



Présentation du projet : Insta_lite 2

Années Universitaire : 2023/2024

Réalisé par : (tous alternants)

HADDAM Younes

AZIZI Fouad

BENKHELOUF TALEB LAID

ZERDANE Ilyes

Présentation du projet :

Dans le cadre de ce projet, nous envisageons de créer une plateforme de gestion d'images qui garantira une expérience utilisateur à la fois efficace et intuitive. Cette plateforme se décomposera en trois parties distinctes : une section publique permettant l'accès aux images publiques du portfolio sans nécessiter d'authentification, une section authentifiée destinée aux utilisateurs n'ayant pas le droit d'accéder aux publications privées, et une section privée réservée aux utilisateurs privilégiés, offrant un accès au contenu privé et public. De plus, une section d'administration sera mise en place, exclusivement réservée aux administrateurs afin de gérer le contenu global de la plateforme. En résumé, l'objectif de cette plateforme est de fournir une solution intégrale pour la gestion des images, des utilisateurs, ainsi que du portfolio de l'administrateur.

Spécification des besoins :

Utilisateur:

Il y a deux catégories d'utilisateurs : ceux autorisés à consulter les publications privées et ceux qui ne le sont pas.

Administrateur:

L'administrateur a la capacité de gérer les utilisateurs en leur attribuant des autorisations pour accéder à la section publique, y compris les noms d'utilisateur. De plus, il a le pouvoir de gérer les images, y compris l'implémentation d'opérations CRUD associées.

Invité

L'invité à accès au portfolio publique seulement.

Diagramme de cas d'utilisation :

