

DOSSIER PROFESSIONNEL (DP)

Nom de naissance

Nom d'usage

Prénom

Adresse

► Oujjet

► Entrez votre nom d'usage ici.

► Salim

► Hyeres 83400

Titre professionnel visé

Cliquez ici pour entrer l'intitulé du titre professionnel visé.

MODALITE D'ACCES :

- ☒ Parcours de formation
- ☐ Validation des Acquis de l'Expérience (VAE)

Présentation du dossier

Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel.
Ce titre est délivré par le Ministère chargé de l'emploi.

Le DP appartient au candidat. Il le conserve, l'actualise durant son parcours et le présente **obligatoirement à chaque session d'examen.**

Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.

Il est consulté par le jury au moment de la session d'examen.

Pour prendre sa décision, le jury dispose :

1. Des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l'entretien professionnel ou de l'entretien technique ou du questionnement à partir de productions.
2. Du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle.
3. Des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d'un parcours de formation
4. De l'entretien final (dans le cadre de la session titre).

[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels du ministère chargé de l'Emploi]

Ce dossier comporte :

- ▶ Pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ;
- ▶ Un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d'un titre, d'un diplôme, d'un certificat de qualification professionnelle (CQP) ou des attestations de formation ;
- ▶ Une déclaration sur l'honneur à compléter et à signer ;
- ▶ Des documents illustrant la pratique professionnelle du candidat (facultatif)
- ▶ Des annexes, si nécessaire.

Pour compléter ce dossier, le candidat dispose d'un site web en accès libre sur le site.



<http://travail-emploi.gouv.fr/titres-professionnels>

Sommaire

Exemples de pratique professionnelle

Intitulé de l'activité-type n° 1 Développer une application sécurisée

p.

5

- ▶ Intitulé de l'exemple n° 1 Installer et configurer son environnement de travail en fonction du projet
- ▶ Intitulé de l'exemple n° 2 Développement d'interfaces utilisateur (HTML/CSS/JavaScript)
- ▶ Intitulé de l'exemple n° 3 Développement de composants métier (Spring Boot/Java)
- ▶ Intitulé de l'exemple n° 4 Contribuer à la gestion d'un projet informatique

Intitulé de l'activité-type n° 2 Concevoir et développer une application sécurisée organisée en couches

p.

- ▶ Intitulé de l'exemple n° 1 Analyser les besoins et maquetter une application
- ▶ Intitulé de l'exemple n° 2 Définir l'architecture logicielle d'une application
- ▶ Intitulé de l'exemple n° 3 Concevoir et mettre en place une base de données relationnelle
- ▶ Intitulé de l'exemple n° 4 Développer des composants d'accès aux données SQL et NoSQL

Intitulé de l'activité-type n° 3 Préparer le déploiement d'une application sécurisée

p.

- ▶ Intitulé de l'exemple n° 1 Préparer et exécuter les plans de tests d'une application
- ▶ Intitulé de l'exemple n° 2 Préparer et documenter le déploiement d'une application
- ▶ Intitulé de l'exemple n° 3 Contribuer à la mise en production dans une démarche DevOps

Titres, diplômes, CQP, attestations de formation *(facultatif)*

p.

Déclaration sur l'honneur

p.

Documents illustrant la pratique professionnelle *(facultatif)*

p.

Annexes *(Si le RC le prévoit)*

p.

EXEMPLES DE PRATIQUE PROFESSIONNELLE

Activité-type 1 Développer une application sécurisée

Exemple n°1 ► *Installer et configurer son environnement de travail en fonction du projet*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Au début de ma formation CDA, dans le contexte de développement d'applications web sécurisées avec Java Spring Boot, j'ai effectué les opérations suivantes :

1- Configuration de l'IDE et environnement Java :

- . Installation et configuration d'IntelliJ IDEA comme environnement de développement principal
- . Sélection et configuration de la JDK adaptée au projet (OpenJDK 17)
- . Configuration des outils de compilation et d'exécution intégrés
- . Activation du support multi-langages (Java, HTML, CSS, JavaScript) pour le développement full-stack

Mise en place des outils de build et gestion des dépendances :

- . Configuration de Maven comme gestionnaire de dépendances principal (avec Gradle en alternative)
- . Paramétrage de la gestion automatique des versions pour éviter les conflits
- . Configuration des repositories Maven pour la récupération des librairies

Configuration Spring Boot :

- . Utilisation de Spring Initiiiez pour générer la structure de projet
- . Sélection des dépendances Spring Boot nécessaires (Web, Security, Data JPA)
- . Configuration du serveur Tomcat embarqué pour le développement local

Organisation et structuration du projet :

- . Création de l'architecture en couches (Controller, Service, Repository)
- . Mise en place de la structure des tests unitaires et d'intégration
- . Configuration des profils de développement (dev, test, prod)
- . Organisation des ressources statiques pour le front-end

Conditions de réalisation :

Environnement de développement local, projets pédagogiques en formation CDA, travail en autonomie avec validation par le formateur.

2. Précisez les moyens utilisés :

IDE : IntelliJ IDEA Community/Ultimate - environnement de développement intégré avec debugger, refactoring et support multi-langages

JDK : OpenJDK 17 - kit de développement Java pour compilation et exécution

Navigateurs : Firefox avec outils de développement intégrés pour le debugging front-end

Gestionnaires de build et dépendances :

Maven 3.8 - outil principal de gestion des dépendances et compilation

Gradle - alternative plus moderne pour certains projets

Spring Initializr - générateur de structure de projet Spring Boot via web ou plugin IDE

Framework et librairies :

Spring Boot 2.7.x/3.3 - Framework principal pour le backend

Spring Security - gestion de la sécurité et authentification

Spring Data JPA - couche d'accès aux données

Thymeleaf - moteur de Template pour les vues

Bootstrap/CSS3 - Framework CSS pour le responsive design

Base de données :

H2 Database - base de données embarquée pour le développement

MySQL - bases de données pour les environnements de test/production

PhpMyAdmin - interfaces d'administration des bases de données

Outils de versioning et collaboration :

Git - système de contrôle de version

GitHub/ - hébergement des repositories et collaboration

Serveurs et déploiement :

Tomcat embarqué - serveur d'application intégré à Spring Boot

Postman - test des APIs REST

Documentation et ressources :

Documentation officielle Spring.io, Oracle Java docs

Stack Overflow et communautés techniques pour la résolution de problèmes

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

J'ai principalement travaillé en autonomie pour l'installation et la configuration de mon environnement de développement, en me basant sur la documentation officielle et les ressources en ligne.

Le formateur CDA m'a accompagné pour :

Valider mes choix technologiques (IDE, JDK, outils de build)

Résoudre les problèmes de configuration rencontrés

Vérifier que mon environnement était correctement configuré et fonctionnel

M'orienter vers les bonnes pratiques de développement

Mode de travail : Apprentissage en autonomie avec supervision et validation du formateur lors des points d'étape réguliers.

4. Contexte

Nom de l'entreprise, organisme ou association ► **LaPlateforme**

Chantier, atelier, service ► Formation Concepteur Développeur d'Applications (CDA) - Parcours développement Java Spring Boot

Période d'exercice ► Du : **07/10/2024** au : **11/07/2025**

5. Informations complémentaires (facultatif)

Technologies et versions spécifiques utilisées :

- JDK OpenJDK 17, IntelliJ IDEA 2024.1, Maven 3.9.4
- Spring Boot 3.1.5 avec Spring Security 6.1
- Base de données H2 (développement) et MySQL 8.0 (production)

Défis rencontrés et solutions apportées :

- Problème de compatibilité entre versions Spring Boot et Java 17 → Résolu en ajustant les dépendances Maven
- Configuration initiale de Spring Security complexe → Accompagnement formateur pour comprendre les bonnes pratiques

Compétences transversales développées :

- Autonomie dans la recherche de solutions techniques
- Lecture et compréhension de documentation officielle en anglais
- Méthodologie de résolution de problèmes techniques

Outils complémentaires explorés :

- Docker pour la containerisation (initiation)
- Postman pour les tests d'API REST

- Git en ligne de commande pour le versioning

Activité-type 1 Développer une application sécurisée

Exemple n°2 ► CP 2 : Développer des interfaces utilisateur

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Conception et développement d'interfaces web :

Création de maquettes et wireframes pour définir l'ergonomie des pages
Développement de pages HTML5 sémantiques et accessibles
Stylisation avec CSS3 (Flexbox, Grid, responsive design)
Implémentation d'interactions dynamiques avec JavaScript vanilla
Intégration de Framework CSS (Bootstrap) pour accélérer le développement
Création de formulaires avec validation côté client
Optimisation pour différents devices (mobile-first approach)
Tests de compatibilité cross-browser

2. Précisez les moyens utilisés :

Langages et technologies :

HTML5 - Structure sémantique des pages
CSS3 - Stylisation avancée (Flexbox, Grid, animations)
JavaScript ES6+ - Interactions dynamiques et validation
Bootstrap 5 - Framework CSS responsive

Outils de développement :

VS Code/IntelliJ IDEA - Éditeur avec extensions front-end
Chrome DevTools - Debugging et optimisation
Figma/Draw.io - Création de maquettes et wireframes

Intégration back-end :

Thymeleaf - Moteur de Template Java pour l'intégration Spring Boot
Spring MVC - Liaison avec les contrôleurs Java

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

Formateur spécialisé front-end pour la validation des interfaces et l'accompagnement sur les bonnes pratiques UX/UI.

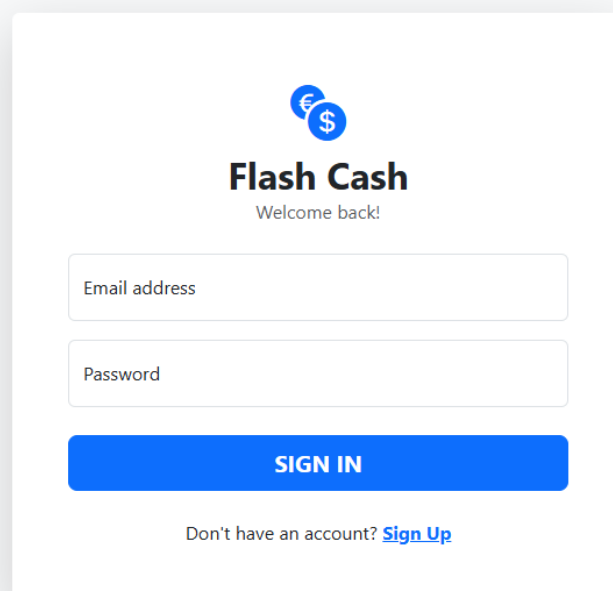
Travail principalement en autonomie avec validation régulière des maquettes et du code produit.

4. Contexte

Nom de l'entreprise, organisme ou association ► **LaPlateforme**

Chantier, atelier, service ► Formation Concepteur Développeur d'Applications (CDA) - Parcours développement Java Spring Boo


Période d'exercice ► Du : 07/10/2024 au : 11/07/2025



The image shows a login form for 'Flash Cash'. At the top, there is a logo consisting of two overlapping circles, one with a Euro symbol (€) and the other with a Dollar symbol (\$). Below the logo, the text 'Flash Cash' is displayed in a bold, sans-serif font, followed by 'Welcome back!' in a smaller font. The form contains two input fields: 'Email address' and 'Password'. Below these fields is a prominent blue button with the text 'SIGN IN' in white, uppercase letters. At the bottom of the form, there is a link that says 'Don't have an account? [Sign Up](#)'.

© 2024 Flash Cash. All rights reserved.

DOSSIER PROFESSIONNEL (DP)

**Flash Cash**
Create your account

First Name

Last Name

Email address


Password


Confirm password


CREATE ACCOUNT


Already have an account? [Sign In](#)


© 2024 Flash Cash. All rights reserved.


