Faculté des sciences
Département d'informatique

Rapport projet de session

Théodore Grignard
grit0301
Sébastien Drezet
dres7036
Xavier Dostie
dosx2601
Tony Besse
best3551

Travail présenté à
M. Armel Ayimdji Tekemetieu
Dans le cadre du cours
Applications Internet et mobilité – IFT604/IFT717
9 décembre 2025

Table des matières

1. Description du projet

1.1 Vue d'ensemble

Le projet L4M Esports consiste en la conception et le développement d'une plateforme mobile centralisée, dédiée à la gestion et à l'animation d'un club e-sport universitaire.

Dans un contexte où l'organisation d'événements compétitifs nécessite une coordination rigoureuse, cette application a pour vocation première de connecter les joueurs, de simplifier l'organisation des équipes et des événements, et de fédérer les membres autour d'une communauté active. La plateforme agit comme un "hub" central pour l'ensemble des activités du club, couvrant aussi bien les sessions d'entraînements amicaux que les tournois officiels, en fournissant des outils de communication et de suivi en temps réel.

1.2 Objectifs du projet

Le développement de cette solution répond à plusieurs objectifs :

- Centralisation : Unifier l'ensemble des activités et informations du club e-sport au sein d'une seule et même interface.
- Automatisation : Réduire la charge de travail des organisateurs en automatisant la gestion des tournois, la génération des arbres de tournois (brackets) et le processus d'inscription.
- Communication : Fluidifier les échanges entre les membres, les équipes constituées et les organisateurs.
- Suivi en temps réel : Offrir une expérience dynamique permettant le suivi direct des matchs, des scores et l'évolution des résultats.
- Accessibilité : Garantir une utilisation flexible grâce à un accès multi-plateformes (Application native Android et Interface Web).

1.3 Public cible

La plateforme s'adresse aux différents acteurs de la vie associative du club :

- Membres du club : Joueurs occasionnels ou compétitifs souhaitant participer aux activités proposées.
- Capitaines d'équipe : Responsables chargés de la gestion administrative et sportive de leur équipe.
- Administrateurs : Organisateurs des tournois et gestionnaires globaux du club et de la plateforme.
- Communauté universitaire : Ensemble des étudiants portant un intérêt à l'e-sport.

1.4 Fonctionnalités principales

Pour répondre aux besoins de ces utilisateurs, l'application intègre les fonctionnalités majeures suivantes :

- Gestion avancée des utilisateurs avec authentification sécurisée.
- Module de création et de gestion d'équipes e-sport.
- Organisation complète d'événements et de tournois.
- Génération algorithmique automatique des brackets (arbres de tournois).
- Suivi des matchs et mise à jour des scores.

- Système complet de notifications (push, in-app).
- Intégration de l'API Twitch pour la mise en avant des profils de streamers.
- Assistant IA contextuel pour l'aide à l'utilisation.
- Géolocalisation intégrée pour faciliter les événements en présentiel.
- Interface multi-plateformes : Mobile (Android natif) et Web (Angular).

2. Analyse et conception

2.1 Analyse des besoins

Cette phase a permis d'identifier et de catégoriser les exigences du système pour garantir une solution robuste et adaptée.

2.1.1 Besoins fonctionnels

Les besoins fonctionnels sont répartis selon les principaux modules de l'application :

Gestion des utilisateurs

- Inscription et mécanismes d'authentification sécurisée.
- Gestion des profils utilisateurs incluant : gamertag, préférences de jeux, et localisation.
- Système de rôles hiérarchiques (Membre, Capitaine, Administrateur).
- Liaison optionnelle avec un compte Twitch externe.

Gestion des équipes

- Création d'équipes spécifiques par jeu (ex: LoL, Valorant, Rocket League).
- Système d'invitations et de demandes d'adhésion.
- Gestion administrative des membres (ajout, retrait, transfert du rôle de capitaine).
- *Règle métier :* Un utilisateur ne peut être membre que d'une seule équipe active par jeu.

