

Rush Master

Elevate Your League Experience

Dossier Projet

Titre RNCP de niveau 6:
Concepteur développeur d'applications

Réalisé par:

Clément Ramos

LaPlatefome

8 rue d'hozier,
13002, Marseille

Lien du site

<https://rushmaster.fr/>

Table des matières

Introduction	3
1. Compétences mises en œuvre	4
1.1 Développer une application sécurisée	4
1.2 Concevoir et développer une application sécurisée organisée en couches	5
1.3 Préparer le déploiement d'une application sécurisée	5
2. L'écosystème League of Legends	6
2.1 Présentation générale de League of Legends	6
2.2 Le mode Ranked : un système compétitif structurant	7
2.3 League of Legends, un esport majeur	8
2.4 Analyse des sites concurrents	8
2.5 Utilisateurs cible : les joueurs et petits entraîneurs amateurs	10
3. Expression des besoins	11
3.1 Contexte et objectifs	11
3.2 Besoins fonctionnels	11
3.3 Besoins non fonctionnels	12
4. Gestion du projet	13
4.1 Organisation et méthodologie	13
4.2 Outils utilisés	14
4.3 Maquettage	14
5. Fonctionnalités de l'application	16
5.1 Présentation des principales pages de l'application	17
5.2 Fonctionnalité Admin	21
6. Environnement technique	25
6.1 Architecture générale	25
6.2 Technologies utilisées	26
6.3 Fonctionnalités techniques principales	27
6.4 Configuration des environnements	27
6.5 Tests automatisés	29
6.6 Hébergement	29
6.7 Base de données	31
7. Conclusion et perspectives	33
Annexes	36

Introduction

Dans le domaine des jeux vidéo en ligne, et plus particulièrement dans des titres compétitifs comme League of Legends, les joueurs attachent une grande importance au suivi de leurs performances et à la compréhension de leur progression. L'accès à des données fiables et à des outils d'analyse devient alors un véritable atout. C'est dans cette optique qu'est né Rush Master, une application web permettant de consulter ses statistiques, son historique de parties, et de comparer ses résultats avec ceux d'autres joueurs.

À l'origine, ce projet a été lancé en dehors du cadre scolaire, simplement par intérêt personnel pour le jeu et par envie de mettre en pratique des compétences techniques. L'idée a progressivement évolué, jusqu'à devenir suffisamment structurée pour être présentée comme projet de fin d'année dans le cadre de la formation de Concepteur Développeur d'Applications (CDA), en vue de la validation du titre RNCP de niveau 6.

L'application repose sur les données publiques fournies par l'API officielle de Riot Games. Ces données sont récupérées, stockées dans notre base de données, puis utilisées pour proposer des indicateurs supplémentaires et des fonctionnalités personnalisées. Cela permet notamment d'enrichir l'expérience utilisateur avec des comparaisons, des classements entre amis ou des tendances de progression. Le système d'authentification OAuth 2.0 proposé par Riot Games a également été intégré, permettant une connexion sécurisée sans manipulation directe d'informations sensibles.

Même s'il s'agit d'un projet réalisé dans un cadre scolaire, nous avons cherché à appliquer des pratiques proches de celles rencontrées en entreprise, notamment en ce qui concerne l'organisation du code, l'intégration de services tiers et la gestion des flux utilisateurs. Ce projet a représenté une bonne occasion de mettre en œuvre de manière concrète les compétences acquises au cours de la formation.

En résumé, Rush Master est un projet né d'une motivation personnelle, qui a trouvé sa place dans un cadre pédagogique, et qui m'a permis de combiner passion, apprentissage et mise en pratique de manière progressive et structurée

1. Compétences mises en œuvre

Ce projet "RushMaster" a permis de mobiliser la grande majorité des compétences professionnelles, conformes au référentiel RNCP de niveau 6 pour le titre de Concepteur et Développeur d'Application. Toutes les compétences ont été validées, à l'exception de celles liées à la gestion des bases de données NoSQL.

1.1 Développer une application sécurisée

- Installer et configurer son environnement de travail en fonction du projet

Nous avons mis en place un environnement complet avec Docker et Docker Compose pour gérer le frontend, backend et base de données. Cela a facilité la configuration et la reproductibilité du projet dans différents environnements.

- Développer des interfaces utilisateur

Le frontend a été développé avec React JS, utilisant Material-UI pour une interface moderne et responsive. Nous avons conçu des composants réutilisables et pris en compte l'accessibilité et l'ergonomie.

- Développer des composants métier

La logique métier, principalement côté backend avec Spring Boot, gère la récupération et le traitement des données issues de l'API Riot Games ainsi que la gestion des utilisateurs et statistiques.

- Contribuer à la gestion d'un projet informatique

Le projet a été mené en binôme avec une organisation agile, des sprints hebdomadaires, une gestion du code via Git, et des revues de code régulières.

1.2 Concevoir et développer une application sécurisée organisée en couches

- Analyser les besoins et maquetter une application
Nous avons réalisé une analyse fonctionnelle complète avec des maquettes initiales sous Figma, validées en binôme avant la phase de développement.
- Définir l'architecture logicielle d'une application
L'architecture a été définie en couches claires : frontend (React), backend (Spring Boot), base de données relationnelle (PostgreSQL). La communication s'appuie sur une API REST sécurisée.
- Concevoir et mettre en place une base de données relationnelle
La base PostgreSQL a été modélisée selon les besoins, avec des scripts d'initialisation et une gestion des données persistantes via Docker.
- Développer des composants d'accès aux données SQL et NoSQL
La compétence SQL est validée avec Spring Data JPA pour l'accès à PostgreSQL. Le volet NoSQL n'a pas été développé dans ce projet.

1.3 Préparer le déploiement d'une application sécurisée

- Préparer et exécuter les plans de tests d'une application
Un plan de test complet a été mis en place, incluant des tests unitaires, d'intégration, de navigation et de charge. Les tests ont été automatisés (JUnit, Jest, Selenium, JMeter) et intégrés à la chaîne d'intégration continue, garantissant une validation régulière du bon fonctionnement de l'application.

- Préparer et documenter le déploiement d'une application
Le déploiement a été préparé à l'aide de Docker et Docker Compose, assurant une portabilité entre les environnements. Une documentation technique a été rédigée pour faciliter l'installation, la configuration et la maintenance de l'application.
- Contribuer à la mise en production dans une démarche DevOps
Une chaîne CI/CD a été mise en place avec GitHub Actions, automatisant les tests et les déploiements. Des outils de suivi et de journalisation ont été utilisés pour assurer la stabilité après la mise en production.

2. L'écosystème League of Legends

Avant de parler réellement du projet RushMaster et des problématiques auxquelles le projet répond, il est nécessaire de vous donner un peu de contexte surtout pour les personnes qui ne jouent pas ou ne connaissent pas forcément le jeu.

2.1 Présentation générale de League of Legends

League of Legends (LoL), développé par Riot Games, est un jeu vidéo de type MOBA (Multiplayer Online Battle Arena) sorti en 2009. Il oppose deux équipes de cinq joueurs dans des parties stratégiques et compétitives, où chaque joueur choisit un champion aux capacités uniques. Le but est de détruire la base adverse par le biais d'une des 3 "voies" présentes sur la carte tout en coopérant avec son équipe.

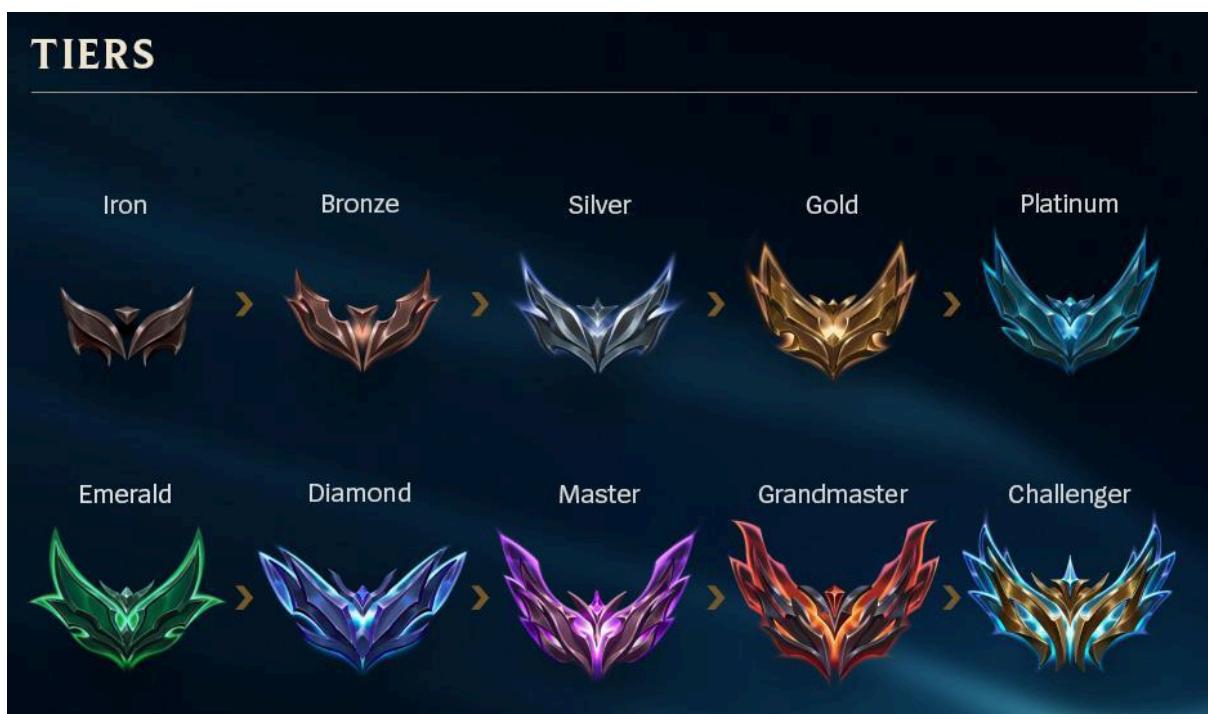
Ce jeu est reconnu pour sa richesse tactique, sa diversité de personnages (plus de 150 champions), et son évolution constante à travers des mises à jour régulières(patch et ajout de champions).

