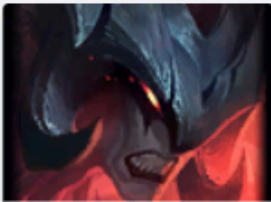


MSP 1

LoL Champions

LoL Champions

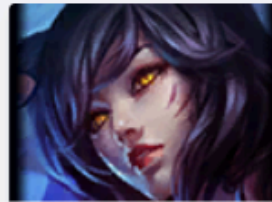
League of Legends Champions



Aatrox
The Darkin Blade



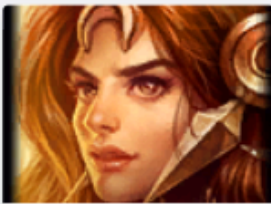
Jax
Grandmaster at Arms



Ahri
The Nine-Tailed Fox



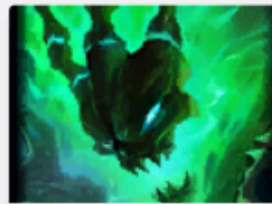
Garen
The Might of Demacia



Leona
The Radiant Dawn



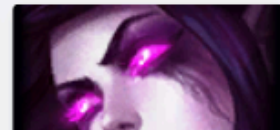
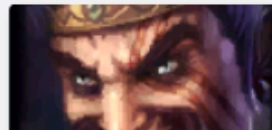
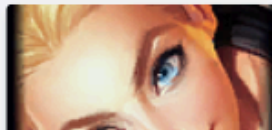
Zed
The Master of Shadows



Thresh
The Chain Warden



Sivir
The Battle Mistress



Rivière Etienne

26/08/2024

Concepteur Développeur d'applications

Introduction.....	1
Installer et configurer son environnement de travail.....	2
Choix de l'IDE : Visual Studio Code.....	2
Configuration pour le backend Spring Boot.....	3
Configuration pour le frontend Next.js.....	3
Gestion de version avec GitHub.....	3
Développer des interfaces utilisateur.....	4
Conception de l'interface utilisateur.....	4
Implémentation avec Next.js.....	6
Développer des composants métiers.....	11
Identification des composants métiers clés.....	11
Implémentation des composants métiers.....	13
Gestion des données.....	14
Tests et validation.....	15
Contribuer à la gestion d'un projet informatique.....	15
Gestion de la qualité.....	15
Gestion des risques et des défis.....	15

Introduction

Dans l'univers en constante évolution des jeux vidéo en ligne, League of Legends se distingue comme l'un des titres les plus populaires et compétitifs. Avec plus de 150 champions jouables, chacun possédant des capacités uniques et une riche histoire, la communauté de LoL est toujours avide d'informations détaillées sur ces personnages.

C'est dans ce contexte que naît le projet "LoL Champions", une application web conçue pour répondre aux besoins des joueurs de League of Legends. L'objectif principal de cette application est de fournir une plateforme centralisée et conviviale permettant aux utilisateurs de consulter l'ensemble des champions du jeu, d'explorer

leurs compétences en détail et de découvrir la variété de leurs apparences à travers leurs différents skins.

"LoL Champions" propose une architecture moderne et performante, intégrant un backend solide basé sur Spring Boot et un frontend réactif conçu avec Next.js. Cette combinaison technologique garantit une expérience utilisateur fluide et réactive, tout en offrant une gestion optimisée des données et une évolutivité durable.

Ce dossier projet présente en détail la conception, le développement et les fonctionnalités de l'application "LoL Champions". Il aborde les choix techniques, les défis rencontrés et les solutions mises en œuvre pour créer une ressource indispensable pour la communauté de League of Legends.

Compétences nécessaire à la réalisation de ce projet

- Installer et configurer son environnement de travail
- Développer des interfaces utilisateur
- Développer des composants métier
- Contribuer à la gestion d'un projet informatique

Installer et configurer son environnement de travail

La mise en place d'un environnement de développement efficace et adapté est une étape cruciale dans la réalisation du projet LoL Champions. Cette phase initiale jette les bases d'un développement fluide et organisé, permettant une gestion optimale du projet tout au long de son cycle de vie.

Choix de l'IDE : Visual Studio Code

Pour ce projet, Visual Studio Code a été sélectionné comme environnement de développement intégré principal. Ce choix s'appuie sur plusieurs avantages clés :

1. **Polyvalence** : VS Code offre un excellent support pour les technologies du backend (Spring Boot/Java) et du frontend (Next.js), permettant de travailler sur l'ensemble du projet dans un seul environnement.

