêt, le script exécute les tests[27]:

```
// Etape 10 : Exécuter les tests Postman
newman run ./server/tests/collection.json \
--environment ./server/tests/environment.json
```

List. 4.55: Execution des tests Postman.

La figure 4.28 illustre un pipeline réussi sur GitHub.[28]

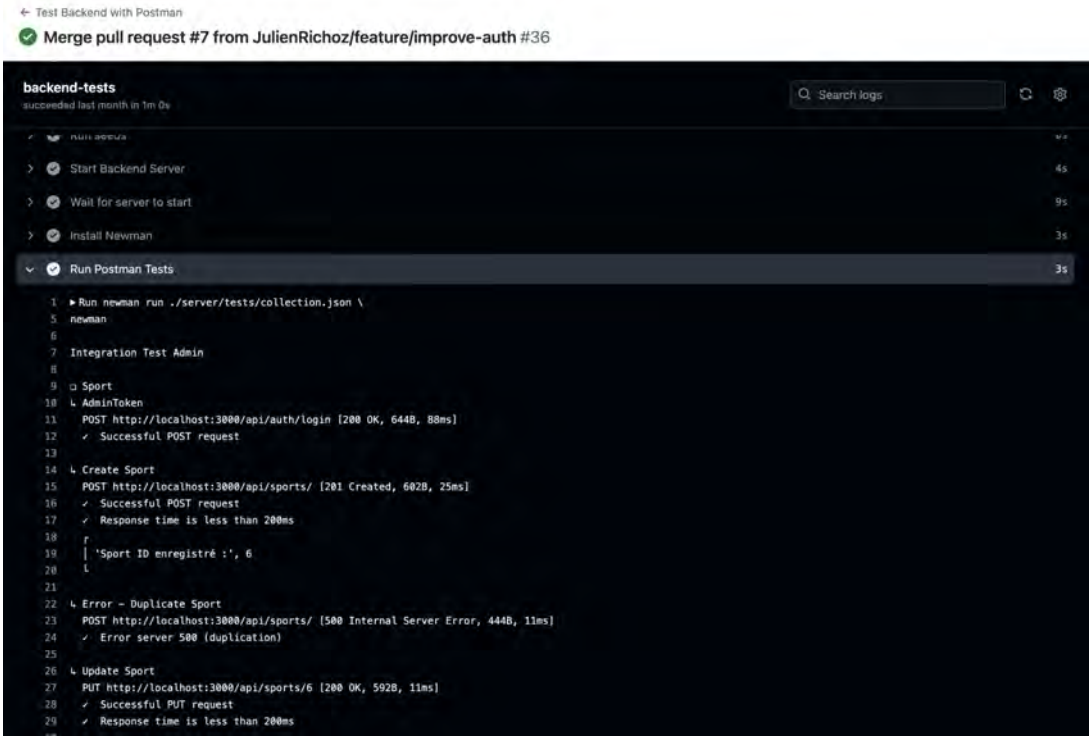


Fig. 4.28.: GitHub Actions testant le backend

[27] Code source-Tests Postman: `https://github.com/JulienRichoz/easy-tourney/tree/0.8.1-thesis/server/tests`

[28] Code source-CI tests: `https://github.com/JulienRichoz/easy-tourney/actions/runs/12383752464`

4.9.5. Déploiement sur Heroku

Le déploiement final de l'application est effectué sur **Heroku** [50], qui propose une plateforme PaaS (Platform as a Service). En raison de la structure monolithique du projet (répertoire `frontend` et `server` dans le même repo Github), le déploiement automatisé (CD) est limité. Actuellement, les deux parties sont déployées manuellement. La figure 4.29 montre que les deux serveurs sont déployés et actifs chez Heroku.