2.2 Le mode Ranked : un système compétitif structurant

Le mode classé, ou Ranked, est le mode central et le plus populaire du jeu. Il permet aux joueurs de se mesurer les uns aux autres dans un système de ligues hiérarchisées, constituant le cœur de l'expérience compétitive.

- Les joueurs commencent à un rang initial (Iron, Bronze, Silver, Gold, Platinum, Diamond, Master, Grandmaster, Challenger).
- Ils progressent ou régressent selon leurs performances dans les parties classées.
- Le rang "Master" (référencé dans le nom de notre application RushMaster) représente les meilleurs joueurs, environ 0,3% du top, ce qui souligne la notion d'excellence et d'expertise.

TIERS



2.3 League of Legends, un esport majeur

Au-delà du simple jeu vidéo, LoL s'est imposé comme un pilier de l'esport mondial, avec des compétitions majeures comme les Championnats du Monde (Worlds) qui attirent des millions de spectateurs. Les équipes professionnelles s'appuient sur des coachs, analystes, et "petits entraîneurs" pour préparer leurs stratégies, analyser les adversaires, et améliorer leurs performances.

Pour Riot Games c'est très souvent leurs modèles, leurs formats économiques qui sont regardé et dont s'inspire la scène esport. C'est donc un jeu très actif en termes de nombres joueurs mais également sur tout ce qui à autour comme les créateur de contenu, la communauté, les événements ou encore les compétitions.

Dans ce contexte, les outils d'analyse de données de jeu sont essentiels, aussi bien pour les pros que pour les amateurs qui souhaitent progresser. Ces outils

permettent de décortiquer les performances individuelles et collectives, identifier les forces et faiblesses, et ajuster les stratégies.

2.4 Analyse des sites concurrents

Dans le cadre du développement de cette application, une analyse approfondie des principaux sites et plateformes existants dans le domaine de l'accompagnement des joueurs de League of Legends a été réalisée. La majorité de ces sites proposent des fonctionnalités visant à améliorer les performances des joueurs à travers des statistiques détaillées, des guides personnalisés, et des outils d'analyse de parties.

Principaux sites concurrents :

- OP.GG

Site très populaire qui fournit des statistiques détaillées sur les joueurs, des analyses de parties, des builds recommandés et un suivi en temps réel. Le modèle économique repose principalement sur la publicité.

- U.GG

Plateforme proposant des builds optimisés, des guides stratégiques et des données de performance à jour, avec un accès gratuit et un modèle publicitaire.

- Mobalytics

Site et application payante en partie (freemium), offrant des analyses avancées de gameplay, un score de performance personnalisé, et des conseils pour améliorer son jeu.

- Porofessor

Outil intégré avec le client de League of Legends, fournissant des statistiques en temps réel, des conseils de builds, et un suivi de progression. Offre une version gratuite et une version premium.

- Blitz.gg

Application qui analyse les parties en temps réel, propose des recommandations de builds et de stratégies, avec un modèle freemium.

- League of Graphs

Site spécialisé dans les statistiques avancées et le suivi des performances des joueurs, totalement gratuit avec financement par la publicité.

Fonctionnalités courantes observées :

Analyse détaillée des parties jouées avec des statistiques de performance (KDA, objectifs pris, farm, etc.)

- Recommandations de builds et runes adaptées à chaque champion
- Suivi des progrès et comparaisons entre joueurs
- Conseils stratégiques et guides d'experts
- Intégration d'API Riot Games pour récupérer les données de jeu en temps réel

Formats économiques majoritairement utilisés :

- Modèle freemium : accès gratuit aux fonctionnalités de base, avec une version payante offrant des analyses plus poussées ou des contenus exclusifs
- Publicités intégrées sur les versions gratuites

- Partenariats ou affiliations avec des marques d'équipement gaming ou d'autres services esports

Notre objectif a pour longtemps de simplement implémenter la plus grande partie de ces fonctionnalités, par curiosité, pour comprendre comment les services Riot Games fonctionnait mais aussi avoir une base de réflexion sur ce que nous pouvions apporter de plus.

2.5 Utilisateurs cible : les joueurs et petits entraîneurs amateurs

RushMaster vise principalement les joueurs de League of Legends et les petits entraîneurs, souvent amateurs, qui n'ont pas accès aux outils sophistiqués des équipes professionnelles. L'application fournit des statistiques détaillées et accessibles pour aider ces utilisateurs à mieux comprendre leurs performances et optimiser leur progression dans le mode Ranked.

Même si notre site se concentre particulièrement sur des statistiques avancées, il permet également aux utilisateurs de simplement consulter à tout moment l'historique complet de leurs parties jouées sans avoir à lancer le jeu sur son ordinateur.

Si vous souhaitez en apprendre encore plus sur League of Legends, l'écosystème et les règles du jeu je vous invite à consulter l'annexe: "League of Legends c'est quoi ?"

3. Expression des besoins

3.1 Contexte et objectifs

Nous avons donc commencé RushMaster en décembre 2024 de manière autonome, puis lorsque l'école nous a fait part de l'idée d'un projet de fin d'année nous avons vu ça comme l'opportunité d'investir du temps dans ce projet.

En février, nous avons donc formalisé le projet pour que ça devienne notre projet de fin bachelor. RushMaster était donc pour nous un challenge pour découvrir et apprendre le react ainsi que mieux se familiariser avec des API professionnelles.

RushMaster est conçu comme une alternative aux outils d'analyse de League of Legends existants, avec une approche personnalisée centrée sur le mode Ranked. L'application a pour but d'aider les joueurs et pourquoi pas, des petits entraîneurs à mieux comprendre leurs performances grâce à des fonctionnalités claires et adaptées, favorisant ainsi leur progression dans le jeu.

3.2 Besoins fonctionnels

- Consultation des statistiques individuelles des champions
Afficher des indicateurs clés tels que le KDA (Kill/Death/Assist), le taux de victoire, le nombre de parties jouées, etc., pour chaque champion utilisé.
- Accès à l'historique des parties
Permettre à l'utilisateur de consulter à tout moment la liste de ses parties récentes, avec des détails comme la date, le rôle joué, les performances, et les résultats.
- Navigation fluide et ergonomique
Offrir une interface intuitive avec une navigation facile entre les différentes pages et sections, adaptée aussi bien aux novices qu'aux utilisateurs expérimentés.
- Intégration avec l'API Riot Games
Utiliser les services fournis par Riot Games pour récupérer des données actualisées et fiables sur les joueurs, champions, et parties.
- Intégration du système d'authentification de Riot Games

Un service fourni par Riot Games si l'on possède une clé production, le système d'authentification OAuth 2.0.

3.3 Besoins non fonctionnels

- Performance

Garantir un temps de réponse rapide, même avec un volume important de données à traiter, tant pour l'affichage que pour le calcul de nos données.

- Portabilité et maintenabilité

Concevoir une architecture modulaire facilitant les évolutions futures et le déploiement sur différents environnements.

- Accessibilité

Veiller à ce que l'application soit utilisable par un large public et sur un maximum de devices.

4. Gestion du projet

4.1 Organisation et méthodologie

Le projet RushMaster a été développé en binôme sur une durée de six mois. La gestion du projet s'est basée sur une organisation flexible, centrée sur une communication constante entre les deux développeurs.

La rigueur a été portée sur la gestion du code et la documentation, avec notamment :

- Des conventions claires pour le nommage et la gestion des branches Git
- Une documentation précise des processus de développement et d'intégration
- Une organisation structurée du dépôt Git pour faciliter la collaboration

Nous nous sommes donc assuré de n'avoir aucun problèmes majeurs lié au versionning ou au convention de nommage.

De manière générale l'évaluation des features que nous voulions implementer c'est vraiment faite au fur et à mesure, car nous ne savions vraiment pas les problématique que nous pouvions rencontré pour l'implémentation mais aussi pour les performances, et ce qu'elles viennent de la clés API ou de notre computing des statistiques.

4.2 Outils utilisés

Gestion de version : Git (GitHub)

Gestion de projet :

- Trello (tableaux Kanban pour le suivi global)
- Notion (documentation et organisation des idées)
- GitHub Issues (gestion des tickets et bugs)
- Communication : Discord, mails, messages

Conception graphique : Figma pour les maquettes et le logo

IDE :

- IntelliJ IDEA pour le backend Java et le frontend React
- Visual Studio Code pour certains Proof of Concept en Python
- Xcode pour le développement Swift mobile prévu

Conteneurisation : Docker et Docker Compose

Hébergement : VPS Hostinger KVM4 (16 Go RAM, 200 Go stockage)

4.3 Maquettting

Comme base de travail nous avons avant le départ du projet fourni un wireframe desktop et mobile de ce à quoi pourrait ressembler notre site internet.