2. **Extensibilité** : Grâce à son vaste écosystème d'extensions, VS Code peut être facilement adapté aux besoins spécifiques du développement Spring Boot et Next.js.
3. **Performance** : Léger et rapide, VS Code assure une expérience de développement fluide, même sur des projets complexes.
4. **Intégration Git** : VS Code propose une intégration native avec Git, facilitant la gestion des versions directement depuis l'IDE.

Pour mener à bien ce projet, il a été nécessaire d'installer et de configurer plusieurs extensions pour VS Code. Parmi celles-ci :

- **Extension Pack for Java** : Ce pack d'extensions pour Java regroupe plusieurs outils essentiels pour le développement en Java, tels que les fonctionnalités de compilation, de débogage, d'analyse de code et de gestion des dépendances. Il inclut des extensions comme **Language Support for Java™ by Red Hat**, **Debugger for Java**, et **Test Runner for Java**, qui améliorent l'expérience de développement en Java dans VS Code en offrant des fonctionnalités avancées et une intégration fluide avec Maven, Gradle, et d'autres outils Java courants.
- **Spring Boot Extension Pack** : Ce pack d'extensions est spécialement conçu pour faciliter le développement d'applications Spring Boot. Il inclut des outils comme **Spring Initializr**, qui permet de créer rapidement des projets Spring Boot, ainsi que **Spring Boot Dashboard** et **Spring Boot Tools**, qui offrent une gestion simplifiée des configurations, du débogage et du déploiement des applications Spring Boot directement depuis VS Code.
- **Prettier - Code Formatter** : Cette extension est utilisée pour formater automatiquement le code selon des règles de style prédéfinies. Prettier garantit un code propre et cohérent en appliquant des normes de style de manière uniforme à travers tous les fichiers, facilitant ainsi la lecture et la maintenance du code.
- **ESLint** : Cette extension permet de détecter et corriger automatiquement les erreurs de style et de syntaxe dans le code JavaScript et TypeScript en se basant sur les règles de codage configurées. ESLint aide à maintenir un code

propre et conforme aux standards définis, ce qui réduit les erreurs et améliore la qualité globale du code.

- **Tailwind CSS IntelliSense** : Cette extension fournit une autocomplétion intelligente et des suggestions pour les classes CSS de Tailwind directement dans VS Code. Elle améliore la productivité en aidant les développeurs à utiliser les classes Tailwind rapidement et sans erreur, tout en offrant des informations contextuelles sur les classes disponibles.

Configuration pour le backend Spring Boot

Pour le développement du backend avec Spring Boot, les étapes suivantes ont été réalisées :

- Installation des extensions nécessaires.
- Configuration du Java Development Kit (JDK) approprié.
- Mise en place des paramètres de build et de débogage spécifiques à Spring Boot.

Pour réaliser ce projet, il a été nécessaire d'installer plusieurs dépendances pour Spring Boot, notamment :

- Spring Boot Starter Data JPA
- Spring Boot Starter Web
- Spring Boot Starter Test
- Spring Boot DevTools
- PostgreSQL JDBC Driver

Configuration pour le frontend Next.js

Pour le développement frontend avec Next.js, l'environnement a été optimisé comme suit :

- Installation des extensions nécessaires.
- Exécution des scripts npm nécessaires au développement de l'application Next.js.

Gestion de version avec GitHub

GitHub a été choisi comme plateforme de gestion de version et de collaboration pour le projet LoL Champions. Les actions suivantes ont été entreprises :

1. Création d'un repository dédié au projet sur GitHub.
2. Configuration de Git en local et connexion au repository distant.

Cette configuration complète de l'environnement de développement assure une base solide pour le projet LoL Champions, favorisant la productivité, la collaboration et la maintenabilité du code tout au long du cycle de développement.