Gestion des événements

- Création d'événements (format en ligne ou présentiel).
- Paramétrage des dates d'ouverture des inscriptions et de déroulement.
- Support de multiples formats de compétition (1v1, 2v2, 3v3, 4v4, 5v5, BATTLE_ROYALE).
- Génération automatique des arbres de compétition.
- Services de géolocalisation pour les événements physiques.

Gestion des matchs

- Instanciation automatique des matchs lors de la génération du bracket.
- Suivi des statuts de match (*upcoming, in_progress, finished, cancelled*).

- Enregistrement, validation des scores et avancement automatique des vainqueurs.
- Clôture automatique de l'événement une fois la finale terminée.

Système de notifications

- Support de plus de 40 types de notifications contextuelles.
- Gestion des notifications programmées et des niveaux de priorité (haute, moyenne, basse).
- Mécanisme de nettoyage automatique des anciennes notifications.

2.1.2 Besoins non fonctionnels

- Sécurité : Mise en œuvre de l'authentification JWT, hachage sécurisé des mots de passe, et validation stricte des entrées utilisateurs.
- Performance : Utilisation d'un cache Redis, indexation de la base MongoDB et optimisation des requêtes.
- Scalabilité : Architecture modulaire favorisant la séparation des responsabilités.
- Maintenabilité : Code structuré et documentation technique complète.
- Disponibilité : Gestion robuste des erreurs et validation des données pour éviter les interruptions de service.

2.2 Modélisation des cas d'utilisation

2.2.1 Acteurs

- Member (Membre) : Utilisateur standard du club. Il peut créer une équipe, rejoindre des équipes existantes et participer aux événements.
- Captain (Capitaine) : Possède les droits du Membre, mais gère en plus son équipe et procède aux inscriptions aux événements.
- Admin (Administrateur) : Possède les droits globaux pour gérer tous les aspects de la plateforme (utilisateurs, tournois, résultats).

2.2.2 Cas d'utilisation principaux (Détails)

Les scénarios suivants décrivent les interactions principales et les règles de gestion associées :

UC1 : S'inscrire et se connecter

- Acteur : Member
- Scénario : L'utilisateur s'inscrit en fournissant email, mot de passe, identité et gamertag. En retour, il reçoit un token JWT permettant son authentification pour les sessions futures.

UC2 : Créer une équipe

- Acteur : Member
- Précondition : L'utilisateur ne doit pas être déjà capitaine d'une équipe active pour le jeu sélectionné.
- Scénario : L'utilisateur crée une équipe et se voit automatiquement attribuer le rôle de *Captain*.

UC3 : Inscrire une équipe à un événement

- Acteur : Captain
- Préconditions : L'événement doit être en statut "open" et la période d'inscription valide.
- Scénario : Le capitaine sélectionne les membres participants et valide l'inscription de son équipe.

UC4 : Générer un bracket

- Acteur : Admin
- Préconditions : Inscriptions closes, aucun match encore joué.
- Scénario : L'administrateur lance la génération. Le système crée l'arbre de tournois et instancie les matchs automatiquement.

UC5 : Gérer un match

- Acteur : Admin
- Scénario : L'admin met à jour le statut du match, saisit le score et valide le résultat. Le système fait avancer automatiquement le vainqueur au tour suivant.

UC6 : Rejoindre une équipe

- Acteur : Member
- Précondition : Ne pas être membre d'une équipe active pour ce jeu.
- Scénario : L'utilisateur envoie une demande. Si le capitaine l'acepte, l'utilisateur rejoint l'effectif.

UC7 à UC9 : Gestion des contraintes d'appartenance

- Ces cas d'utilisation gèrent les échecs logiques : tentative de création d'une seconde équipe pour un même jeu (UC7), tentative de rejoindre une seconde équipe (UC8), ou tentative de quitter une équipe alors qu'un événement est en cours (UC9). Dans ces cas, le système rejette l'action avec un message d'erreur explicite.