Au vu de la croissance du projet, le wireframe ne comporte que les pages principales sur lesquelles nous voulions mettre l'accent. Aujourd'hui le site est dans une phase plus avancée que ce que vous pouvez voir ci dessous.



Nous avons également travaillé sur une charte graphique pour pouvoir visualiser rapidement et facilement plusieurs palette de couleurs, plusieurs polices d'écritures.

Cette dernière à également était faite sous figma et comporte un thème clair et un thème foncé que l'on retrouve sur le site web

GraphChart (Light Theme)

Typography
Styles for headings, paragraphs, lists...etc

h1
Taxing Laughter: The Joke Tax Chronicles

h2
The People of the Kingdom

h3
The Joke Tax

h4
People stopped telling jokes

p
The king, seeing how much happier his subjects were, realized the error of his ways and repealed the joke tax.

blockquote
"After all," he said, "everyone enjoys a good joke, so it's only fair that they should pay for the privilege."

table

King's Treasury	People's happiness
Empty	Overfowing
Middle	Satisfied
Empty	Overfowing

list

- 1st level of puns: 5 gold coins
- 2nd level of jokes: 10 gold coins
- 3rd level of one-liners : 20 gold coins

inline code

```
function doSomething() {
```

lead

A modal dialog that interrupts the user with important content and expects a response.

large

Are you sure absolutely sure?

small

Email address

subtle

Enter your email address.

GraphChart (Dark Theme)

Typography
Styles for headings, paragraphs, lists...etc

h1
Taxing Laughter: The Joke Tax Chronicles

h2
The People of the Kingdom

h3
The Joke Tax

h4
People stopped telling jokes

p
The king, seeing how much happier his subjects were, realized the error of his ways and repealed the joke tax.

blockquote
"After all," he said, "everyone enjoys a good joke, so it's only fair that they should pay for the privilege."

table

King's Treasury	People's happiness
Empty	Overfowing
Middle	Satisfied
Empty	Overfowing

list

- 1st level of puns: 5 gold coins
- 2nd level of jokes: 10 gold coins
- 3rd level of one-liners : 20 gold coins

inline code

```
function doSomething() {
```

lead

A modal dialog that interrupts the user with important content and expects a response.

large

Are you sure absolutely sure?

small

Email address

subtle

Enter your email address.

5. Fonctionnalités de l'application

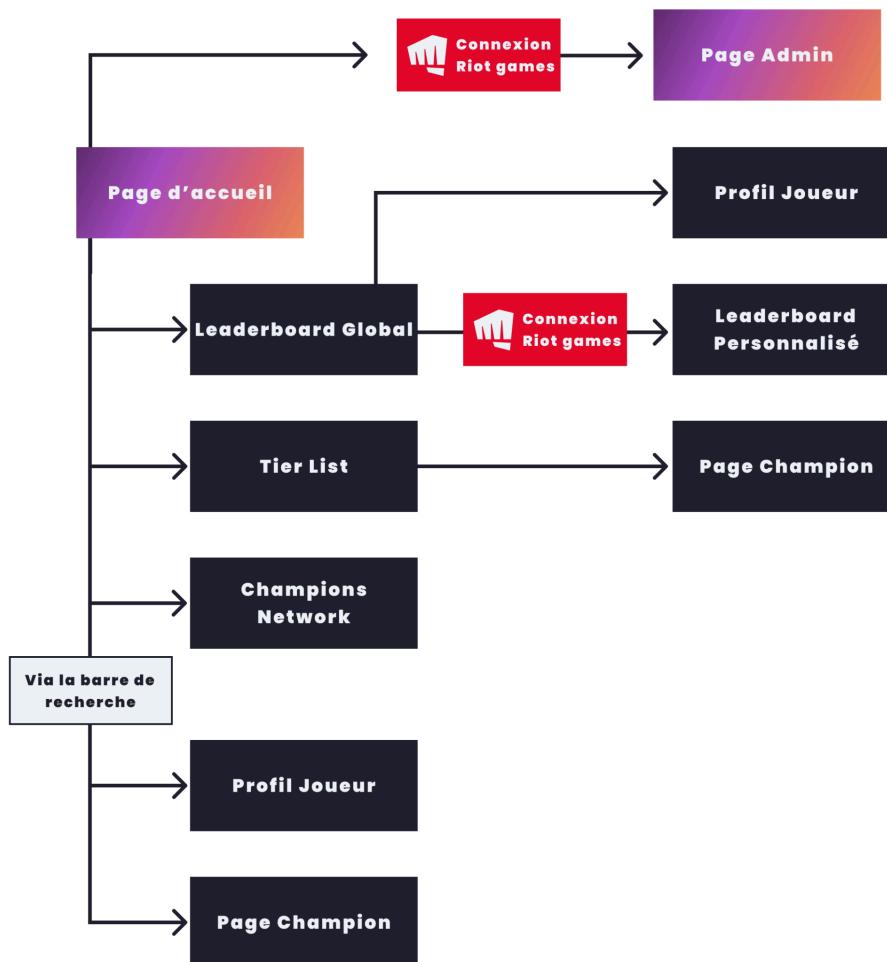
L'application RushMaster propose une interface riche et intuitive permettant de consulter, comparer et explorer les statistiques de joueurs et de champions sur League of Legends. Le site s'adresse à tous les profils de joueurs : du joueur occasionnel au compétiteur souhaitant optimiser ses performances.

Il est important de noté que l'intégralité des données sur lesquelles nous nous basons sont mise à disposition par l'API Riot Games et sont publiques, nous nous détachons donc de tous processus de protection de donnée

5.1 Présentation des principales pages de l'application

L'interface de RushMaster est découpée en plusieurs pages dédiées, accessibles via la barre de navigation ou par une barre de recherche pour les pages "champions" et "joueurs".

Pour que vous compreniez au mieux l'utilisation et l'agencement du site je vais vous détailler son organisation ainsi que les fonctionnalité présentes sur chaque pages



Tout au long du projet nous avions le souhait d'avoir un minimum de points de friction dans le parcours utilisateur (Pas de coupure dans sa traversée du site). C'est pourquoi nous n'avons à aucun moment la nécessité de créer un compte ou se connecter sur le site pour en profiter.

Les 2 seules petites parties réservée sont, le leaderboard personnel ainsi que la page admin disponible on compte éligible (choisi par nous même)

Page d'accueil

Objectif : Introduction à l'application et accès rapide aux fonctionnalités principales.

Fonctionnalités :

- Présentation succincte de l'application.
- Barre de recherche multifonction :
 - Recherche d'un joueur via pseudo Riot + tag.
 - Recherche d'un champion par nom.
- Accès rapide aux sections clés du site :
 - Leaderboard
 - Tier list

- Autres pages principales

Page Leaderboard

Objectif : Afficher le classement global des joueurs.

Fonctionnalités :

- Classement par Elo.
- Possibilité de consulter le profil d'un joueur en cliquant sur son pseudo.
- Si l'utilisateur est connecté :
 - Accès à un leaderboard personnalisé basé sur sa liste d'amis Riot (pour des comparaisons plus ciblées).

Page Tier List

Objectif : Évaluer la performance actuelle des champions.

Fonctionnalités :

- Affichage du classement des champions par efficacité.
- Filtres dynamiques :
 - Patch du jeu
 - Elo (niveau de classement)
- Permet d'identifier les champions méta ou ceux à éviter.

Page Champion Network

Objectif : Visualiser les relations entre champions en jeu.

Fonctionnalités :

- Graphique interactif (type graphe / nœuds).
- Deux types d'affichage 2D et 3D.
- Analyse des synergies et compositions fréquentes.

Page Profil Joueur

Objectif : Afficher les données de jeu d'un joueur.

Fonctionnalités :

- Informations de base : pseudo, rang.
- Statistiques par champion récemment joué.
- Historique de parties interactif :
 - Affichage des 10 joueurs de chaque partie.
 - Mini-carte dynamique des événements clés.
 - Vue détaillée des builds (objets, sorts, runes...).
- Accessible à tous les visiteurs, même non connectés.

Page Champion

Objectif : Présenter en détail un champion.

