

Rapport de Projet Filé

League of Legends Ensi.gg



Création d'un site visant à aider les joueurs de League of Legends à travers des statistiques et optimiser leurs créations de parties.

Rapport de conception

Mai 2024

Table des matières

1 Résumé du projet	3
2 Introduction	4
3 Présentation de la solution	5
4 Description conceptuelle du problème	6
5 Conception du Frontend	7
5.1 Visuels réalisés	7
5.2 Découpage des composants	8
5.3 Gestion des Tests	8
6 Conception du Backend	9
6.1 Introduction	9
6.2 Structure globale du backend	9
6.3 Database	10
6.4 WebSocket	11
6.5 Gestion des Tests	13
7 Discussion	15
7.1 Général	15
7.2 Deploiement	15
7.2.1 Difficulté	15
7.2.2 Méthode simpliste	16
7.2.3 Méthode NextJS	16
7.3 Amélioration possibles	16
8 Conclusion	17

Table des figures

1	User case du site web. Red = Page ProDraft, Green = Page Stat	5
2	Détail des différents appels d'API implémenté pour chaque page du site. Gris = Requêtes à l'API, Vert = Pages utilisant l'API	6
3	Page Index où l'on écrit son nom d'utilisateur ou un champion	7
4	Page décrivant toutes les statistiques d'un joueur ici Haruyukis	7
5	Page montrant la liste des différents champions	7
6	Page focalisée sur le champion Vayne en reprenant les données de la database	7
7	Page de génération de liens : un lien par équipe + un lien pour les spectateurs	7
8	Page Draft permettant la création des compositions d'équipes entre joueurs	7
9	Découpage des composants	8
10	Fonctionnement du projet dans l'ensemble	9
11	Exemple par Swagger (fourni par FastAPI) d'une requête de backend pour le frontend	9
12	Schéma de la structure du backend rangé par fonctionnalité	10
13	Schéma des relations dans la Base de données	11
14	Résultat d'un pytest	14
15	Couverture du code à 79%	14
16	Exemple d'une erreur avec une reponse 404	14
17	Deploiement basique en Fullstack	16

1 Résumé du projet

Dans le cadre de notre projet de création d'un site destiné à aider les joueurs de League of Legends à travers des statistiques et à optimiser leurs compositions de parties, nous avons effectué diverses tâches au niveau du frontend, du backend et des websockets. Nous avons réalisé cela afin de fournir un site regroupant tout le nécessaire afin de fournir aux joueurs un outil convivial et informatif pour améliorer leur expérience de jeu.

Notre travail jusqu'à présent comprend la mise en place de plusieurs éléments essentiels. Tout d'abord, nous avons développé une **page d'accueil** intuitive permettant aux utilisateurs de rechercher les statistiques d'un joueur ou d'un champion en utilisant simplement leur nom. Cette fonctionnalité offre un moyen rapide et efficace d'accéder aux informations pertinentes.

Ensuite, nous avons conçu la page **Summoners**, qui offre aux joueurs la possibilité de consulter leur historique de parties, leurs statistiques personnelles, ainsi que d'autres données utiles pour analyser et améliorer leur performance en jeu.

De plus, notre site propose une page dédiée aux **Champions**, offrant aux utilisateurs un accès facile à une liste complète des personnages disponibles dans le jeu, classés selon différents rôles. Cette fonctionnalité permet aux joueurs de mieux comprendre les options de composition disponibles et de choisir en conséquence en fonction des différentes statistiques de chaque champion.

Chaque champion bénéficie d'une page détaillée fournissant des statistiques approfondies sur ses performances et son utilisation. Ces informations détaillées permettent aux joueurs de prendre des décisions plus éclairées lors de la sélection et de l'utilisation des champions dans leurs parties.

Enfin nous avons réalisé une page de **draft** qui permet à 2 équipes de pouvoir préparer les champions qu'ils vont choisir d'une manière beaucoup plus rapide et pratique grâce à l'intermédiaire de websockets afin de voir en temps réel les choix de l'autre équipe. Pour pouvoir accéder aux liens gérant la room, nous avons aussi établi une **page de générations de liens**.

Pour réaliser cela, nous allons utiliser le modèle de l'atomic design pour effectuer chaque partie du frontend, nous allons créer un cache pour éviter le nombre de requêtes de l'API LOL en faisant notre propre base de données. De plus nous allons utiliser les websockets afin de permettre aux différents utilisateurs de réaliser leur composition d'équipe.

Malgré ces avancées, notre projet rencontre certaines limitations. Notamment, l'obtention des données nécessaires pour alimenter notre site peut parfois être difficile en raison des restrictions d'accès et de disponibilité de l'API. De plus, l'analyse et la présentation des données peuvent être complexes et nécessitent un travail approfondi pour garantir leur exactitude et leur pertinence.

Cependant, malgré ces défis, notre solution représente une avancée significative dans le domaine de l'aide aux joueurs de League of Legends. En fournissant un accès facile à des données et des analyses précieuses, notre site vise à améliorer la compréhension et les performances des joueurs, contribuant ainsi à une expérience de jeu plus enrichissante et gratifiante.

En résumé, notre travail jusqu'à présent a posé les bases d'une solution complète et utile pour les joueurs de League of Legends. Nous sommes convaincus que notre site continuera à évoluer et à apporter de la valeur à la communauté des joueurs dans les mois/années à venir.

2 Introduction

League of Legends (LoL) est un jeu stratégique nécessitant énormément de compétences à maîtriser. Malheureusement, le jeu en lui-même ne permet pas d'obtenir les données nécessaires permettant à un joueur de progresser comme les données de certaines de leurs parties, les personnages forts du moment ou bien les objets à acheter. D'autres soucis comme le choix des personnages sur le jeu qui est excessivement long et pas très bien géré font que de nombreuses équipes choisissent leur personnage en dehors du jeu.

