Rapport Technique Kuby

# Introduction

Ce document présente une architecture technique pour un projet comprenant :

- Un backend en Spring Boot (Java) exposant une API REST et MySQL.

- Un client léger développé avec Vue.js et Vuetify.

- Un client lourd utilisant Electron, Vue.js et Vuetify.

Technologies utilisées

• Spring Boot

• JAVA

• Base de données : MySQL

• JWT pour l'authentification

• Swagger pour la documentation API

Contraintes techniques et fonctionnelles

• Utilisation du pattern MVC (Modèle-Vue-Contrôleur)

• Respect des bonnes pratiques RESTful

• Validation des entrées et gestion des erreurs

• Gestion des endpoints pour une adéquation avec les besoins front du client

lourd et du client léger.

MCD

Une image contenant texte, capture d’écran, Logiciel multimédia, conception

Le contenu généré par l’IA peut être incorrect.

UML

Diagram Use Case

Une image contenant texte, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, diagramme, ligne

Le contenu généré par l’IA peut être incorrect.

Diagram Sequence (client léger)

Une image contenant texte, diagramme, capture d’écran, Parallèle

Le contenu généré par l’IA peut être incorrect.

Diagram Sequence (client lourd)

Une image contenant texte, Parallèle, nombre, diagramme

Le contenu généré par l’IA peut être incorrect.

Architecture

Schéma de l’architecture

Une image contenant diagramme, croquis, Dessin technique, Plan

Le contenu généré par l’IA peut être incorrect.

L’API suit une architecture monolithique organisée en plusieurs couches :

•Contrôleurs (Controllers) : Gèrent les requêtes HTTP et orchestrent les

opérations avec les services. Elle met à disposition les endpoints utilisés par

les clients lourd et léger.

• Services (Services) : Contiennent la logique métier.

• Dépôts (Repositories) : Permettent l’interaction avec la base de données avec toutes les fonctions du CRUD (Create,Read,Update,Delete) avec des spécifités de tri sur l’ID, nécessaire lors du filtre sur les différents clients.

Backend : API en Spring Boot

Structure du projet :

Kuby\_API/

├── src/main/java/com/example/kuby\_api/

│ ├── config/

│ ├── controller/

│ ├── model/

│ ├── repository/

│ ├── service/

│ ├── KubyApiApplication.java

│

├── src/main/resources/

│ ├── application.yml

│ ├── data.sql

│

├── build.gradle

Endpoints REST

- /users/

- GET getAllUsers : Récupérer la liste de tous les utilisateurs

- GET getOneUser/{id} : Récupérer un utilisateur spécifique par son ID

- GET getUsersByType/{type} : Récupérer tous les utilisateurs filtrés par type

- POST createUser : Ajouter un nouvel utilisateur

- PUT updateUser/{id} : Modifier les informations d'un utilisateur

- DELETE deleteUser/{id} : Supprimer un utilisateur

- GET {userId}/orders : Ajouter une commande pour un utilisateur

- GET {userId}/adresses : Ajouter une adresse pour un utilisateur

- POST login : Authentification d'un utilisateur

- POST loginAdmin : Authentification d'un administrateur

- POST updatePassword/{id} : Mettre à jour le mot de passe d'un utilisateur

- /adresses/

- GET getAllAdresses : Récupérer toutes les adresses

- GET getOneAdress/{id} : Récupérer une adresse spécifique

- POST createAdress : Ajouter une nouvelle adresse

- POST createUserAdress/{userId} : Associer une adresse à un utilisateur

- PUT updateAdress/{id} : Modifier une adresse

- DELETE deleteAdress/{id} : Supprimer une adresse

- /articles/

- GET getAllArticles : Récupérer tous les articles disponibles

- GET getAllArticlesDetails : Récupérer les détails de tous les articles

- GET getOneArticle/{id} : Récupérer un article spécifique

- POST createArticle/{userId} : Ajouter un nouvel article et l'assigné à un fournisseur

- PUT updateArticle/{id} : Modifier un article existant

- DELETE deleteArticle/{id} : Supprimer un article

- /families/

- GET getAllFamilies : Récupérer toutes les familles de produits

- GET getOneFamily/{id} : Récupérer une famille de produits spécifique

- POST createFamily : Ajouter une nouvelle famille de produits

- PUT updateFamily/{id} : Modifier une famille de produits

- DELETE deleteFamily/{id} : Supprimer une famille de produits

- /grapes/

- GET getAllGrapes : Récupérer tous les cépages

- GET getOneGrape/{id} : Récupérer un cépage spécifique

- POST createGrape : Ajouter un nouveau cépage

- PUT updateGrape/{id} : Modifier un cépage existant

- DELETE deleteGrape/{id} : Supprimer un cépage

- /localisations/

- GET getAllLocalisations : Récupérer toutes les localisations

- GET getOneLocalisation/{id} : Récupérer une localisation spécifique

- POST createLocalisation : Ajouter une nouvelle localisation

- GET getLocalisationByAdressId/{adressId} : Récupérer une localisation liée à une adresse spécifique

- PUT updateLocalisation/{id} : Modifier une localisation existante

- DELETE deleteLocalisation/{id} : Supprimer une localisation

- /orders/

- GET getAllOrders : Récupérer toutes les commandes

- GET getAllOrdersDetails : Récupérer les détails de toutes les commandes

- GET getOneOrder/{id} : Récupérer une commande spécifique

- POST createOrder : Ajouter une nouvelle commande

- PUT updateOrder/{id} : Modifier une commande existante

- DELETE deleteOrder/{id} : Supprimer une commande

- /orderItems/

- GET getAllOrderItems : Récupérer tous les articles de commande

- GET getOrderItemById/{id} : Récupérer un article de commande spécifique

- POST createOrderItems : Ajouter un ou plusieurs articles à une commande

- PUT updateOrderItem/{id} : Modifier un article de commande

- DELETE deleteOrderItem/{id} : Supprimer un article de commande

- /stocks/

- GET getAllStocks : Récupérer tous les stocks

- GET getOneStock/{id} : Récupérer un stock spécifique

- POST createStock : Ajouter un nouveau stock

- PUT updateStock/{id} : Modifier un stock existant

- DELETE deleteStock/{id} : Supprimer un stock

- /stocks/

- GET byArticle/{articleId} : Récupérer le stock d'un article spécifique

- GET byUser/{userId} : Récupérer les stocks associés à un utilisateur

Client Léger : Vue.js & Vuetify

Structure du projet :

Kuby\_ClientLeger/

├── src/

│ ├── api/

│ ├── interfaces/

│ ├── services/

│ ├── endpoints.ts/

│ ├── index.ts/

│ ├── components/

│ ├── views/

│ ├── router/

│ ├── store/

│ ├── utils/

│

├── public/

├── package.json

Communication avec l'API

- api/index.ts gère les appels API avec Axios

- api/endpoints.ts stock les endpoints de l'api

- api/services/ fonctions qui permettent de faire les appels api

- api/intefaces/ définit les interfaces TypeScript pour structurer les données des réponses API

- Stockage des données dans Pinia

- Affichage dynamique avec Vuetify

Client Lourd : Electron, Vue.js & Vuetify

Structure du projet :

Kuby\_ClientLourd/

├── src/

│ ├── api/

│ ├── interfaces/

│ ├── services/

│ ├── endpoints.ts/

│ ├── index.ts/

│ ├── components/

│ ├── views/

│ ├── router/

│ ├── store/

│ ├── utils/

│

├── electron.cjs

├── package.json

Déploiement & Configuration

- Client Lourd : Packagé avec Electron Builder (Windows, Linux)