UC10 : Transférer le rôle de capitaine

- Acteur : Captain
- Scénario : Le capitaine délègue son rôle à un autre membre. Les rôles et permissions sont mis à jour instantanément (l'ancien capitaine devient simple membre).

UC11 : Retirer un membre d'une équipe

- Acteur : Captain
- Contrainte : Impossible si le membre participe à un événement dont le statut est "en cours".
- Scénario : Le capitaine exclut un membre de l'équipe.

UC12 & UC13 : Modification d'événement et Annulation

- L'administrateur peut modifier un événement (UC12) et un capitaine peut annuler une inscription (UC13), à la condition stricte que l'événement ne soit pas déjà en statut *in_progress*.

UC14 : Validation du format d'équipe à l'inscription

- Acteur : Captain
- Scénario : Lors de l'inscription à un tournoi (ex: 5v5), le système vérifie que le nombre de joueurs sélectionnés correspond exactement au format requis. Si ce n'est pas le cas, l'inscription est bloquée.

UC15 : Sécurité de la régénération de bracket

- Acteur : Admin
- Scénario : Empêche la régénération accidentelle d'un arbre de tournoi si des matchs ont déjà été joués, garantissant l'intégrité de la compétition.

2.3 Règles de gestion

Afin de garantir l'intégrité des compétitions et la cohérence des données, le système applique rigoureusement les règles métier suivantes :

2.3.1 Règles relatives aux équipes

- Unicité du capitanat : Un utilisateur ne peut exercer la fonction de capitaine que pour une seule équipe active par jeu donné.
- Unicité de l'appartenance : Un utilisateur ne peut être membre que d'une seule équipe active par jeu.
- Restriction administrative : Les comptes ayant le rôle *Admin* ne peuvent pas intégrer d'équipes en tant que joueurs.
- Verrouillage des effectifs :
    - Il est impossible de retirer un membre d'une équipe si celui-ci participe à un événement dont le statut est "en cours".
    - Un membre ne peut pas quitter son équipe de son plein gré s'il est engagé dans un événement en cours.

2.3.2 Règles relatives aux événements

- Privilège d'inscription : Seul le capitaine de l'équipe est habilité à inscrire son équipe à un tournoi.
- Conformité du format : L'équipe doit inscrire le nombre exact de joueurs requis par le format de l'événement (ex: un tournoi 5v5 exige strictement 5 titulaires).
- Unicité de participation : Un membre ne peut participer à un événement qu'au sein d'une seule équipe.
- Respect des délais : Les inscriptions ne sont possibles que durant la période définie.
- Verrouillage "En cours" :
    - Aucune annulation d'inscription n'est permise une fois l'événement démarré.
    - Les paramètres de l'événement (dates, format, nom) ne sont plus modifiables une fois le statut passé à "en cours".

- Intégrité du bracket : La régénération de l'arbre de tournois est bloquée dès lors qu'au moins un match a été joué.