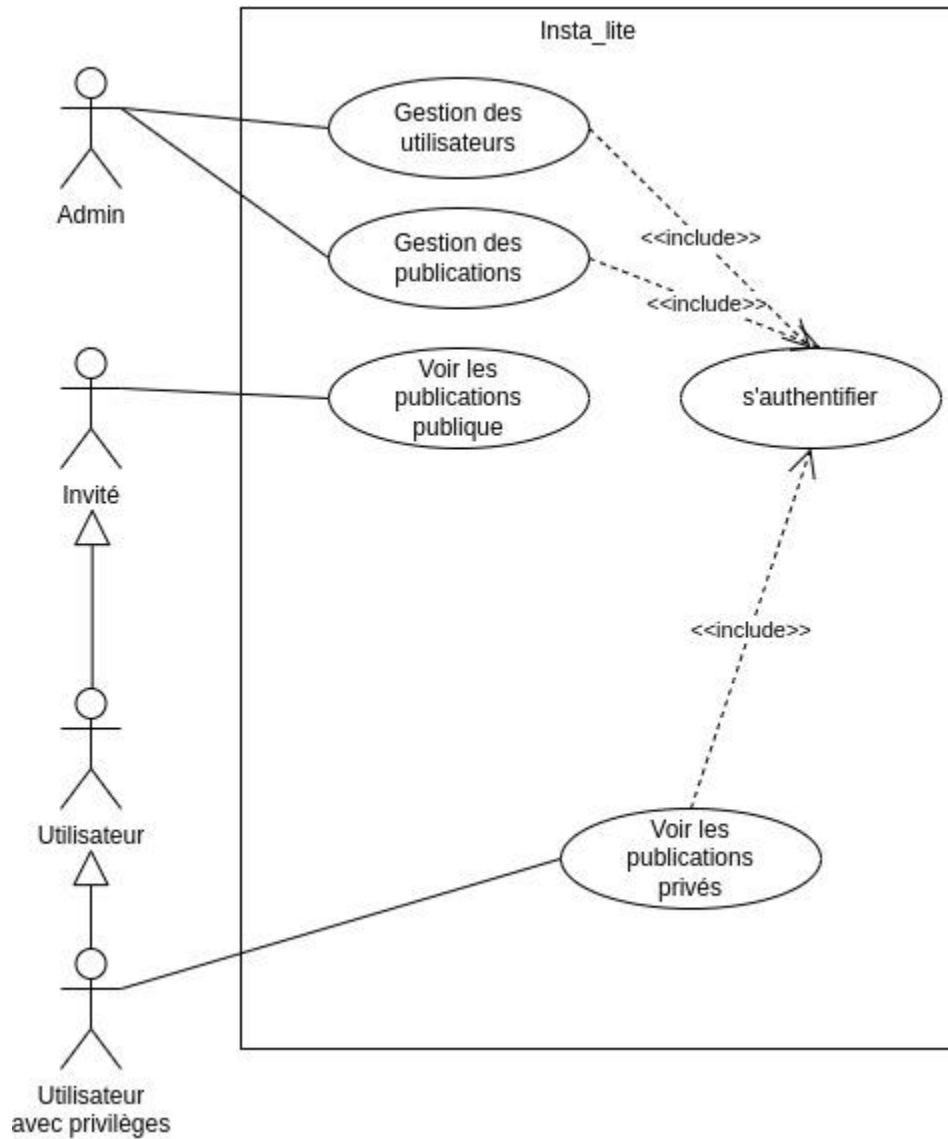
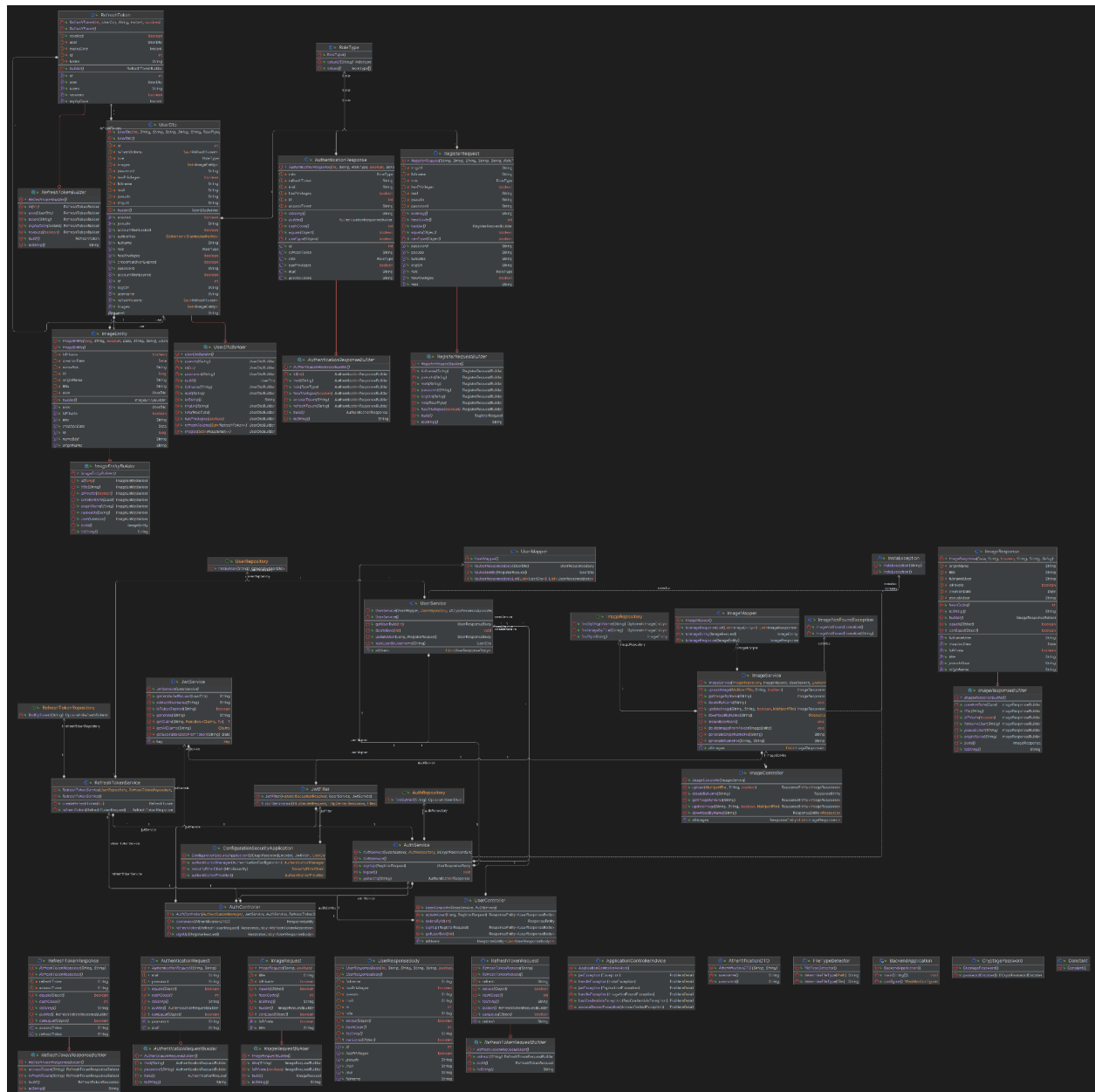


