

SORBONNE UNIVERSITÉ



L'ANNÉE SCOLAIRE 2023-2024

Projet PC3R: Virtual Farm

Étudiants :

Rachid BOUHMAD
Elhadj Alseiny DIALLO
Do Truong Thinh TRUONG

Encadrant :

Romain DEMANGEON

Table des matières

1	Introduction	2
2	Base de données et Entités	2
2.1	Diagramme des relations	2
2.2	Utilisation d'Hibernate	3
2.3	DAO et Service	4
3	Endpoints et API	5
3.1	Endpoints	5
3.2	Servlets	6
3.3	ResponseDTO	6
4	Utilisation de l'API externe	7
5	Middleware	8
5.1	Auth	8
5.2	CORS	8
5.3	Filter mapping	9
6	Front end en ReactJS	9
6.1	Landing Page	10
7	Conclusion	14
7.1	Perspective	14
7.2	Mot de la fin	14

1 Introduction

Dans le cadre du cours Programmation Concurrente Réactive, Répartie et Réticulaire 2023-2024, nous avons été chargés de développer un projet d'application web en utilisant Jakarta Servlet (anciennement Java Servlet), le conteneur Tomcat, et des API externes. Notre première idée était de créer un jeu en ligne basé sur des comètes fournies par l'API de la NASA et les utiliser en tant qu'animal de compagnie. Après plusieurs sessions de brainstorming, nous avons décidé d'abandonner celle-ci et de créer une application de jeu en ligne qui génère des donjons en s'appuyant sur des API externes. Le jeu utilise également la géolocalisation pour afficher les donjons proches de l'utilisateur et permet aux joueurs de générer et d'utiliser des pets (animaux de compagnie virtuels) pour explorer ces donjons. Les joueurs peuvent acheter et vendre des pets, et les utiliser pour entrer dans les donjons.

Notre projet vise à combiner des aspects ludiques et techniques, en utilisant des technologies web modernes et des concepts avancés de programmation concurrente et répartie. Le backend de l'application est conçu pour être évolutif et sécurisé, assurant une expérience utilisateur fluide et fiable.

2 Base de données et Entités

2.1 Diagramme des relations

Pour gérer les données du jeu, nous avons conçu une base de données relationnelle détaillée. Cette base de données stocke des informations sur les utilisateurs, les pets, les donjons, les combats, et plus encore. Le schéma de la base de données est illustré dans le diagramme suivant :

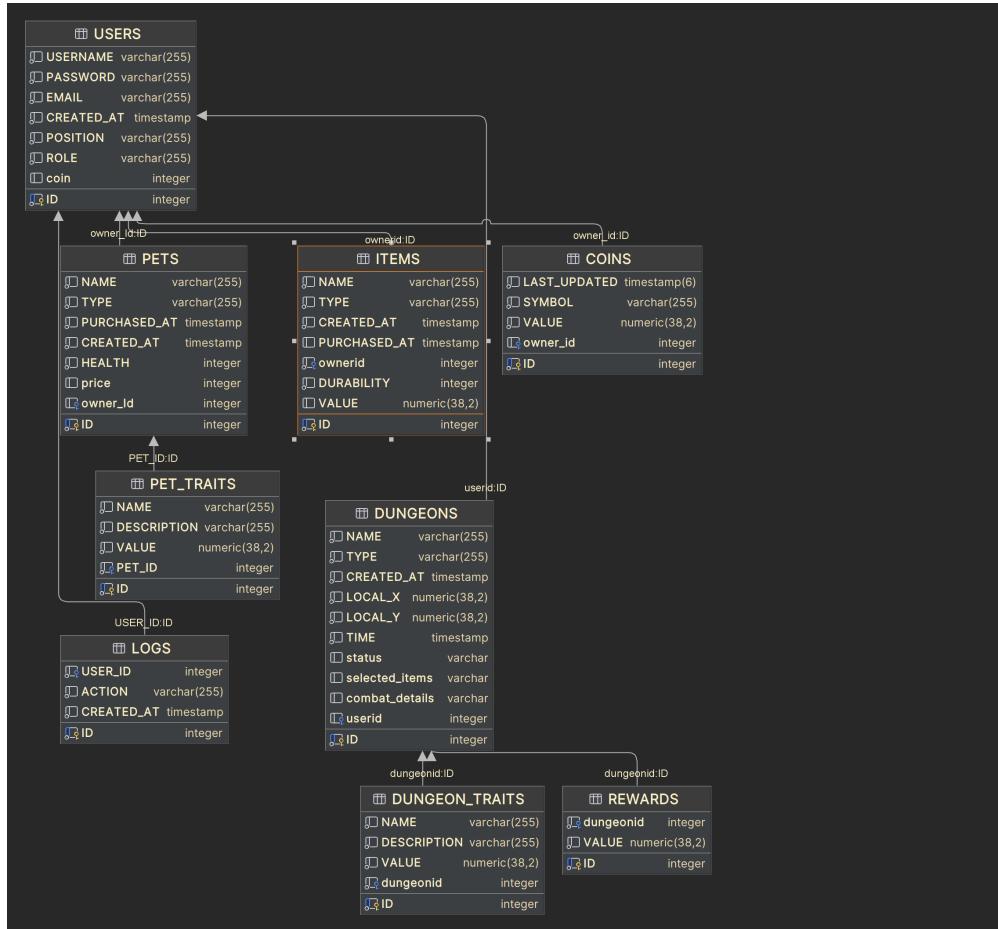


FIGURE 1 – Diagramme DB

2.2 Utilisation d'Hibernate

Nous utilisons Hibernate JPA (Java Persistence API) pour mapper les relations de notre base de données vers les entités Java. Hibernate facilite la gestion des opérations de base de données en fournissant un cadre de travail robuste et flexible pour le mapping objet-relationnel (ORM). Grâce à Hibernate, nous pouvons manipuler les données de la base de données à travers des objets Java sans avoir à écrire des requêtes SQL complexes.