2.3.3 Règles relatives aux matchs

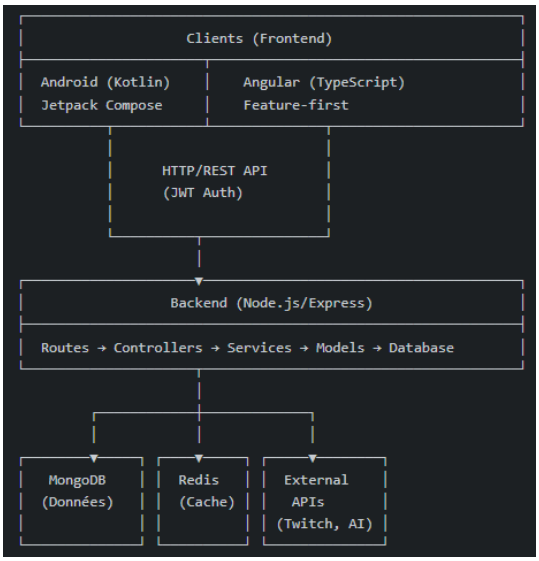
- Droits d'administration : Seuls les administrateurs ont les droits pour modifier le statut d'un match, saisir les scores et valider le résultat final.
- Saisie des scores : La modification du score n'est autorisée que si le match est au statut *in_progress*.
- Clôture automatique : La validation manuelle par un administrateur bascule automatiquement le statut du match à *finished*.
- Progression automatique : Le vainqueur est déterminé par le système selon le score saisi et est automatiquement promu au tour suivant dans l'arbre de tournoi.
- Fin de tournoi : L'événement se termine automatiquement (statut *finished*) une fois le vainqueur de la finale déterminé.

3. Architecture Technique

3.1 Architecture générale

Le projet repose sur une architecture multi-tier assurant une séparation claire des responsabilités entre les interfaces clients et la logique serveur.

Le schéma ci-dessous illustre les interactions entre les différentes couches du système :



3.2 Architecture Backend

Le backend est développé en Node.js avec le framework Express, structuré selon une approche modulaire MVC (Modèle-Vue-Contrôleur) adaptée aux API REST.

3.2.1 Structure du projet

L'arborescence du serveur est organisée comme suit :

- config/ : Configuration de la base de données et variables d'environnement.

- models/ : Définition des schémas de données Mongoose (*User, Team, Event, Match, etc.*).
- routes/ : Définition des points d'entrée de l'API REST.
- controllers/ : Gestion de la logique des requêtes entrantes et des réponses HTTP.
- services/ : Implémentation de la logique métier pure (*Business Logic*).
- middlewares/ : Fonctions intermédiaires pour l'authentification (JWT), la gestion d'erreurs et la validation.
- utils/ & scripts/ : Outils transverses et scripts d'initialisation des données (seeding).

3.2.2 Flux de traitement d'une requête

Le traitement d'une requête suit un flux linéaire et sécurisé :

1. Requête HTTP entrante.
2. Passage par les Middlewares (CORS, Parsing JSON).
3. Routing vers la Route Express correspondante.
4. Vérification par le Middleware d'authentification (si la route est protégée).
5. Traitement par le Controller (validation des formats d'entrée).
6. Appel au Service (exécution de la logique métier).
7. Interaction avec le Model (accès aux données MongoDB/Redis).
8. Renvoi de la Réponse JSON au client.

3.2.3 Stack technique Backend

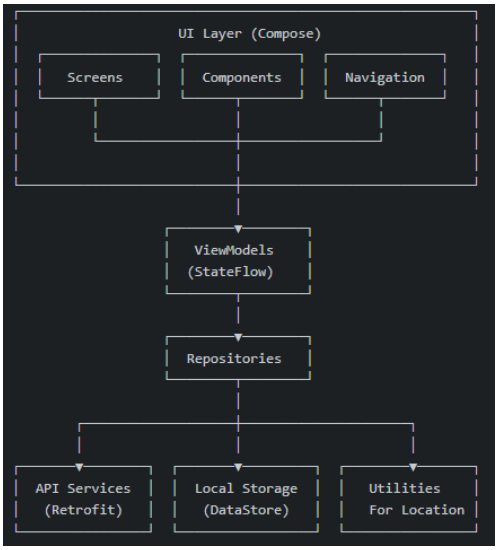
- Runtime : Node.js
- Framework Web : Express.js
- Base de données : MongoDB (NoSQL) avec Mongoose (ODM)
- Cache : Redis (optimisation des performances)
- Sécurité : JWT (Authentification stateless) et bcryptjs (Hachage)
- Clients externes : Axios (Intégration API Twitch et OpenAI)

3.3 Architecture Android

L'application mobile est développée en Kotlin et utilise l'approche moderne de l'interface déclarative avec Jetpack Compose. Elle respecte le pattern architectural MVVM (Model-View-ViewModel) recommandé par Google.