 Flash Cash





 Dashboard

 Transactions


 Faire un dépôt

 Friends

 Sign Out




Welcome back, Salim Oujjet!
Here's your financial overview



15000.00 €

Current Account

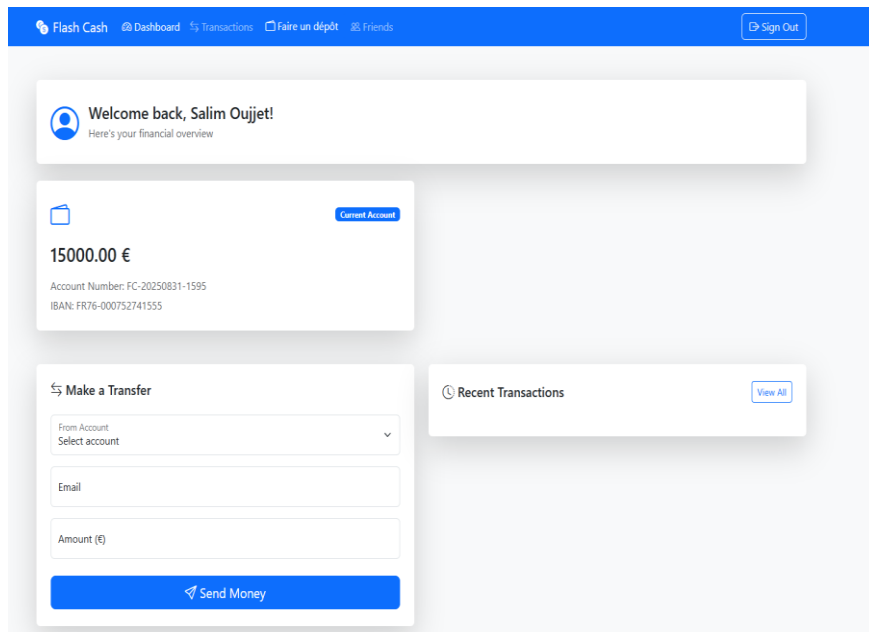
Account Number: FC-20250831-1595
IBAN: FR76-000752741555



Make a Transfer

From Account

DOSSIER PROFESSIONNEL (DP)



```
btn-primary {  
  background-color: #0d6efd;  
  border-color: #0d6efd;  
  transition: all 0.2s;  
}  
  
.btn-primary:hover {  
  background-color: #0b5ed7;  
  border-color: #0a58ca;  
  transform: translateY(-1px);  
}  
  
.card {  
  transition: all 0.3s;  
}  
  
.card:hover {  
  transform: translateY(-5px);  
}  
  
/* Amélioration des alertes */  
.alert {  
  border-radius: 10px;  
}
```

```
<!DOCTYPE html>
<html lang="fr" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flash Cash - Dépôt d'argent</title>

  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.1/font/bootstrap-icons.css">
  <link rel="stylesheet" th:href="@{/index.css}">
</head>
<body class="bg-light">
<!-- Navigation -->
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <div class="container">
    <a class="navbar-brand d-flex align-items-center" th:href="@{/dashboard}">
      <i class="bi bi-currency-exchange me-2"></i>
      Flash Cash
    </a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
```

Page d'accueil (Index) :

J'ai développé une page d'accueil responsive servant de point d'entrée à l'application. Cette page présente les fonctionnalités principales et guide l'utilisateur vers les différentes sections de l'application. L'interface s'adapte automatiquement aux différents formats d'écran (desktop, tablette, mobile) grâce à l'utilisation de Bootstrap et des media queries CSS.

Formulaires interactifs :

J'ai créé des formulaires responsifs en utilisant Bootstrap comme Framework CSS principal, avec très peu de surcharge CSS personnalisée. Ces formulaires s'adaptent parfaitement aux différents supports (ordinateur, tablette et mobile) et intègrent des validations côté client en JavaScript pour améliorer l'expérience utilisateur.

Pages dynamiques avec Thymeleaf :

J'ai développé des pages HTML intégrant Thymeleaf comme moteur de Template, stylisées avec Bootstrap. Thymeleaf, étant le moteur de Template natif de Spring, me permet de charger dynamiquement les informations provenant du back-end Java directement dans mes Template HTML. Cette intégration est rendue possible grâce aux contrôleurs Spring MVC qui exposent les données via des routes définies côté serveur, rendant ces données accessibles et manipulables dans les vues Thymeleaf.

Intégration front-end/back-end :

L'ensemble de ces interfaces s'intègre parfaitement avec l'architecture Spring Boot, permettant une communication fluide entre la couche présentation et la logique métier développée en Java.

Activité-type 1 Développer une application sécurisée

Exemple n°1 ► CP 3 : Développer des composants métier

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Développement de la logique métier avec Spring Boot :

Architecture en couches :

- Création des entités JPA pour modéliser les données métier
- Développement des contrôleurs REST (@RestController) pour exposer les API
- Implémentation des services métier (@Service) contenant la logique applicative
- Création des repositories (@Repository) pour l'accès aux données avec Spring Data JPA

Gestion de la sécurité :

- Configuration de Spring Security pour l'authentification et l'autorisation
- Implémentation de la gestion des rôles et permissions utilisateur
- Sécurisation des Endpoint avec des annotations (@PreAuthorize, @Secured)
- Gestion des sessions et tokens JWT pour l'authentification

Services métier développés :

- Service de gestion des utilisateurs (inscription, connexion, profils)
- Service de gestion des données métiers spécifiques au projet
- Service de validation des données avec Bean Validation
- Gestion des exceptions métier avec @ControllerAdvice

Conditions : Développement en architecture MVC avec Spring Boot, respect des principes SOLID, tests unitaires avec JUnit, intégration avec base de données relationnelle

2. Précisez les moyens utilisés :

Framework et technologies :

- **Spring Boot 3.2** - Framework principal pour l'architecture
- **Spring MVC** - Gestion des contrôleurs et routing
- **Spring Security 6** - Sécurisation de l'application
- **Spring Data JPA** - Abstraction de la couche d'accès aux données

Outils de développement :

- **IntelliJ IDEA** - IDE avec support Spring Boot
- **Maven** - Gestion des dépendances
- **Postman** - Tests des APIs REST
- **JUnit 5** - Tests unitaires

Base de données :

- **H2** - Base de données embarquée pour le développement
- **MySQL** - Base de données pour les environnements de test/production

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

Formateur Java/Spring Boot pour l'accompagnement sur l'architecture des composants métier et la validation des bonnes pratiques de développement.

4. Contexte

Nom de l'entreprise, organisme ou association ► *LaPlateforme*

Chantier, atelier, service ► Module développement de composants métier Spring Boot

Période d'exercice ► Du : *07/10/2024* au : *11/07/2025*

5. Informations complémentaires (facultatif)

Principe de séparation des responsabilités (Single Responsibility Principle)

Injection de dépendances avec Spring IoC

Gestion centralisée des exceptions

Documentation des APIs avec Swagger/OpenAPI

```
import jakarta.persistence.*;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.time.LocalDateTime;
import java.util.Collection;
import java.util.Collections;

@Entity
@Table(name = "users")
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    @NotBlank(message = "Le nom d'utilisateur est obligatoire")
    @Size(min = 3, max = 50, message = "Le nom d'utilisateur doit contenir entre 3 et 50 caractères")
    private String username;

    @Column(nullable = false)
    @NotBlank(message = "L'email est obligatoire")
    @Email(message = "L'email doit être valide")
    private String email;

    @Column(nullable = false)
    @NotBlank(message = "Le mot de passe est obligatoire")
    @Size(min = 8, message = "Le mot de passe doit contenir au moins 8 caractères")
    private String password;

    // ... (autres méthodes) ...
}
```

Annotations JPA (@Entity, @Table, @Id, @GeneratedValue) pour la persistance en base de données

Validation des données avec Bean Validation (@NotBlank, @Email, @Size) pour garantir l'intégrité des données

Intégration Spring Security via l'implémentation de UserDetails pour l'authentification

Gestion des rôles avec un enum Role pour contrôler les autorisations

Audit des données avec createdAt et isActive pour traçabilité et soft delete

Encapsulation avec constructeurs, getters/setters selon les bonnes pratiques Java

```
import com.example.app.entity.User;
import com.example.app.repository.UserRepository;
import com.example.app.dto.UserRegistrationDto;
import com.example.app.exception.UserAlreadyExistsException;
import com.example.app.exception.UserNotFoundException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

@Service
@Transactional
public class UserService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    /**
     * Crée un nouveau utilisateur dans le système
     * @param registrationDto données d'inscription
     * @return utilisateur créé
     * @throws UserAlreadyExistsException si l'utilisateur existe déjà
     */
    public User createUser(UserRegistrationDto registrationDto) {
        // Vérification que l'utilisateur n'existe pas déjà
        if (userRepository.existsByUsername(registrationDto.getUsername()))
            throw new UserAlreadyExistsException("Un utilisateur avec ce nom
```

Annotation @Service pour l'injection de dépendances Spring

Gestion transactionnelle avec @Transactional pour assurer la cohérence des données

Sécurité des mots de passe avec PasswordEncoder et hashage BCrypt

Gestion des exceptions métier avec des exceptions personnalisées

Validation métier (unicité username/email, vérifications de droits)

Implémentation UserDetailsService pour Spring Security

Méthodes CRUD complètes avec soft delete et mise à jour sécurisée


```
import com.example.app.dto.UserRegistrationDto;
import com.example.app.dto.UserResponseDto;
import com.example.app.dto.PasswordChangeDto;
import com.example.app.entity.User;
import com.example.app.service.UserService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "http://localhost:3000")
public class UserController {

    @Autowired
    private UserService userService;

    /**
     * Inscription d'un nouveau utilisateur
     * @param registrationDto données d'inscription
     * @return utilisateur créé
     */
    @PostMapping("/register")
    public ResponseEntity<UserResponseDto> registerUser(@Valid @RequestBody UserRegistrationDto registrationDto) {
        User user = userService.createUser(registrationDto);
        UserResponseDto response = convertToDto(user);
        return new ResponseEntity<>(response, HttpStatus.CREATED);
    }
}
```

Annotations Spring MVC (@RestController, @RequestMapping) pour la gestion des requêtes HTTP

Sécurisation des endpoints avec @PreAuthorize pour contrôler l'accès selon les rôles

Validation des données avec @Valid sur les DTO d'entrée

Gestion des codes de statut HTTP avec ResponseEntity pour des réponses appropriées

CORS configuré pour permettre les appels depuis le front-end

Séparation des responsabilités avec conversion entité/DTO pour ne pas exposer les données sensibles

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Map;

@ControllerAdvice
public class GlobalExceptionHandler {

    /**
     * Gère les exceptions UserNotFoundException
     */
    @ExceptionHandler(UserNotFoundException.class)
    public ResponseEntity<ErrorResponse> handleUserNotFoundException(
        UserNotFoundException ex, WebRequest request) {

