Projet 3 : Backend et sécurité

CHRISTOPHE PIERRÈS,
LE MERCREDI 28 MAI 2025, DURÉE : 16MN

JORDI CHARPENTIER

DIFFICULTÉS RENCONTRÉES

- Sur le projet lui-même, pas vraiment
  - Du fait de l'architecture claire et simple
  - Quelques subtilités :
    - Avec mapstruct : mapping sur Rental (picture)
    - Avec lombok : dans certains cas d'usage (entités avec @MappedSuperclass)

- Difficultés anecdotiques

DIFFICULTÉS RENCONTRÉES

- Sur le projet lui-même pas vraiment
  - Du fait de l'architecture claire et simple
  - Quelques subtilités :
    - Avec mapstruct : mapping sur Rental (picture)
    - Avec lombok dans certains cas d'usage (entités avec @MappedSuperclass)

- Difficultés anecdotiques

DU TABLEAU D'ANALYSE A LA DOC OPENAPI/SWAGGER

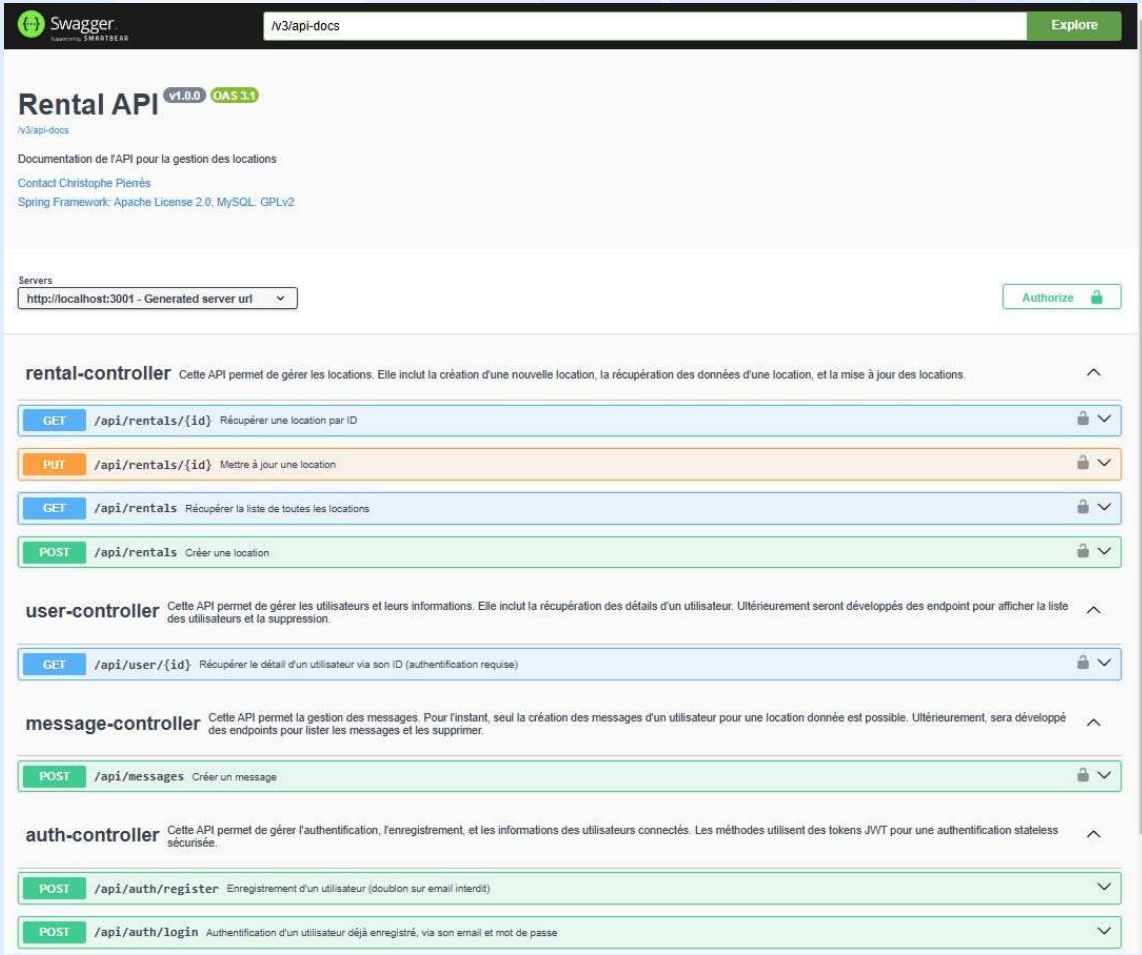
Route Angular	Verbe	api	description	Param	Réponses (Mockoon)