Développer des interfaces utilisateur

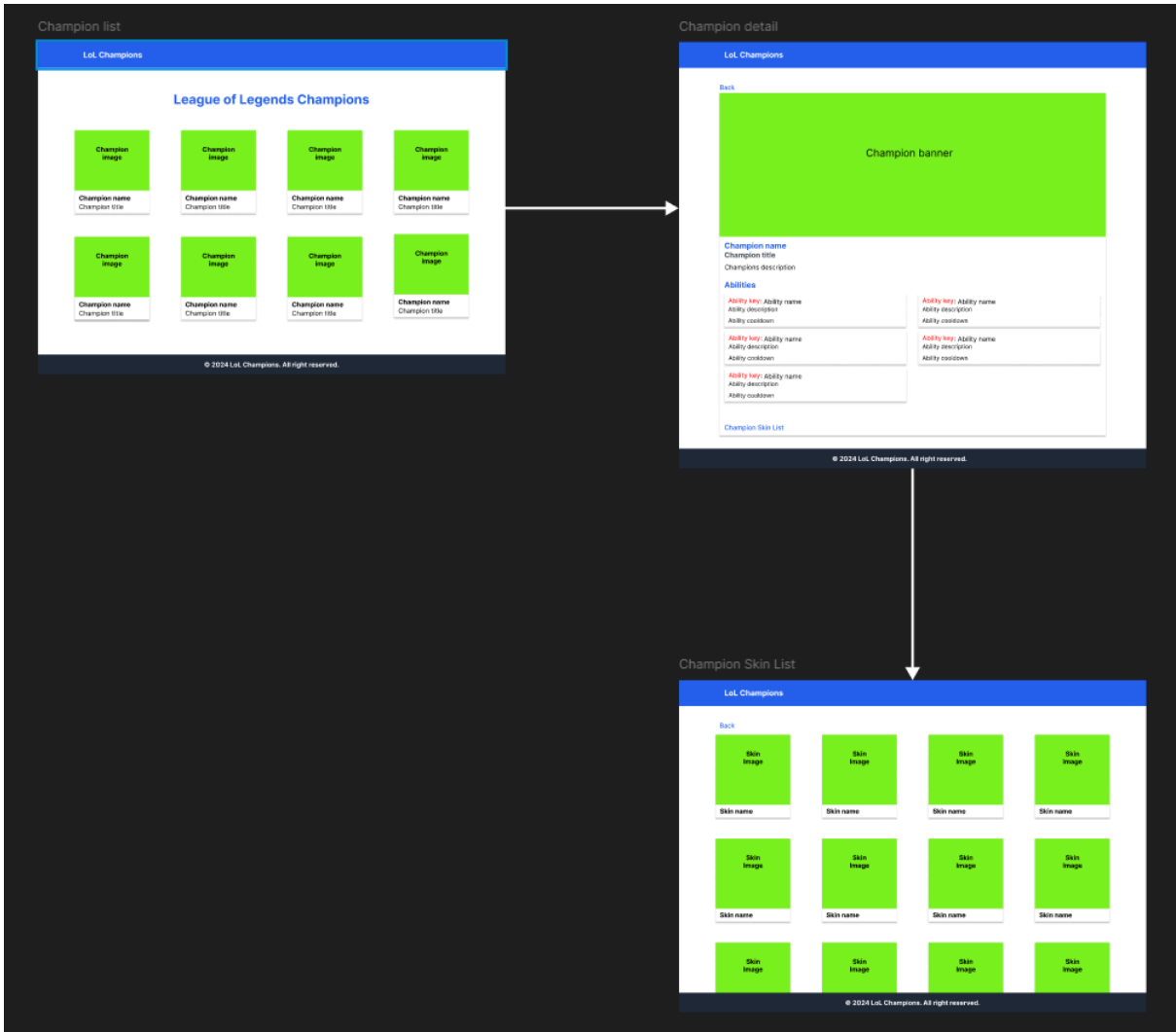
La conception d'interfaces utilisateur attractives, intuitives et performantes est essentielle au succès d'une application. Cette compétence englobe la conception, l'implémentation et l'optimisation des éléments visuels et interactifs de l'application, assurant une expérience utilisateur de haute qualité.