        ErrorResponse errorResponse = new ErrorResponse(
            HttpStatus.NOT_FOUND.value(),
            "Utilisateur non trouvé",
            ex.getMessage(),
            LocalDateTime.now(),
            request.getDescription(false)
        );

        return new ResponseEntity<>(errorResponse, HttpStatus.NOT_FOUND);
    }

    /**
```

Annotation @ControllerAdvice pour intercepter toutes les exceptions des contrôleurs

Gestion spécialisée avec @ExceptionHandler pour chaque type d'exception métier

Réponses structurées avec des classes ErrorResponse standardisées

Gestion de la validation avec traitement détaillé des erreurs de Bean Validation

Sécurité avec gestion des AccessDeniedException sans exposer d'informations sensibles

Logging et traçabilité avec timestamp et path de la requête

Codes HTTP appropriés (404, 409, 400, 403, 500) selon le type d'erreur

Activité-type 1 Développer une application sécurisée

Exemple n°1 ► Contribuer à la gestion d'un projet informatique

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Planification et suivi de projet :

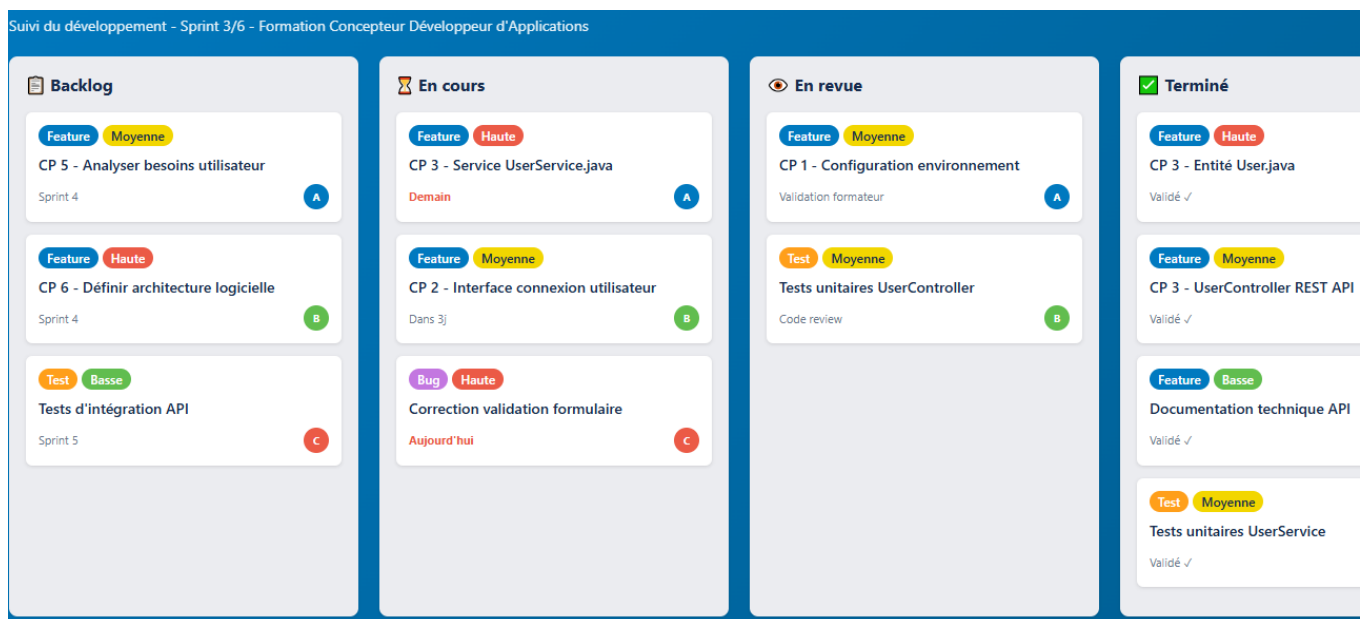
- Participation à l'analyse des besoins fonctionnels et techniques du projet
- Estimation des charges de développement pour les différentes fonctionnalités
- Création et mise à jour du planning de développement avec jalons et livrables
- Suivi de l'avancement des tâches et reporting régulier au formateur/équipe
- Identification et remontée des risques techniques ou de planning

Gestion de la documentation projet :

- Rédaction de spécifications techniques des composants développés
- Documentation du code source et des APIs REST
- Création de guides d'installation et de déploiement
- Tenue à jour du README du projet avec instructions d'utilisation
- Documentation des tests et procédures de validation

Gestion des versions et collaboration :

- Utilisation de Git pour le versioning du code avec stratégie de branches
- Application des conventions de nommage et de commit messages
- Gestion des merge requests et revues de code
- Résolution des conflits de fusion entre branches
- Création de tags pour les versions stables



```
import com.parkit.parkingsystem.constants.Fare;
import com.parkit.parkingsystem.constants.ParkingType;
import com.parkit.parkingsystem.dao.ParkingSpotDAO;
import com.parkit.parkingsystem.dao.TicketDAO;
import com.parkit.parkingsystem.model.ParkingSpot;
import com.parkit.parkingsystem.model.Ticket;
import com.parkit.parkingsystem.util.InputReaderUtil;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Date;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("Tests du service de parking - ParkingService")
class ParkingServiceTest {

    @Mock
    private InputReaderUtil inputReaderUtil;

    @Mock
    private ParkingSpotDAO parkingSpotDAO;

    @Mock
    private TicketDAO ticketDAO;

    private ParkingService parkingService;

    @BeforeEach
    void setUp() {
```

Test 1 : processIncomingVehicle_Car_Success

Description : Ce test vérifie le processus d'entrée réussi d'une voiture dans le parking. Il simule la sélection du type de véhicule (1 = voiture), la saisie de la plaque d'immatriculation, et confirme qu'une place est trouvée, réservée, et qu'un ticket est créé en base de données.

Test 2 : processIncomingVehicle_Bike_Success

Description : Test similaire au précédent mais pour une moto (type 2). Vérifie que le système gère correctement les deux types de véhicules avec leurs places dédiées et leurs tarifs différents.

Test 3 : processIncomingVehicle_NoAvailableSpot

Description : Test du cas d'erreur où aucune place n'est disponible. Vérifie que le système refuse l'entrée (retourne -1) et n'effectue aucune opération de sauvegarde quand le parking est complet.

```
import com.parkit.parkingsystem.constants.Fare;
import com.parkit.parkingsystem.constants.ParkingType;
import com.parkit.parkingsystem.dao.TicketDAO;
import com.parkit.parkingsystem.model.ParkingSpot;
import com.parkit.parkingsystem.model.Ticket;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Date;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("Tests du service de calcul de tarifs - FareCalculatorService")
class FareCalculatorServiceTest {

    @Mock
    private TicketDAO ticketDAO;

    private FareCalculatorService fareCalculatorService;

    @BeforeEach
    void setUp() {
        fareCalculatorService = new FareCalculatorService();
    }