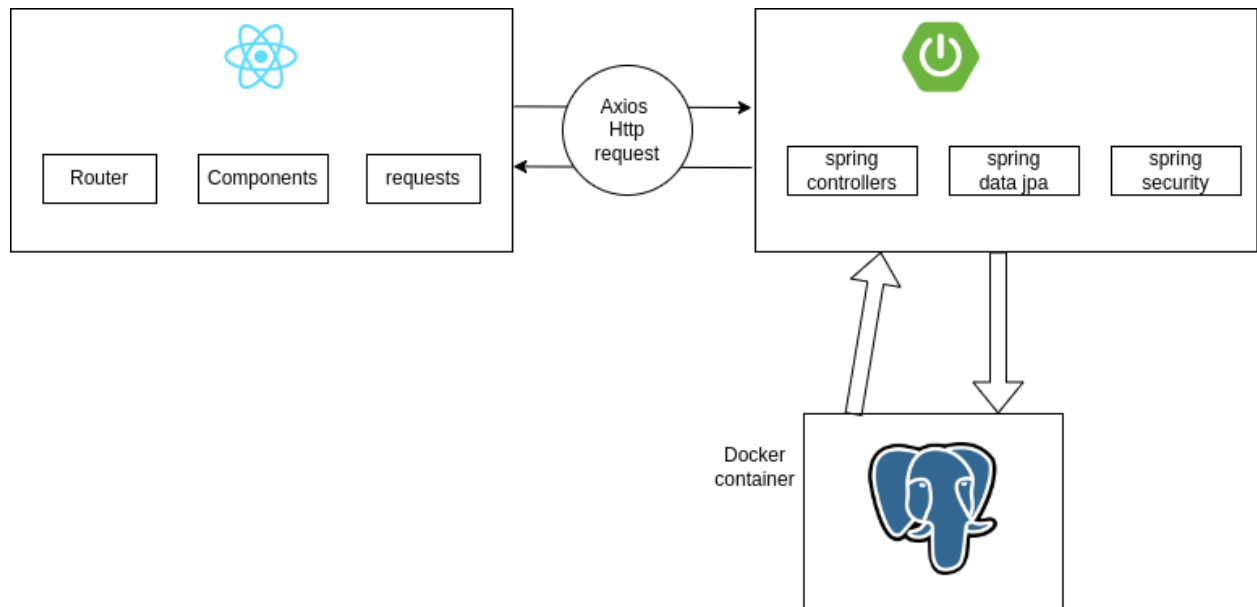
Diagramme de cas d'utilisation générale

Conception:



Technologie de développement utilisée:

Le schéma ci-dessous montre ce qu'on a fait :



Pourquoi React ?

React utilise JSX, une syntaxe particulière qui facilite l'intégration d'HTML et de JavaScript, permettant ainsi de tirer parti des avantages des deux langages. Cette approche simplifie immédiatement la compréhension de ReactJS. Grâce à son modèle basé sur des composants, un cycle de vie clairement défini et l'utilisation de JavaScript de manière simple, React devient une plateforme facile à apprendre et à utiliser pour la construction de sites web professionnels.