Fonctionnalités :

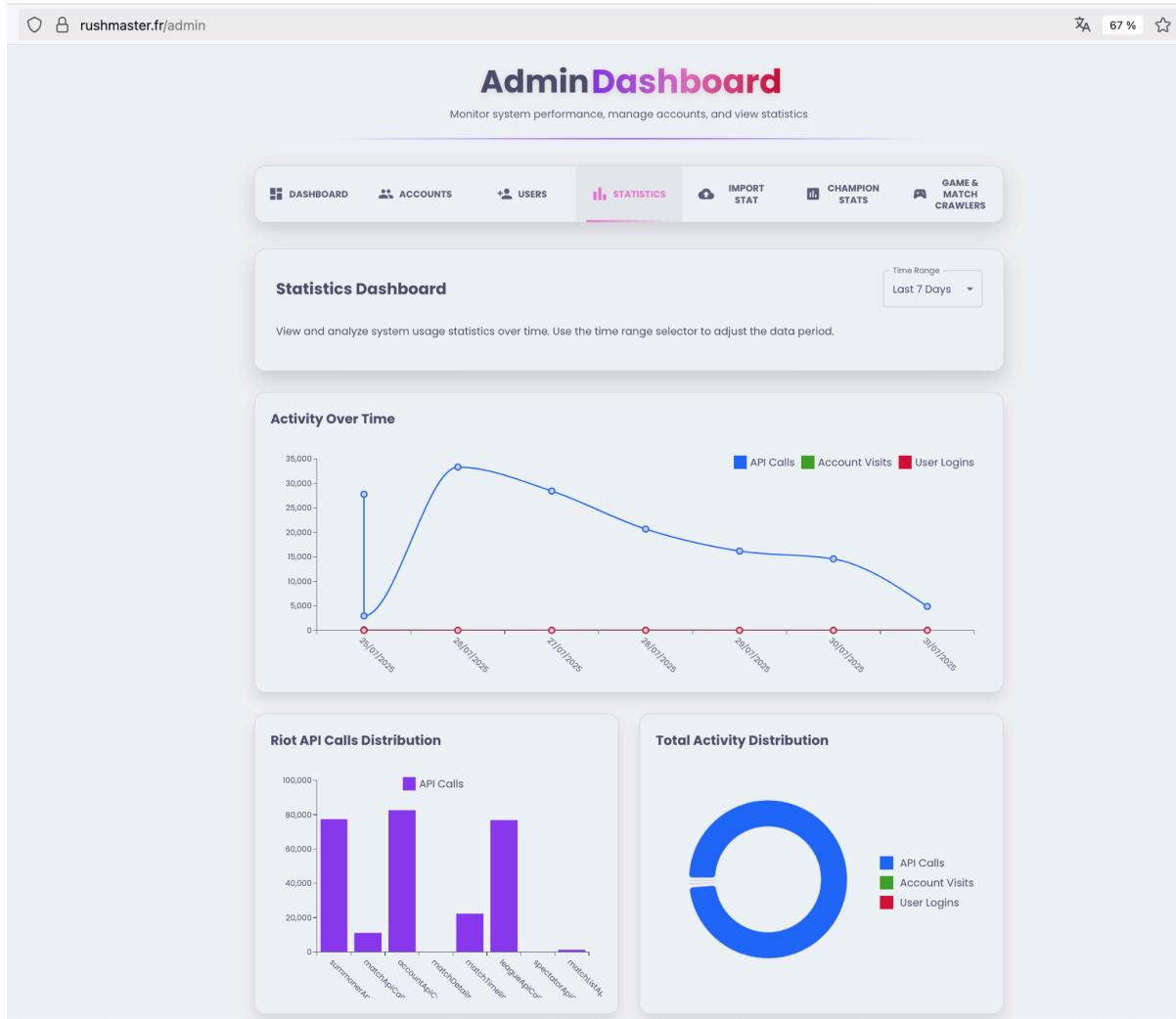
- Statistiques : winrate, pickrate, banrate.
- Tier actuel selon les données.

- Builds les plus utilisés :
 - Runes
 - Objets
 - Ordre des compétences
 - Sorts d'invocateur

5.2 Fonctionnalité Admin

[Connecté uniquement et Admin] Page Admin

Objectif : Avoir des informations importantes affichées, monitorer notre base de données avec l'ajout et la suppression de joueurs par différentes méthodes.



Le Dashboard admin comporte 7 pages donc je nous vous présenterez pas 7 captures d'écrans de celle-ci mais ci dessous vous trouverez le détails des utilité de celles-ci.

Fonctionnalités :

- Dashboard : Vue d'ensemble des performances et des statistiques du système.
- Accounts : Gestion des comptes administrateurs.
- Users : Gestion des utilisateurs (ajout, modification, suppression).

- Statistiques : Graphiques permettant d'observer l'usage de notre clé API en fonction du temps.
- Import Stat : Peuple la base de données de joueurs via leur classement dans le jeu.
- Champion Stats : Permet de manuellement déclenché le calcul des "Tier list" de champions.
- Game Crawler : Autre solution pour peupler la base de joueurs et de parties, en naviguant de parties en parties qui ont été disputées récemment.

En plus de ces 7 pages, le site comporte une page de contact : "Contact Us" qui permet aux utilisateurs de nous adresser un message de feedback ou une remarque sur le site.

Initialement nous avions fait ceci par mail, mais nous ne trouvions pas ça idéal car dur d'en être notifié de plus d'avoir la contrainte de partager une adresse mail avec mon binôme.

De ce constat nous avons décidé de faire un Bot discord responsable de ces messages.

Have a question, suggestion, or just want to say hello? Fill out the form below and we'll get back to you as soon as possible.

Name*
Clément

Email*
clement.ramos83@gmail.com

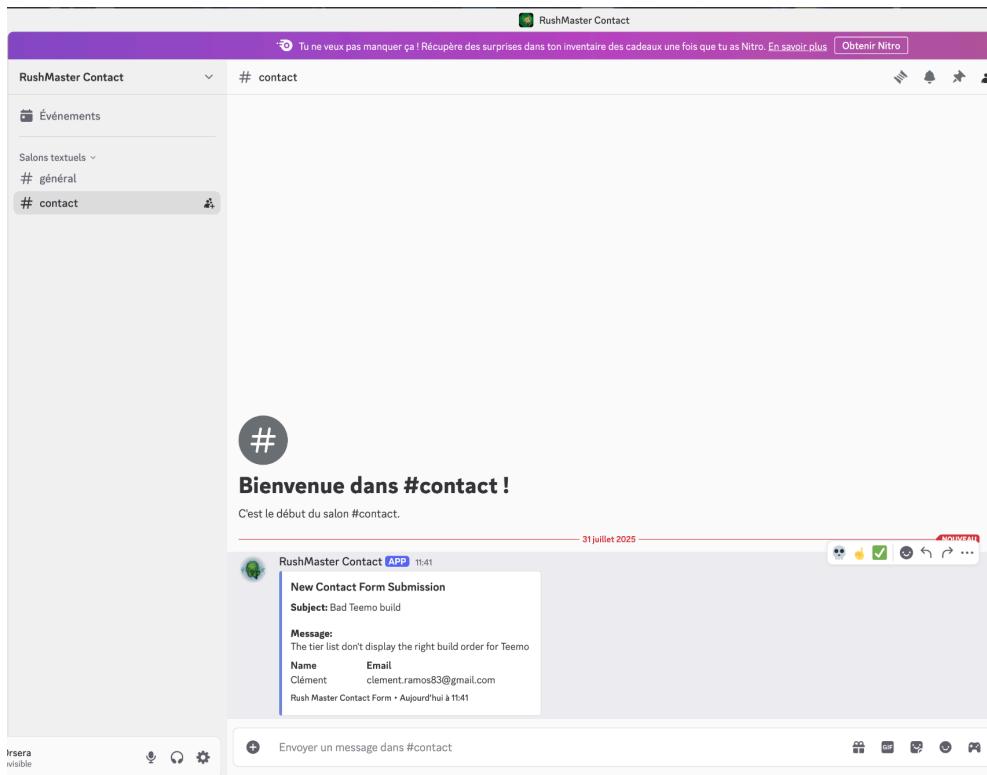
Subject*
Bad Teemo build

Message*
The tier list don't display the right build order for Teemo.

SEND MESSAGE ➤

De cette manière une fois le messages envoyés nous sommes tous les 2 notifiée sans avoir les problèmes que nous avions ci-dessus avec le mail.

Sur discord nous avons donc:

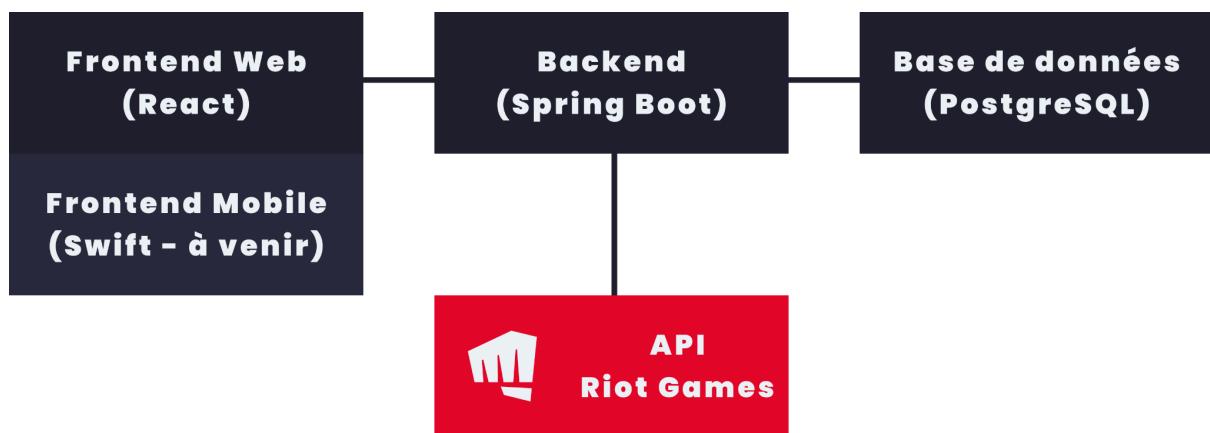


6. Environnement technique

6.1 Architecture générale

RushMaster est une application web full-stack composée de deux modules principaux :

- Frontend : développé avec React, utilisant JavaScript, Vite pour le build, et Material-UI pour la gestion de l'interface utilisateur et du thème (dont le mode sombre).
- Backend : développé avec Java et Spring Boot, fournissant la logique métier et des API REST sécurisées pour communiquer avec le frontend.
- Base de données : PostgreSQL, utilisée pour stocker les données utilisateurs, les statistiques, et l'historique des parties.



Les services sont conteneurisés avec Docker et orchestrés via Docker Compose pour simplifier le déploiement et la gestion des environnements.