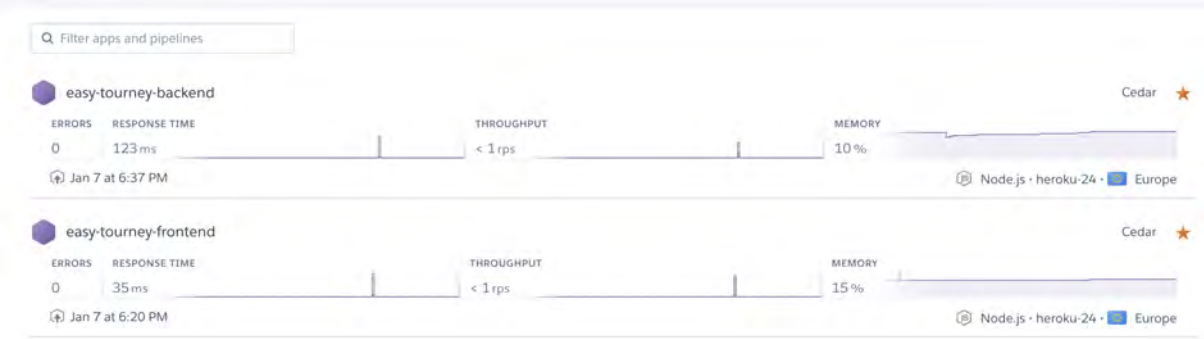


Fig. 4.29.: Serveurs Heroku Frontend (Vue) et Backend (Node/Express

Frontend : Le frontend est une application statique, générée via `npm run build`, qui produit un dossier `dist` contenant uniquement les fichiers nécessaires pour servir l'application (HTML, CSS, JavaScript). Sur Heroku, ce dossier est configuré avec un serveur statique (`serve`) pour être accessible en production.

Backend : Le backend, construit avec Node.js et Express, utilise une base de données MySQL hébergée sur **JawsDB** [51]. Les variables d'environnement, comme `DB_PASSWORD` et `JWT_SECRET`, sont configurées directement dans le tableau de bord Heroku, permettant une gestion centralisée et sécurisée.

Limitations et perspectives : La structure mono-repo complique l'automatisation du déploiement, car Heroku ne détecte pas automatiquement les builds pour les sous-dossiers. Pour résoudre ce problème, une séparation en deux dépôts GitHub (`frontend` et `backend`) est envisagée, ce qui permettrait une intégration plus fluide avec les outils de déploiement d'Heroku.
Malgré ces défis, l'application est entièrement fonctionnelle et performante sur Heroku. Les utilisateurs peuvent créer un compte et tester l'application en tant qu'utilisateur final. Cependant, sans rôle d'administrateur, ils seront limités, car ils ne pourront pas accéder aux tournois.
Lien EasyTourney : https://easy-tourney-frontend-899109fd59ab.herokuapp.com/login
Note: Il est possible que le lien ne soit plus disponible lors de la consultation. Merci de me contacter aux besoins d'un test ou d'installer directement l'application localement en suivant les instructions[29] sur Github.

[29]Code source-README Github: `https://github.com/JulienRichoz/easy-tourney/`

Conclusion de l'intégration continue

La mise en place des migrations et seeders, l'automatisation des tests d'intégration via Postman et le déploiement sur Heroku assurent un cycle de développement globalement complet. Chaque *push* ou *pull request* est validé par des tests, ce qui limite efficacement les régressions.
Cependant, des améliorations restent possibles, notamment avec l'ajout de tests unitaires pour renforcer la robustesse du code et la mise en place d'un déploiement continu (CD) pour automatiser entièrement le processus. Malgré ces points, les bases du pipeline CI et le déploiement sur Heroku constituent une réussite notable, apportant une réelle valeur ajoutée au projet.