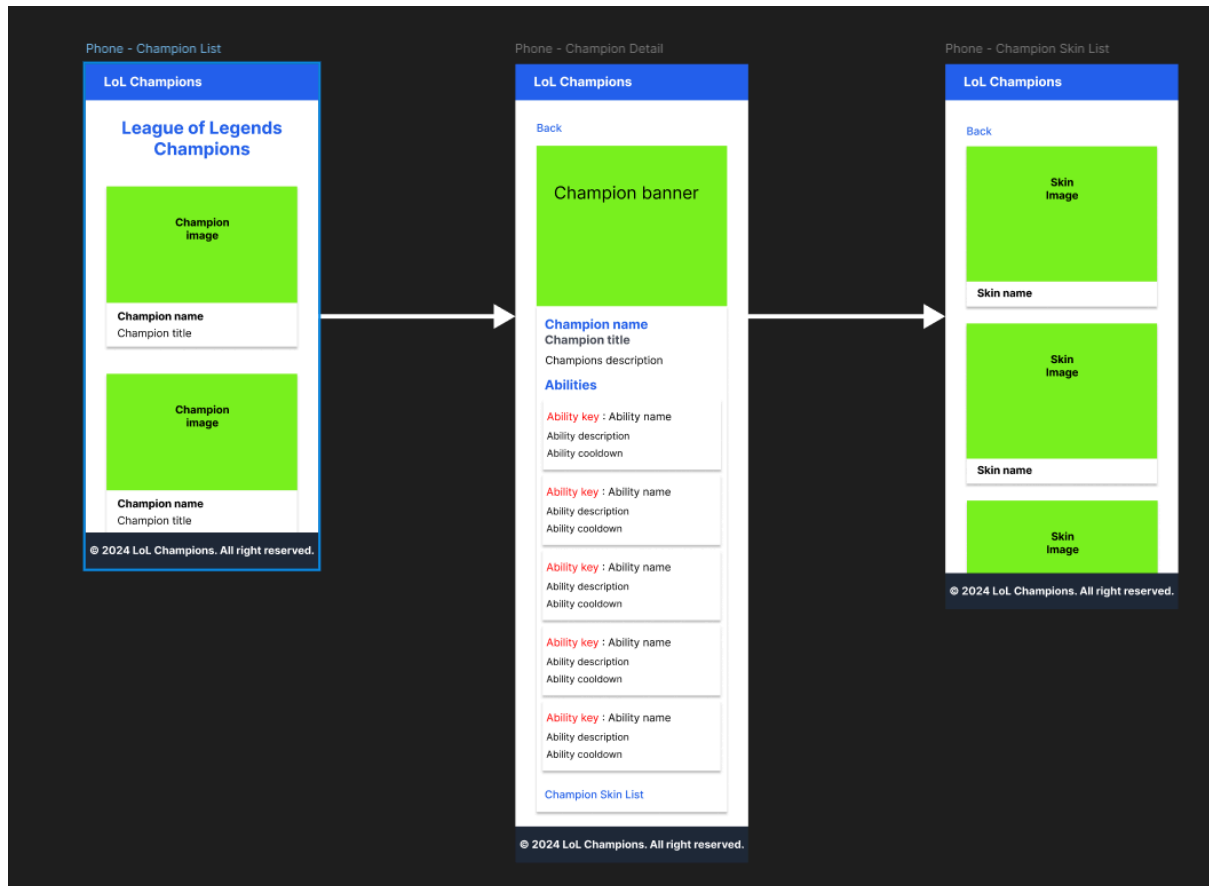
Conception de l'interface utilisateur

Wireframing et maquettage:

- Création de wireframes pour les principales pages de l'application (liste des champions, détails d'un champion, galerie de skins d'un champion).
- Réalisation de maquettes détaillées pour visualiser l'apparence finale de l'application.

Maquette globale version bureau :



Maquette globale version téléphone :**Implémentation avec Next.js****1. Structure du projet :**

- Organisation des composants React de manière modulaire et réutilisable.
- Mise en place d'une architecture de dossiers claire pour les pages, composants, styles et assets.

2. Développement des composants :

- Création de composants React pour les éléments récurrents.
- Utilisation de Tailwind Css pour le style.

Tailwind CSS permet de styliser directement les balises en utilisant la propriété **className**, offrant ainsi un moyen rapide et flexible d'appliquer des styles sans écrire de CSS personnalisé.

```
<h2 className="text-xl font-bold mb-2 text-blue-600">Abilities</h2>
<div className="mb-4 grid grid-cols-1 sm:grid-cols-2 gap-4">
  {sortedAbilities.map((ability) => (
    <div key={ability.id} className="bg-gray-100 rounded-lg p-3">
      <h3 className="text-lg font-bold mb-1 text-gray-800">
        <span className="text-red-600">{ability.abilityKey}</span> :{" "}
        {ability.name}
      </h3>
      <p className="mb-1 text-sm text-gray-700 line-clamp-2">
        {ability.description}
      </p>
      <p className="text-xs text-gray-600">
        Cooldown: {ability.cooldown}s
      </p>
    </div>
  ))}
</div>
```

3. Intégration des données :

- Implémentation de la logique de récupération des données depuis l'API backend.