3.3.1 Structure et Pattern MVVM

L'architecture sépare l'interface utilisateur de la logique de données pour faciliter la testabilité et la maintenance.



Composants clés :

- UI Layer : Écrans (*Screens*) et composants réutilisables (*Components*) construits avec Jetpack Compose. La navigation est gérée par *Navigation Compose*.
- ViewModels : Ils exposent l'état de l'interface via des StateFlow et survivent aux changements de configuration.
- Data Layer : Les *Repositories* font l'abstraction entre les sources de données distantes (API via Retrofit) et locales (DataStore).

3.3.2 Stack technique Android

- Langage : Kotlin
- UI : Jetpack Compose
- Injection de dépendances : Hilt
- Réseau : Retrofit & Gson
- Concurrence : Coroutines & Flow
- Persistance locale : DataStore (Préférences utilisateur)
- Multimédia : Coil (Chargement asynchrone d'images)
- Services Google : Google Play Services Location

3.4 Architecture Angular

L'interface Web est une Single Page Application (SPA) développée avec Angular. Le projet adopte les standards récents du framework (version 17+).

3.4.1 Principes architecturaux

- Feature-first : L'organisation des dossiers se fait par fonctionnalité métier (*Auth, Games, Teams, Events*...) et non par type de fichier, ce qui favorise la modularité.

- Standalone Components : Utilisation de composants autonomes, supprimant la complexité des NgModules.
- Signals : Utilisation du nouveau système de réactivité d'Angular pour une gestion d'état fine et performante.
- Lazy Loading : Chargement différé des modules de fonctionnalités (features) pour optimiser le temps de chargement initial.
- Services Centralisés : Un client API centralisé dans src/app/api/ gère les appels REST de manière typée et cohérente.

4. Mise en œuvre

4.1 Technologies utilisées

Le choix de la stack technique a été guidé par des objectifs de performance, de maintenabilité et de capacité à gérer des données en temps réel.

4.1.1 Backend (Serveur & Base de données)

Le backend repose sur un environnement Node.js (v18+), choisi pour sa gestion asynchrone efficace des entrées/sorties.

- Framework : Express.js (v4.18.2) pour la structuration de l'API REST.
- Base de données : MongoDB couplé à l'ODM Mongoose (v8.0.3) pour la flexibilité des schémas de données.
- Cache : Redis (v4.6.12) pour l'optimisation des performances et le stockage temporaire.
- Sécurité : Implémentation de JWT (v9.0.2) pour l'authentification stateless et bcryptjs pour le hachage des mots de passe.
- Temps réel : Socket.io (v4.7.4) assure la communication bidirectionnelle instantanée.
- Intégrations : Axios pour les requêtes vers les API tierces (Twitch, OpenAI).

4.1.2 Application Mobile (Android)

L'application native est développée en Kotlin (1.9+) et adopte les derniers standards de développement Android.

- Interface Utilisateur : Jetpack Compose (Material 3) pour une UI déclarative moderne.
- Architecture : Utilisation de Hilt pour l'injection de dépendances et de MVVM.
- Réseau : Retrofit combiné à Gson pour la consommation de l'API REST.
- Asynchronisme : Utilisation intensive des Coroutines Kotlin.
- Navigation & Données : Navigation Compose pour le flux d'écrans et DataStore pour la persistance légère.
- Services : Google Play Services Location pour la géolocalisation et Coil pour le chargement d'images.

4.1.3 Application Web (Angular)

Le client Web est une Single Page Application (SPA) robuste basée sur Angular 17+.

- Langage : TypeScript.
- Gestion d'état : Utilisation des *Signals* pour la réactivité et RxJS pour la gestion des flux asynchrones.
- Architecture : *Standalone Components* (suppression des NgModules) pour alléger la structure.
- Temps réel : Client Socket.io intégré pour la synchronisation des données.