Notre objectif est de réaliser un site permettant de remédier à ces deux problèmes et donc améliorer l'expérience de jeu des différents joueurs.

3 Présentation de la solution

Pour le développement du frontend de ce projet, nous avons opté pour utiliser **React** en conjonction avec **Next.js** afin de réaliser une application web réactive et dynamique. Du côté du style, nous avons choisi **Tailwind CSS** pour son côté esthétique. Les tests du frontend ont été réalisé avec l'aide de **Cypress** qui nous offre une automatisation de tests efficace.

Pour le développement du backend en **Python**, nous avons choisi ce langage en raison de sa rapidité d'exécution. Étant donné que nous manipulons d'énormes quantités de données, plusieurs millions, il est essentiel d'avoir des appels rapides du backend.

Le frontend est rigoureusement testé à l'aide de l'outil **Cypress**

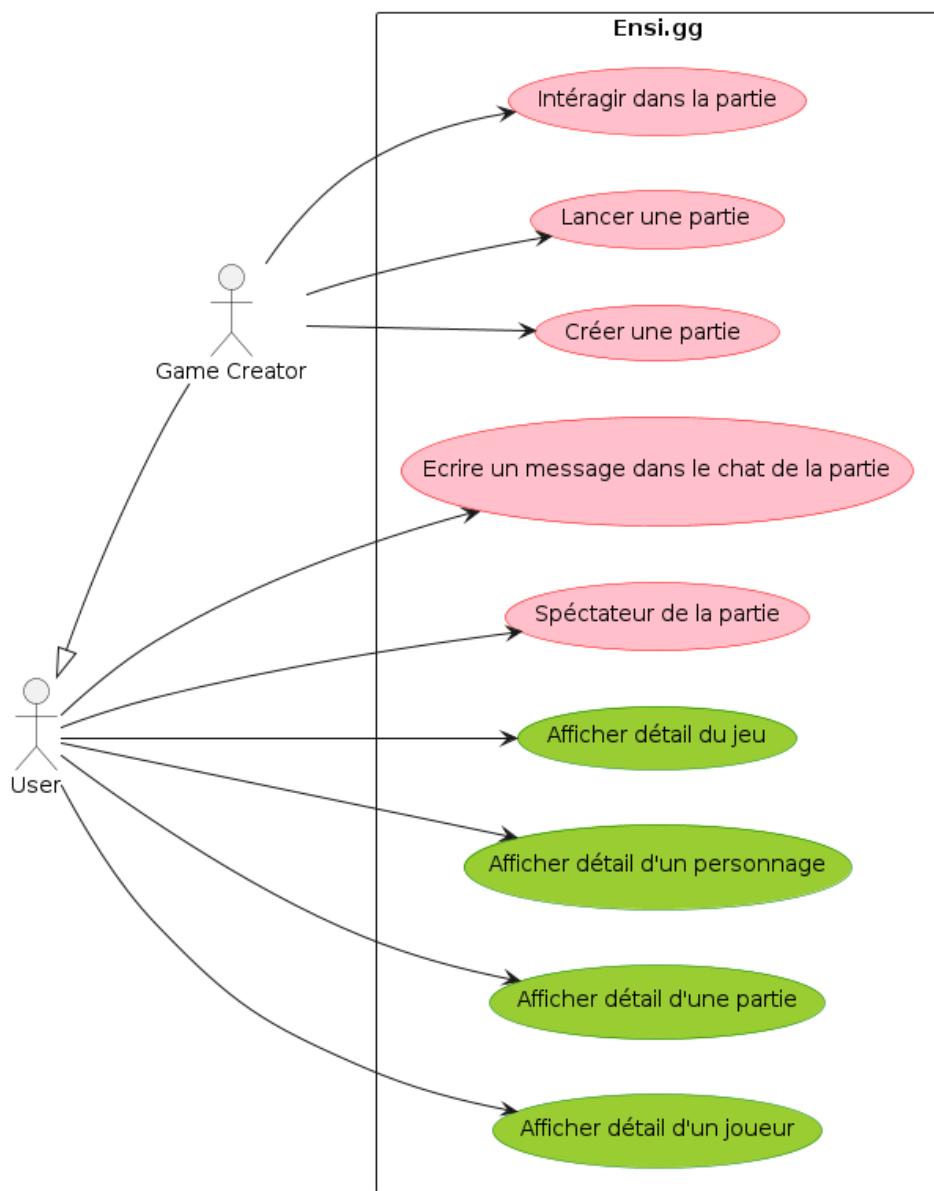


Figure 1: User case du site web. Red = Page ProDraft, Green = Page Stat

4 Description conceptuelle du problème

Notre solution consiste à développer un site web intuitif et convivial qui rassemble toutes les informations nécessaires pour aider les joueurs à prendre des décisions éclairées. Cela comprend des statistiques détaillées sur les joueurs individuels, les champions, ainsi que des analyses approfondies sur les tendances de jeu et les stratégies gagnantes.

Le développement de cette solution nécessitera une intégration étroite avec les API de League of Legends pour obtenir des données en temps réel sur les joueurs, les champions, les parties en cours, etc. De plus, nous devrons concevoir une interface utilisateur attrayante et intuitive qui facilite la navigation et la visualisation des données.