4.10. Conclusion Point de Vue Développeur

À travers ce chapitre, nous avons mis en évidence l'architecture de "Easy Tourney" (Vue.js + Node/Express + Sequelize + MySQL) et son approche modulaire (MVC, migrations, seeders, Vuex) qui facilite l'évolution du code. Le diagramme de séquence "login à l'affichage des tournois" illustre comment chaque composant (composant Vue, store, API REST, contrôleur, etc.) coopère en pratique. Nous avons également détaillé plusieurs fonctionnalités avancées : **rôles de tournoi**, **token d'invitation**, **génération de pools via Strategy**, **WebSockets** pour le temps réel et des fonctionnalités hors ligne tels que le téléchargement du tournoi en Excel ou l'installation d'une PWA. Enfin, la mise en place d'un **pipeline CI** et de **tests d'intégration** garantit la fiabilité avant le déploiement sur Heroku, prouvant que le projet se veut aisément maintenable.

Easy Tourney offre ainsi un exemple concret d'une application web Full Stack unissant UX conviviale, architecture logicielle solide et algorithmes de planification répondant aux besoins d'événements sportifs scolaires.

5
Conclusion

5.1. Résultats

Le projet *Easy Tourney* a répondu efficacement aux défis rencontrés par les enseignants dans l'organisation de tournois multisports. Il fournit une plateforme robuste et intuitive, intégrant des fonctionnalités avancées telles que :

- La gestion des inscriptions via des liens d'invitation.
- La planification automatique des matchs.
- Un système de rôles utilisateurs (Admin, Assistant, Joueur, Invité).
- Un arbitrage des matchs et une visualisation des scores en temps réel.

En combinant des concepts modernes tels que le *Strategy Pattern* pour la génération de plannings et l'architecture MVC avec Node.js et Vue.js, le projet garantit modularité et évolutivité. La mise en œuvre de tests d'intégration, l'utilisation d'un pipeline CI léger et le déploiement sur Heroku témoignent d'un respect des bonnes pratiques de développement logiciel. Grâce à GitHub, une méthodologie Agile a été appliquée, favorisant des itérations régulières et une gestion efficace des tâches.

En résumé, *Easy Tourney* est une solution exploitable qui répond aux besoins des établissements scolaires, tout en posant les bases d'évolutions futures grâce à une architecture soignée et une approche centrée sur l'utilisateur.

5.2. Améliorations et Perspectives

Voici les différentes améliorations à entreprendre, regroupées en catégories :

- **Sécurité** :
  - Migration du stockage des tokens vers des cookies sécurisés (1 semaine) : Améliorer la sécurité contre les attaques XSS en remplaçant `localStorage` par des cookies *HttpOnly*.
- **Performance et fiabilité** :
  - Uniformisation sur le fuseau horaire UTC (2 semaines) : Permettre une gestion cohérente des horaires, essentielle pour les tournois internationaux ou multi-journées.

- **Nouvelles fonctionnalités** :
  - Gestion des tournois multi-journées (2 semaines) : Étendre les capacités de planification pour des événements s'étendant sur plusieurs jours.
  - Ajout de types de tournois (1-2 mois) : Implémenter des systèmes comme les éliminatoires, en collaboration avec des enseignants pour bien cerner les besoins.
  - Système de gestion des scores avancés (2 semaines) : Intégrer des sports individuels ou chronométrés, avec des calculs automatiques des moyennes par équipe.
- **Expérience utilisateur** :
  - Optimisation PWA (1 semaine) : Ajouter des fonctionnalités hors ligne et des notifications push pour enrichir l'expérience utilisateur.
- **Processus de développement** :
  - Pipeline CI/CD (variable) : Améliorer le processus de déploiement en automatisant les tests et la mise en production.

À long terme, l'intégration de fonctionnalités collaboratives, telles qu'un tableau de bord partagé pour plusieurs enseignants, ou la possibilité de créer des tournois inter-établissements, pourrait renforcer l'utilité d'*Easy Tourney*.

5.3. Retour personnel

Ce projet a été un véritable défi, notamment en raison de la complexité de la base de données, qui comprenait une quinzaine de tables. J'ai rapidement constaté que la complexité des relations entre les entités et les besoins de synchronisation croissent de manière non linéaire avec le nombre de tables. Cela m'a poussé à repenser plusieurs aspects de la conception, à améliorer ma rigueur et à approfondir ma compréhension des outils comme Sequelize.