4.2 Fonctionnalités implémentées

Cette section détaille l'implémentation des fonctionnalités clés à travers les différentes couches de l'application.

4.2.1 Authentification et gestion des utilisateurs

Le système garantit une sécurité robuste et une gestion fine des identités.

- Backend : Gestion complète du cycle de vie utilisateur (inscription avec validation d'unicité email/gamertag, connexion JWT, révocation de token). Le système gère trois niveaux de rôles (*Member, Captain, Admin*) et permet la liaison avec un compte Twitch.
- Android : Interface sécurisée comprenant le stockage chiffré du token JWT via DataStore, la gestion de profil et une prévisualisation de l'intégration Twitch.
- Angular : Protection des routes via des *Guards*, intercepteurs HTTP pour l'injection automatique des tokens, et interfaces d'administration des utilisateurs.

4.2.2 Gestion des jeux

- Backend : CRUD complet réservé aux administrateurs. Le système supporte une variété de formats compétitifs (du 1v1 au 5v5, incluant le Battle Royale) et permet la définition de règles spécifiques.
- Android : Catalogue de jeux avec filtrage et affichage détaillé.

4.2.3 Gestion des équipes

L'accent est mis sur la flexibilité de la composition des équipes tout en respectant les contraintes compétitives.

- Backend : Logique complexe incluant la création d'équipes, l'invitation, les demandes d'adhésion, et le transfert de capitanat. Des règles strictes empêchent le retrait de membres ou la dissolution d'équipe lorsqu'un événement est actif.
- Android : Gestion ergonomique des effectifs (listes filtrables, validation des demandes d'adhésion) synchronisée en temps réel via Socket.io.

4.2.4 Gestion des événements

Cœur du système compétitif, ce module gère l'organisation des tournois de A à Z.

- Backend : Prise en charge des événements en ligne et présentiels (avec coordonnées GPS). Le système gère le cycle de vie complet : ouverture des inscriptions, génération automatique des arbres de tournois (*brackets*), et clôture automatique une fois le vainqueur déterminé.

- Android : Affichage interactif des brackets, filtrage des événements (notamment par proximité géographique via Google Maps) et visualisation en temps réel de l'avancement.

4.2.5 Inscriptions aux événements

- Backend : Contrôle rigoureux des conditions d'inscription (rôle de capitaine requis, respect du format d'équipe, dates limites).
- Android : Interface permettant aux capitaines de sélectionner leur "roster" et de gérer leurs inscriptions (modification/annulation sous conditions).

4.2.6 Gestion des matchs

L'automatisation est centrale pour réduire la charge de travail des administrateurs.

- Backend : Création automatique des matchs via le bracket. Le système gère les transitions d'état complexes (*in_progress → pending_validation → finished*), valide les scores, désigne les vainqueurs et fait avancer automatiquement les équipes gagnantes au tour suivant.
- Android : Suivi en direct des scores, interfaces de validation pour les administrateurs et mise à jour instantanée via WebSockets.

4.2.7 Système de notifications

Un moteur de notifications complet informe les utilisateurs des actions importantes.

- Backend : Gestion de plus de 40 types d'événements déclencheurs, classés par catégories :
  - *Matchs :* Création, début imminent, changement de score, résultat final.
  - *Équipes :* Invitations, adhésions, transferts de rôles.
  - *Événements :* Ouvertures d'inscriptions, mises à jour de brackets. Le système supporte les notifications programmées (ex: rappels avant match) et la gestion des priorités.
- Android : Centre de notifications avec filtrage, badges de non-lus et actions rapides.