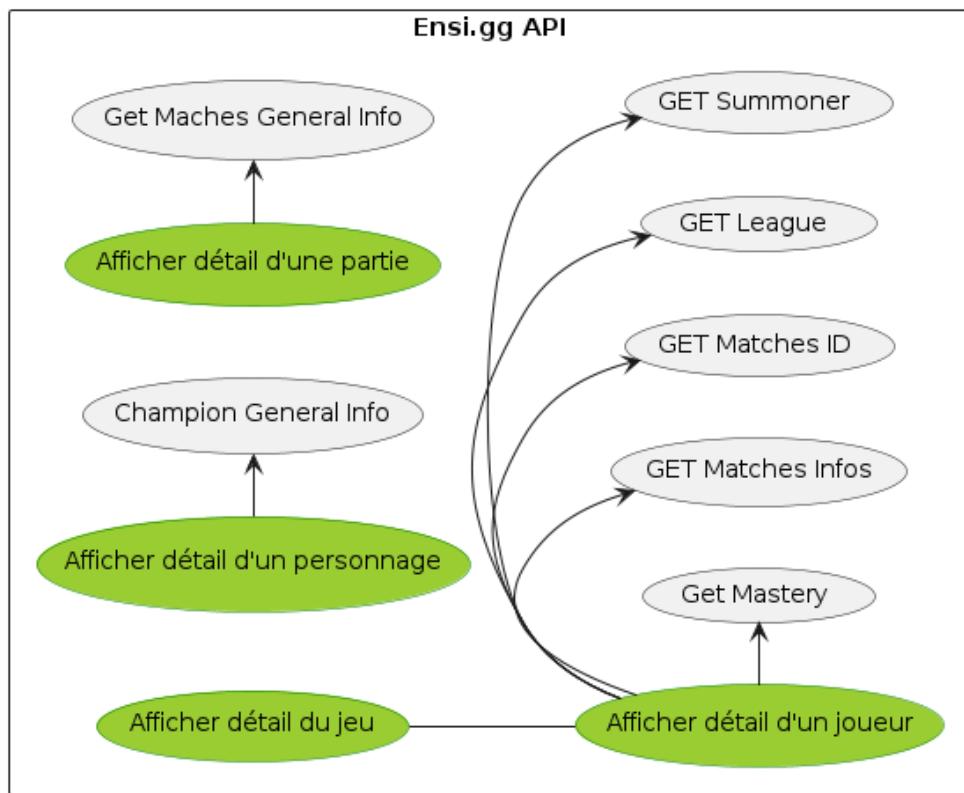


Figure 2: Détail des différents appels d’API implémenté pour chaque page du site. Gris = Requêtes à l’API, Vert = Pages utilisant l’API

5 Conception du Frontend

5.1 Visuels réalisés

Voici un aperçu de nos résultats au niveau du frontend :

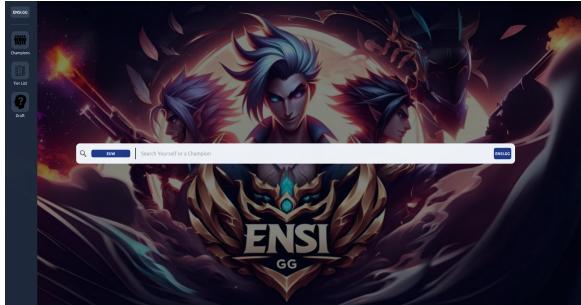


Figure 3: Page Index où l'on écrit son nom d'utilisateur ou un champion

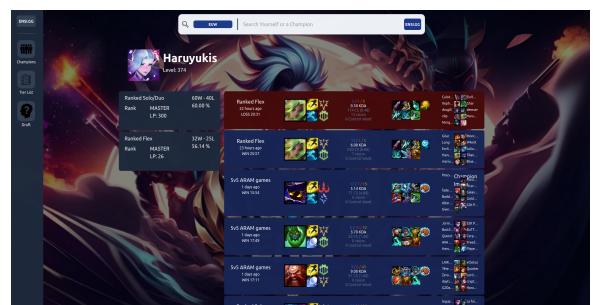


Figure 4: Page décrivant toutes les statistiques d'un joueur ici Haruyukis

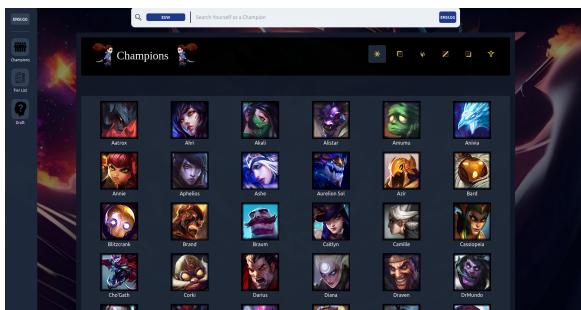


Figure 5: Page montrant la liste des différents champions

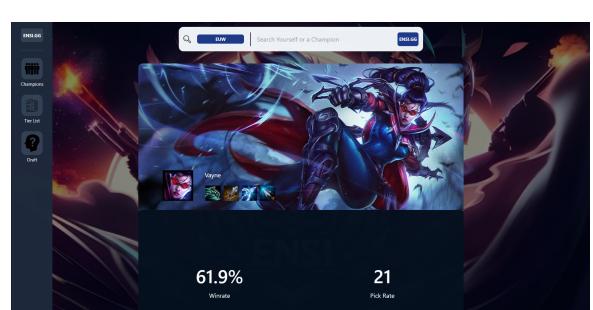


Figure 6: Page focalisée sur le champion Vayne en reprenant les données de la database