/register	POST	/api/auth/register	Création d'un utilisateur
			
			Contrôle d'unicité sur email
				In :
				export interface RegisterRequest {
				email: string;
				name: string;
				password: string;
				}
				out :
				export interface AuthSuccess {
				token: string;
				}
					200
					400 (pas de structure définie)

/login	POST	/api/auth/login	Tente un logging	In :
				export interface LoginRequest {
				email: string;
				password: string;
				}
				out :
				export interface AuthSuccess {
				token: string;
				}
					200
					401

			Si authentifié		

/me	GET	/api/auth/me	Utilisateur connecté	Out : User
				export interface User {
				id: number,
				name: string,
				email: string,
				created_at: Date,
				updated_at: Date
				}
					200
					401



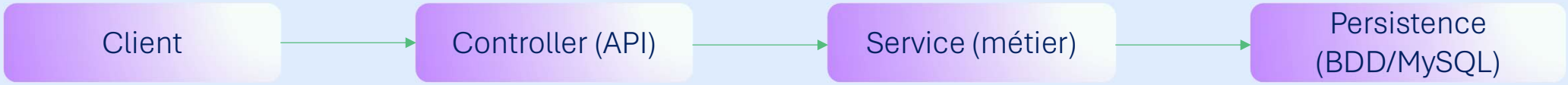
DÉPENDANCES PRINCIPALES

Dépendance	Rôle	Apport
jakarta.persistence-api	API de persistance pour JPA	Gestion des entités et des opérations CRUD.
spring-boot-starter-data-jpa	Starter Spring pour JPA	Intégration JPA/Hibernate avec Spring.
spring-boot-starter-web	Développement API REST	Gestion des routes HTTP (ex : contrôleurs REST).
spring-boot-starter-validation	Validation avec Hibernate Validator	Validation des DTOs (annotations @Valid, etc.).
spring-boot-starter-security	Authentification et sécurité	Gestion des mots de passe et des autorisations.
spring-boot-starter-oauth2-resource-server	Configure l'application en tant que serveur de ressources OAuth2	Responsable de protéger et exposer des API sécurisées. Valide les tokens d'accès envoyés avec les requêtes HTTP par des clients. Supporte des fournisseurs d'identité courants (Keycloak, etc.) ou des solutions personnalisées.
jjwt	Gestion des tokens JWT	Tokens sécurisés pour l'authentification.
mapstruct	Framework de mapping entre objets	Simplifie la transformation des données entre DTOs ⇔ entités.
lombok	Génération automatique de code	Amélioration de la lisibilité et réduction du code boilerplate.
mysql-connector-java	Driver JDBC pour MySQL	Communication efficace avec la base de données MySQL.
cloudinary	Plateforme cloud spécialisée dans gestion	Utilisé pour le stockage, l'optimisation et la récupération des médias
springdoc-openapi-starter-webmvc-ui	Extrait la doc des api REST et annotations	Interface de tester entièrement les API

PRINCIPES GÉNÉRAUX DE L'ARCHITECTURE

- Architecture en couches
  - Couche **Présentation** : Gestion des requêtes HTTP (API REST)
  - Couche **Métier** : Logique applicative et règles métiers.
  - Couche **Persistance** : Accès et manipulation des données avec JPA/Hibernate

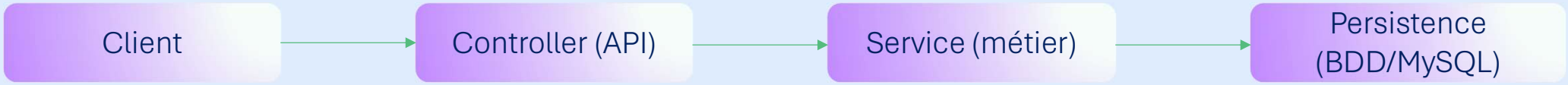
- Principes appliqués
  - **Principe SOLID** : Responsabilité unique et injection de dépendances
  - **Séparation des préoccupations** : API, validation, logique métier, persistance, mapping (mapstruct)
  - **Sécurité** avec Spring Security (JWT, PasswordEncoder)