4.2.8 Intégration Twitch & Assistant IA

- Twitch : Récupération et affichage des métadonnées des streamers (statut live, followers, avatar) pour enrichir les profils joueurs.
- Assistant IA : Intégration de l'API OpenAI (GPT) offrant un chatbot contextuel capable de répondre aux questions des utilisateurs en fonction de leur rôle et de la page consultée.

4.2.9 Géolocalisation

- Backend : Utilisation d'index géospatiaux MongoDB pour permettre la recherche d'événements à proximité (/api/events/nearby).
- Android : Exploitation du GPS du téléphone pour proposer des événements pertinents et affichage des lieux de tournoi sur Google Maps.

4.3 Points techniques remarquables

L'implémentation a nécessité la résolution de plusieurs défis techniques :

1. Gestion des erreurs : Mise en place d'un middleware centralisé et de classes d'erreurs personnalisées (NotFoundError, ValidationError) pour uniformiser les réponses API.
2. Optimisation des performances : Utilisation stratégique de Redis pour le cache et d'index optimisés (géospatiaux et relationnels) sur MongoDB.
3. Sérialisation polymorphique (Android) : Développement de *Deserializers* Gson personnalisés pour gérer les réponses API où certains champs peuvent être soit une simple chaîne de caractères (ID), soit un objet complet peuplé (*populated object*).
4. Mécanisme de Bracket : Algorithme de génération automatique pour les tournois à élimination directe, incluant des sécurités anti-régénération pour préserver l'historique des matchs joués.

4.4 Scripts d'initialisation et Tests

Pour faciliter le déploiement et le développement, le projet inclut :

- Scripts de seeding : Un ensemble de scripts (initDefaultAdmin.js, initDefaultGames.js, etc.) permet de peupler la base de données avec un jeu de données cohérent, incluant des scénarios complexes (tournois en cours, événements géolocalisés à Montréal).
- Stratégie de test : Le backend bénéficie de tests unitaires (Jest) et de tests d'intégration couvrant les flux critiques (Auth, Équipes, Jeux), exécutés sur une instance *MongoDB Memory Server* pour l'isolation.

4.5 Déploiement

L'infrastructure est conçue pour être conteneurisée et modulaire :

- Backend : Orchestration via docker-compose.yml incluant les services Node.js, MongoDB et Redis. Configuration flexible via variables d'environnement.
- Clients : Configuration Gradle optimisée pour Android (Support SDK 24 à 36) et build de production pour Angular prêt pour le déploiement statique.

5. Présentation du résultat

5.1 Application Mobile (Android)

L'application Android constitue le point d'entrée principal pour les utilisateurs finaux. Elle respecte les principes du Material Design 3.

5.1.1 Authentification et Onboarding

L'expérience utilisateur débute par une sécurisation de l'accès :

- Connexion : Un formulaire épuré permet l'accès via email et mot de passe, avec une gestion d'erreurs explicite en cas d'identifiants invalides.
- Inscription : La création de compte valide en temps réel les données saisies (unicité du gamertag et de l'email) pour garantir la cohérence de la base d'utilisateurs.

5.1.2 Navigation et Ergonomie

La navigation principale s'articule autour d'une barre inférieure (*Bottom Navigation Bar*) donnant un accès rapide aux quatre piliers de l'application : Jeux, Équipes, Notifications (avec badge de compteur dynamique) et Profil.

5.1.3 Modules fonctionnels

- Catalogue de Jeux : Présentation visuelle des jeux supportés sous forme de cartes. Les administrateurs disposent ici d'un bouton d'action flottant pour ajouter ou modifier des jeux. Chaque jeu détaille les événements qui lui sont associés, avec une distinction claire pour les tournois en présentiel.
- Gestion des Équipes : Ce module permet de visualiser les équipes existantes avec des options de filtrage par jeu et statut. La vue détaillée d'une équipe offre des actions contextuelles selon le rôle de l'utilisateur (inviter un membre, gérer les demandes d'adhésion, ou quitter l'équipe).
- Événements et Tournois : C'est le cœur de l'application. Les cartes d'événements affichent les informations cruciales (format, dates, statut). La vue détaillée intègre le Bracket visuel (arborescence du tournoi), permettant de suivre l'évolution des matchs du premier tour jusqu'à la finale. Pour les événements physiques, une section "Lieu" intègre un lien direct vers Google Maps.
- Suivi des Matchs : Les utilisateurs peuvent consulter les scores en temps réel. Les administrateurs accèdent à des outils de modération pour modifier le statut du match, saisir les scores et valider officiellement les résultats.