Travailler seul sur un projet de cette envergure a été formateur, mais a mis en évidence certaines limites, telles que la fatigue et l'importance de la relecture par une autre personne pour détecter rapidement des erreurs. Néanmoins, l'utilisation d'un système de versionnement sur GitHub a été salvatrice, en me permettant de retourner facilement à des versions stables ou en identifiant l'origine des problèmes plus facilement.

En réalisant ce projet, j'ai confirmé ma volonté de combiner mes compétences techniques avec des activités à impact humain, comme l'éducation sportive.

En conclusion, ce projet m'a permis de concrétiser une application fonctionnelle répondant aux besoins des enseignants, tout en renforçant mes compétences en génie logiciel et en gestion de projet. Cette expérience restera une étape marquante de mon parcours académique et professionnel.

A

Code Source, Site Web et Installation

GitHub

Le code source complet est disponible sur GitHub :
`https://github.com/JulienRichoz/easy-tourney/tree/0.8.1-thesis`
La version **0.8.1-thesis**, datée du 18 janvier 2025, garantit un code conforme à celui décrit dans ce document.

Tester l'application en ligne

L'application est déployée et hébergée sur Heroku. Toutefois, **cet accès n'est pas garanti** en raison de possibles changements de nom de domaine ou de l'arrêt de l'hébergement. Vous pouvez y accéder via le lien suivant :
`https://easy-tourney-frontend-899109fd59ab.herokuapp.com/login`
Note: Pour toute demande de test ou d'accès à l'application en ligne, merci de me contacter sur mon mail: ***julienrichoz@outlook.com***

Installation

Pour tester l'application, vous pouvez suivre les instructions fournies dans le fichier `README.md` sur le GitHub. Pour accéder à la dernière version, rendez-vous sur :
`https://github.com/JulienRichoz/easy-tourney`

B

Acronymes Communs

API	Application Program Interface
CI/CD	Continuous Integration / Continuous Deployment
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheet
DB	DataBase
DBMS	Database Management System
DOM	Document Object Model
ERM	Entity Relationship Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JWT	JSON Web Token
JSON	JavaScript Object Notation
MVC	Model-View-Controller
ORM	Object-Relational Mapping
OS	Operating System
PWA	Progressive Web Application
REST	Representational State Transfer
SQL	Structured Query Language
UML	Unified Modeling Language
UNIFR	Université de FRibourg
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UI	User Interface
UX	User Experience

C

Licence de la documentation

Ressources web référencées

[1] Tournify - Planificateur de tournois en ligne. https://www.tournify.fr/ (accédé le: 28 Août, 2024).

[2] Challenge Place - Planificateur de tournois en ligne. https://challenge.place/ (accédé le: 29 Août, 2024).

[3] Expérience Utilisateur - Meilleures pratiques (lucidspark). https://lucidspark.com/blog/ux-design-best-practices (accédé le: 10 Septembre, 2024).

[4] Vue.JS - Frontend JavaScript Framework. https://vuejs.org/ (accédé le: 10 Septembre, 2024).

[5] Introduction à EdgeJS. https://edgejs.dev/docs/introduction (accédé le: 30 Septembre, 2024).

[6] Documentation AdonisJS. https://docs.adonisjs.com/guides/preface/introduction (accédé le: 1 Octobre, 2024).

[7] Tutoriel AdonisJS 6. https://adocasts.com/series/lets-learn-adonisjs-6 (accédé le: 1 Octobre, 2024).

[8] NodeJS - Javascript Platform. https://nodejs.org/en (accédé le: 5 Octobre, 2024).

[9] ExpressJS - Minimalist web framework for Node.js. https://expressjs.com/ (accédé le: 5 Octobre, 2024).

[10] RestFul API. https://restfulapi.net/ (accédé le: 08 Octobre, 2024).

[11] UseCase Diagrams with draw io. https://drawio-app.com/blog/uml-use-case-diagrams-with-draw-io/ (accédé le: 10 Septembre, 2024).