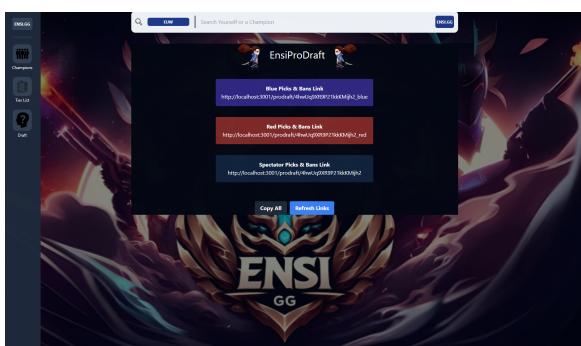


Figure 7: Page de génération de liens : un lien par équipe + un lien pour les spectateurs

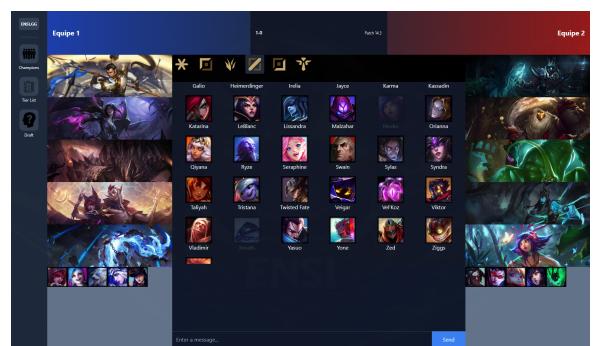


Figure 8: Page Draft permettant la création des compositions d'équipes entre joueurs

5.2 Découpage des composants

Pour la partie Frontend, nous avons utilisé le modèle de conception atomique en décomposant nos interfaces en différentes parties modulaires (Atomic Design by Brad Frost) : "atomes", "molecules", "organismes", "templates" et "pages". Le but de modèle était de nous assurer une cohérence de structure au niveau des pages (pour ne pas les surcharger) tout en gardant une structure réutilisable afin notamment de pouvoir recycler des organismes à différents endroits. Un des exemples peut être la sidebar qui doit être présente sur chaque page afin de pouvoir changer de page efficacement.

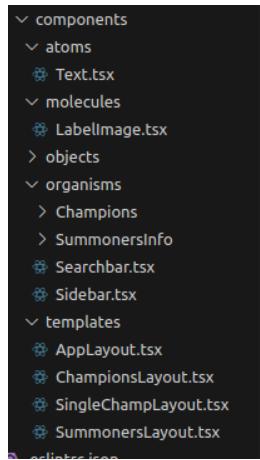


Figure 9: Découpage des composants

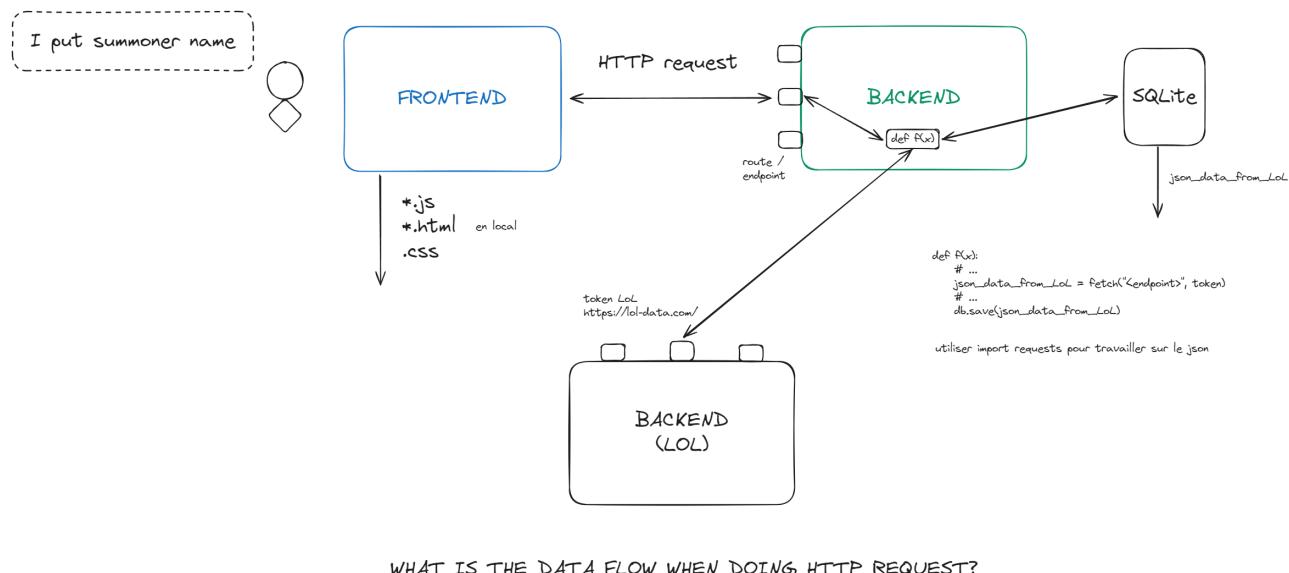
5.3 Gestion des Tests

Pour la gestion des tests du Frontend, nous avons utilisé Cypress pour des raisons de facilité d'utilisation, son côté visuel qui permet de bien voir les différentes possibilités et sa rapidité. Grâce à cela, il était utile pour nous de tester les différentes fonctionnalités utiles et de vérifier que le comportement attendu était le bon. Par exemple, vérifier la redirection vers la page du personnage associé quand on le recherche au niveau de la page index pour rechercher ou bien vérifier que le choix des personnages sur les différents liens s'effectue de manière cohérentes avec les websockets.