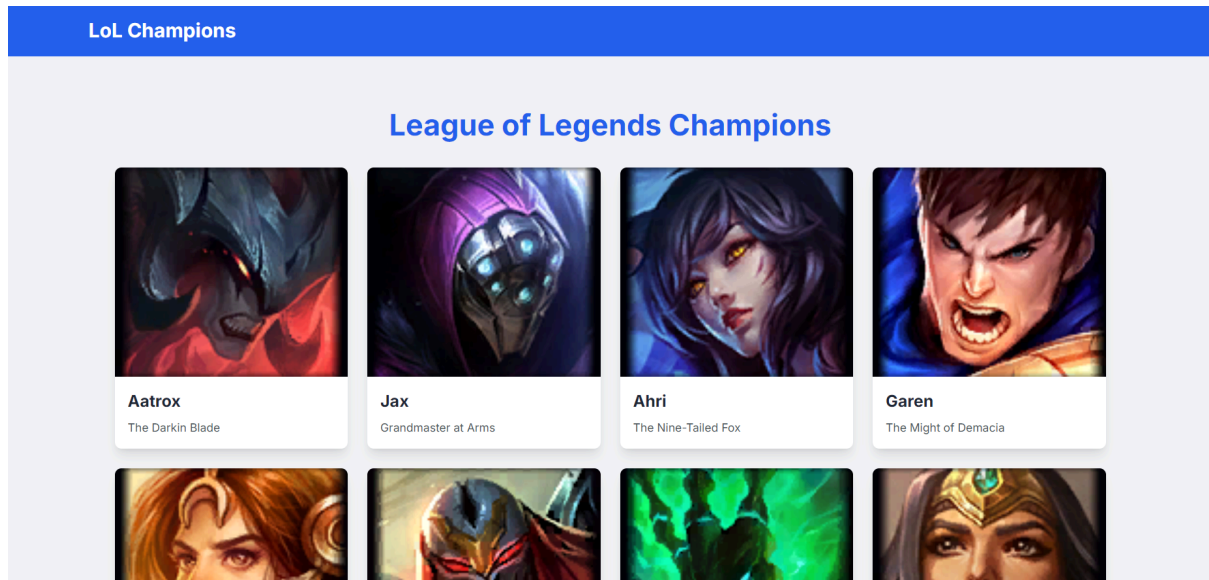
```
const API_BASE_URL = "http://localhost:8080";

export async function getChampions(): Promise<Champion[]> {
  const response = await fetch(`${API_BASE_URL}/champions`);
  if (!response.ok) {
    throw new Error("Failed to fetch champions");
  }
  return response.json();
}

export async function getChampion(name: string): Promise<Champion> {
  const response = await fetch(`${API_BASE_URL}/champions/${name}`);
  if (!response.ok) {
    throw new Error(`Failed to fetch champion with name ${name}`);
  }
  return response.json();
}
```


4. Design Responsive :

- Développement d'une interface adaptative fonctionnant sur desktop, tablette et mobile.

Version bureau et tablette :

LoL Champions

← Back



Aatrox

The Darkin Blade

Once honored defenders of Shurima against the Void, Aatrox and his brethren would eventually become an even greater threat to Runeterra, and were defeated only by cunning mortal sorcery.

Abilities

Passive : Deathbringer Stance

Aatrox's next basic attack has increased range and deals a percentage of the target's maximum health as bonus physical damage, while also healing him for a portion of the damage dealt.

Cooldown: 15s

Q : The Darkin Blade

Aatrox slams his greatsword down, dealing physical damage. This ability can be recast twice, with each cast increasing in damage and area of effect.

Cooldown: 14s

W : Infernal Chains

Aatrox smashes the ground, damaging and slowing the first enemy hit. If it's a champion or large monster, they are chained to the impact area. If they don't escape within a short duration, they are...

Cooldown: 26s

E : Umbral Dash

Aatrox dashes, gaining attack damage for a few seconds. This ability can store multiple charges.