    @Test
    @DisplayName("Calcul tarif voiture - 1 heure de stationnement")
    void calculateFare_Car_OneHour() {
        // Given
```

Tests FareCalculatorServiceTest.java :

Test 1 : calculateFare_Car_OneHour

Description : Test de base du calcul tarifaire pour 1 heure de voiture. Vérifie que le prix correspond exactement au tarif horaire défini dans la constante CAR_RATE_PER_HOUR.

Test 2 : calculateFare_Bike_OneHour

Description : Équivalent du test précédent pour moto. Confirme l'application du tarif BIKE_RATE_PER_HOUR pour une heure complète.

Test 3 : calculateFare_Car_FreeParking_30Minutes

Description : Test de la gratuité des 30 premières minutes pour voiture. Partie intégrante des nouvelles spécifications métier demandées par Will.

Test 4 : calculateFare_Bike_FreeParking_30Minutes

Description : Même test de gratuité mais pour moto, confirmant que la règle s'applique à tous types de véhicules.

Test 5 : calculateFare_Car_RecurringUser_5PercentDiscount

Description : Test de la fonctionnalité de fidélisation avec 5% de réduction. Utilise un mock du TicketDAO pour simuler un utilisateur ayant 3 tickets précédents.

Test 6 : calculateFare_Bike_RecurringUser_5PercentDiscount

Description : Test équivalent de la réduction fidélité pour moto. Vérifie que la réduction s'applique quel que soit le type de véhicule.

Test 7 : calculateFare_Car_45Minutes

Description : Test du calcul proportionnel pour 45 minutes. Vérifie que le système calcule correctement $0.75 \times \text{tarif_horaire}$.

2. Précisez les moyens utilisés :

Outils de gestion de projet :

- **Trello** - Gestion des tâches et suivi du backlog
- **Slack/Teams** - Communication d'équipe et notifications
- **Notion** - Documentation partagée du projet

Outils de versioning et collaboration :

- **Git** - Système de contrôle de version distribué
- **GitHub** - Hébergement des repositories et collaboration

Outils de qualité et tests :

- **JUnit 5** - Framework de tests unitaires Java
- **Mockito** - Mock des dépendances pour les tests

DOSSIER PROFESSIONNEL (DP)

Documentation et communication :

- **Swagger/OpenAPI** - Documentation automatique des APIs
- **Javadoc** - Documentation du code Java
- **Diagrammes UML** - Modélisation avec PlantUML/Draw.io

3. Avec qui avez-vous travaillé ?

Formateur chef de projet pour la validation des livrables et l'accompagnement méthodologique

4. Contexte

Nom de l'entreprise, organisme ou association ► *LaPlateforme*

Chantier, atelier, service ► **Projet fil rouge de gestion de projet informatique**

Période d'exercice ► Du : *07/10/2024* au : *11/07/2025*

5. Informations complémentaires (facultatif)

Activité-type 2

Concevoir et développer une application sécurisée organisée en couches

Exemple n° 2 ► Analyser les besoins et maquetter une application

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Pendant ma formation CDA, nous avons eu pour projet de créer une application web sécurisée de e-commerce en équipe. L'objectif était de concevoir une boutique en ligne complète en suivant une méthodologie structurée d'analyse des besoins et de maquettage.

Analyse des besoins et spécifications : Nous avons commencé par établir un cahier des charges détaillé en recueillant les besoins fonctionnels auprès du formateur (Product Owner) et en définissant les user stories. Cela incluait :

- Page d'accueil avec présentation des produits phares et promotions
- Catalogues produits avec système de filtrage et recherche avancée
- Pages détail produit avec galerie photos, descriptions et avis clients
- Système de panier et processus de commande sécurisé
- Dashboard client avec historique d'achats, wishlist et gestion du profil
- Interface d'administration pour la gestion des produits et commandes
- Système d'authentification et gestion des rôles utilisateur

Maquettage et conception UX/UI : Pour réaliser les maquettes et organiser l'architecture de l'information, nous avons utilisé Figma comme outil principal. Cette approche nous a permis d'avoir une vision claire de l'organisation avant le développement. Figma nous a particulièrement aidés grâce à :

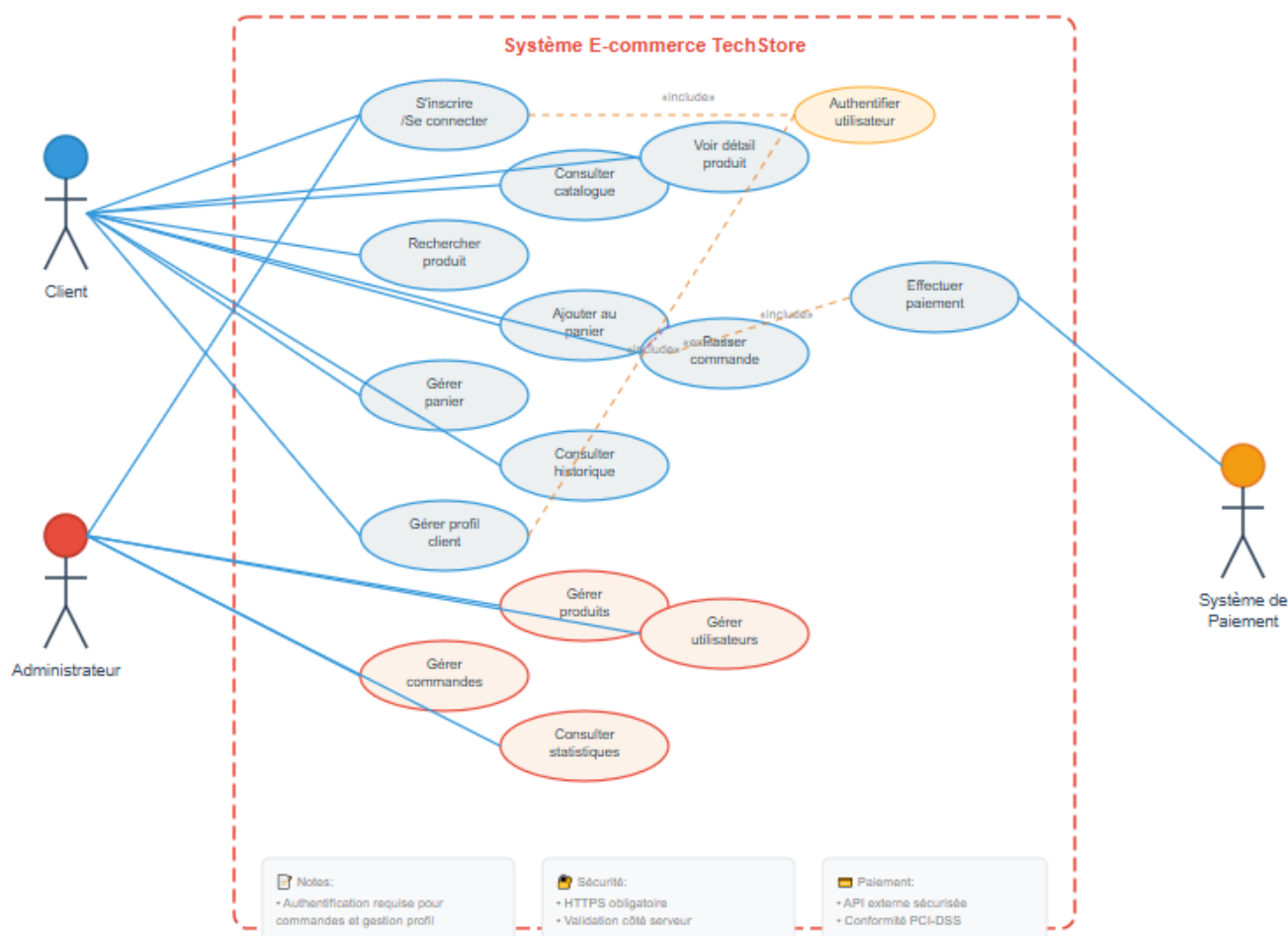
- **Wireframes basse fidélité** pour définir la structure et hiérarchie des informations
- **Maquettes haute fidélité** avec charte graphique cohérente et responsive design
- **Prototypage interactif** montrant les liens entre les pages et le parcours logique
- **Design system** avec composants réutilisables (boutons, formulaires, cards)
- **User journey mapping** pour optimiser l'expérience d'achat

Modélisation et documentation : En parallèle du maquettage, nous avons produit :

- Diagrammes de cas d'usage UML pour formaliser les interactions utilisateur-système
- Personas et scénarios d'usage pour guider nos choix de conception
- Spécifications techniques détaillées pour chaque fonctionnalité
- Critères d'acceptation pour valider l'implémentation des user stories

DOSSIER PROFESSIONNEL (DP)

Diagramme de Cas d'Usage - TechStore E-commerce





TechStore

Accueil

Produits

Categories

Contact



(2)

Connexion

S'inscrire

Les Meilleures Tech

Découvrez notre sélection de produits high-tech

Découvrir

Calques

- Page Accueil
- Header Component
- Logo TechStore
- Navigation Menu
- Hero Section
- Product Grid
- Product Card x4
- Footer Newsletter



Ordinateurs



Smartphones



Audio



Montres



Gaming



Image Produit
300x200px

MacBook Pro 14"

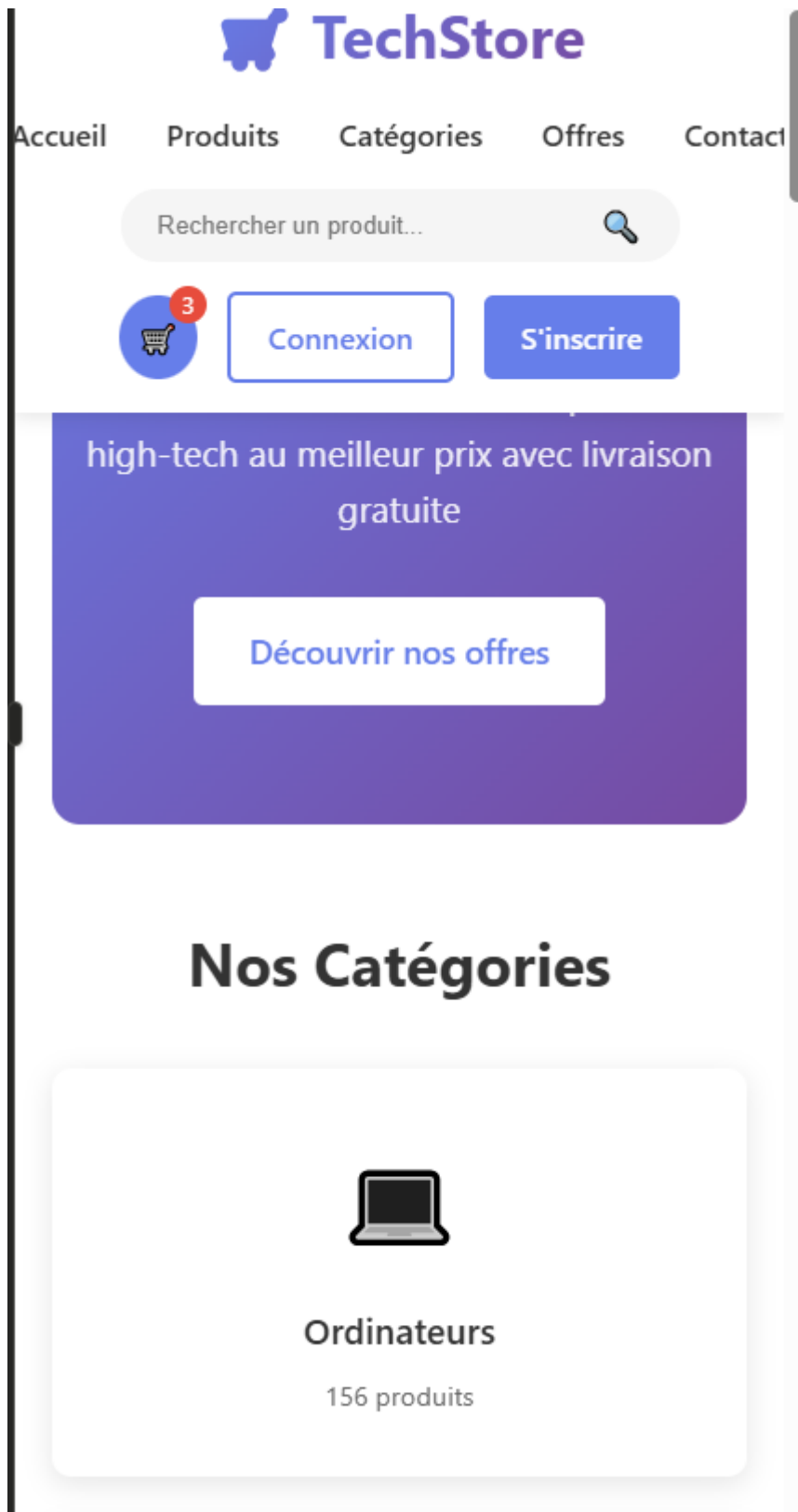
2 499,00 €

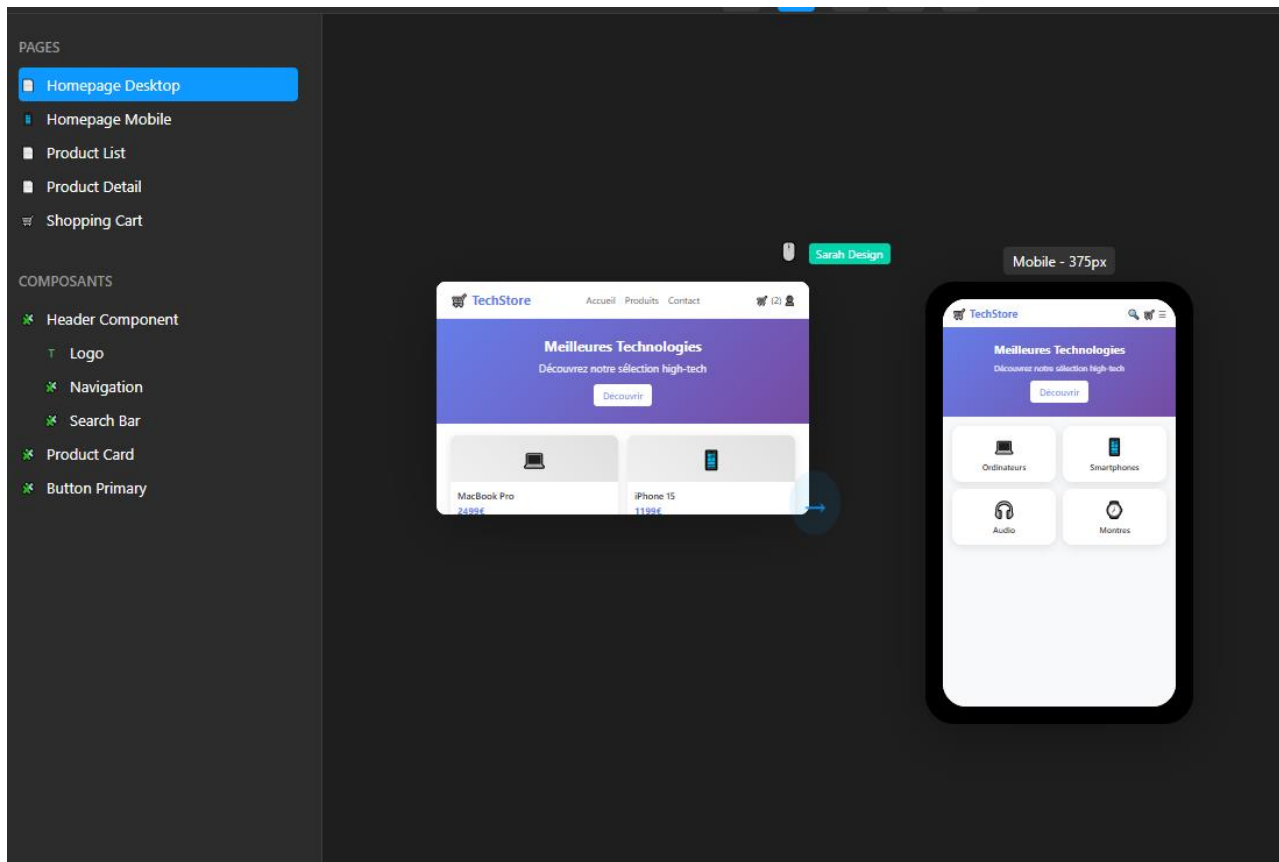


Image Produit
300x200px

iPhone 15 Pro

1 229,00 €





"Prototype Figma Interactif - Interface de Conception Collaborative"

Cette capture présente l'interface Figma avec le prototype interactif du projet TechStore. L'écran montre :

Interface Figma complète :

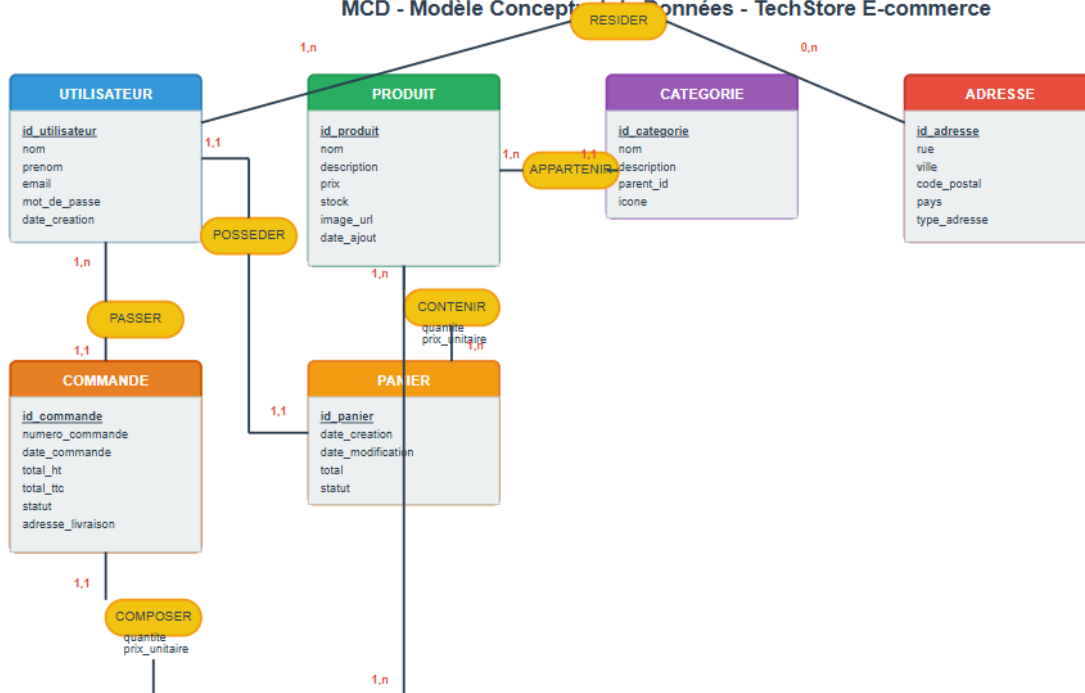
- **Barre d'outils** avec outils de design, bouton "Présenter" et partage collaboratif
- **Panneau des calques** (gauche) montrant l'organisation hiérarchique : pages, composants réutilisables, éléments imbriqués
- **Canevas central** avec frames responsive Desktop (1200px) et Mobile (375px)
- **Panneau propriétés** (droite) avec paramètres d'interaction, design system, et historique des versions

Fonctionnalités de prototypage :

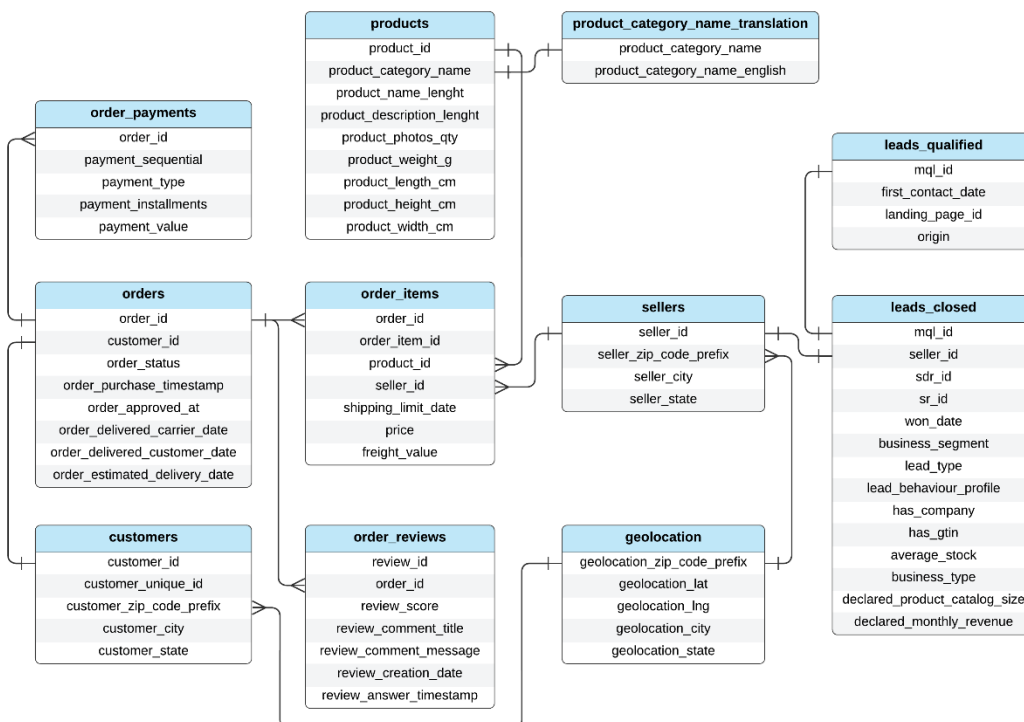
- **Interactions configurées** : On Click → Navigation, On Hover → Preview, Scroll → Header fixe
- **États interactifs** : Default, Hover, Pressed pour tous les composants
- **Flèches de flux** montrant la navigation entre écrans
- **Design system cohérent** : couleurs, typographies, espacements standardisés

DOSSIER PROFESSIONNEL (DP)

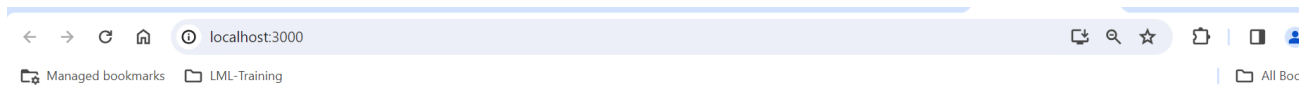
MCD - Modèle Conceptuel des Données - TechStore E-commerce



DATABASE SCHEMA



DOSSIER PROFESSIONNEL (DP)



Ecommerce Management System

ReactStore

Login

Email:

Password:

Login

New Account? [Register Here](#)

Product Management System

ReactStore Logout Product CartItems



ID: 1001
NAME: laptop
PRICE: 80000

View Details

Edit

Delete



ID: 1003
NAME: Tv
PRICE: 20000

View Details

Edit

Delete



ID: 1004
NAME: laptop
PRICE: 40000

View Details

Edit

Delete



ID: 1005
NAME: Tablet
PRICE: 2200

View Details

Edit

Delete



ID: 1006
NAME: Mobile
PRICE: 80000

View Details

Edit

Delete



ID: 1007



ID: 1008



ID: 1009



ID: 10010



ID: 10011

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

Outils de modélisation de données :

- **Looping** - Logiciel français de modélisation MCD/MLD avec génération automatique des scripts SQL
- **Draw.io** - Création de diagrammes conceptuels avec Template base de données intégrés
- **MySQL Workbench** - Modélisation physique et reverse engineering pour les bases MySQL

Méthodes de conception :

- **Méthode Merise** - Approche française structurée (MCD → MLD → MPD)
- **Notation Crow's Foot** - Représentation des cardinalités et relations
- **Dictionnaire des données** - Documentation exhaustive des attributs et contraintes

Ressources et références :

- **Repositories GitHub** - Analyse d'exemples concrets d'e-commerce (ramortegui/e-commerce-db, shreyasbhat132/Designing-Fashion-E-commerce-Database)
- **Documentation technique** - Standards de modélisation et bonnes pratiques
- **Forums spécialisés** - Developpez.net pour résolution de problèmes de modélisation

3. Avec qui avez-vous travaillé ?

pratiques en classe avec travail individuel sur des cas d'étude e-commerce, suivi de **corrections collectives** et **échanges entre apprenants** pour comparer les approches de modélisation.

4. Contexte

Nom de l'entreprise, organisme ou association ► Cliquez ici pour taper du texte.

Chantier, atelier, service ► Cliquez ici pour taper du texte.

Période d'exercice ► Du : 07/10/2024 au : 11/07/2025

5. Informations complémentaires (facultatif)

Activité-type 2 Concevoir et développer une application sécurisée organisée en couches

Exemple n° 1 ► Définir l'architecture logicielle d'une application

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Dans le cadre de mon projet de formation, j'ai conçu et développé une API Java sécurisée destinée à faciliter la gestion des informations critiques entre les casernes de pompiers et les résidents vivant dans des zones sujettes aux inondations. L'objectif était de centraliser les données sensibles des habitants (notamment les enfants, les antécédents médicaux et les situations d'urgence) afin de permettre une intervention rapide et ciblée en cas de crise.

J'ai travaillé en autonomie sur l'ensemble du projet, de la conception à l'implémentation, en respectant une architecture logicielle en couches distinctes et en intégrant des mécanismes de sécurité adaptés à la nature sensible des données.

Les principales tâches réalisées sont :

- Analyse des besoins fonctionnels : identification des entités clés (caserne, résident, enfant, alerte, dossier médical, zone inondable).
- Conception d'une architecture en couches :
 - Couche de présentation (API REST),
 - Couche métier (règles de gestion),
 - Couche de persistance (base de données sécurisée).
- Développement de l'API avec Spring Boot, en respectant les principes de séparation des responsabilités.
- Modélisation des données : création d'entités JPA pour gérer les résidents, leurs enfants, leurs antécédents médicaux, les alertes téléphoniques et les zones à risque.
- Mise en œuvre de la sécurité via Spring Security et JWT (JSON Web Tokens) pour garantir l'authentification et l'autorisation des utilisateurs (pompiers, responsables de caserne, administrateurs).
- Gestion des alertes : implémentation d'un système d'alertes téléphoniques simulées (via API externe ou logs), déclenchées en cas d'urgence (ex. : niveau critique d'inondation).
- Fonctionnalités spécifiques aux inondations : ajout d'un module de suivi des zones à risque, avec possibilité de marquer un résident comme "en danger" ou "sauvé".
- Tests fonctionnels des endpoints avec Postman, notamment pour la création de dossiers médicaux, la génération d'alertes et la mise à jour des statuts d'urgence.

Ce projet a été mené dans un cadre pédagogique, sans accès à des données réelles, mais en respectant les exigences de confidentialité, sécurité et disponibilité liées à la gestion de données sensibles (type données médicales ou personnelles). L'accent a été mis sur la fiabilité et la réactivité de l'application en situation d'urgence.