Voici un diagramme concis du mapping des entités :



FIGURE 2 – Mapping des entités

Hibernate JPA joue un rôle crucial dans notre application en simplifiant les interactions avec la base de données. Voici quelques avantages et aspects de son utilisation :

- **Mapping Objet-Relationnel (ORM)** : Hibernate permet de mapper les tables de la base de données à des classes Java. Cela rend le code plus lisible et maintenable, car nous pouvons manipuler les données de la base de données en utilisant des objets Java.
- **Gestion Automatique des Transactions** : Hibernate gère automatiquement les transactions, réduisant ainsi le risque d'erreurs liées à la gestion manuelle des transactions.
- **HQL (Hibernate Query Language)** : Hibernate propose un langage de requête orienté objet, HQL, qui est similaire à SQL mais fonctionne avec les entités Java. Cela permet d'écrire des requêtes de manière plus intuitive et naturelle pour les développeurs Java.

— **Cache de Second Niveau** : Hibernate offre des mécanismes de cache de premier et second niveau pour améliorer les performances des opérations de base de données. Le cache de premier niveau est associé à la session, tandis que le cache de second niveau peut être partagé entre plusieurs sessions.

— **Gestion des Relations** : Hibernate gère efficacement les relations entre les entités, telles que les relations un-à-un, un-à-plusieurs, et plusieurs-à-plusieurs, en utilisant des annotations ou des fichiers de configuration XML.

Grâce à Hibernate, nous avons pu abstraire la complexité des interactions avec la base de données et nous concentrer sur la logique métier de notre application. L'utilisation de JPA avec Hibernate a également facilité le développement et les tests en fournissant un cadre cohérent et standardisé pour la persistance des données.

2.3 DAO et Service

Pour travailler avec nos entités et les relations dans la base de données, nous avons également conçu des DAO (Data Access Objects) pour effectuer les modifications sur la base de données comme suit :

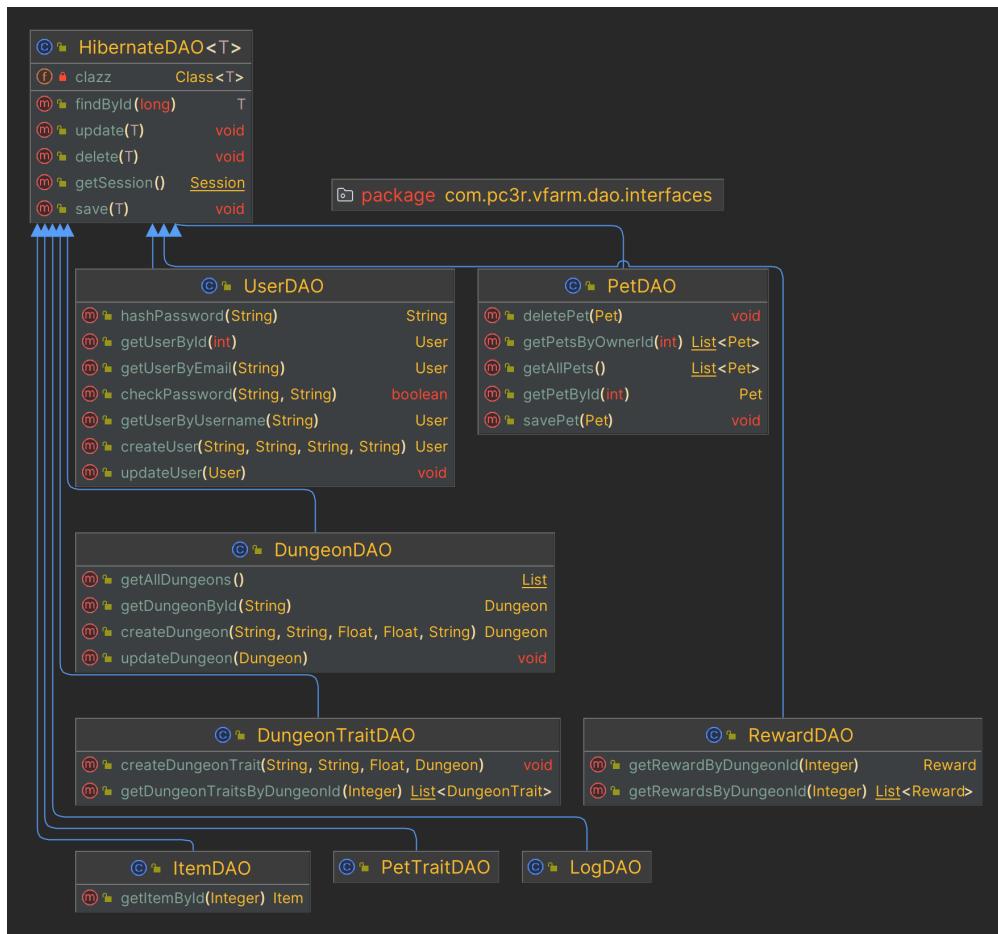


FIGURE 3 – DAO Classes

Nous avons un DAO générique qui prend les DAO concrets et offre les opérations basiques de la base de données. En effet, nous pouvons étendre les fonctions des DAO par y ajouter les méthodes qui vont exécuter les requêtes

pour récupérer les données exigées. Ensuite, nous créons des classes Service qui seront lesquelles qui interagissent avec les requêtes HTTP de notre application, ici, nous mettons des méthodes spécifiques du domaine.