6.2 Technologies utilisées

Frontend :

- JavaScript
- React
- Vite (build tool)
- Material-UI (bibliothèque UI)
- PropTypes (validation des props)

Backend :

- Java
- Spring Boot
- Gradle (gestion des dépendances et build)
- Spring Security (authentification et encodage des mots de passe)
- JUnit 5 et Mockito (tests unitaires et d'intégration)

Base de données :

- PostgreSQL
- Scripts SQL pour l'initialisation

Infrastructure :

- Docker et Docker Compose

- Réseau Docker bridge pour connecter les services

6.3 Fonctionnalités techniques principales

Frontend :

- Affichage dynamique des statistiques des champions et des parties
- Navigation fluide entre les différentes pages
- Mode sombre via Material-UI
- Validation des types avec PropTypes

Backend :

- Gestion sécurisée des utilisateurs (authentification, encodage des mots de passe)
- Exposition d'API REST pour communication avec le frontend
- Intégration avec l'API Riot Games pour récupérer les données actualisées
- Gestion des variables d'environnement pour sécuriser les clés API et données sensibles

Base de données :

- Stockage et persistance des utilisateurs, statistiques, et historiques de parties

6.4 Configuration des environnements

La mise en place d'un environnement de développement robuste et reproductible était une priorité pour assurer la qualité et la maintenabilité du projet RushMaster. J'ai opté pour une approche basée sur la conteneurisation, qui présente plusieurs avantages.

- Isolation des composants
- Reproductibilité de l'environnement
- Facilité de déploiement
- Gestion simplifiée des dépendances

J'ai structuré l'environnement de développement autour de trois composants principaux, chacun isolé dans son propre conteneur Docker :

Frontend : Conteneur Node.js hébergeant l'application React

Backend : Conteneur Java exécutant l'application Spring Boot

Base de données : Conteneur PostgreSQL pour le stockage des données

L'orchestration de ces conteneurs est gérée via Docker Compose, permettant de démarrer l'ensemble de l'application avec une seule commande. Cette approche présente l'avantage de garantir que tous les développeurs travaillent dans un environnement strictement identique, éliminant ainsi les problèmes classiques de "ça marche sur ma machine"

Intégration de l'API Riot Games

Le backend communique avec l'API officielle de Riot Games pour récupérer :

- Les informations des joueurs (pseudo, rang, MMR...)
- Les historiques de match
- Les statistiques globales sur les champions
- Les données de métadonnées (patchs, sorts, runes...)

Au début du projet, nous disposions d'une clé de développement limitée en nombre de requêtes. Par la suite, nous avons obtenu une clé de production permettant un volume plus important de requêtes. Cette clé de production nous a également permis de demander et d'obtenir l'accès à l'OAuth2 de Riot Games, facilitant ainsi l'authentification sécurisée des utilisateurs via leur compte Riot.

6.5 Tests automatisés

Backend :

- Tests unitaires avec JUnit 5 et Mockito
- Tests d'intégration des composants backend
- Tests API REST pour valider les endpoints

Frontend :

- Tests unitaires et d'intégration avec React Testing Library et Jest
- Tests de rendu et interaction utilisateur

Ces tests garantissent la qualité, la robustesse et facilitent la maintenance du projet.

6.6 Hébergement

L'application est hébergée chez Hostinger sous l'abonnement KVM4, offrant les caractéristiques suivantes :

- Type d'hébergement : VPS (serveur privé virtuel)
- Ressources allouées :

- 4 vCPU
- 16 Go de RAM
- 200 Go d'espace disque SSD NVMe
- Bande passante illimitée
- Système d'exploitation : Linux (Ubuntu recommandé)
- Accès root complet pour gérer l'environnement serveur
- Support : assistance 24/7 et infrastructure redondante

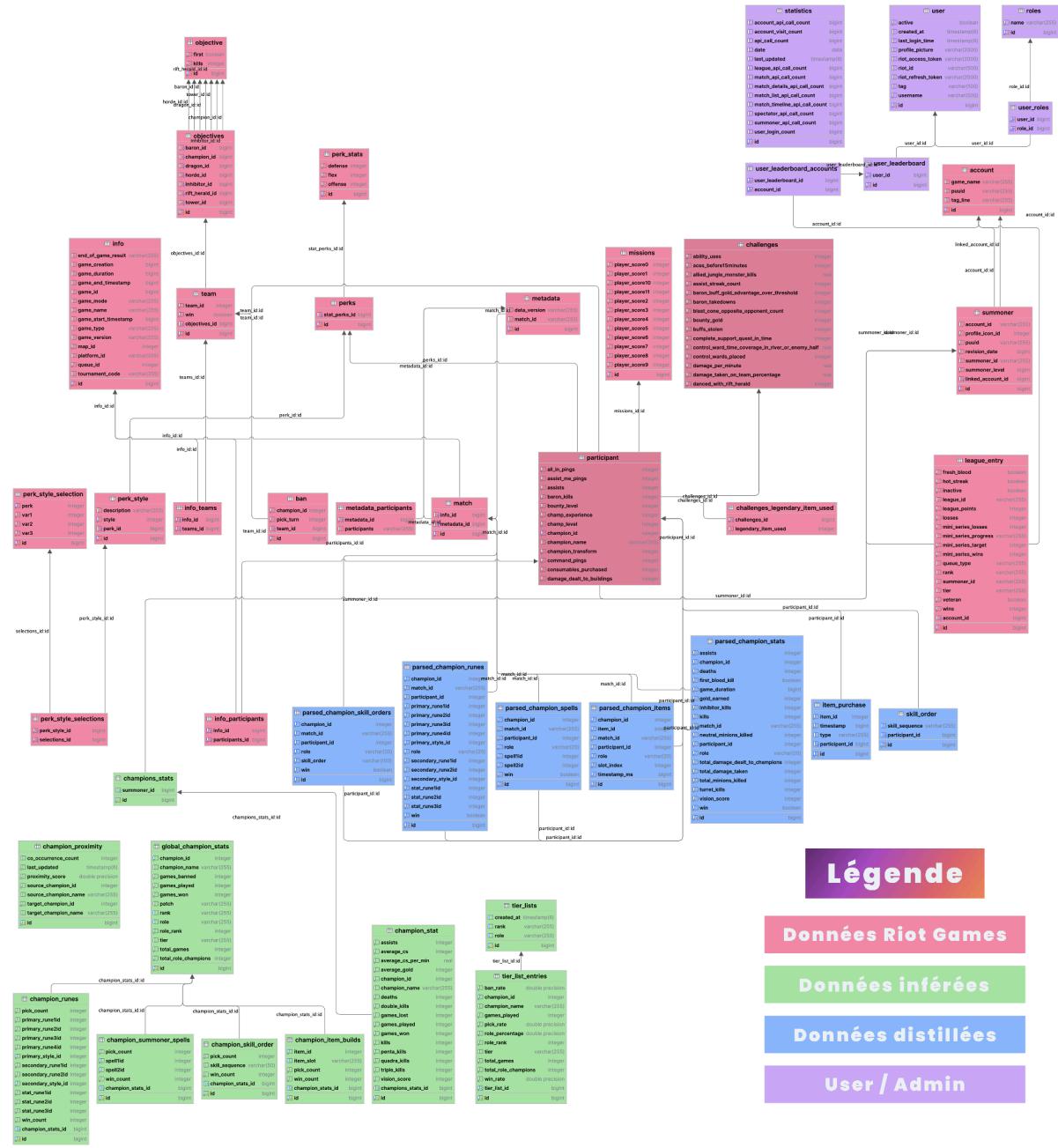
Cet hébergement assure une bonne performance pour la gestion des bases de données, des API et des accès simultanés utilisateurs.

The screenshot shows the Hostinger VPS overview page. On the left, there's a sidebar with links like 'Menu principal', 'Aperçu', 'Kodee', 'Paramètres', 'Système d'exploitation et panel', 'Sauvegardes et surveillance', 'Sécurité', 'API', 'Gestionnaire DNS', and 'Tutoriels'. The main content area has a header 'Aperçu général' and a 'Terminal de navigateur'. It displays various metrics: CPU usage (1%), Memory usage (16%), Disk usage (6 GB / 200 GB), Traffic (2.3 MB in, 0.0 MB out), and other stats like SSH keys (1), Firewall rules (0), and snapshots (2). A purple box encourages users to switch to automatic backups. At the bottom, there are sections for 'Informations du VPS' (Server location: Germany - Frankfurt, OS: Ubuntu 24.04 LTS, Hostname: orseraeclipse.com, Uptime: 36 days 17 hours) and 'Informations du plan' (Plan: KVM 4, Renewal: 2026-06-13, Active: Yes).

6.7 Base de données

Vous trouverez donc ci dessous la base de données qui est assez fournie dû à plusieurs contraintes et choix techniques qui ont été fait. Il est important de noter que dans le schéma NOM DU SCHEMA 2 tables ont été réduite pour gagner en lisibilité mais les tables challenges et participant comportent respectivement NOMBRE DE COLLONNES.

Le diagramme est normalement en bonne qualité et vous permet de zoomer pour le consulter.



Sans rentrer dans le détail de chaque table ce qu'il est important de relever ce sont les différents cas d'usage des tables que j'ai mis en évidence via ce diagramme et sa légende.

La plupart des tables sont rouges et sont des données que l'on obtient tel quel via l'API Riot Games, seulement nous les mettons en base de données pour éviter des requêtes inutiles vers l'API et également améliorer les performances de notre site.