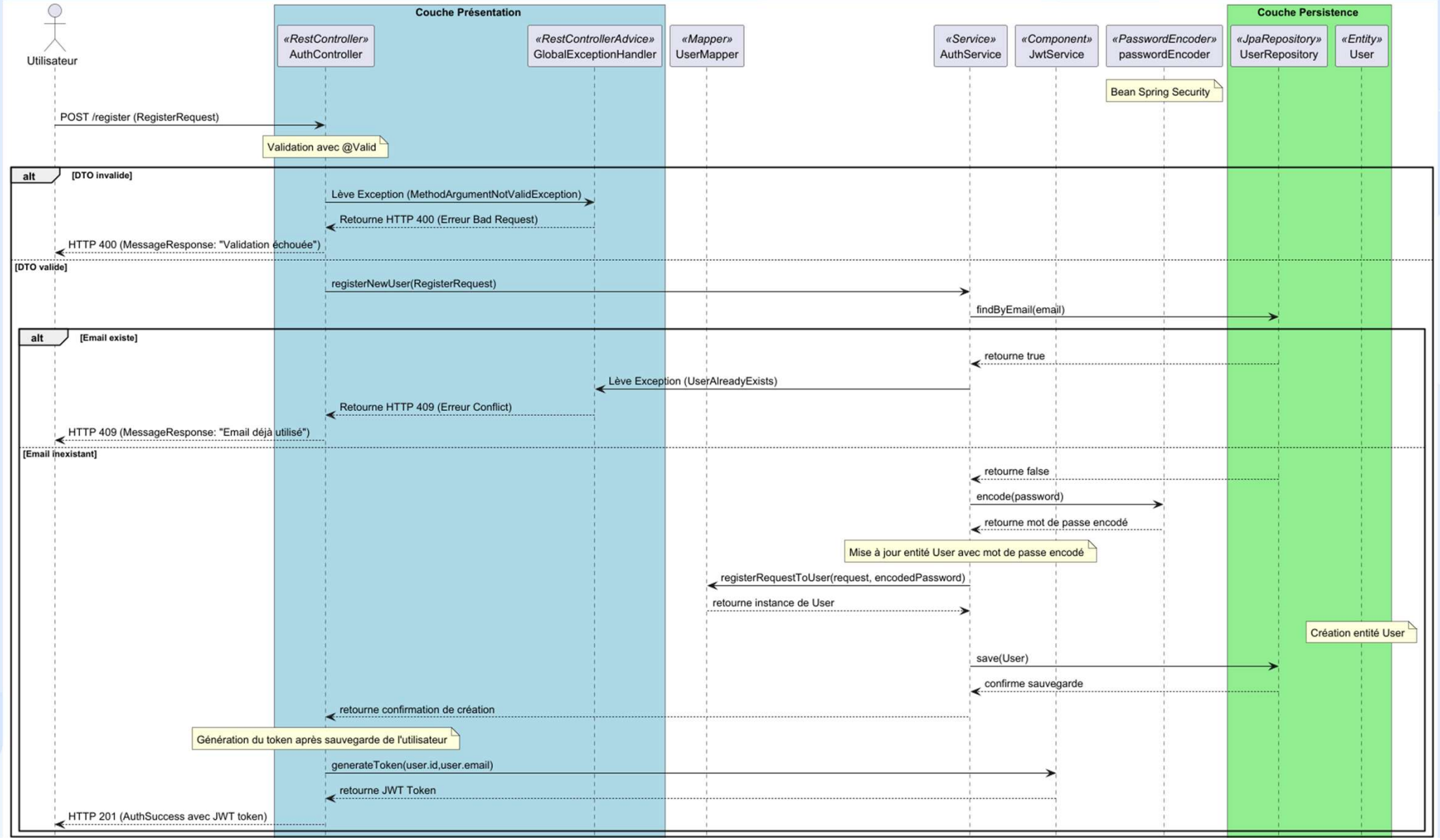
PRINCIPES GÉNÉRAUX DE L'ARCHITECTURE

- Architecture en couches
  - Couche **Présentation** : Gestion des requêtes HTTP (API REST)
  - Couche **Métier** : Logique applicative et règles métiers.
  - Couche **Persistance** : Accès et manipulation des données avec JPA/Hibernate