```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"
    <modelVersion>4.0.0</modelVersion>

    <!-- Utilise Spring Boot Starter Parent pour hériter des configurations standards -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.4.0</version>
        <relativePath/>
    </parent>

    <!-- Informations sur le projet -->
    <groupId>com.safetynetalerts</groupId>
    <artifactId>emergency-services</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>EmergencyServices</name>
    <description>Application de gestion des services d'urgence</description>

    <!-- Version Java utilisée -->
    <properties>
        <java.version>17</java.version>
    </properties>

    <!-- Dépendances nécessaires -->
    <dependencies>
        <!-- Spring Data JPA -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <!-- Spring Web pour construire des APIs REST -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- DevTools pour le rechargement à chaud -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
    </dependencies>
```

```
1  package com.emergencyservices.entity;
2
3  import jakarta.persistence.*;
4  import lombok.AllArgsConstructor;
5  import lombok.Data;
6  import lombok.NoArgsConstructor;
7
8  import java.util.List;
9
10 @Entity
11 @Data
12 @NoArgsConstructor
13 @AllArgsConstructor
14 @Table(name = "fire_station")
15 public class FireStation {
16
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long id;
20
21     @Column(name = "station_number")
22     private Integer stationNumber;
23
24     private String address;
25     private String city;
26     private String zip;
27
28     @ManyToMany
29     @JoinTable(
30         name = "fire_station_personnel",
31         joinColumns = @JoinColumn(name = "fire_station_id"),
32         inverseJoinColumns = @JoinColumn(name = "person_id")
33     )
34     private List<Person> personnel;
35 }
```

DOSSIER PROFESSIONNEL (DP)

```
1 package com.emergencyservices.entity;
2
3 import jakarta.persistence.*;
4 import lombok.Data;
5 import com.vladmihalcea.hibernate.type.json.JsonType;
6 import org.hibernate.annotations.Type;
7 import java.util.List;
8
9 @Data
10 @Entity
11 @Table(name = "medical_record")
12 public class MedicalRecord {
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Long id;
16
17     @Type(JsonType.class)
18     @Column(columnDefinition = "json")
19     private List<String> medications;
20
21     @Type(JsonType.class)
22     @Column(columnDefinition = "json")
23     private List<String> allergies;
24
25     @Type(JsonType.class)
26     @Column(columnDefinition = "json")
27     private String prescription;
28
29     @OneToOne
30     @JoinColumn(name = "person_id", nullable = false)
31     private Person person;
32 }
```