Cooldown: 9s

R : World Ender

Aatrox unleashes his true demonic form, increasing his size and gaining attack damage, lifesteal, and movement speed. Upon activating, he terrifies nearby minions and resets his cooldowns for a short...

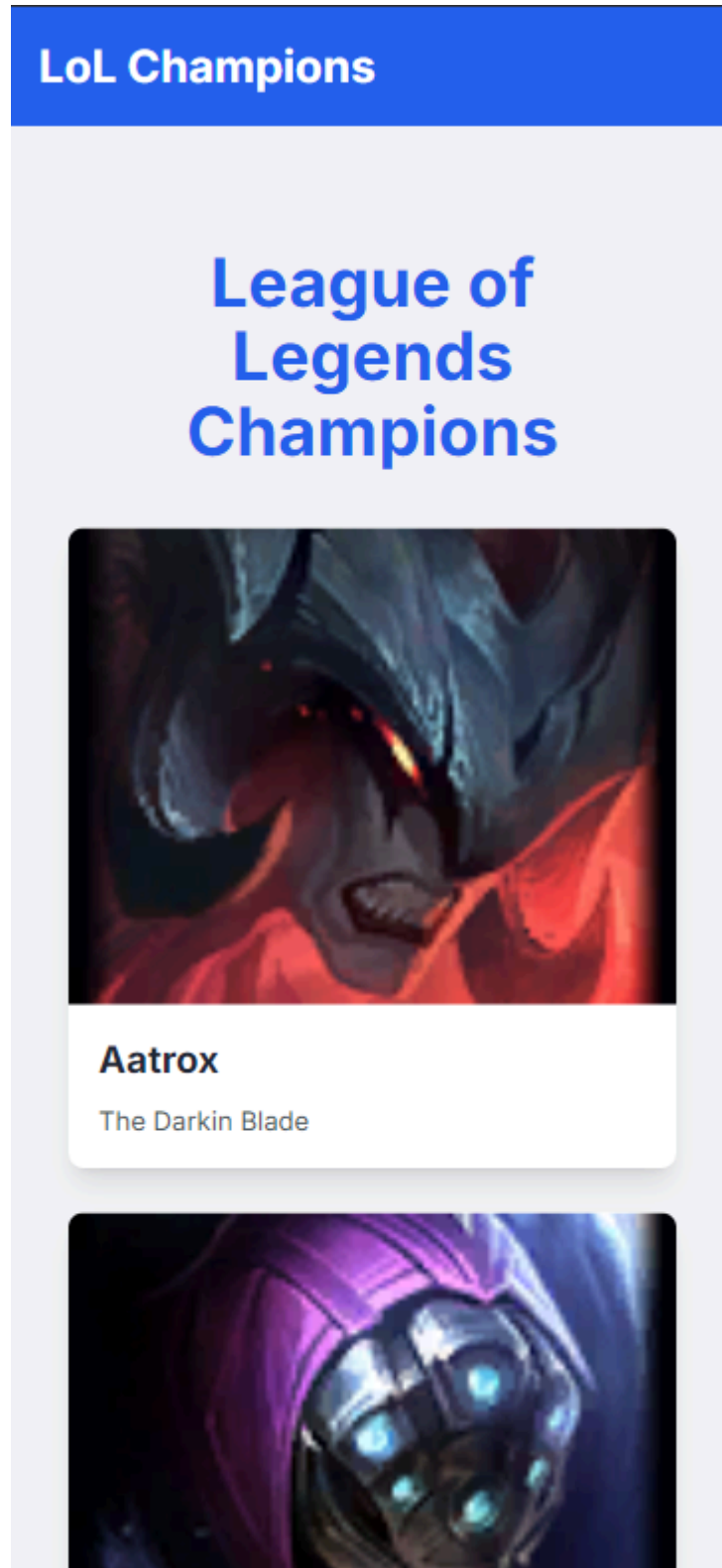
Cooldown: 120s

[Aatrox Skin List](#)

© 2024 LoL Champions. All rights reserved.

10

Version téléphone :



5. Tests automatisés :

- Implémentation des tests unitaires avec Vitest.