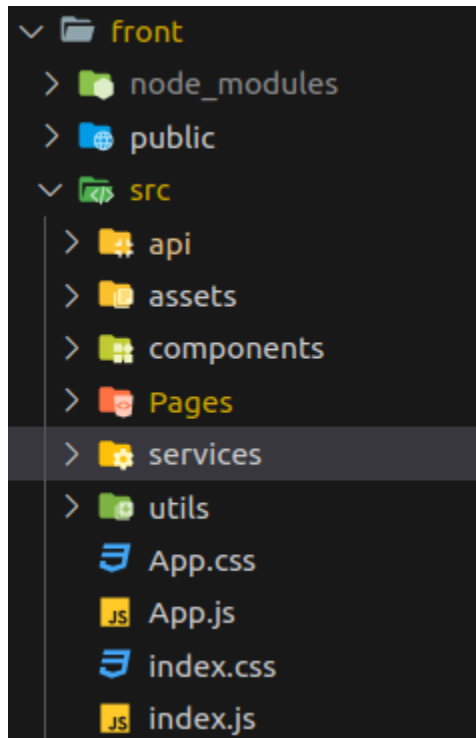
Performance

ReactJS intègre le concept d'état, considérant chaque composant comme une fonction qui peut être manipulée en fonction de l'état qui lui est transmis. Cela permet d'observer la sortie ainsi que les actions, événements et fonctions déclenchés.

Préférence de l'équipe :

L'équipe a choisi de travailler avec React en raison de sa facilité d'apprentissage, et également parce que nous possédons déjà une base de connaissances solide dans ce framework.

Structure de notre projet Front :



Api : le dossier contenant des fichiers pour nos requêtes (on a utilisé axios)

Components : Ce dossier contient nos composants réutilisables.

Pages : Ce dossier contient tout les pages avec des sous dossier pour chaque page (admin, users, home...)

utils : Ceci contient nos fonctions personnalisé

Requêtes api :

```
import axios from "axios";
let apiUrl = "http://localhost:8082/";

export const signup = (user) => {
  return axios.post(apiUrl + "inscription", user);
};

export const login = (user) => {
  return axios.post(apiUrl + "connexion", user);
};
```

Gestion des inputs

nous avons opté pour l'utilisation de la bibliothèque Formik, reconnue pour sa capacité à gérer efficacement les saisies utilisateur. Formik offre des fonctionnalités avancées pour la gestion des champs de formulaire, simplifiant ainsi le processus de collecte et de validation des données d'entrée.

Par exemple :

```
const LoginSchema = Yup.object().shape({
  mail: Yup.string().email("Invalid email").required("Email is required"),
  password: Yup.string().required("Password is required"),
});

export default function Login() {
  const [, setUser] = useAtom(userAtom);

  const navigate = useNavigate();

  const formik = useFormik({
    initialValues: {
      mail: "",
      password: "",
    },
    validationSchema: LoginSchema,
  });
```

La validation des formulaire est gérée par Yup

Gestion des rôles & token

Pour ceci on a utilisé localStorage pour sauvegarder le token et le rôle de l'utilisateur

```
// save token to local storage
localStorage.setItem("user-token", res.data.access_token);
localStorage.setItem("user-role", res.data.role);
```

Docker :

Postgres:

Postgres est un serveur de base de données relationnelles SQL qui fonctionne sur de nombreux systèmes d'exploitation. Il permettra la gestion et l'interaction de notre application avec la basé de donnees sql associée.

```
version: "3.8"

services:
  postgres:
    image: postgres:13-alpine
    environment:
      POSTGRES_USER: dockeruser
      POSTGRES_PASSWORD: dockerpassword
      POSTGRES_DB: dockerdb
    ports:
      - "5432:5432"
    volumes:
      - postgres-data:/var/lib/postgresql/data

volumes:
  postgres-data:
```


Spring :



Spring Boot, un framework open-source largement adopté dans l'écosystème Java, a été spécifiquement développé pour la création d'applications modernes, évolutives et performantes. Fondé sur le cadre bien établi de Spring, il simplifie le processus de développement des nouvelles applications tout en réduisant considérablement le temps nécessaire pour les mettre sur le marché.

En se basant sur Spring, Spring Boot offre un ensemble préconfiguré de composants et d'outils, permettant aux développeurs de commencer rapidement avec une configuration minimale. Grâce à cette approche, les développeurs peuvent rapidement construire et déployer des applications qui sont à la fois faciles à entretenir, sécurisées et hautement évolutives.