- Principes appliqués
  - **Principe SOLID** : Responsabilité unique et injection de dépendances
  - **Séparation des préoccupations** : API, validation, logique métier, persistance, mapping (mapstruct)
  - **Sécurité** avec Spring Security (JWT, PasswordEncoder)

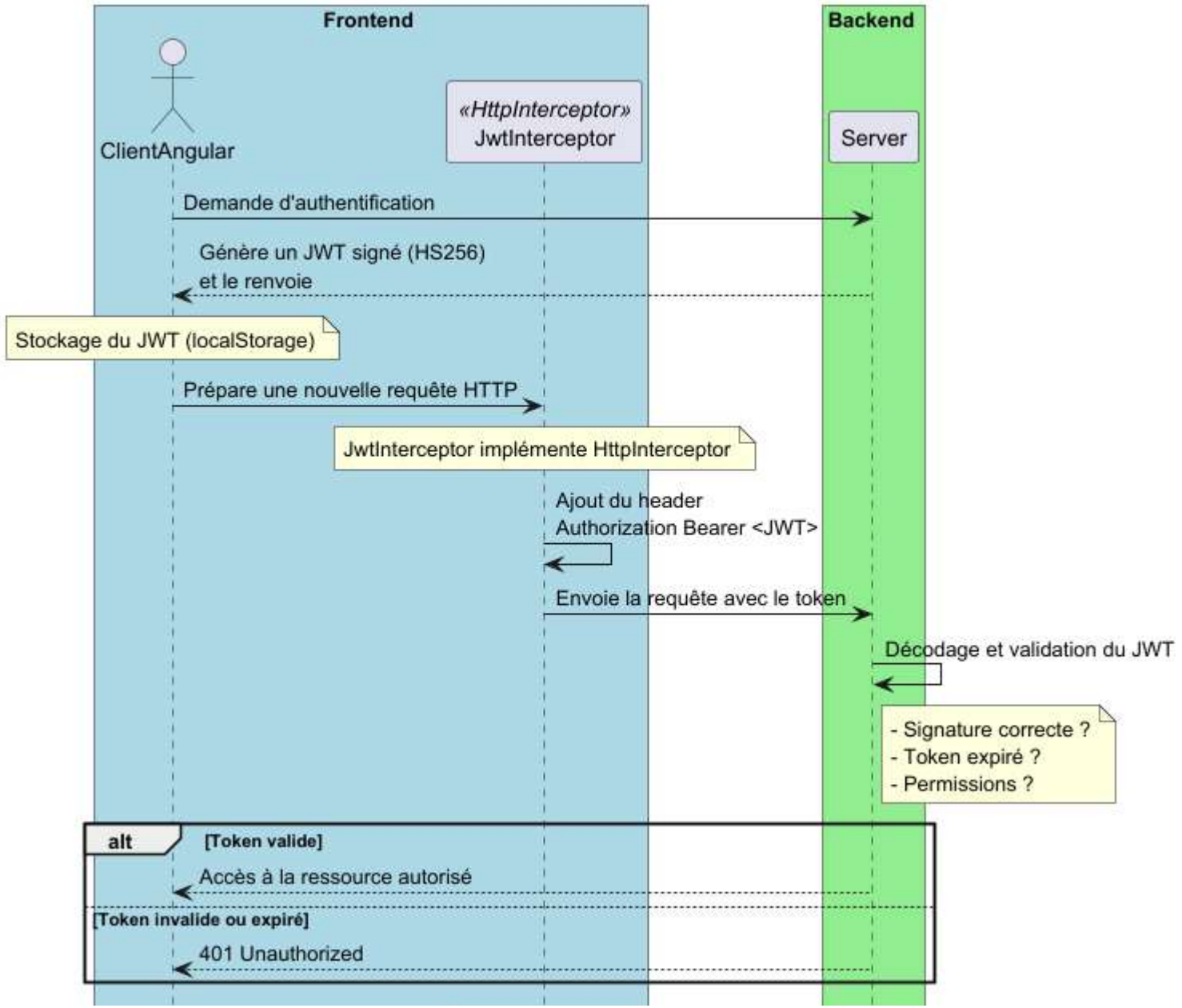


SCÉNARIO REGISTER

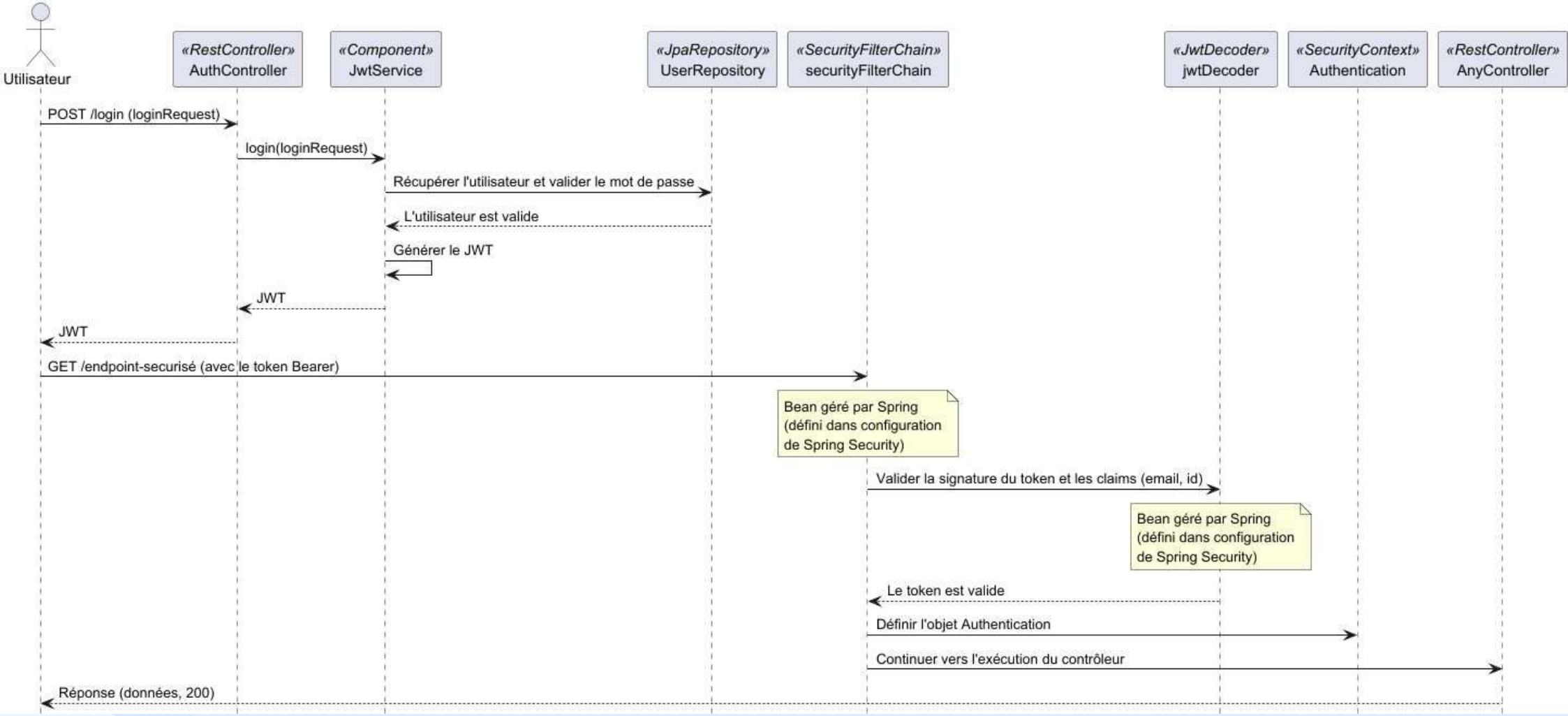


SECURITE

DIAG. DE SEQUENCES
- LOGIN
- REQUETE APRÈS LOGIN



FOCUS SUR SECURITÉ SPRING



Merci
à vous

CHRISTOPE PIERRÈS
+33 (0) 781 425 406
CPIERRES@HOTMAIL.COM

Principe	Description	Exemple dans Spring
S - Single Responsibility Principle (SRP)	Une classe (ou un composant) doit avoir une seule et unique responsabilité. Elle doit répondre uniquement à une source de changement.	Dans Spring, un **Controller** est responsable uniquement de gérer les requêtes HTTP, tandis qu'un **Service** gère la logique métier et un **Repository** s'occupe des opérations liées à la base de données (exemple : une classe ne fait qu'une seule chose).
O - Open/Closed Principle (OCP)	Une classe doit être ouverte à l'extension mais fermée à la modification. Cela permet de rajouter du comportement sans altérer le code existant.	L'utilisation des **interfaces** dans Spring (ex. : **UserDetailsService** pour **l'authentification**) permet d'étendre les fonctionnalités d'authentification en fournissant de nouvelles implémentations sans toucher au contrat ou à la logique de Spring Security.