Exemple de code :

```
import { expect, test } from "vitest";
import { render, screen } from "@testing-library/react";
import Page from "../app/page";

test("Render the main page", () => {
  render(<Page />);
  expect(
    screen.getByRole("heading", {
      level: 1,
      name: "League of Legends Champions",
    })
  ).toBeDefined();
});
```

Réponse favorable dans le terminal :

```
RERUN __tests__/page.test.tsx x2

✓ __tests__/page.test.tsx (1)
  ✓ Render the main page

Test Files  1 passed (1)
Tests      1 passed (1)
Start at   13:13:18
Duration   280ms

PASS Waiting for file changes...
press h to show help, press q to quit
```

Réponse avec une erreur dans le terminal :

```
RERUN __tests__/page.test.tsx x4

> __tests__/skins/[name]/page.test.tsx (1)
  ✗ Render the skin list page
✓ __tests__/page.test.tsx (1)
  ✓ Render the main page

Test Files  1 failed | 1 passed (2)
Tests      1 failed | 1 passed (2)
Start at   13:21:57
Duration   247ms

PASS Waiting for file changes...
press h to show help, press q to quit
```

Cette approche globale du développement des interfaces utilisateur garantit une application non seulement esthétiquement plaisante, mais aussi fonctionnelle, performante et accessible à tous les utilisateurs, quel que soit leur appareil.

Développer des composants métiers

Le développement de composants métiers est au cœur de l'application LoL Champions. Ces composants encapsulent la logique spécifique au domaine de League of Legends et constituent la base fonctionnelle de l'application. Ils assurent la gestion, le traitement et la présentation des données relatives aux champions, à leurs compétences et à leurs skins.

Identification des composants métiers clés

1. Gestionnaire de champions :

- Responsable de la récupération et de la gestion des données de tous les champions.

Contrôleur du gestionnaire de champions :

```
@RestController
@RequestMapping("/champions")
@CrossOrigin(origins = "http://localhost:3000")
public class ChampionController {

    @Autowired
    ChampionService championService = new ChampionService();

    @GetMapping("")
    public List<Champion> getAllChampions() {
        return championService.getAllChampions();
    }

    @GetMapping("/{name}")
    public Champion getChampionById(@PathVariable String name) {
        return championService.getChampionByName(name);
    }

    @PostMapping("/create")
    public Champion creatChampion(@RequestBody Champion champion) {
        return championService.createChampion(champion);
    }
}
```

Dépôt (Repository) du gestionnaire de champions :

```
package com.msp1.lol_champions.repository;

import org.springframework.data.repository.ListCrudRepository;
import com.msp1.lol_champions.model.Champion;

public interface ChampionRepository extends ListCrudRepository<Champion, Long> {
    Champion findChampionByName(String name);
}
```

Service du gestionnaire de champions :

```
@Service
public class ChampionService {

    @Autowired
    private ChampionRepository championRepository;

    public List<Champion> getAllChampions() {
        return championRepository.findAll();
    }

    public Champion getChampionByName(String name) {
        return championRepository.findChampionByName(name);
    }

    public Champion createChampion(Champion champion) {
        return championRepository.save(champion);
    }

    public void deleteChampionById(Long id) {
        championRepository.deleteById(id);
    }
}
```

2. Gestionnaire de compétences :

- Traite et présente les détails des compétences de chaque champion.
- Chaque champion possède un passif et quatre compétences uniques


```
@Entity
@Table(name = "abilities")
public class Ability {

    @Id
    @GeneratedValue
    private Long id;

    private float cooldown;

    @Column(nullable = false, unique = true)
    private String name;

    private String description;
    private String imageUrl;
    private String abilityKey;

    @ManyToOne
    @JoinColumn(name = "champion_id", nullable = false)
    private Champion champion;

    /* Constructor(s) */
    public Ability() {
    }

    /* Getters & Setters */
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

3. Visualiseur de skins :

- Gère l'affichage et l'organisation des skins pour chaque champion.
- Tous les champions du jeu ont au minimum un skin qui est celui par défaut.