[12] MySQL - Pourquoi le choisir ? . https://www.mysql.com/fr/why-mysql/ (accédé le: 13 Octobre, 2024).

[13] Comparaison entre base de données relationnelles et non relationnelles. https://aws.amazon.com/compare/the-difference-between-relational-and-non-relational-databases/ (accédé le: 13 Octobre, 2024).

[14] Transaction ACID.
`https://www.databricks.com/fr/glossary/acid-transactions` (accédé le: 13 Octobre, 2024).

[15] MongoDB - Aggregation.
`https://welovedevs.com/fr/articles/aggregation-mongo/` (accédé le: 13 Octobre, 2024).

[16] Type de tournoi Round Robin - Wikipedia.
`https://fr.wikipedia.org/wiki/Tournoi_toutes_rondes` (accédé le: 15 Octobre, 2024).

[17] Sequelize ORM - Documentation. `https://sequelize.org/docs/v6/` (accédé le: 15 Octobre, 2024).

[18] ORM - Object Relational Mapping - definition. `https://www.theserverside.com/definition/object-relational-mapping-ORM` (accédé le: 15 Octobre, 2024).

[19] Nomenclature de versionnement (pour GitHub dans mon cas).
`https://semver.org/` (accédé le: 15 Octobre, 2024).

[20] MVC Framework Introduction (Geeks for geeks).
`https://www.geeksforgeeks.org/mvc-framework-introduction/` (accédé le: 15 Octobre, 2024).

[21] Axios - http request and introduction. `https://axios-http.com/docs/intro` (accédé le: 16 Octobre, 2024).

[22] Postman - Application permettant de tester les API. `https://www.postman.com/` (accédé le: 16 Octobre, 2024).

[23] Reddit Forum - Discussion sur comment gérer les scores dans une base de données. `https://www.reddit.com/r/csharp/comments/hfpz4b/for_a_database_of_game_leaderboards_is_it_better/` (accédé le: 17 Octobre, 2024).

[24] ERM - Entité Relation Model (sqllearning). `https://sqlearning.com/sql-server-introduction/entity-relationship-model/` (accédé le: 18 Octobre, 2024).

[25] Store - Guide sur le store vuex. `https://vuex.vuejs.org/guide/` (accédé le: 23 Octobre, 2024).

[26] FullCalendar - Documentation. `https://fullcalendar.io/docs/vue` (accédé le: 24 Octobre, 2024).

[27] Reddit - Looking for a full calendar option (forum). `https://www.reddit.com/r/vuejs/comments/1dq2y27/seeking_recommendations_for_a_fullfeatured/` (accédé le: 24 Octobre, 2024).

[28] Composants Vue - Introduction et explications.
`https://vuejs.org/guide/essentials/component-basics` (accédé le: 25 Octobre, 2024).

[29] What is CRUD ? - codecademy.
`https://www.codecademy.com/article/what-is-crud` (accédé le: 30 Septembre, 2024).

[30] Draggable - Librairie de drag n drop pour Vue. `https://www.npmjs.com/package/vuedraggable/v/next` (accédé le: 3 Novembre, 2024).

[31] Comparaison de la scalabilité horizontale et verticale pour les bases de données. `https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling` (accédé le: 3 Novembre, 2024).

[32] Progressive Web App (PWA) - developer-mozilla. `https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps` (accédé le: 4 Novembre, 2024).

[33] OAuth2 - Service d'authentification. `https://oauth.net/2/` (accédé le: 4 Novembre, 2024).

[34] JSON Web Token (JWT) - site officiel. `https://jwt.io/` (accédé le: 4 Novembre, 2024).

[35] Comparaison entre Bootstrap et TailwindCSS. `https://strapi.io/blog/bootstrap-vs-tailwind-css-a-comparison-of-top-css-frameworks` (accédé le: 4 Novembre, 2024).

[36] FullCalendar - Resource Time Grid pour gérer l'affichage vertical. `https://fullcalendar.io/docs/v4/vertical-resource-view` (accédé le: 4 Novembre, 2024).