Pour les données vertes, ce sont des données qui sont parlantes statistiquement et au travers du jeu, mais qui ne sont pas directement fournies par l'API. Nous les calculons donc via les autres données.

Un exemple parlant est le KDA (Ratio entre Kills/Deaths/Assists) qui est souvent un bon indicateur sur la dynamique de vos parties. Nous possédons les 3 données pour le calculer mais pas directement le KDA.

Pour les données bleues, ces données sont apparues vers les dernières semaines du projet et ont 2 fonctions.

Améliorer les performances du site.

Conserver la cohérence de nos statistiques même si l'on supprime de parties datées.

Pour les données violettes elles sont lié au différentes pages admin ainsi qu'à la gestions des rôles que l'on fait par dessus la gestion de compte fournies par Riot (OAuth 2.0)

7. Conclusion et perspectives

je pensais une conclusion en 3 parties:

- Difficulté rencontrées
- Conclusion
- perspectives d'améliorations

Difficultés rencontrées et solutions apportées

- Limitation initiale de la clé API Riot Games, remplacée par une clé production avec accès OAuth2 pour augmenter les quotas de requêtes
- Gestion complexe des états sur le frontend, solutionnée par l'adoption progressive de Redux
- Complexité de l'implémentation OAuth2, avec des ajustements des flux d'authentification pour garantir la sécurité et la stabilité

- Optimisation des requêtes SQL pour améliorer les performances de la base de données

Le projet RushMaster a représenté bien plus qu'un simple exercice de développement : il a été l'occasion de mobiliser l'ensemble des compétences acquises tout au long de la formation de Concepteur Développeur d'Applications, à travers un cas concret, ambitieux et motivant.

D'un point de vue technique, le projet nous a permis de travailler sur une architecture complète en full-stack, en intégrant à la fois la gestion de données via des API externes (Riot Games), le traitement côté serveur avec Spring Boot, et le rendu dynamique d'interfaces réactives avec React. L'aspect DevOps a également été pleinement intégré grâce à la conteneurisation via Docker et à l'organisation d'une CI claire et structurée.

Nous pensons avoir fait de bons compromis dans la répartition de notre temps et de nos efforts : en nous appuyant sur nos acquis solides (Java, SQL, développement web), nous avons pu concentrer notre énergie sur ce qui comptait le plus pour nous dans ce projet :

- proposer une application réellement fonctionnelle et finalisée,
- explorer deux nouvelles technologies front-end (React pour le web, Swift en cours pour mobile),
- intégrer un grand nombre de fonctionnalités concrètes pour l'utilisateur,
- et maintenir une organisation rigoureuse du projet, avec une gestion de version, de tickets et des tâches bien cadrée.

Ce que nous avons appris

- La nécessité de concevoir une application orientée utilisateur, avec une logique de navigation claire et des interfaces accessibles.

- L'importance d'isoler les responsabilités entre couches applicatives, de documenter son code et de prévoir des tests automatisés dès les premières étapes du projet.
- L'apprentissage concret de la gestion d'API tierces, de leurs limites (quotas, OAuth2, structure des données), et de leur intégration dans notre architecture.

Pistes d'évolution

- Trouver des solutions de stockage et de computing plus puissantes pour gérer un plus grand nombre de joueurs ou accélérer leur insertion et traitement dans la base de données.
- (À compléter : autres pistes techniques ou fonctionnelles prévues)

Annexes

Terme / Acronyme – Fonctionnel

Acronyme	Définition
ELO	Système de classement utilisé dans League of Legends pour déterminer le niveau d'un joueur.
KDA	Kills / Deaths / Assists – Statistique utilisée pour évaluer la performance d'un joueur dans LoL.
LoL	League of Legends – Jeu vidéo de type MOBA développé par Riot Games.
LPs	League Points – Points utilisés pour classer un joueur dans un rang spécifique dans le mode classé.
MMR	Matchmaking Rating – Score caché utilisé pour équilibrer les parties en classé.
OP.GG	Plateforme concurrente analysant les statistiques des joueurs de League of Legends.
RushMaster	Nom de l'application développée. "Master" fait référence au rang élevé dans LoL.
Skill Order	Ordre dans lequel un joueur monte ses compétences lors d'une partie.
Summoner Spells	Sorts d'invocateur – capacités spécifiques choisies en début de partie par un joueur.
Tag Riot	Identifiant unique d'un joueur de LoL (nom + # + ID serveur).
Tier List	Classement hiérarchisé des champions selon leur performance ou popularité.
Winrate	Pourcentage de parties gagnées par un joueur ou un champion.

Terme / Acronyme – Technique

Acronyme	Définition
API	Application Programming Interface – Interface de programmation applicative permettant à deux systèmes de communiquer.
CI/CD	Continuous Integration / Continuous Deployment – Méthodologies visant à automatiser les tests, l'intégration et le déploiement du code.
CRUD	Create, Read, Update, Delete – Ensemble des opérations de base sur une base de données.
DevOps	Ensemble de pratiques qui unifient le développement logiciel (Dev) et les opérations informatiques (Ops).
Docker	Plateforme de conteneurisation permettant d'isoler et de déployer des applications dans des environnements légers.
Docker Compose	Outil permettant de définir et de gérer des applications multi-conteneurs Docker via un fichier de configuration YAML.
IDE	Integrated Development Environment – Environnement de développement intégré (ex : IntelliJ, VS Code, Xcode).
IntelliJ	IDE Java développée par JetBrains, très utilisé pour les projets Spring Boot.
Jest	Framework de tests JavaScript utilisé notamment avec React.
JUnit	Framework de test unitaire pour les applications Java.
Material UI (MUI)	Librairie de composants React inspirée du design system de Google (Material Design).
Node (diagramme)	Dans le contexte des graphes, un "nœud" représente une entité connectée à d'autres entités.
OAuth2	Protocole d'autorisation standardisé pour l'accès sécurisé aux API.

POC	Proof of Concept – Prototype rapide servant à tester une idée ou fonctionnalité.
PostgreSQL	Système de gestion de base de données relationnelle open-source.
REST	Representational State Transfer – Architecture standard pour la conception d'API.
React	Librairie JavaScript utilisée pour créer des interfaces utilisateur dynamiques.
Riot Games	Éditeur du jeu League of Legends et fournisseur des API utilisées dans le projet.
RNCP	Répertoire National des Certifications Professionnelles – Référentiel officiel français des certifications professionnelles.
SGBD	Système de Gestion de Base de Données.
Spring Boot	Framework Java permettant de créer des applications web robustes de façon rapide.
Spring Security	Module de sécurité pour les applications Spring (authentification, autorisation).
Swift	Langage de programmation utilisé pour développer des applications iOS (mobile).
Vite	Outil de build rapide pour projets front-end (utilisé avec React).
VS Code	IDE léger pour divers langages de programmation, notamment JavaScript et Python.
Xcode	IDE officiel pour le développement d'applications Apple (macOS, iOS).

League of Legends c'est quoi ?

Concept de Base

- Type de Jeu : League of Legends (LoL) est un jeu vidéo de stratégie en équipe où deux équipes, composées chacune de cinq joueurs, s'affrontent. Chaque joueur contrôle un personnage unique, appelé "champion", doté de compétences spécifiques.
- Objectif : L'objectif principal est de détruire la base adverse, appelée le Nexus, située à l'extrême opposée de la carte de jeu. Pour y parvenir, les joueurs doivent s'affronter et explorer la carte afin de collecter des ressources, ce qui leur permet de renforcer leurs champions pour finalement détruire le Nexus adverse.

Gameplay

- Carte : Le jeu se déroule sur une carte unique divisée en trois voies principales. Les joueurs doivent naviguer sur cette carte pour atteindre et attaquer la base ennemie.

Rôles : Chaque équipe est composée de cinq rôles distincts, chacun ayant ses spécificités :

- Toplaner : Positionné sur la voie du haut, ce rôle met l'accent sur les confrontations directes avec l'adversaire.
- Midlaner : Situé sur la voie centrale, ce rôle est crucial tant en termes de communication que de positionnement stratégique sur la carte.
- Botlaner/ADC : Positionné sur la voie du bas, ce rôle est axé sur l'inflation de dégâts mais est généralement plus vulnérable.

- Jungler : Ce rôle consiste à se déplacer entre les voies dans une zone appelée "la jungle", intervenant pour soutenir les joueurs sur les voies au moment opportun.
- Support : Joué en binôme avec le Botlaner, ce rôle vise à compenser la vulnérabilité du Botlaner en fournissant une assistance stratégique et tactique.
- Stratégie : La stratégie et la coopération sont essentielles. Les joueurs doivent choisir leurs champions avec soin, planifier leurs attaques et s'adapter aux mouvements de l'équipe adverse.

Tournois et Championnats

- Ligue Professionnelle : La scène compétitive de LoL est très développée, avec des ligues professionnelles dans différentes régions du monde, telles que la LEC (League of Legends European Championship) en Europe et la LCS (League of Legends Championship Series) en Amérique du Nord.
- Championnat du Monde : Chaque année, le Championnat du Monde de League of Legends, souvent appelé Worlds, est organisé. Cet événement majeur rassemble les meilleures équipes du monde entier pour concourir pour le titre de champion du monde et attire des millions de téléspectateurs.