L - Liskov Substitution Principle (LSP)	Une classe dérivée doit pouvoir être utilisée comme substitut de sa classe de base sans comportement inattendu.	Les implémentations d'interfaces comme **JpaRepository** respectent LSP : n'importe quelle classe qui implémente un repository peut être substituée sans casser le code (par exemple, passer d'une implémentation en mémoire à une implémentation réelle basée sur Data JPA).
I - Interface Segregation Principle (ISP)	Une interface ne doit contenir que les méthodes nécessaires à son utilisation. Les clients ne doivent pas être forcés de dépendre de méthodes inutilisées.	Avec **Spring Data JPA**, vous pouvez **définir** des **interfaces minimalistes spécifiques** à votre cas d'utilisation (exemple : **UserRepository** avec seulement **findByEmail**() au lieu d'alourdir votre interface avec des méthodes inutiles).
D - Dependency Inversion Principle (DIP)	Une classe doit dépendre d'abstractions (interfaces) plutôt que d'implémentations concrètes.	Spring utilise l'injection de dépendances via configuration (ex. : annotations **@Autowired** ou **@Bean**). Par exemple, un service peut dépendre d'une interface NotificationService sans connaître l'implémentation concrète (mail, SMS, etc.), respectant ainsi le DIP.

SÉCURITÉ – BONNES PRATIQUES

- Authentification puis Autorisation
- Spring Boot répond à une hiérarchie
  - Créer ses packages sous celui de la méthode principale afin que celle-ci dernière soit la première à s'exécuter
  - Classe de configuration dans sous-package configuration avec annotations :
    - ```
      @Configuration
      @EnableWebSecurity
      public class SpringSecurityConfig {
      ```
  - Sécurisation par défaut
    - Nécessite d'être revue
      ```

SÉCURITÉ – BONNES PRATIQUES

- Attentif aux dépendances dans le bon ordre pour initialisation correcte:
  - Oauth 2.0 client (en premier) puis Spring Security puis Spring Web
- 3 niveaux :
  - Pare-feu HTTP
  - DelegatingFilterProxy (dirige flux HTTP vers filtres de sécurité)
  - Chaîne de filtres sécurisée (héberge les règles de sécurité)
- Choix de l'authentification par token (JSon Web Token)
  - JWT : Object JS encode et transmet les infos d'authentification
  - Détient infos codées que seul le serveur comprend
    - Pas besoin de sauvegarder les infos de session sur le serveur
    - Plus performant et sécurisé qu'un cookie de session

SÉCURITÉ – OAUTH VS HTTP BASIC

- L'authentification HTTP de base :
  - nécessite que les informations d'identification de l'utilisateur soient partagées avec chaque ressource
  - envoie les informations d'identification de l'utilisateur non chiffrées dans l'en-tête HTTP. Peuvent être compromises.
- Avec OAuth, infos d’identification partagées uniquement avec le serveur d’Authentification
- Rôles de sécurité :
  - L'authentification HTTP de base n’a pas ce concept
  - Avec OAuth2, ils sont définis dans des scopes et passés dans le token

BONNES PRATIQUES POUR LE DÉVELOPPEMENT DU BACKEND

- 2 processeurs d’annotations pour accélérer l’écriture du code
  - Lombok
  - MapStruct :
    - génère les classes Mapper (plus nécessaire au runtime)
    - Mappage des entités JPA vers des DTOs et vice-versa

DIFFICULTES

- Champs d’audit communs placés dans une @MappedSuperclass
  - @Data pas compatible, obligé d’utiliser d’autres annotations :
    - @EqualsAndHashCode(callSuper = true)
    - @ToString(callSuper = true)
    - @Getter
    - @Setter
- RentalMapper spécifique pour le champ picture