[37] TailwindCSS - Light CSS Framework. `https://tailwindcss.com/` (accédé le: 4 Novembre, 2024).

[38] TailwindCSS - Thème de couleur Dark Mode/Light Mode. `https://tailwindcss.com/docs/dark-mode` (accédé le: 8 Novembre, 2024).

[39] Transaction avec Sequelize - Comment faire ? `https://sequelize.org/docs/v6/other-topics/transactions/` (accédé le: 20 Novembre, 2024).

[40] Strategy Pattern - Refactoring Guru. `https://refactoring.guru/design-patterns/strategy` (accédé le: 20 Novembre, 2024).

[41] Problème N + 1 avec les ORM (StackOverFlow). `https://stackoverflow.com/questions/97197/what-is-the-n1-selects-problem-in-orm-object-relational-mapping` (accédé le: 5 Décembre, 2024).

[42] Single Page Application (SPA) - Qu'est-ce que c'est ? `https://developer.mozilla.org/en-US/docs/Glossary/SPA` (accédé le: 6 Décembre, 2024).

[43] SocketIO - What is a Socket.IO ? `https://socket.io/docs/v4/` (accédé le: 10 Décembre, 2024).

[44] Comparison of a WebSocket and HTTP. `https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/` (accédé le: 10 Décembre, 2024).

[45] Tutoriel pour implémenter les websocket. `https://medium.com/@emperorbrains/building-real-time-applications-with-vue-js-and-websockets-3db2dd8d5d7c` (accédé le: 13 Décembre, 2024).

[46] Websocket - explications approfondies. https://vishalrana9915.medium.com/understanding-websockets-in-depth-6eb07ab298b3 (accédé le: 13 Décembre, 2024).

[47] Newman - Executer script postman (repo github). https://github.com/postmanlabs/newman (accédé le: 14 Décembre, 2024).

[48] ExcelJS - librairie pour travailler avec les fichiers Excel. https://www.npmjs.com/package/exceljs (accédé le: 18 Décembre, 2024).

[49] PWA - Service workers: exemple de configuration. https://cli.vuejs.org/core-plugins/pwa.html#example-configuration (accédé le: 22 Décembre, 2024).

[50] Heroku - Platform where EasyTourney is deployed. https://www.heroku.com/ (accédé le: 22 Décembre, 2024).

[51] JawsDB - Service MySQL sur Heroku (AddOn). https://www.heroku.com/ (accédé le: 22 Décembre, 2024).

[52] Différence entre un état 'stateless' et 'stateful'. https://www.redhat.com/fr/topics/cloud-native-apps/stateful-vs-stateless (accédé le: 3 Janvier, 2025).

[53] GitHub Actions - Pipeline et actions. https://github.com/features/actions (accédé le: 5 Janvier, 2025).

[54] Qu'est-ce que l'intégration continue et le développement continu (CI/CD) ? https://www.redhat.com/fr/topics/devops/what-is-ci-cd (accédé le: 5 Janvier, 2025).

[55] Reddit Forum - Meilleures pratiques concernant le stockage du JWT. https://www.reddit.com/r/webdev/comments/x15xvg/jwt_storage_best_practices/ (accédé le: 10 Janvier, 2025).

[56] Cross Site Script Attack (XSS) - Qu'est-ce que c'est ? https://www.kaspersky.fr/resource-center/definitions/what-is-a-cross-site-scripting-attack (accédé le: 10 Janvier, 2025).

[57] Attaque CSRF. https://owasp.org/www-community/attacks/csrf (accédé le: 11 Janvier, 2025).

[58] GitFlow - Comment travailler avec les branches github. https://docs.github.com/en/get-started/using-github/github-flow (accédé le: 15 Janvier, 2025).

[59] Principes des méthodes agiles. https://www.techtarget.com/searchsoftwarequality/definition/agile-software-development (accédé le: 16 Janvier, 2025).

[60] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (accédé le: 20 Décembre, 2024).