Popularité et Culture

- Audience : LoL est l'un des jeux les plus regardés et possède une énorme communauté de fans à travers le monde. Sa diffusion se fait principalement sur Twitch, un site de streaming, en raison de l'origine de l'e-sport qui provient des gamers, des streamers et des créateurs de contenu. Contrairement au sport traditionnel, la diffusion est entièrement gratuite et ne fournit pas de rémunération aux clubs car il n'y a pas de droits télévisuels.

- Impact Culturel : Le jeu a influencé la culture populaire, avec des références dans la musique, les films, et même des collaborations avec des marques de vêtements et des artistes.

Économie

- Sponsors et Récompenses : Les tournois de LoL offrent des prix importants, et les équipes professionnelles sont souvent sponsorisées par de grandes entreprises. Les joueurs professionnels peuvent gagner leur vie en jouant à LoL, grâce aux salaires, aux sponsors et aux gains des tournois.

Documentations

Ci-dessous vous trouverez les documentations que nous avons mis en place pour le projet celles-ci sont en anglais car nous avions peut être pour objectif de rendre le repository public et open source.

Branch Management Strategy for Rush Master

1. Overview

This document outlines the branch management strategy for the Rush Master project, which uses a Git submodule structure where the main components (frontend, backend, and mobile) are separate repositories included as submodules in the main project.

The Rush Master project follows a structured branching strategy to ensure smooth development and deployment processes across all components. This strategy accounts for the project's submodule architecture and coordinates development across frontend, backend, and mobile applications.

2. Main Repository (Rush_Master)

The main repository containing all submodules adheres to the following rules:

- Always on main branch
The main repository should always remain on the main branch and no other branches.
- Purpose
Serves as an integration point for all submodules and handles deployment configurations.
- Updates
Updated when new versions of submodules are ready to be integrated.

3. Frontend and Backend Submodules

Both frontend and backend follow the same branching strategy:

Primary Development Branch

- dev
 - All ongoing development work happens in the dev branch.
 - Serves as the integration point for feature branches.

Feature Branches

- feature/feature-name
 - Created from the dev branch.
 - Used when multiple developers work on the same component.

- Focused on implementing specific features or fixes.
- Merged back to dev when complete.

Version Releases

- When a new version is ready for release:
 - Merge dev → main.
 - The main branch always represents the production-ready state of the code.

Deployment Automation

1. Overview

This document explains the automated deployment process set up for the Rush Master application.

The GitHub Actions workflow in `.github/workflows/main-ci.yml` is configured to automatically deploy updates to the VPS whenever:

- Changes are pushed to the main branch.
- The workflow is manually triggered.

2. Workflow Steps

1. Validation

- Checks that the Docker Compose configuration files are valid.

2. Deployment

- Connects to the VPS.
 - Navigates to `/opt/rush-master`.
 - Pulls the latest changes from the main repository.
 - Updates all submodules to their latest commits.
 - Rebuilds and restarts the Docker containers.
-

3. Required GitHub Secrets

For deployment to work correctly, add the following secrets to your GitHub repository:

1. VPS_HOST

- The IP address or hostname of your VPS server.

2. VPS_SSH_USERNAME

- The username for SSH access to the VPS (usually root or another user with appropriate permissions).

3. VPS_SSH_PRIVATE_KEY

- The private SSH key for authentication to the VPS.
- Must include the complete key content with BEGIN and END lines.

How to Add Secrets to GitHub Repository

1. Go to your GitHub repository.
2. Click on Settings.
3. In the left sidebar, select Secrets and variables → Actions.
4. Click New repository secret.
5. Add secrets one by one:

VPS_HOST

- Name: VPS_HOST
- Value: IP address or hostname of your VPS (e.g., 91.108.102.201)

VPS_SSH_USERNAME

- Name: VPS_SSH_USERNAME
- Value: SSH username for the VPS

VPS_SSH_PRIVATE_KEY

- Name: VPS_SSH_PRIVATE_KEY
- Value: Copy the entire private key file content (e.g., `~/.ssh/id_ed25519`), including:

Deploying Rush Master on a VPS Server

This guide provides step-by-step instructions for deploying the Rush Master application on a Virtual Private Server (VPS).

The Rush Master project uses a Git submodule structure where the main components (frontend, backend, and mobile) are separate repositories included as submodules in the main project. This deployment guide takes this structure into account and provides specific instructions for handling submodules during deployment and maintenance.

Prerequisites

- A VPS with at least 2GB RAM and 2 CPU cores
- Ubuntu 20.04 LTS or newer (recommended)
- Domain name pointing to your VPS IP address
- Basic knowledge of Linux command line
- SSL certificate for your domain (or ability to generate one)

1. Initial Server Setup

1.1 Update System Packages

```
sudo apt update  
sudo apt upgrade -y
```

1.2 Install Required Dependencies

Install Docker and Docker Compose

```
sudo apt install -y apt-transport-https ca-certificates curl  
software-properties-common  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"  
sudo apt update  
sudo apt install -y docker-ce docker-compose
```

Start and enable Docker service

```
sudo systemctl start docker  
sudo systemctl enable docker
```

Add your user to the docker group (optional, for running docker without sudo)

```
sudo usermod -aG docker $USER
```

Log out and log back in for this to take effect

1.3 Configure Firewall

Install UFW if not already installed

```
sudo apt install -y ufw
```

Allow SSH, HTTP, and HTTPS

```
sudo ufw allow 22/tcp
```

```
sudo ufw allow 80/tcp
```

```
sudo ufw allow 443/tcp
```

Enable the firewall

```
sudo ufw enable
```

2. Clone the Repository

Create a directory for the application

```
mkdir -p /opt/rush-master  
cd /opt/rush-master
```

Clone the main repository (replace with your actual repository URL)
git clone https://github.com/yourusername/rush-master.git .

Initialize and update submodules (frontend, backend, and mobile)
git submodule init
git submodule update

Alternatively, you can clone with submodules in one command
git clone --recurse-submodules
https://github.com/yourusername/rush-master.git .

This project uses Git submodules for the frontend, backend, and mobile components. The above commands ensure all submodules are properly initialized and updated during the cloning process.

3. SSL Certificate Setup

Proper file permissions are critical for SSL certificates and keystore files. The application runs as a non-root user in Docker containers, and if the permissions are too restrictive, the application will fail with `AccessDeniedException`. All certificate files must have at least 644 permissions (readable by all users) to ensure the application can access them.

3.1 Option 1: Using Let's Encrypt (Recommended for Production)

Install Certbot
sudo apt install -y certbot

Obtain certificates (replace rushmaster.fr with your actual domain)

```
sudo certbot certonly --standalone -d rushmaster.fr -d www.rushmaster.fr
```

Copy certificates to the appropriate locations

```
sudo mkdir -p frontend/ssl  
sudo cp /etc/letsencrypt/live/rushmaster.fr/privkey.pem  
frontend/ssl/rushmaster.key  
sudo cp /etc/letsencrypt/live/rushmaster.fr/fullchain.pem  
frontend/ssl/rushmaster.crt
```

Set proper permissions for frontend certificates

```
sudo chmod 644 frontend/ssl/rushmaster.key  
sudo chmod 644 frontend/ssl/rushmaster.crt
```

Generate DH parameters

```
cd frontend/ssl  
openssl dhparam -out dhparam.pem 2048  
chmod 644 dhparam.pem  
cd ../../..
```

Create backend keystore

```
cd backend/ssl  
openssl pkcs12 -export \  
-in ../../frontend/ssl/rushmaster.crt \  
-inkey ../../frontend/ssl/rushmaster.key \  
-out rushmaster.p12 \  
-name rushmaster \  
-password pass:rushmaster
```

Set proper permissions for backend keystore

```
chmod 644 rushmaster.p12  
cd ../../..
```

3.2 Option 2: Using Self-Signed Certificates (For Testing Only)

```
cd frontend/ssl  
chmod +x generate-self-signed-cert.sh  
../generate-self-signed-cert.sh
```

Set proper permissions for frontend certificates
chmod 644 rushmaster.key
chmod 644 rushmaster.crt
chmod 644 dhparam.pem

```
cd ../../backend/ssl  
chmod +x generate-keystore.sh  
../generate-keystore.sh
```

Set proper permissions for backend keystore (in case the script doesn't set it)
chmod 644 rushmaster.p12
cd ..