```
1 package com.emergencyservices.entity;
2
3 import jakarta.persistence.*;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 import java.util.Date;
9
10 @Entity
11 @Data
12 @AllArgsConstructor
13 @NoArgsConstructor
14 @Table(name = "person")
15 public class Person {
16
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long id;
20
21     @Column(name = "first_name", nullable = false)
22     private String firstName;
23
24     @Column(name = "last_name", nullable = false)
25     private String lastName;
26
27     @Column(name = "address")
28     private String address;
29
30     @Column(name = "city")
31     private String city;
32
33     @Column(name = "zip")
34     private String zip;
35
36     @Column(name = "phone")
37     private String phone;
```

2. Précisez les moyens utilisés :

- Langage de programmation : Java 17
- Framework principal : Spring Boot (version 3.2.x)
- Sécurité : Spring Security + JWT (authentification stateless)

DOSSIER PROFESSIONNEL (DP)

- Base de données : H2 (en mode embarqué pour le développement), avec possibilité de migration vers PostgreSQL
- ORM : Hibernate / Spring Data JPA
- API REST : Contrôleurs REST, gestion des DTO, pagination, validation des entrées
- Documentation : Swagger / OpenAPI (pour la documentation automatique des endpoints)
- Outils de développement : IntelliJ IDEA, Postman, Git, Maven
- Système d'exploitation : Windows
- Simulateur d'alertes : logs événementiels + endpoints dédiés pour simuler un appel d'urgence

3. Avec qui avez-vous travaillé ?

Ce projet a été réalisé en autonomie, dans le cadre d'un travail individuel de fin de module. J'ai bénéficié d'un accompagnement pédagogique de mes formateurs, avec lesquels j'ai échangé sur les choix d'architecture, la modélisation des données et la sécurisation des endpoints. Aucun travail collaboratif direct n'a été mené, mais des retours ont été intégrés après des revues de code et des présentations orales.

4. Contexte

Nom de l'entreprise, organisme ou association ☐ *LaPlateforme*

Chantier, atelier, service ▶ Cliquez ici pour taper du texte. Projet de développement logiciel / Formation en développement d'applications

Période d'exercice ▶ Du : *07/10/2024* au : *11/07/2025*

5. Informations complémentaires (facultatif)

Activité-type 2

Concevoir et développer une application sécurisée
Organisée en couches

Exemple n° 3 ► Concevoir et mettre en place une base de données relationnelle

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Dans le cadre de mon projet de formation, j'ai conçu et mis en œuvre une base de données relationnelle destinée à supporter une application bancaire simple, permettant la gestion des utilisateurs, des comptes, des transactions et des relations entre utilisateurs (amitiés). Cette base a été intégrée à une application Java développée avec Spring Boot, dans le but de garantir une architecture modulaire, sécurisée et évolutive.

Les principales tâches réalisées sont :

Analyse des besoins fonctionnels : identification des entités clés (utilisateurs, comptes, transactions, amitiés).

Conception du modèle conceptuel : définition des relations entre entités (un utilisateur possède un compte, une transaction implique deux comptes, etc.).

Modélisation du schéma relationnel : création d'un diagramme d'entités-associations (ERD) avec les tables suivantes :

User : stocke les informations des utilisateurs (email, mot de passe, nom d'utilisateur),

Account : contient les détails des comptes (IBAN, solde, lien vers l'utilisateur),

Transaction : enregistre chaque opération financière (montant, date, description, frais, comptes émetteur et récepteur),

Friendship : gère les relations entre utilisateurs (relation bidirectionnelle via une table intermédiaire).

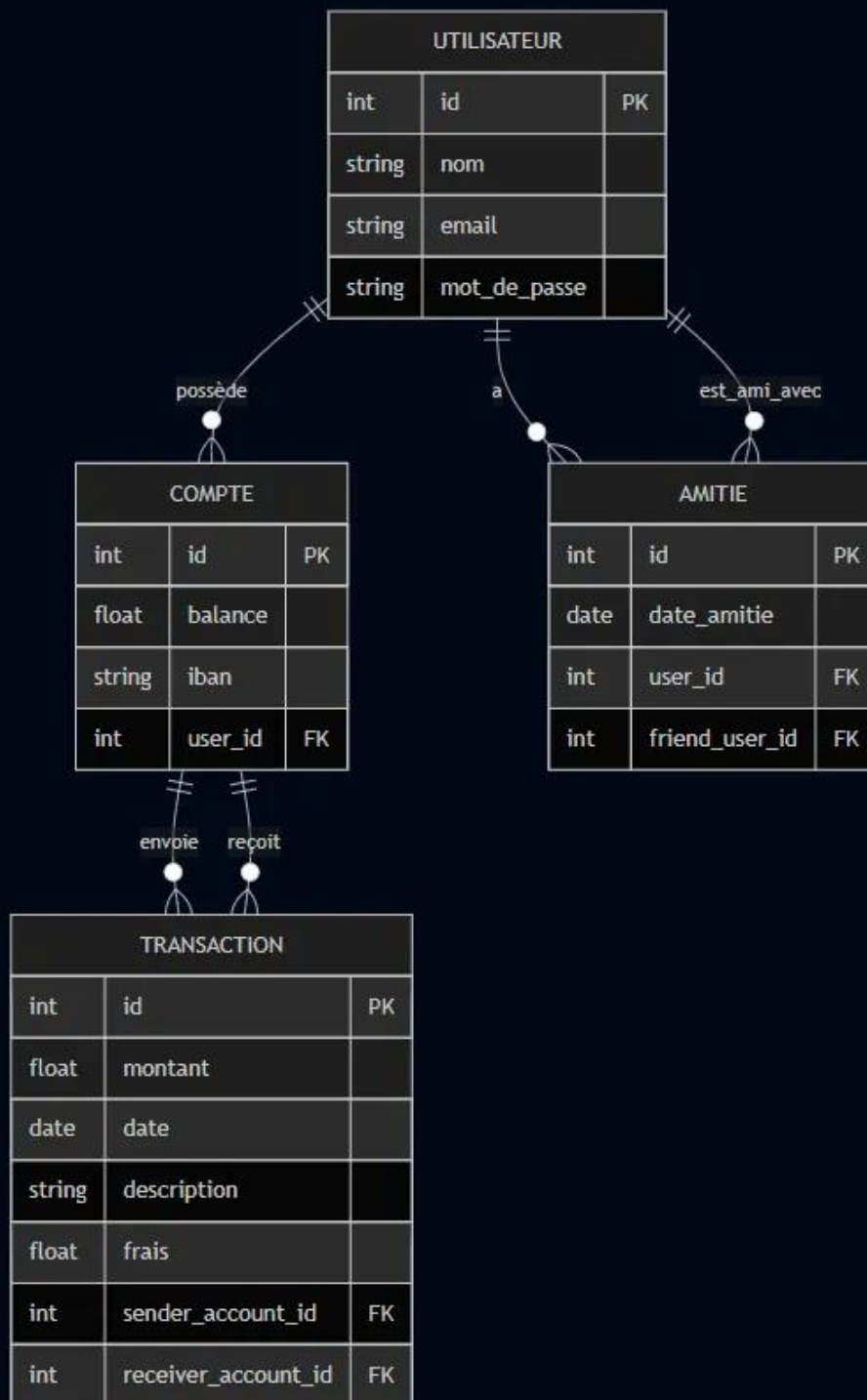
Définition des contraintes : clés primaires, clés étrangères, indices, types de données adaptés (ex. : BIGINT pour les identifiants, VARCHAR(255) pour les chaînes, DATETIME(6) pour la précision milliseconde).

Mise en œuvre technique : utilisation de H2 Database comme base embarquée pour le développement, avec Hibernate pour la gestion automatique de la persistance.

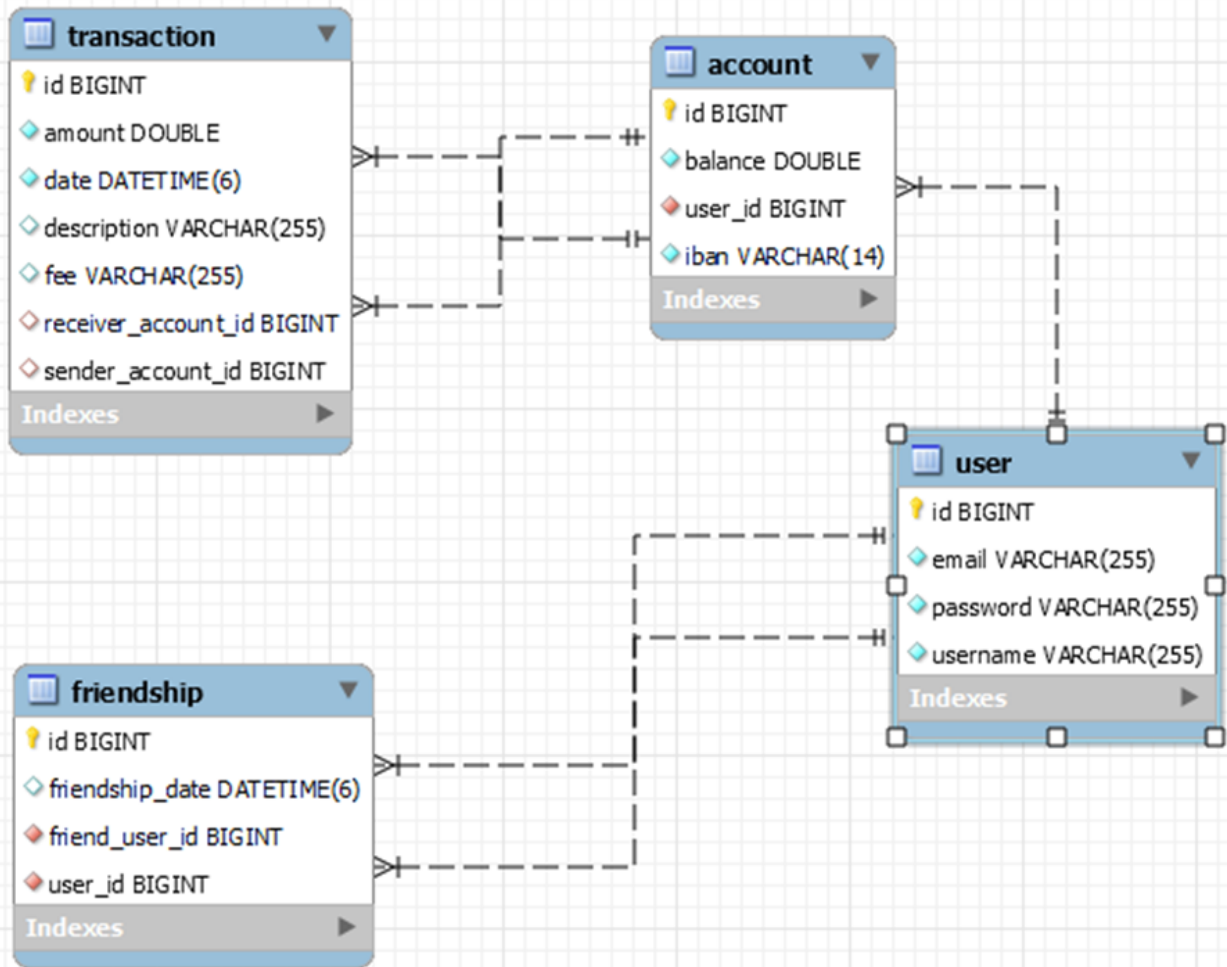
Validation du modèle : tests de création de données, vérification des relations et intégrité référentielle via des requêtes SQL et des tests unitaires.

Ce travail a été réalisé dans un environnement de développement local, sans accès à une infrastructure de production. L'objectif était de maîtriser les bonnes pratiques de conception de bases de données relationnelles, notamment la normalisation, la gestion des relations complexes (many-to-many), et l'intégration avec une couche métier sécurisée.

DOSSIER PROFESSIONNEL (DP)



DOSSIER PROFESSIONNEL (DP)



2. Précisez les moyens utilisés :

Outil de modélisation : Diagramme ERD (via un outil comme dbdiagram.io, MySQL Workbench, ou Draw.io)

Base de données : H2 (mode embarqué, idéal pour le développement rapide)

ORM : Hibernate / Spring Data JPA

Framework : Spring Boot (avec sprint-boot-starter-data-jpa)

Langage : Java 17

Gestion des dépendances : Maven

Outils de développement : IntelliJ IDEA, Postman, Git

Systeme d'exploitation : Windows

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

Ce projet a été réalisé en autonomie, dans le cadre d'un travail individuel de fin de module. J'ai toutefois bénéficié de l'accompagnement de mes formateurs, qui m'ont aidé à valider la cohérence du modèle relationnel et à corriger certaines erreurs de conception (ex. : gestion des relations many-to-many).

4. Contexte

Nom de l'entreprise, organisme ou association ☒ *LaPlateforme*

Chantier, atelier, service ► Développement logiciel / Formation en développement d'applications

Période d'exercice ► Du : *07/10/2024* au : *11/07/2025*

5. Informations complémentaires (facultatif)

User : Table principale contenant les identifiants des utilisateurs (id, email, password, username).
Account : Chaque utilisateur a un compte bancaire (clé étrangère user_id), avec IBAN et solde.
Transaction : Chaque transaction relie deux comptes (sender_account_id et receiver_account_id), avec montant, date, frais et description.
Friendship : Table intermédiaire pour gérer les relations d'amitié entre utilisateurs (relation bidirectionnelle).
Ce modèle est conforme aux bonnes pratiques de conception de base de données :

Normalisation (pas de redondance inutile),
Intégrité référentielle (clés étrangères),
Scalabilité (possibilité d'ajouter des tables comme notification, audit log, etc.).
L'intégration de cette base avec une API REST sécurisée (via JWT) permet de garantir un accès contrôlé aux données sensibles.

Activité-type 2

Concevoir et développer une application sécurisée
Organisée en couches

Exemple n° 1 ► *Développer des composants d'accès aux données SQL et NoSQL*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Pendant ma formation, j'ai conçu et développé une application médicale nommée HealthCare, destinée à permettre aux médecins de gérer leurs patients et de créer des notes de suivi clinique. Le projet a été réalisé dans le cadre d'un travail pédagogique, en environnement local, avec pour objectif de mettre en œuvre une architecture moderne basée sur des microservices et de maîtriser la gestion de données à la fois structurées et non structurées.

L'application est composée de deux microservices principaux :

- Un microservice relationnel utilisant MySQL pour gérer les données structurées (médecins, patients, rendez-vous, etc.).
- Un microservice NoSQL utilisant MongoDB pour stocker les notes médicales, qui sont souvent textuelles, semi-structurées ou volumineuses (ex. : rapports complets, observations détaillées).

Les principales tâches réalisées sont :

- Analyse des besoins fonctionnels : identification des entités clés (médecin, patient, rendez-vous, note médicale).
- Conception d'une architecture en microservices : découpage logique de l'application en deux services indépendants.
- Mise en place de la couche de persistance :
 - Pour le service MySQL : utilisation de JPA/Hibernate pour mapper les classes Java aux tables de la base, automatiser la création des tables, et gérer les opérations CRUD via des repositories Spring Data JPA.
 - Pour le service MongoDB : intégration de Spring Boot Starter Data MongoDB, permettant de manipuler des documents JSON via des interfaces similaires à celles de JPA.
- Création de modèles métier : définition des entités Java (ex. : Doctor, Patient, MedicalNote) et de leurs relations.
- Implémentation de la logique métier : gestion des droits d'accès (un médecin ne peut voir que ses propres patients), création et consultation des notes.
- Tests des composants d'accès aux données : validation des requêtes simples (CRUD) et de la gestion des relations entre entités via des tests unitaires et des appels API (Postman).

Ce projet a été mené en autonomie, dans un environnement de développement local, avec l'objectif de valider les compétences en gestion de données SQL et NoSQL, ainsi qu'en conception d'architecture modulaire.

2. Précisez les moyens utilisés :

- Langage de programmation : Java 17
- Framework principal : Spring Boot (version 3.x)
- Base de données relationnelle : MySQL
- Base de données NoSQL : MongoDB
- ORM / ODM :
 - Hibernate (via spring-boot-starter-data-jpa) pour MySQL,
 - Spring Data MongoDB (via spring-boot-starter-data-mongodb) pour MongoDB.
- Gestion des dépendances : Maven
- Outils de développement : IntelliJ IDEA, Postman, Git, Docker (optionnel)
- Système d'exploitation : Windows
- Outils de modélisation : Diagrammes UML (classes, séquences), diagrammes d'entités
- Documentation : Swagger/OpenAPI pour l'API REST