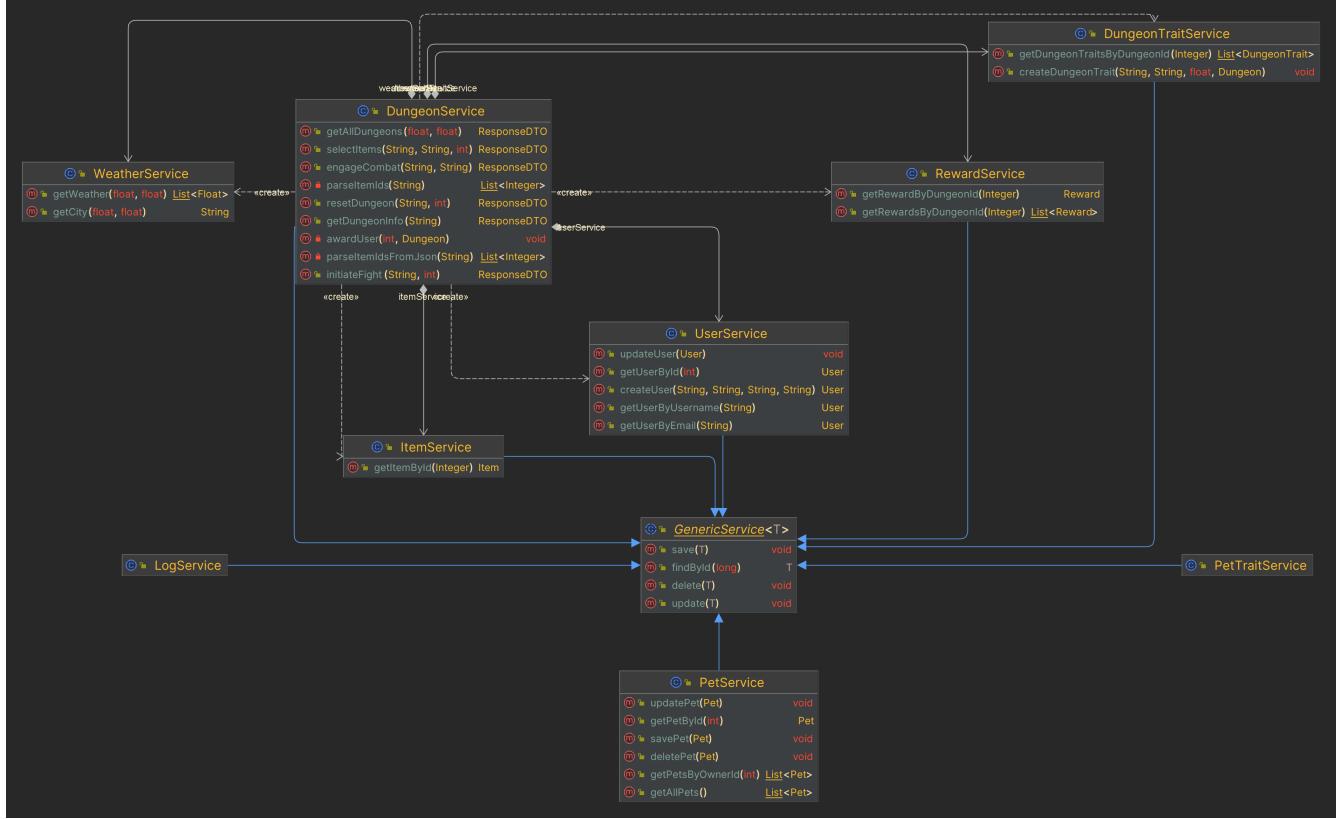


FIGURE 4 – Couche Service

3 Endpoints et API

3.1 Endpoints

Notre application expose plusieurs endpoints pour interagir avec les différentes fonctionnalités du jeu. Les endpoints sont conçus pour être sécurisés et performants, assurant une communication efficace entre le client (front-end) et le serveur (back-end). Voici une brève description des principaux endpoints et de leur rôle dans l'application :

- **Authentification des Utilisateurs** : Ce endpoint gère l'inscription des nouveaux utilisateurs, la connexion des utilisateurs existants et la gestion des sessions. Lors de l'inscription, les informations des utilisateurs sont validées et stockées en toute sécurité dans la base de données. Pour la connexion, les informations d'identification sont vérifiées et, si elles sont correctes, une session utilisateur est créée. Des mécanismes de sécurité tels que le hachage des mots de passe et la gestion sécurisée des sessions sont utilisés pour protéger les données des utilisateurs.
- **Gestion des Pets** : Ce endpoint permet aux utilisateurs de créer, acheter, vendre et gérer leurs pets. Les utilisateurs peuvent générer de nouveaux pets en utilisant des ressources ou de la monnaie virtuelle, acheter des pets d'autres utilisateurs sur un marché en ligne, ou vendre leurs propres pets. Les opérations de gestion incluent la mise à jour des attributs des pets, la visualisation de leurs caractéristiques, et l'affectation de pets spécifiques à des tâches ou des combats. La gestion des transactions et des échanges est sécurisée et assure l'intégrité des données.