4. Environment Configuration

Create a ` `.env` file in the root directory with the necessary environment variables:
cat > .env << EOL

Database Configuration

```
DB_URL=jdbc:postgresql://db:5432/rush_master_db  
DB_USERNAME=postgres  
DB_PASSWORD=your_secure_password_here
```

Riot API Configuration

```
RIOT_API_KEY=your_Riot_api_key_here
```

Application Configuration

```
APP_BASE_URL=https://rushmaster.fr  
EOL
```

5. Update Application Configuration

5.1 Update Email Configuration (Optional)

If you want to use your own email service, update the email configuration in `backend/src/main/resources/application.properties`:

Replace with your email configuration

```
sed -i  
's/spring.mail.username=rushmaster.confirmation@gmail.com/spring.mail.usern  
ame=your_email@example.com/'  
backend/src/main/resources/application.properties  
sed -i 's/spring.mail.password=utdj wirb ykay  
ajns/spring.mail.password=your_email_password/'  
backend/src/main/resources/application.properties
```

5.2 Update SSL Password (Recommended)

For better security, update the SSL keystore password:

Generate a new password

```
NEW_PASSWORD=$(openssl rand -base64 12)
```

Update the password in the keystore

```
cd backend/ssl  
openssl pkcs12 -export \  
-in ../../frontend/ssl/rushmaster.crt \  
-inkey ../../frontend/ssl/rushmaster.key \  
-out rushmaster.p12 \  
-name rushmaster \  
-password pass:$NEW_PASSWORD
```

Set proper permissions for backend keystore

```
chmod 644 rushmaster.p12
```

```
cd ../..
```

Update the password in application.properties

```
sed -i
```

```
"s/server.ssl.key-store-password=rushmaster/server.ssl.key-store-password=$NEW_PASSWORD/" backend/src/main/resources/application.properties
```

6. Build and Deploy the Application

Build and start the containers in detached mode

```
docker-compose up -d
```

Check if all containers are running

```
docker-compose ps
```

7. Post-Deployment Verification

7.1 Check Application Status

Check container logs

```
docker-compose logs
```

Check frontend health

```
curl -k https://localhost/health
```

Check backend health

```
curl -k https://localhost:8443/health/heartbeat
```

7.2 Access the Application

Open your browser and navigate to `https://rushmaster.fr`

8. Maintenance and Updates

8.1 Updating the Application

Pull the latest changes from the main repository

```
git pull
```

Update all submodules to their latest commits

```
git submodule update --remote --merge
```

Alternatively, you can update specific submodules if needed

```
git submodule update --remote --merge frontend
```

```
git submodule update --remote --merge backend
```

```
git submodule update --remote --merge mobile
```

Rebuild and restart the containers

```
docker-compose down
```

```
docker-compose up -d
```

Since this project uses Git submodules, it's important to update both the main repository and all submodules to ensure all components are at their latest versions.

8.2 Backing Up the Database

Create a backup directory

```
mkdir -p /opt/rush-master/backups
```

Backup the database

```
docker exec -t rush-master_db_1 pg_dump -U postgres rush_master_db >
/opt/rush-master/backups/rush_master_db_$(date +%Y%m%d).sql
```

8.3 SSL Certificate Renewal (Let's Encrypt)

Let's Encrypt certificates expire after 90 days. There are two ways to handle certificate renewal:

8.3.1 Automatic Renewal with GitHub Actions (Recommended)

The GitHub Actions workflow automatically renews Let's Encrypt certificates during deployment when pushing to the main branch. This is the recommended approach as it ensures certificates are regularly checked and renewed as part of the deployment process.

This is configured in the ` `.github/workflows/main-ci.yml` file and requires no additional setup beyond the initial Let's Encrypt configuration.

Benefits of this approach:

- Certificates are checked for renewal with every deployment
- No need to set up separate cron jobs on the server
- Certificate renewal is part of the deployment process, ensuring the application always has valid certificates
- Any issues with certificate renewal will be visible in the GitHub Actions logs

8.3.2 Manual Renewal with Cron Job (Alternative)

If you prefer to manage certificate renewal separately from deployment, you can set up a cron job on the server:

Test certificate renewal

```
sudo certbot renew --dry-run
```

Add a cron job for automatic renewal

```
echo "0 3 * * * certbot renew --quiet && cp  
/etc/letsencrypt/live/rushmaster.fr/privkey.pem  
/opt/rush-master/frontend/ssl/rushmaster.key && cp  
/etc/letsencrypt/live/rushmaster.fr/fullchain.pem  
/opt/rush-master/frontend/ssl/rushmaster.crt && cd  
/opt/rush-master/backend/ssl && openssl pkcs12 -export -in  
../../frontend/ssl/rushmaster.crt -inkey ../../frontend/ssl/rushmaster.key -out  
rushmaster.p12 -name rushmaster -password pass:$NEW_PASSWORD && cd  
/opt/rush-master && docker-compose restart" | sudo tee -a /etc/crontab
```

9. Troubleshooting

9.1 Container Issues

Check container logs
docker-compose logs frontend
docker-compose logs backend
docker-compose logs db

Restart a specific container
docker-compose restart frontend

9.2 SSL Issues

Check Nginx SSL configuration
docker exec -it rush-master_frontend_1 nginx -t

Check SSL certificate validity
openssl x509 -in frontend/ssl/rushmaster.crt -text -noout

9.3 Database Issues

Connect to the database

```
docker exec -it rush-master_db_1 psql -U postgres -d rush_master_db
```

Check database status

```
docker exec -it rush-master_db_1 pg_isready -U postgres
```

10. Security Considerations

1. Change Default Passwords: Update all default passwords in the application.
2. Regular Updates: Keep your system and application updated.
3. Firewall Configuration: Only expose necessary ports.
4. Database Backups: Regularly backup your database.
5. SSL Certificates: Use valid SSL certificates and renew them before expiration.
6. Environment Variables: Secure your `.env` file and other configuration files.

11. Performance Optimization

For better performance on a VPS:

1. Increase Docker Memory Limits: Adjust Docker memory limits based on your VPS resources.
2. Database Tuning: Optimize PostgreSQL configuration for your workload.
3. Nginx Caching: Enable caching in Nginx for static assets.
4. CDN Integration: Consider using a CDN for static content.

Automatic Deployment with GitHub Actions

Rush Master is configured with a GitHub Actions workflow that automatically deploys the latest version to the VPS whenever changes are pushed to the main branch.

How It Works

1. The workflow is defined in `github/workflows/main-ci.yml`
2. When code is pushed to the main branch, the workflow:
 - Validates the docker-compose configuration
 - Connects to the VPS via SSH
 - Pulls the latest code from the repository
 - Updates all submodules to their latest versions
 - Renews Let's Encrypt SSL certificates (if needed)
 - Updates the application with the renewed certificates
 - Rebuilds and restarts the Docker containers

Required GitHub Secrets

To enable automatic deployment, the following secrets must be configured in the GitHub repository settings:

1. VPS_HOST: The IP address or hostname of your VPS server
2. VPS_SSH_USERNAME: The username for SSH access to the VPS
3. VPS_SSH_PRIVATE_KEY: The private SSH key for connecting to the VPS

Setting Up SSH Keys

1. Generate an SSH key pair on your local machine:
`ssh-keygen -t ed25519 -C "github-actions-deploy"`
2. Add the public key to the authorized_keys file on the VPS:
On your local machine

```
cat ~/.ssh/id_ed25519.pub | ssh root@your_vps_ip_address "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
```

3. Generate the known_hosts entry:

```
ssh-keyscan your_vps_ip_address
```

4. Add the private key as the VPS_SSH_PRIVATE_KEY secret in your GitHub repository settings:

- Go to your GitHub repository
- Click on "Settings"
- In the left sidebar, click on "Secrets and variables" > "Actions"
- Click on "New repository secret"
- Name: VPS_SSH_PRIVATE_KEY
- Value: Copy the entire content of your private key file (~/.ssh/id_ed25519), including the BEGIN and END lines

5. Add your VPS hostname or IP address as the VPS_HOST secret:

- Follow the same steps as above
- Name: VPS_HOST
- Value: Your VPS IP address or hostname (e.g., 91.108.102.201)

6. Add your SSH username as the VPS_SSH_USERNAME secret:

- Follow the same steps as above
- Name: VPS_SSH_USERNAME
- Value: Your SSH username for the VPS (usually "root" or another user with appropriate permissions)

Manual vs. Automatic Deployment

With automatic deployment configured, you have two options for deploying updates:

1. Automatic Deployment: Simply push changes to the main branch, and GitHub Actions will handle the deployment.

- Manual Deployment: Follow the steps in the "Updating the Application" section if you need to deploy manually.

Conclusion

Your Rush Master application should now be successfully deployed on your VPS and configured for automatic updates whenever changes are pushed to the main branch. If you encounter any issues, refer to the troubleshooting section or check the application logs for more information.

Rush Master Versioning Strategy

Overview

This document outlines the versioning strategy for the Rush Master project, covering both backend and frontend components.

Versioning Scheme

Rush Master follows [Semantic Versioning (SemVer)](<https://semver.org/>) principles:

- MAJOR version for incompatible API changes (e.g., 1.0.0)
- MINOR version for backward-compatible functionality additions (e.g., 0.1.0)
- PATCH version for backward-compatible bug fixes (e.g., 0.1.1)

Current Version

- Current version: 0.1.0 (pre-1.0.0 development phase)
- This version is reflected in:
 - Backend: `build.gradle` file
 - Frontend: `package.json` file

Git Tags

Git tags are used to mark specific versions in the repository history:

- Format: `v{MAJOR}.{MINOR}.{PATCH}`
- Example: `v0.1.0`

Version Bumping Guidelines

- PATCH version: Increment for bug fixes and minor changes
- MINOR version: Increment for new features that don't break existing functionality
- MAJOR version: Increment for breaking changes (will be 1.0.0 when the API is considered stable)

Release Process

1. Update version numbers in both backend and frontend configuration files
2. Create a git tag for the new version
3. Update this documentation if versioning strategy changes

Version History

The following versions have been tagged in the repository:

- v0.0.1: Initial project setup
- v0.0.2: CI/CD workflows implementation
- v0.0.3: Basic features and UI components implementation
- v0.0.4: Core functionality additions
- v0.0.5: Authentication and user features implementation
- v0.0.6: Deployment and CI/CD improvements
- v0.1.0: First minor release with complete feature set

Rush Master

Elevate Your League Experience

Remerciements

Tout d'abord, je remercie mon tuteur de stage, **Samuel Drillaud**, pour son encadrement, ses conseils avisés et son soutien constant tout au long de mon alternance.

Un grand merci également à **Hervé Milesi**, qui à pris le temps de me partager son savoir ainsi que ses automatismes qui ont grandement contribué à ma courbe d'apprentissage.