```
[
  {
    id: 1,
    firstName: "John",
    lastName: "Doe",
    birthday: "1990-01-01",
    gender: "Male",
    address: "456 Updated St",
    phone: "+1234567890",
    email: "john.doe.updated@example.com"
  },
  {
    id: 2,
    firstName: "John",
    lastName: "Doe",
    birthday: "1990-01-01",
    gender: "Male",
    address: "456 Updated St",
    phone: "+1234567890",
    email: "john.doe.updated@example.com"
  }
]
```

DOSSIER PROFESSIONNEL (DP)

```
Rename usages

package fr.santesysteme.patient.enums;

public enum Role { 4 usages

    USER, 1 usage

    ADMIN, no usages
}
```

localhost:3081/webapp/

Système de Santé Accueil Patients admin

Tableau de bord

Accueil

Patients

Gérez vos patients, consultez leurs dossiers et suivez leur historique médical.

[Voir les patients](#)

Rendez-vous

Planifiez et gérez les rendez-vous avec vos patients.

[Gérer les rendez-vous](#)

Traitements

Consultez et suivez les traitements prescrits à vos patients.

[Voir les traitements](#)

Activités récentes

- Patient ajouté**
Jean Dupont a été ajouté à la base de patients. il y a 3 jours
- Rendez-vous planifié**
Rendez-vous avec Marie Martin le 15 Mars à 14h00. il y a 4 jours
- Dossier mis à jour**
Le dossier médical de Paul Bernard a été mis à jour. il y a 1 semaine

Statistiques des patients

Nouveaux patients

Rendez-vous à venir

Patient	Date	Heure
Sophie Petit	12/03/2025	09:30
Michel Thomas	14/03/2025	11:00
Amélie Roux	15/03/2025	14:45

```
@Data 16 usages  Esteban
@NoArgsConstructor
@AllArgsConstructor
@Document(collection = "notes")
public class Note {

    @MongoId
    private String id;

    private String title;

    private String content;

    private String patientId;
}
```

```
@Autowired
private NoteRepository noteRepository;

@Autowired
public MsPatientFeignClient msPatientFeignClient;

public List<Note> findAll() { return noteRepository.findAll(); }

public ResponseEntity<String> save(Note note) { 1 usage  Esteban
    try {
        ResponseEntity<String> response = msPatientFeignClient.patientExists(note.getPatientId());

        Note savedNote = noteRepository.save(note);
        return ResponseEntity.ok(body: "Note saved");
    } catch (feign.FeignException.NotFound e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
            .body(null);
    } catch (feign.FeignException e) {
        return ResponseEntity.status(e.status())
            .body("Error saving note");
    } catch (Exception e) {
        return ResponseEntity.badRequest().body("Error saving note");
    }
}
```

DOSSIER PROFESSIONNEL (DP)

main

1 Branch

0 Tags

Go to file

Code

salim-OU Update api-gateway.yml

9959262 · 4 months ago

76 Commits

api-gateway.yml	Update api-gateway.yml	4 months ago
eureka-server.yml	Update eureka-server.yml	5 months ago
ms-assessment.yml	Update ms-assessment.yml	5 months ago
ms-notes.yml	Update ms-notes.yml	5 months ago
ms-patients.yml	Update ms-patients.yml	5 months ago
ms-user.yml	Update ms-user.yml	5 months ago
ms-webapp.yml	Update ms-webapp.yml	5 months ago

About

No description, website, or topics

Activity

0 stars

1 watching

0 forks

Report repository

Releases

No releases published

Packages

No packages published

```
@Entity
@Getter
@Setter
@ToString
@AllArgsConstructor
@NoArgsConstructor
public class Patient {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String firstName;
    private String lastName;
    private LocalDate birthday;
    private String gender;
    private String address;
    private String phone;
    private String email;
}
```

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

Ce projet a été réalisé en autonomie, dans le cadre d'un travail individuel de fin de module. J'ai bénéficié de l'accompagnement de mes formateurs, qui m'ont aidé à valider les choix techniques (notamment l'utilisation de deux bases de données distinctes) et à corriger certaines erreurs de conception (ex. : gestion des clés étrangères entre microservices). Aucun travail collaboratif direct n'a été mené, mais des retours ont été intégrés après des revues de code.

4. Contexte

Nom de l'entreprise, organisme ou association ► *LaPlateforme*

Chantier, atelier, service ► Projet de développement logiciel / Module de base de données

Période d'exercice ► Du : *07/10/2024* au : *11/07/2025*

5. Informations complémentaires (facultatif)

Le projet HealthCare illustre parfaitement la combinaison de deux types de bases de données selon leur usage :

- MySQL pour les données fortement structurées (relation médecin-patient),
- MongoDB pour les données flexibles et riches (notes médicales, historiques d'évolution).

Des images accompagnent ce rapport, montrant :

- L'utilisation de JPA/Hibernate pour générer automatiquement les tables à partir des entités Java,
- L'implémentation de repositories Spring Data JPA pour les opérations CRUD,
- Des exemples de requêtes personnalisées (ex. : liste des patients d'un médecin),
- L'intégration de MongoDB via des MongoRepository pour gérer les documents MedicalNote.

Cette approche permet une scalabilité accrue, une flexibilité accrue dans la gestion des données, et une meilleure performance pour les opérations sur des volumes importants de données textuelles.

Activité-type 3 Préparer le déploiement d'une application sécurisée

Exemple n° 1 ▶ Préparer et exécuter les plans de tests d'une application

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Dans le cadre de mon projet de formation intitulé "Parking System", l'un de mes objectifs principaux était d'assurer une haute couverture des tests unitaires sur le code applicatif. Pour y parvenir, j'ai intégré JaCoCo (Java Code Coverage), un plugin Maven permettant de mesurer précisément le taux de couverture du code par les tests.

J'ai développé les tests à l'aide des frameworks JUnit 5 et Mockito, en mettant l'accent sur la validation des composants métier et des services métiers. Grâce à JaCoCo, j'ai pu générer un rapport détaillé mettant en évidence les parties du code couvertes — ou non — par les tests. Ce rapport m'a permis d'identifier les lignes et les branches de code non testées, afin de renforcer la qualité et la fiabilité de l'application.

Cette démarche m'a aidé à améliorer progressivement la couverture, à corriger des cas non prévus et à garantir un code plus robuste et maintenable.