- **Exploration des Donjons** : En utilisant les services de géolocalisation, ce endpoint affiche les donjons situés à proximité des utilisateurs. Les donjons sont générés dynamiquement en fonction de la position de l'utilisateur et des données provenant d'API externes. Les utilisateurs peuvent sélectionner un donjon proche et entrer dans celui-ci en utilisant leurs pets. L'exploration des donjons comprend la navigation à travers différentes salles, la découverte de trésors, et la rencontre de divers défis et ennemis.
- **Gestion des Combats** : Ce endpoint gère les interactions et les combats dans les donjons. Lorsqu'un utilisateur engage un combat, le système récupère les données pertinentes sur les pets de l'utilisateur et les ennemis dans le donjon. Les combats sont simulés en temps réel, prenant en compte les attributs des pets, les objets utilisés, et les stratégies de combat. Les résultats des combats, incluant les récompenses et les dommages subis, sont mis à jour dans la base de données et présentés à l'utilisateur.

3.2 Servlets

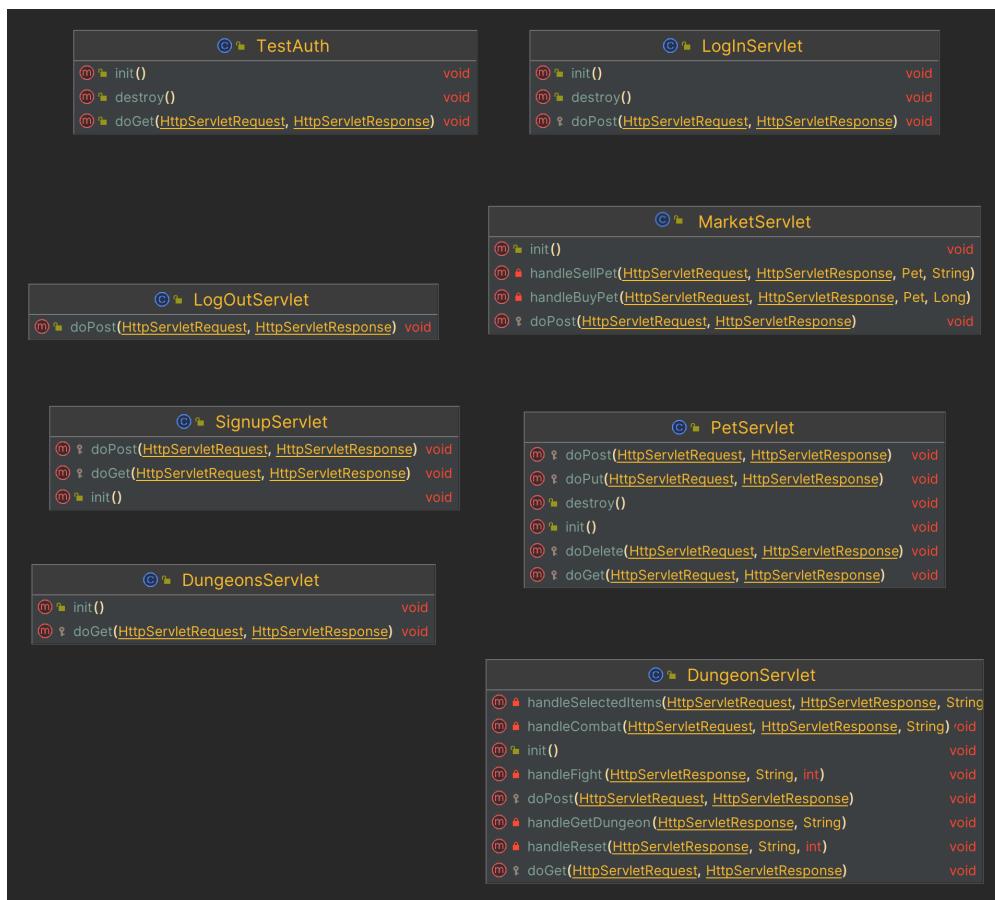


FIGURE 5 – Servlets

3.3 ResponseDTO

Chaque endpoint est développé en suivant les meilleures pratiques en matière de développement d'API RESTful, garantissant une intégration fluide et une évolutivité future de l'application. Afin de répondre à une requête, nous avons conçu des classes DTO qui nous permet d'avoir un standard modelisé des réponses HTTP.



FIGURE 6 – Couche DTO

4 Utilisation de l'API externe

Dans le cadre du projet, nous utilisons une API qui retourne la météo des coordonnées passées en paramètres de <http://api.weatherapi.com/> pour la génération des donjons. Les donjons devraient avoir des attributs qui sont basés sur les mesures de la météo de la localisation du joueur.

```
int randomeDungeons = (int) (Math.random() * 5 + 1);
for (int i = 0; i < randomeDungeons; i++){
    float posXRandomized = posX + (float) (Math.random() * 0.025);
    float posYRandomized = posY + (float) (Math.random() * 0.025);
    Dungeon dungeon = ((DungeonDAO) dao).createDungeon(weatherService.getCity(posX, posY),
    "city", posXRandomized, posYRandomized, "idle");
    dungeonTraitService.createDungeonTrait("Temperature", "Temperature of the dungeon", temp, dungeon);
    dungeonTraitService.createDungeonTrait("Wind", "Wind speed of the dungeon", wind, dungeon);
    dungeonTraitService.createDungeonTrait("Humidity", "Humidity of the dungeon", humidity, dungeon);
    dungeonTraitService.createDungeonTrait("Pressure", "Pressure of the dungeon", pressure, dungeon);
    dungeons.add(dungeon);
}
```