6 Conception du Backend

6.1 Introduction

Le jeu LOL fournit une API qui renvoie un bon nombre de données du jeu soit sur un joueur, soit une partie et bien d'autres données. Cependant, il y a une limite dans le nombre de requête possible dans un certain laps de temps. Sachant que le nombre de données à récupérer est astronomiquement grand, il est donc important de créer un cache pour fournir au Frontend des données tout en limitant le nombre d'appel à l'API LOL. On passe donc dans notre propre backend pour faire l'intermédiaire entre le Frontend et le Backend de LOL (Figure 8). Par ailleurs, pour une même requête API de LOL, certaines données ne sont pas nécessaire ou bien n'existe pas. Avoir notre propre base de donnée permet donc également de limiter les données envoyées au Frontend, donc d'améliorer les performances et de faire des statisques sur les personnages jouables dans le jeu (non fourni par l'API LOL).



WHAT IS THE DATA FLOW WHEN DOING HTTP REQUEST?

Figure 10: Fonctionnement du projet dans l'ensemble

```
200 Response body
[{"summonerId": "NGIurIL2EiRx0MSuhmaQOX1ESvkdsuNGOFyTPoAR0WhILpY", "summonerPuuid": "HJjicLx0bFNuecnPFK13skchhQ_h-q7xhFCfsmplNa6FGK3kEayIKUbj0VAYHD9716z5tz_ZU50A", "summonerName": "haruyukis", "summonerTag": "eum", "summonerLevel": 377, "summonerProfileIconId": 4666}]
```

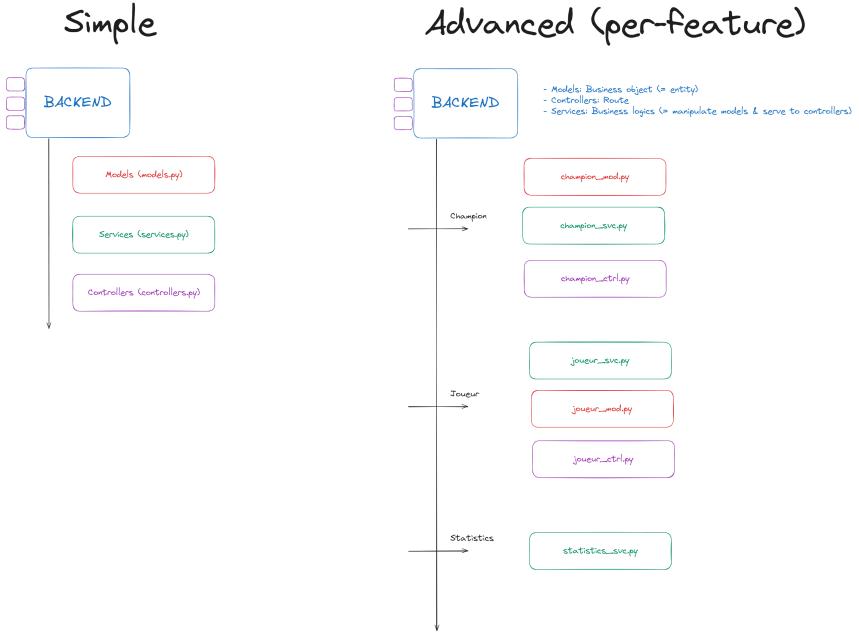
Figure 11: Exemple par Swagger (fourni par FastAPI) d'une requête de backend pour le frontend

6.2 Structure globale du backend

Notre backend suit la structure suivante par fonctionnalité: Modèle Service Contrôleur, une structure classique de backend pour de l'échange d'information depuis une base de donnée pour garantir une maintenance de code simple et concis.

MCS: Models / Controllers / services

- Models: Business object (= entity)
- Controllers: Route
- Services: Business logics (= manipulate models & serve to controllers)



```

Backend_lol
├── __pycache__
│   └── champions
│       ├── __pycache__
│       └── champion_dto.py
│       └── champion_svc.py
└── match
    ├── __pycache__
    └── models
        └── match_svc.py
└── user
    ├── __pycache__
    ├── models
    │   └── user_dto.py
    │   └── user_svc.py
    ├── .coverage
    ├── controllers.py
    ├── cred.py
    ├── database.db
    ├── helpers.py
    ├── init_db.py
    ├── models.py
    ├── services.py
    ├── test_champions.py
    ├── test_league.py
    ├── test_match.py
    ├── test_personnage.py
    └── test_summoner_info.py

```

Figure 12: Schéma de la structure du backend rangé par fonctionnalité

6.3 Database

La technologie utilisé ici pour la base de donnée est SQLite, en passant par la librairie de python SQLAlchemy pour pouvoir fournir rapidement, facilement et bien structuré une base de donnée fonctionnelle.

Les attributs des différentes tables proviennent principale des données reçus de l'API LOL. Chaque utilisateur (User) possède deux rangs: ils sont donc identifiés par les attributs de Ranks **summonerId** et **queueId**. Chaque **rank** est lui-même reliés aux personnages qui ont été joué par l'utilisateur. Les données des personnages permettent de savoir lequel possède les meilleurs statisques. Cela permet à l'utilisateur de savoir quel personnage est le plus adapté pour gagner ses parties.

Ensuite, pour Match est identifié par un identifiant (matchId) et chaque match est relié à 10 autres participants (2 équipes de 5 joueurs). Un participant est donc forcément identifier par son Id et quel match, il a participé. Stocker cela permet donc de faire les statisques que ce soit dans la table Personnage (vérifier si un joueur est bon avec un certains personnage) ou bien avoir une statistique globale Champion (statistique d'un personnage sur l'ensemble des joueurs qui l'a joué).