```
import com.parkit.parkingsystem.dao.ParkingSpotDAO;
import com.parkit.parkingsystem.dao.TicketDAO;
import com.parkit.parkingsystem.model.ParkingSpot;
import com.parkit.parkingsystem.model.Ticket;
import com.parkit.parkingsystem.util.InputReaderUtil;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Date;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("Tests du service de parking - ParkingService")
class ParkingServiceTest {

    @Mock
    private InputReaderUtil inputReaderUtil;

    @Mock
    private ParkingSpotDAO parkingSpotDAO;

    @Mock
    private TicketDAO ticketDAO;

    private ParkingService parkingService;

    @BeforeEach
    void setUp() {
        parkingService = new ParkingService(inputReaderUtil, parkingSpotDAO, ticketDAO);
    }

    @Test
```


java

```
1 @Test
2 void shouldCalculateFeeForLessThanOneHour() {
3     when(clock.now()).thenReturn(LocalDate.of(2024, 10, 5, 10, 0));
4     ParkingService service = new ParkingService(clock);
5     double fee = service.calculateFee(Duration.ofMinutes(45));
6     assertEquals(2.0, fee);
7 }
```

```
package com.parkit.parkingsystem.service;

import com.parkit.parkingsystem.constants.Fare;
import com.parkit.parkingsystem.constants.ParkingType;
import com.parkit.parkingsystem.dao.TicketDAO;
import com.parkit.parkingsystem.model.ParkingSpot;
import com.parkit.parkingsystem.model.Ticket;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Date;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("Tests du service de calcul de tarifs - FareCalculatorService")
class FareCalculatorServiceTest {

    @Mock
    private TicketDAO ticketDAO;

    private FareCalculatorService fareCalculatorService;

    @BeforeEach
    void setUp() {
        fareCalculatorService = new FareCalculatorService();
    }

    @Test
    @DisplayName("Calcul tarif voiture - 1 heure de stationnement")
    void calculateFare_Car_OneHour() {
        // Given
```

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

- Langage : Java 17
- Framework : Spring Boot (version 3.x)
- Tests unitaires : JUnit 5, Mockito
- Couverture de code : JaCoCo (plugin Maven)
- Gestion des dépendances : Maven
- Outils de développement : IntelliJ IDEA, Git
- Système d'exploitation : Windows 11
- Génération de rapports : Rapport HTML généré par JaCoCo (target/site/jacoco/index.html)
- Contrôle de version : Git (dépôt local)

3. Avec qui avez-vous travaillé ?

Ce projet a été réalisé en autonomie, dans le cadre d'un travail individuel de fin de module. J'ai bénéficié de l'accompagnement de mes formateurs, qui m'ont aidé à comprendre les métriques de couverture (lignes, branches, méthodes) et à interpréter correctement les rapports JaCoCo. Des points réguliers ont été effectués pour valider l'approche de test et l'évolution de la couverture.

4. Contexte

Nom de l'entreprise, organisme ou association ► *LaPlateforme*

Chantier, atelier, service ► Cliquez ici pour taper du texte. Préparer et exécuter les plans de tests d'une application

Période d'exercice ► Du : *07/10/2024* au : *11/07/2025*

5. Informations complémentaires (facultatif)

Activité-type 3 Préparer le déploiement d'une application sécurisée

Exemple n° 10 ► Préparer et documenter le déploiement d'une application

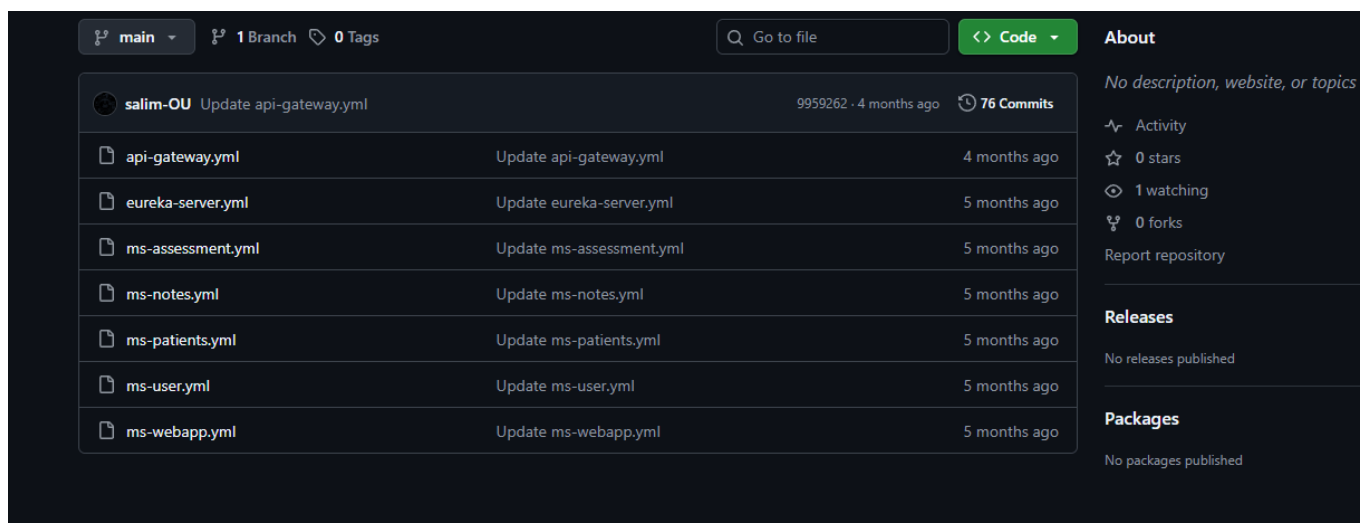
1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Dans le cadre de mon projet de formation "MyHealthTracker", j'ai développé une application médicale modulaire basée sur une architecture microservices. L'objectif était de permettre aux médecins de gérer leurs patients, d'enregistrer des notes cliniques et de suivre les antécédents médicaux, tout en garantissant une mise en production fiable, automatisée et sécurisée.

Pour assurer une gestion efficace du déploiement, j'ai mis en œuvre une stratégie complète intégrant :

Spring Cloud Config pour la gestion centralisée des configurations,
Docker Compose pour le déploiement local et homogène de tous les services,
Jib (plugin Maven) pour la création automatique des images Docker,
Spring Boot Actuator pour la vérification de l'état des services.

Le projet est composé de plusieurs microservices, comme illustré dans l'image ci-dessous :



Chaque microservice (ex. : gestion des utilisateurs, suivi des rendez-vous, historique médical) est indépendant et déployable séparément.

Pour gérer les configurations selon les environnements (dev, test, prod), j'ai utilisé Spring Cloud Config Server, qui lit les fichiers de configuration depuis un repository GitHub centralisé. Chaque microservice récupère automatiquement sa configuration via une API REST, en fonction de son nom et de l'environnement actuel.

Pour le déploiement, j'ai utilisé Docker Compose avec un fichier docker-compose.yml structuré pour :

Démarrer en premier le Config Server et la base de données MySQL,
Attendez que ces services soient prêts (grâce à Spring Boot Actuator),
Puis lancer les microservices principaux.

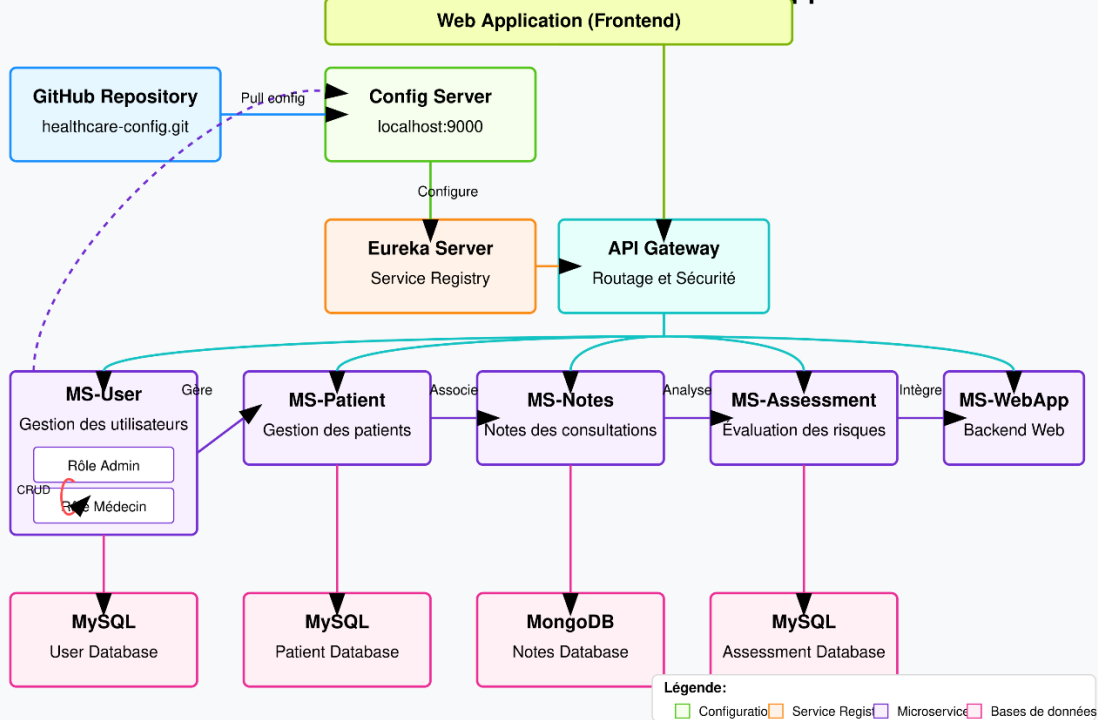
Les images Docker sont générées automatiquement par Jib, un plugin Maven qui s'intègre directement dans le cycle de build. Avant de créer l'image, Jib exécute les tests unitaires et d'intégration, garantissant que seul un code valide est déployé.

Enfin, j'ai rédigé un fichier README.md complet, expliquant :

La structure du projet,
La configuration des environnements,
Les commandes pour lancer le déploiement (docker-compose up),
Les URLs des endpoints Swagger pour chaque service.
Ce système me permet de :

Garantir que chaque microservice démarre avec la bonne configuration,
Automatiser le test et le build avant le déploiement,
Faciliter la transition entre environnements (dev → test → prod),
Documenter clairement le processus de déploiement pour toute équipe technique.

Architecture Spring Cloud - Healthcare App



Flux d'information:

1. Les utilisateurs accèdent à l'application via la WebApp (frontend)
2. L'API Gateway assure le routage et la sécurité des requêtes vers les microservices appropriés
3. MS-Assessment analyse les notes des patients pour évaluer les risques médicaux

2. Précisez les moyens utilisés :

Langage : Java 17

Framework : Spring Boot (version 3.x)

Architecture : Microservices

Gestion de configuration : Spring Cloud Config + GitHub

Conteneurs : Docker, Docker Compose

Build & Images : Maven + Jib

Outils de monitoring : Spring Boot Actuator

Outils de développement : IntelliJ IDEA, Git, Postman

Documentation : Markdown (README.md)

Système d'exploitation : Windows /

3. Avec qui avez-vous travaillé ?

Ce projet a été réalisé en autonomie, dans le cadre d'un travail individuel de fin de module. J'ai bénéficié de l'accompagnement de mes formateurs, qui m'ont aidé à valider la configuration de Spring Cloud Config, l'ordre de démarrage des conteneurs et l'intégration de Jib dans le pipeline de build.

DOSSIER PROFESSIONNEL (DP)

Aucun travail collaboratif direct n'a été mené, mais des retours ont été intégrés après des revues techniques.

4. Contexte

Nom de l'entreprise, organisme ou association ► *LaPlateforme*

Chantier, atelier, service ► Projet de développement logiciel / Module DevOps & déploiement

Période d'exercice ► Du : *07/10/2024* au : *11/07/2025*

5. Informations complémentaires (facultatif)

Mettre en place une infrastructure de déploiement automatisée,
Gérer les configurations multi-environnements,
Utiliser des outils modernes de containerisation et de CI/CD,
Documenter clairement le processus de déploiement.

🔗 Exemples de fichiers inclus dans le projet :

config-server/src/main/resources/application.yml

docker-compose.yml

pom.xml (avec plugin Jib)

README.md (guide de déploiement)

DOSSIER PROFESSIONNEL (DP)

Activité-type 3 Cliquez ici pour entrer l'intitulé de l'activité

Exemple n° 1 ► Cliquez ici pour entrer l'intitulé de l'exemple

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

2. Précisez les moyens utilisés :

3. Avec qui avez-vous travaillé ?

4. Contexte

Nom de l'entreprise, organisme ou association ► Cliquez ici pour taper du texte.

Chantier, atelier, service ► Cliquez ici pour taper du texte.

Période d'exercice ► Du : Cliquez ici au : Cliquez ici

5. Informations complémentaires (facultatif)

Titres, diplômes, CQP, attestations de formation

(facultatif)

Intitulé	Autorité ou organisme	Date
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.

DOSSIER PROFESSIONNEL (DP)

Déclaration sur l'honneur

Je soussigné(e) [prénom et nom] [Cliquez ici pour taper du texte.](#) ,
déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis
l'auteur(e) des réalisations jointes.

Fait à [Cliquez ici pour taper du texte.](#) le [Cliquez ici pour choisir une date](#)
pour faire valoir ce que de droit.

Signature :

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé

Cliquez ici pour taper du texte.

DOSSIER PROFESSIONNEL (DP)

ANNEXES

(Si le RC le prévoit)