Dans le WeatherService, nous faisons appel à l'API externe pour récupérer les données avec notre clé

```
public class WeatherService {
    private static final String API_KEY = "c3053ddd2658400182e170051242404";
```

```
private static final String WEATHER_URL = "http://api.weatherapi.com/v1/current.json";

public List<Float> getWeather(float posX, float posY) throws IOException {
    URL url = new URL(WEATHER_URL + "?key=" + API_KEY + "&q=" + posX + "," + posY);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("GET");
    conn.setRequestProperty("Accept", "application/json");

    if (conn.getResponseCode() != 200) {
        throw new IOException("Failed : HTTP error code : " + conn.getResponseCode());
    }

    BufferedReader br = new BufferedReader(new InputStreamReader((conn.getInputStream())));
    String output;
    StringBuilder weatherData = new StringBuilder();
    while ((output = br.readLine()) != null) {
        weatherData.append(output);
    }
    output = weatherData.toString();
    ...
}
```

5 Middleware

5.1 Auth

Afin de passer toutes nos requêtes à un couche qui va protéger nos données, il nous faut un couche d'autorisation pour vérifier le session id de chaque requête.

```
HttpServletRequest req = (HttpServletRequest) request;
HttpServletResponse res = (HttpServletResponse) response;

HttpSession session = req.getSession(false);

if (session == null) { //checking whether the session exists
    this.context.log("Unauthorized access request");
    PrintWriter out = res.getWriter();
    res.setContentType("application/json");
    res.setStatus(401);
    out.println("{\"status\": \"error\", \"message\": \"Unauthorized access\"}");
    out.close();
} else {
    // pass the request along the filter chain
    chain.doFilter(request, response);
}
```

5.2 CORS

Par d'ailleurs, lors de la conception du front end, nous sommes rendu compte qu'il y avait des problèmes de CORS vu que les appels API du front end en ReactJS ne viennent pas du même domaine, nous avons du construire également un middleware pour gérer la CORS.

```
HttpServletResponse response = (HttpServletResponse) servletResponse;
HttpServletRequest request = (HttpServletRequest) servletRequest;
```

```
// Set CORS headers
response.setHeader("Access-Control-Allow-Origin", "http://localhost:5173");
response.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS");
response.setHeader("Access-Control-Allow-Headers", "Authorization, Content-Type, XSRF-Token");
response.setHeader("Access-Control-Expose-Headers", "Location");
response.setHeader("Access-Control-Allow-Credentials", "true");

if ("OPTIONS".equalsIgnoreCase(request.getMethod())) {
    response.setStatus(HttpServletResponse.SC_OK);
} else {
    filterChain.doFilter(request, response);
}
```

5.3 Filter mapping

Finalement, voici notre mapping pour toutes les requêtes vers l'application backend :

```
<filter>
    <filter-name>AuthFilter</filter-name>
    <filter-class>com.pc3r.vfarm.middleware.AuthFilter
    </filter-class>
</filter>
<filter>
    <filter-name>CorsFilter</filter-name>
    <filter-class>com.pc3r.vfarm.middleware.CorsFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>CorsFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>AuthFilter</filter-name>
    <url-pattern>/api/*</url-pattern>
</filter-mapping>
```

6 Front end en ReactJS

Pour la partie front end, nous avons construit une application en ReactJS pour notre jeu en ligne. Nous avons utilisé ReactJS bootstrappé par Vite et MUI pour mettre des styles à nos composants, ce qui donne un produit assez satisfaisant.