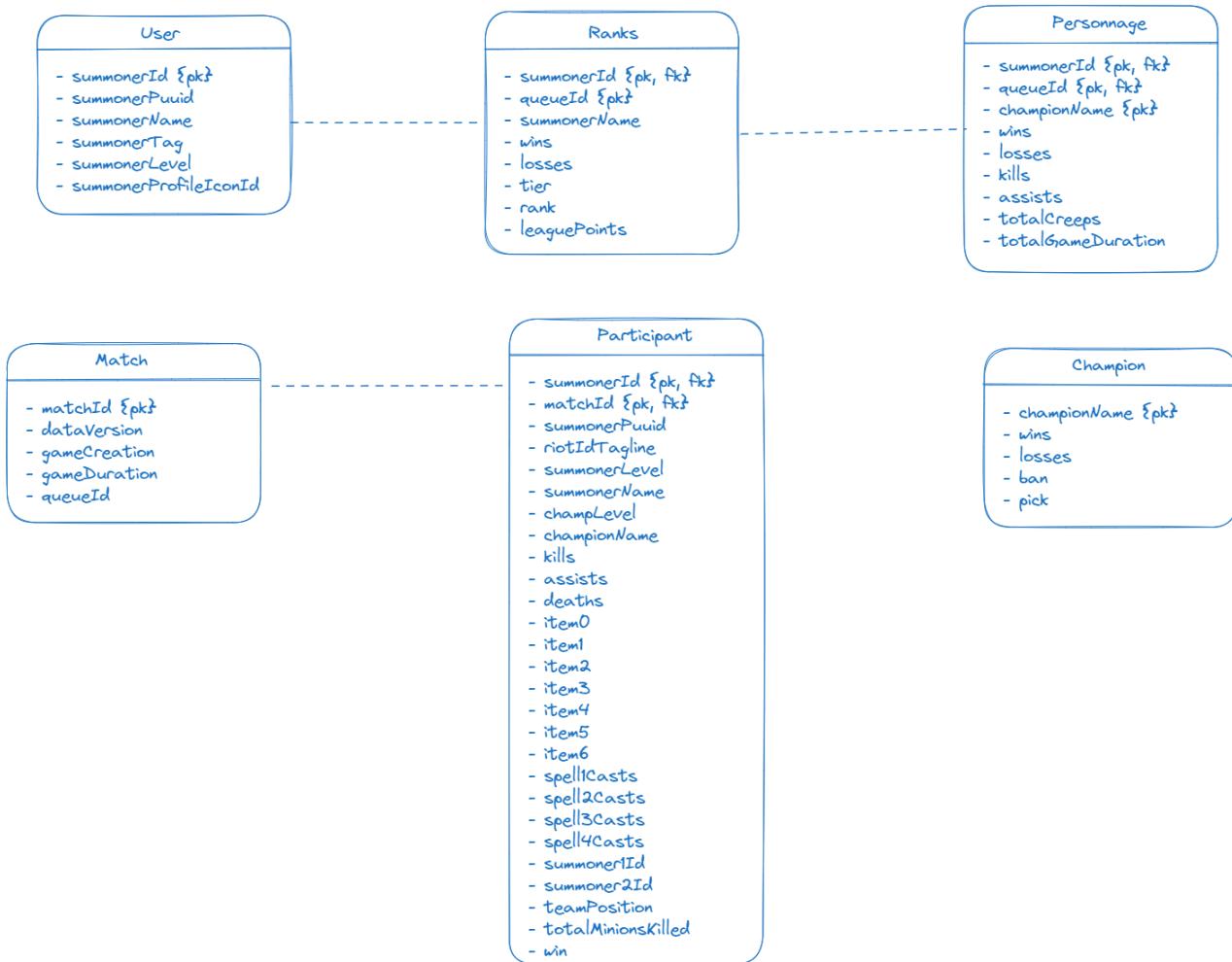


Figure 13: Schéma des relations dans la Base de données

6.4 WebSocket

Les WebSockets permettent une communication entre le serveur et le client en temps réel ce qui est essentiel pour la page ProDraft de notre projet. L'utilisation des WebSockets garantit que les actions entreprises par un Game Creator soit immédiatement visibles par tous les autres user connectés à la même salle.

Pour l'implémentation des Websockets, nous avons utilisé la bibliothèque **socket.io** qui est compatible avec JavaScript et Node.js.

Nous avons configuré le serveur WebSocket pour accepter les connexions provenant de notre client hébergé à <http://localhost:3001>, le serveur, lui, écoute sur le port 3000 (Cela est valable uniquement car le site n'est pas déployé).

Fonctionnalités de la WebSocket :

Connexion et Déconnexion des Utilisateurs

- **Log des Connexions** : Enregistre les connexions des utilisateurs avec leur identifiant unique (`socket.id`).
- **Log des Déconnexions** : Enregistre les déconnexions des utilisateurs avec leur identifiant unique (`socket.id`).

Rejoindre une Salle

- **Rejoindre une Salle** : Les utilisateurs peuvent rejoindre une salle spécifique en utilisant l'événement `join_room`.
- **Normalisation des Noms de Salle** : Les noms de salle sont normalisés en supprimant les suffixes `_blue` ou `_red`.
- **Initialisation de l'État de la Salle** : Si la salle n'existe pas encore dans `roomData`, son état est initialisé.
- **Envoi de l'État Initial** : L'état initial de la salle est envoyé à l'utilisateur qui vient de rejoindre.

Changement d'État

- **Événement state_change** : Les utilisateurs peuvent envoyer des changements d'état via l'événement `state_change`.
- **Mise à Jour des Données** : Les données de la salle dans `roomData` sont mises à jour avec les nouvelles informations.
- **Émission des Changements d'État** : Les changements d'état sont émis à tous les utilisateurs de la salle.