```
@Service
public class SkinService {

    @Autowired
    private SkinRepository skinRepository;

    public List<Skin> getAllSkins() {
        return skinRepository.findAll();
    }

    public List<Skin> getChampionSkins(String name) {
        return skinRepository.findChampionSkin(name);
    }

    public Skin createSkin(Skin skin) {
        return skinRepository.save(skin);
    }
}
```

Implémentation des composants métiers

1. Architecture et design patterns :

- Application du pattern Repository pour la gestion des accès aux données (voir illustration ci-dessus).

2. Développement backend avec Spring Boot :

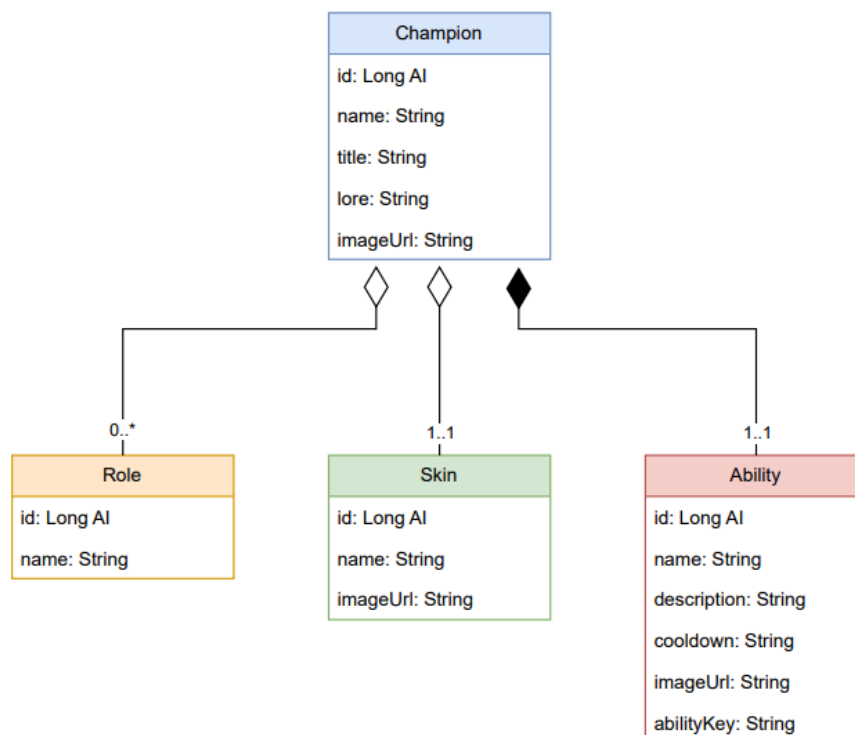
- Création de services RESTful pour exposer les données des champions, compétences et skins.
- Implémentation de la logique métier dans des services dédiés.
- Utilisation de Spring Data JPA pour la persistance des données.

```
spring.application.name=lol_champions
spring.jpa.database=postgresql
spring.datasource.url=jdbc:postgresql://localhost:5432/lolChampions
spring.datasource.username=username
spring.datasource.password=password
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.hibernate.ddl-auto=update
```

Gestion des données

1. Modélisation des données :

- Conception de schémas de base de données optimisés pour les champions, compétences et skins.
- Définition des relations entre les entités.



2. Récupération des données :

- Récupération de certaines données essentielles depuis l'API officielle de Riot Games.

Tests et validation

1. Tests unitaires :

- Écriture de tests unitaires pour chaque composant métier avec JUnit et Mockito.

- Mise en place de tests d'intégration pour valider les interactions entre les composants.

Cette approche du développement des composants métiers assure que l'application LoL Champions dispose d'une base solide et performante, capable de gérer efficacement les données complexes liées aux champions de League of Legends tout en offrant une expérience utilisateur fluide et réactive.

Plan d'évolution de l'application

Au fur et à mesure du développement de cette application, de nombreuses idées se sont manifestées. J'envisage de les intégrer dans de futures mises à jour. Voici les fonctionnalités que je prévois d'ajouter :

- Ajout d'une fonctionnalité permettant de trier les personnages en fonction de leur rôle dans le jeu.
- Intégration d'une barre de recherche pour faciliter la recherche de personnages ou de skins.
- Mise en place de comptes utilisateurs, permettant aux utilisateurs de se connecter et de créer leurs propres personnages. Ces créations seront ensuite soumises à un vote mensuel ou hebdomadaire. Les personnages les mieux notés seront présentés sur une page dédiée aux créations de la communauté.
- Ajout de la possibilité pour les utilisateurs de choisir la version du jeu dont ils souhaitent obtenir les informations. Certains personnages ayant subi des refontes visuelles ou de compétences, il peut être utile de consulter leur apparence et leurs compétences d'origine.