6.1 Landing Page

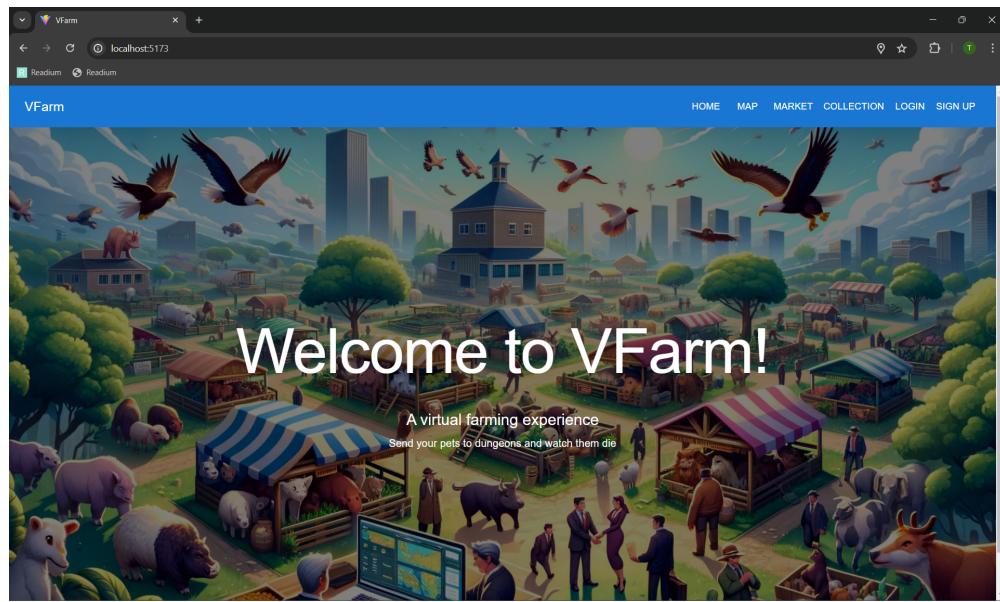
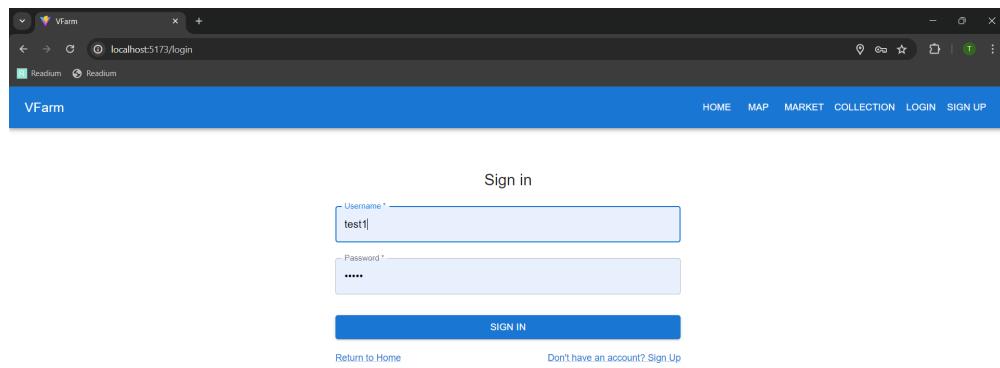


FIGURE 7 – Landing Page

Il s'agit d'une page d'accueil standard avec les boutons en navbar en tête. L'application comprend des pages suivantes

A screenshot of a web browser displaying the VFarm sign up page. The page has a blue header with the VFarm logo and a navigation bar with links for HOME, MAP, MARKET, COLLECTION, LOGIN, and SIGN UP. The main content area is titled "Sign Up" and contains three input fields: "Username *", "Email *", and "Password *". The "Username" field has "test1" entered. The "Email" field is empty. The "Password" field has "*****" entered. Below the fields is a large blue "SIGN UP" button. At the bottom left is a link "Return to Home" and at the bottom right is a link "Already have an account? Sign In".

FIGURE 8 – Sign Up Page



The screenshot shows a web browser window for the VFarm application. The URL is `localhost:5173/login`. The page has a blue header with the VFarm logo. Below the header, there is a "Sign in" form with two input fields: "Username*" containing "test1" and "Password*" containing "*****". A blue "SIGN IN" button is centered below the inputs. At the bottom of the form, there are links for "Return to Home" and "Don't have an account? Sign Up". The top right of the browser window shows standard navigation icons.