Envoi de Messages

- **Événement message** : Les utilisateurs peuvent envoyer des messages textuels à la salle via l'événement `message`.
- **Normalisation des Noms de Salle** : Les noms de salle sont normalisés en supprimant les suffixes `_blue` ou `_red`.
- **Formatage des Messages** : Les messages sont formatés avec le nom d'utilisateur et le contenu du message.
- **Émission des Messages** : Les messages formatés sont émis à tous les utilisateurs de la salle.

Fonction Utilitaire : `removeColorFromRoom`

- **Normalisation des Noms de Salle** : La fonction supprime les suffixes `_blue` et `_red` des noms de salle pour normalisation.

6.5 Gestion des Tests

Pour pouvoir tester notre backend python, nous avons décidé d'utiliser pytest pour faire des tests E2E et qui fournit également la couverture du code (Figure 12). Pour chaque **End point**, il y a deux choses très importantes à vérifier:

- la véracité des données obtenus
- la gestion des erreurs. (Figure 13)

En effet, les tests permettent de vérifier les données obtenus (si elles coïncident avec celle attendues) et tests les données manquantes, par exemple dans le cas où le frontend demanderait un joueur inexistant.

```

PS C:\Users\jiang\Documents\projet-de-specialite-lol\backend\backend_lol> pytest
=====
platform win32 -- Python 3.12.2, pytest-8.1.1, pluggy-1.4.0
rootdir: C:\Users\jiang\Documents\projet-de-specialite-lol\backend
configfile: pyproject.toml
plugins: anyio-4.3.0, cov-5.0.0
collected 13 items

test_champions.py .
test_league.py ...
test_match.py ....
test_personnage.py ...
test_summoner_info.py .....

[ 7%]
[ 23%]
[ 53%]
[ 69%]
[100%]

===== 13 passed in 6.31s =====

```

Figure 14: Résultat d'un pytest

Name	Stmts	Miss	Cover
<hr/>			
champions\champion_dto.py	7	0	100%
champions\champion_svc.py	26	13	50%
controllers.py	44	3	93%
cred.py	1	0	100%
helpers.py	3	0	100%
init_db.py	11	11	0%
match\match_svc.py	79	33	58%
match\models\match_dto.py	37	0	100%
models.py	79	0	100%
services.py	6	0	100%
test_champions.py	10	0	100%
test_league.py	15	0	100%
test_match.py	25	0	100%
test_personnage.py	15	0	100%
test_summoner_info.py	27	0	100%
user\user_dto.py	28	0	100%
user\user_svc.py	113	52	54%
<hr/>			
TOTAL	526	112	79%

Figure 15: Couverture du code à 79%

```
{
  "detail": {
    "status": {
      "status_code": 404,
      "message": "Data not found - No results found for player with riot id gfdsfsdf#dgsgdg"
    }
  }
}
```

Figure 16: Exemple d'une erreur avec une reponse 404

7 Discussion

7.1 Général

À ce stade, nous avons implémenté les fonctionnalités décrites dans notre diagramme UML. Pour offrir une expérience utilisateur optimale, il serait avantageux de rendre notre application compatible avec différents appareils en adoptant une approche de design responsive. En effet, en plus de l'importance des performances, il est absolument nécessaire que notre application soit simple d'utilisation et esthétique afin de pouvoir se démarquer de la concurrence.

Une autre chose importante à signaler est qu'il est nécessaire qu'un administrateur rajoute les données des nouveaux personnages et objets dans les jsons correspondants puisque malheureusement ces données ne sont pas disponibles via l'API de Lol.

7.2 Deploiement

Le déploiement de notre projet est pour l'instant assez fastidieux. Par manque de temps, nous n'avons donc pour l'instant pas pu le déployer. Il y a deux méthodes pour le déployer, une assez limitée et l'autre qui demande normalement un travail plus importants.

7.2.1 Difficulté

Dans notre projet, nous avons décidé d'utiliser NextJS pour sa simplicité pour le routage. Cependant, ce choix a été mauvais pour la partie déploiement. En effet, NextJS est un framework SSR (Server Side Rendering). C'est donc un serveur qui va afficher les différents composants. Ce gros problème oblige donc d'avoir des pages non statiques et donc difficile à déployer. En effet, déployer un projet fullstack classique (hors NextJS), il faut fournir des fichiers statiques (index.html, *.js et *.css) générés avec la commande **npm run build** et ensuite fournir comme **End Point** principale à FastAPI, ces fichiers statiques. En conclusion, c'est donc le backend qui gérera l'affichage et la gestion de l'API créée lors d'un déploiement.