5.1.4 Fonctionnalités transverses

- Centre de Notifications : Une liste centralisée permet de gérer les alertes (invitations, rappels de matchs, résultats). L'utilisateur peut marquer les notifications comme lues ou les supprimer.
- Profil Utilisateur : Au-delà des informations de base, cette section permet de lier et tester la connexion avec un compte Twitch (récupération de l'avatar et du statut live) et de gérer la sécurité du compte (déconnexion, suppression).
- Assistant IA : Un widget flottant, accessible en permanence, ouvre une interface de chat. L'IA, consciente du contexte (page visitée et rôle de l'utilisateur), répond aux questions d'assistance.

5.2 Application Web (Angular)

L'interface web offre une expérience miroir de l'application mobile, adaptée aux écrans de bureau. Elle reprend l'ensemble des fonctionnalités (Dashboard, gestion des tournois, administration) avec une mise en page optimisée (navigation latérale, zone de contenu large), idéale pour les tâches d'administration lourdes effectuées par les organisateurs de tournois.

5.3 Interface de Programmation (API)

Le Backend expose une API RESTful structurée permettant l'interaction avec les clients Mobile et Web. Voici les principaux points d'entrée :

Authentification & Utilisateurs

- POST /api/auth/* : Gestion complète (inscription, connexion, déconnexion).
- GET/PUT/DELETE /api/users/me : Gestion autonome du profil utilisateur connecté.

Ressources Métier

- GET /api/games & /api/teams : Consultation des catalogues.
- CRUD /api/events : Gestion complète des événements, incluant l'endpoint spécifique /api/events/:id/generate-bracket pour la création automatique des arbres de tournois.
- GET /api/events/nearby : Recherche géolocalisée.

Gestion Sportive (Matchs)

- PUT /api/matches/:id/score : Mise à jour des scores.
- POST /api/matches/:id/validate : Validation administrative déclenchant l'avancement dans le bracket.

Services Tiers & Utilitaires

- GET /api/twitch/user/:username : Proxy vers l'API Twitch.
- POST /api/ai/assist : Interface avec le service OpenAI.

6. Conclusion et perspectives

6.1 Bilan du projet

Le projet L4M Esports a permis de réaliser une plateforme de gestion de tournois complète, répondant aux exigences d'un environnement universitaire dynamique. La solution délivrée est robuste, sécurisée et centralisée. Elle réussit le pari de simplifier la logistique complexe des organisateurs tout en offrant une expérience fluide et moderne aux joueurs via une application mobile native et une interface web.

6.2 Perspectives d'évolution

Pour aller plus loin, plusieurs axes d'amélioration sont envisageables :

- Tournois complexes : Support des formats "Double Elimination" (Loser bracket).
- Streaming in-app : Intégration du lecteur vidéo Twitch directement dans l'application.
- Aspect social : Ajout d'un système de chat en direct pour chaque match.
- Monétisation : Intégration d'une passerelle de paiement pour des tournois à inscription payante.
- Portage iOS : Développement d'une version Apple pour couvrir l'ensemble du parc mobile étudiant.

6.3 Acquis pédagogiques

Ce projet a permis de consolider des compétences en développement Fullstack, notamment la gestion de bases de données NoSQL, le développement mobile déclaratif avec Jetpack Compose, l'intégration d'APIs tierces et la mise en œuvre d'une architecture logicielle professionnelle.