FIGURE 9 – Log In Page

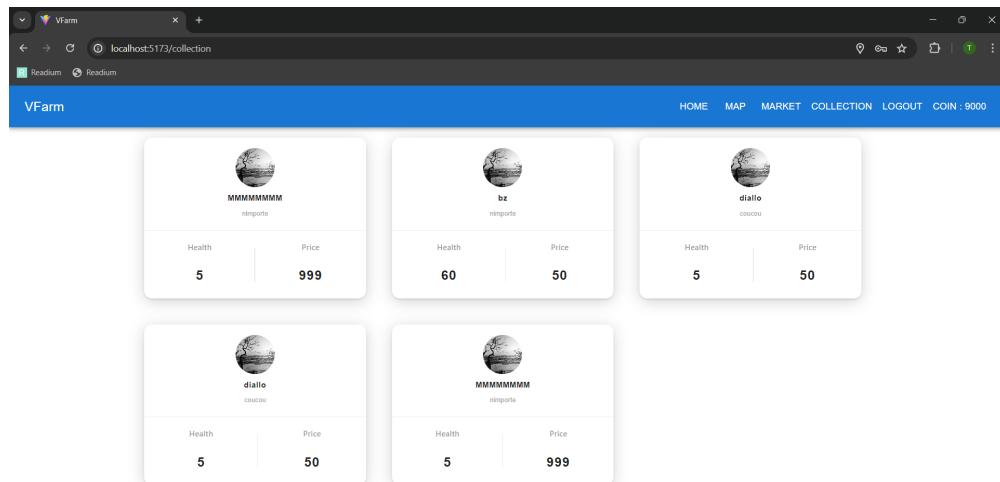


FIGURE 10 – Collection de pets

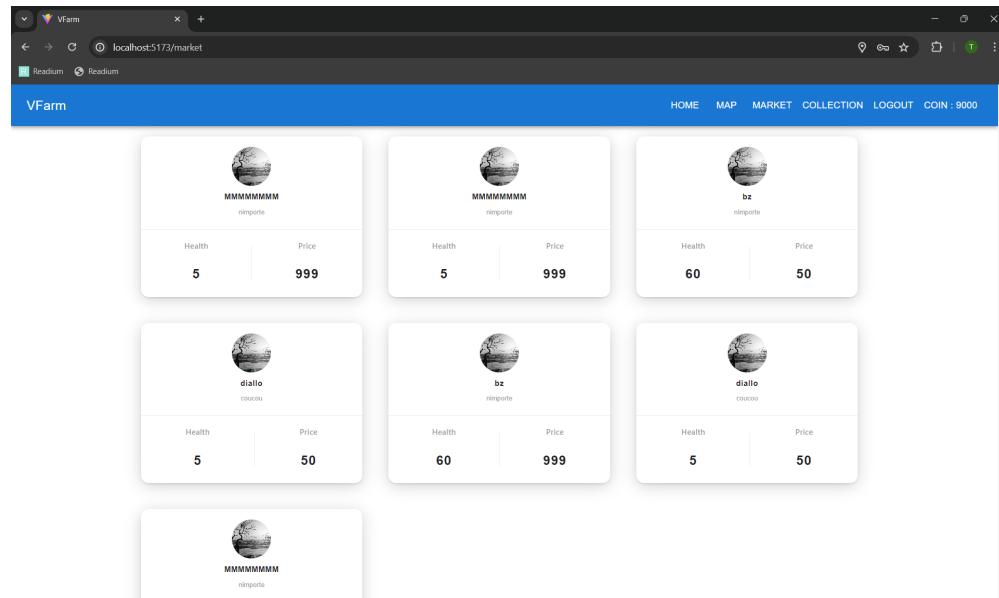


FIGURE 11 – Marché

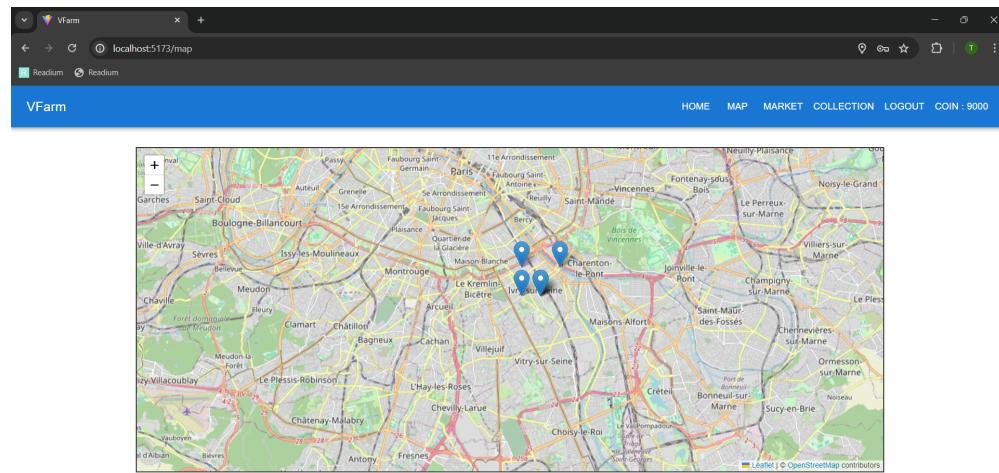


FIGURE 12 – Carte avec les donjons

Le joueur peut acheter les pets sur le marché ou vendre les pets dans sa collection. La page principale de l'application est la page de la carte où il y a des donjons. Afin de initialiser un donjon, il faut cliquer dessus pour verrouiller le donjon pour commencer la composition de l'équipe qui va engager en combat.