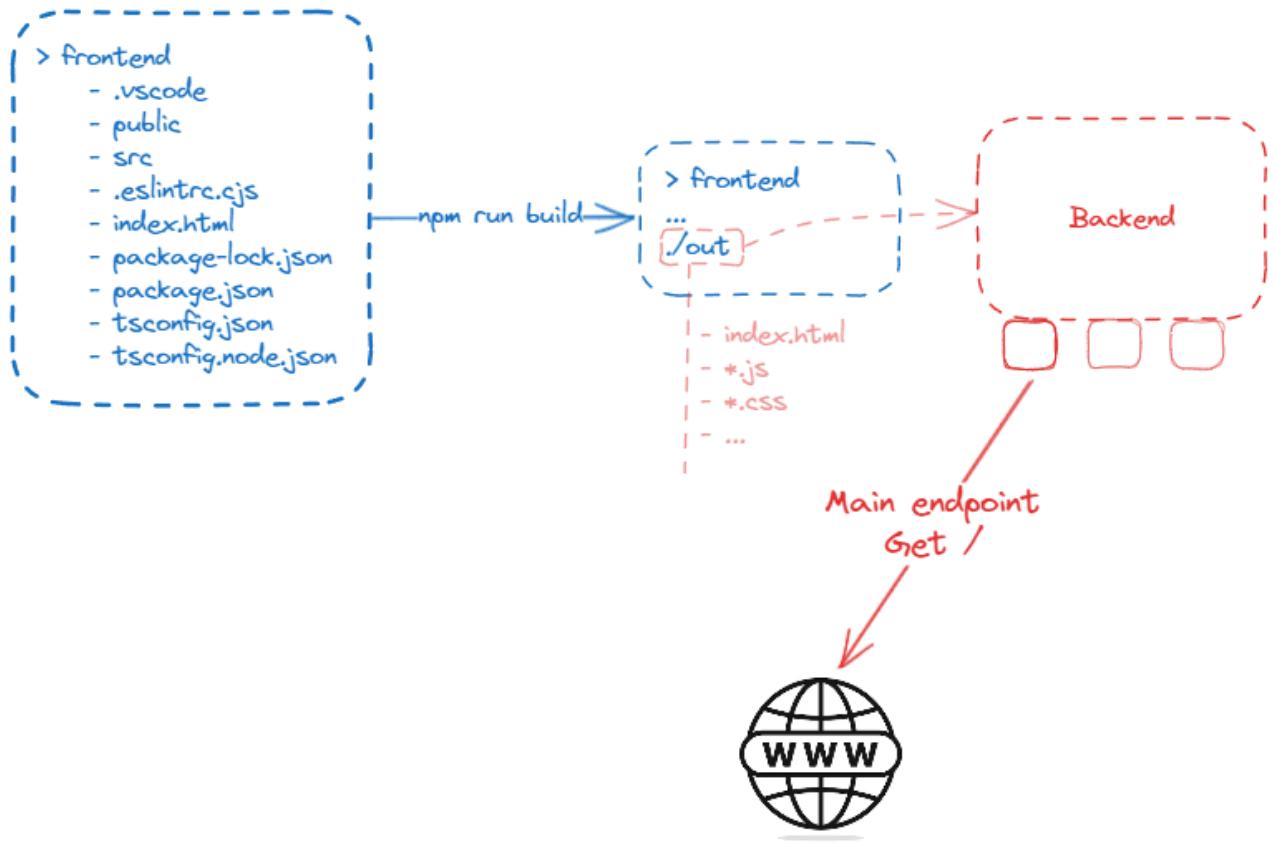


Figure 17: Deploiement basique en Fullstack

7.2.2 Méthode simpliste

Il est possible de déployer seulement le backend et pas le frontend au lieu du projet fullstack en entier via Scalingo par exemple. Le frontend pourrait donc discuter directement avec le backend déployer.

7.2.3 Méthode NextJS

NextJS propose leur propre méthode pour pouvoir déployer un application SSR Fullstack gratuitement. Par manque de temps, la méthode n'a pas été étudié assez profondément pour être bien expliquer. Il est clair que notre projet actuel a quelques lacunes par rapport à cela: déployer en SSR. Il faudrait donc changer la structure de beaucoup de code.

7.3 Amélioration possibles

Par rapport au déploiement, il était très dommage de s'y intéressé à la toute fin. Il est clair que cette erreur aurait pu être éviter si on se serait directement intéressé au déploiement dès le début pour pouvoir coder en fonction ou même de decider d'utiliser un autre framework.

Une autre amélioration serait de créer un site responsive. En effet, ce n'est pas le cas pour l'instant. Notre but dans le projet était vraiment de touché à plein de technologies et de méthodes qui se fait dans la vie d'un programmeur fullstack de tous les jours pour en apprendre le plus possible.

8 Conclusion

Nous avons atteint quasiment tous les objectifs que nous nous étions fixés pour notre application web. Les différentes pages ont été développées avec succès, notamment les pages statistiques des joueurs et des personnages. Le frontend a été complété pour l'ensemble des pages, incluant un début de design réactif, le design complet et les tests Cypress.

Du côté backend, nous avons achevé les travaux principalement pour la page **Champions**. Le schéma relationnel de la base de données a été finalisé, et nous avons réussi à récupérer l'ensemble des données de tous les joueurs, ce qui nous a permis de créer des statistiques détaillées pour chaque personnage jouable. Des tests pour le backend ont été mis en place à l'aide de **supertest**.

Nous avons également implémenté les **WebSocket** à l'aide de **socket.io** pour permettre une connexion en temps réel entre deux équipes, facilitant ainsi l'élaboration de stratégies avant une partie. Trois liens sont générés pour une room, deux pour les game creators qui s'occupent d'intégrer les modifications sur la page, et le dernier pour un utilisateur qui est spectateur des modifications faites par les game creators. La page dispose aussi d'un chat en temps réel, permettant à tous les utilisateurs présents dans la room de discuter entre eux.

Les résultats actuels et la structure de l'ensemble du code sont clairs et compréhensibles, ce qui a permis de poursuivre le développement de manière efficace. La page **Summoners**, qui constitue notre principale réalisation, offre une vue exhaustive sur chaque joueur et intègre les principales technologies que nous avons utilisées, telles que le frontend et les appels d'API depuis le frontend vers notre backend.

En somme, tous les objectifs ont été atteints, et le projet est maintenant pleinement fonctionnel et prêt à être utilisé, il ne manque que le déploiement.