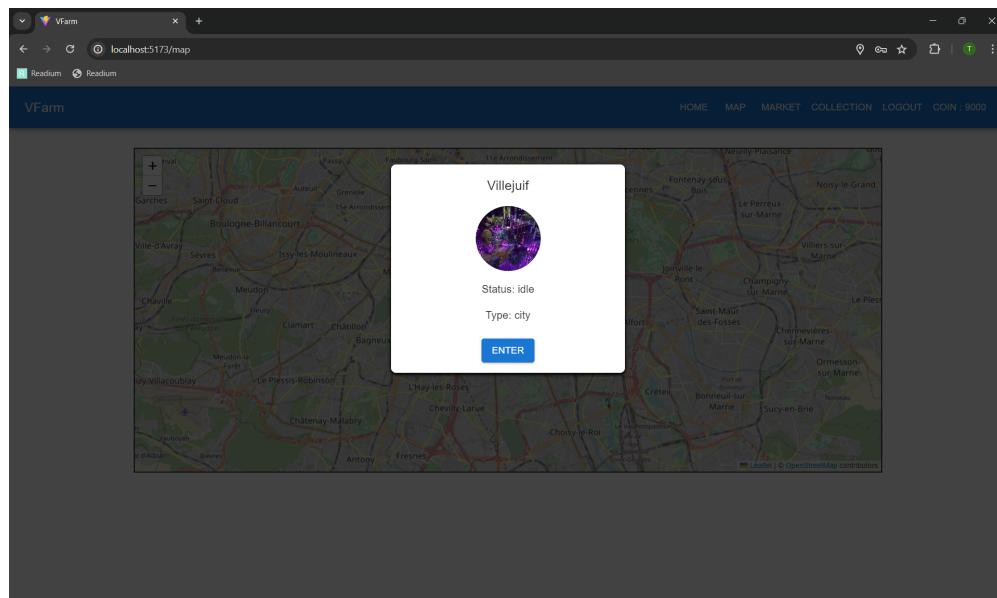


FIGURE 13 – Donjon

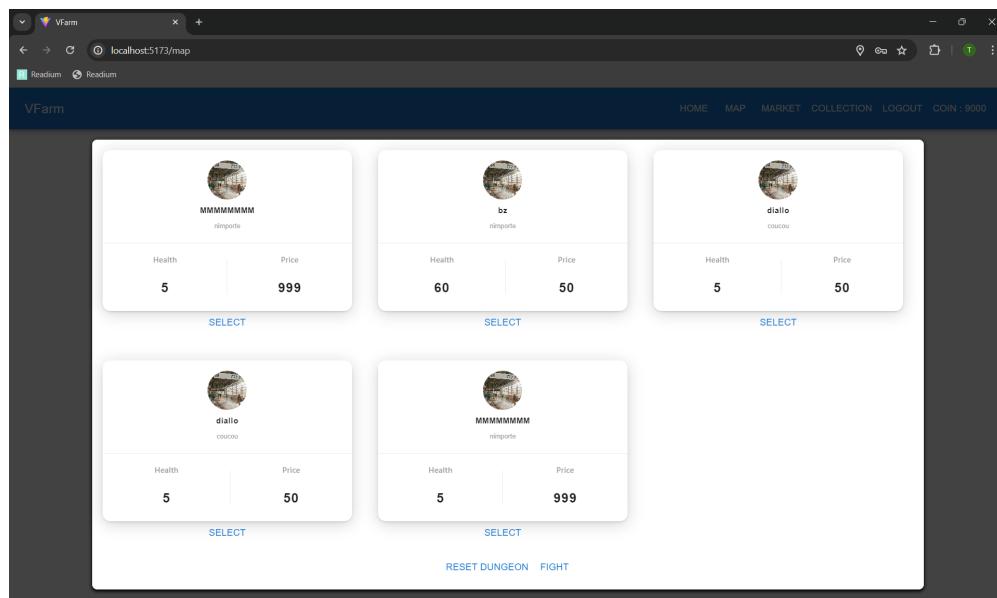


FIGURE 14 – Composition d'équipe

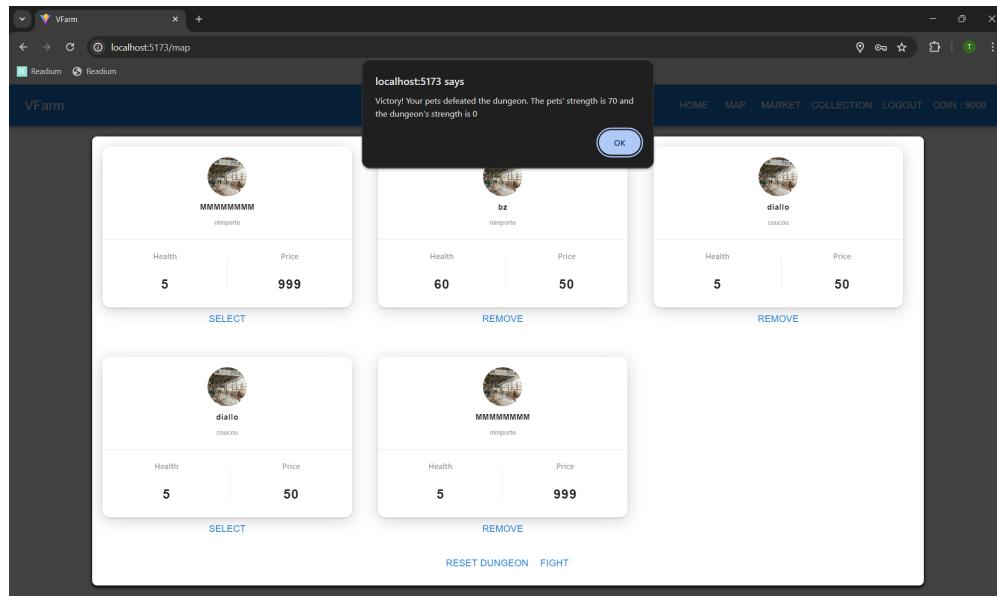


FIGURE 15 – Victoire

7 Conclusion

Ce projet nous a permis d'explorer le développement d'une application web en utilisant un serveur Tomcat. En complément, il a donné une vision claire de l'implémentations des techniques et des algorithmes classiques dans un environnement web. Ce projet nous a donné et appris plusieurs techniques utilisés dans le développement web sécurisé, ce qui nous sera très utile pour notre futur.

7.1 Perspective

Nous envisageons plusieurs améliorations possibles pour le projet :

- **Authentification et Authorisation plus robustes** : Le projet pourrait bénéficier d'un véritable système de Auth qui va garder les session id des utilisateurs.
- **Simulation plus riche des fonctionnalités de l'application** : Nous pourrions implémenter des fonctionnalités supplémentaires pour les utilisateurs, telles que le mélange des attributs pour les donjons et le combat.
- **Système de gestion des utilisateurs** : Nous devrions implémenter un système de gestion des utilisateurs plus avancé, incluant des niveaux de permissions et des profils utilisateurs détaillés.
- **Fonctionnalités avancées** : Il serait bénéfique d'ajouter des fonctionnalités avancées telles que des notifications en temps réel et des tableaux de bord analytiques pour les utilisateurs.
- **Logique du jeu plus diversifié** : Un système de jeu plus robuste avec des générations des pets et des objets.
- **Art et Graphisme** : Le jeu a besoin d'un autre niveau de graphisme.

7.2 Mot de la fin

Nous sommes ravis d'avoir mené à bien ce projet et d'avoir créé un jeu vidéo qui n'est certainement pas de haute qualité mais jouable. Nous sommes fiers du résultat final et espérons qu'il répondra aux attentes du sujet et ainsi de l'UE.