Détails des interfaces de services :

ImageService:

Le service ImageService est responsable de la gestion des images, de leur téléchargement, de leur mise à jour, de leur suppression, de leur récupération, et de leur téléchargement.

Dans ce service, et pour implémenter un mécanisme permettant de sécuriser notre application, nous avons appliqué une méthode vu en cours de web sécurité qui base sur le fait que l'utilisateur n'ait jamais le nom sous lequel l'image est sauvegardé en interne, pour le faire, dans notre base de données nous avons deux champs (origineName, et nameBdd) le premier c'est le nom de l'image retourné à l'utilisateur lorsqu'il souhaite publier une image et le deuxième, est le nom sous lequel l'image est stocké en interne.

Méthode	Signature	Type de retour	Description
uploadImage	(MultipartFile image, String title, boolean isPrivate)	ImageResponse	Permet de stocker une image donnée avec un titre et un paramètre de confidentialité.
getAllImages	()	List<ImageResponse>	Récupérer toutes les images disponibles selon les droits de l'utilisateur.
updateImage	(String name, String title, boolean isPrivate, MultipartFile newImage)	ImageResponse	Met à jour une image existante avec de nouveaux détails et/ou une nouvelle image.
getImageByName	(String name)	ImageResponse	Récupérer une image par son nom d'origine.
deleteByName	(String name)	void	Supprime une image par son nom d'origine.
downloadByName	(String name)	Resource	Fournit une ressource pour télécharger une image par son nom
deleteImageFromFolder (private)	(ImageEntity imageEntity)	void	Supprime l'image du système de fichiers.
instantiateHost (private)	()	void	Initialise l'adresse hôte pour le service.
generateNameFile	(String name, String extension)	String	Génère un nom de fichier unique.
generateOriginName File	(String extension)	String	Génère un nom d'origine unique pour le fichier.

ImageService

UserService

Le service UserService gère les utilisateurs, y compris la récupération de la liste de tous les utilisateurs, la récupération d'un utilisateur par son ID, la mise à jour des informations de l'utilisateur, et la suppression d'un utilisateur. Il gère également le chargement de l'utilisateur connecté actuel.

Méthode	Signature	Return	Description
getAllUsers	()	List<UserResponse Body>	Récupère tous les utilisateurs enregistrés dans le système et renvoie leurs informations sous forme de liste.
getUserById	(int id)	UserResponseBody	Récupérer les détails d'un utilisateur spécifique par son identifiant.
deleteById	(int id)	void	Supprime un utilisateur du système en fonction de son identifiant.
loadUserByUsername	(String username)	UserDTO	Cette méthode est appelée par Spring Security pendant le processus d'authentification pour charger les détails d'un utilisateur donné par son nom d'utilisateur. Elle doit récupérer les informations de l'utilisateur (comme les autorités, le mot de passe, etc.) et les retourner sous forme d'objet qui implémente l'interface UserDetails. Si aucun utilisateur n'est trouvé avec le nom d'utilisateur fourni, cette méthode doit lever une exception <code>UsernameNotFoundException</code>
updateUser	(@PathVariable Long id,	UserResponse Body	Met à jour les informations d'un utilisateur existant avec les

	RegisterRequest userDtoRequest)		données fournies.
--	------------------------------------	--	-------------------

RefreshTokenService :

Le service RefreshTokenService est responsable de la gestion des tokens de rafraîchissement. Il génère un token de rafraîchissement lorsqu'un utilisateur se connecte et peut également rafraîchir les tokens d'authentification lorsque cela est nécessaire.

Méthode	Signature	Return	Description
createRefreshToken	(int userDto_id)	RefreshToken	Créer un nouveau token de rafraîchissement pour un utilisateur donné. Le paramètre userDto_id est l'identifiant de l'utilisateur pour lequel le token est créé. Cette méthode retourne un objet RefreshToken contenant les détails du token de rafraîchissement.
refreshToken	(RefreshTokenRequest refreshTokenRequest)	RefreshTokenResponse	Accepte une demande de token de rafraîchissement et renvoie une nouvelle paire de tokens d'accès et de rafraîchissement. Cette méthode est appelée lorsqu'un client souhaite

			rafraîchir son token d'accès expiré ou sur le point d'expirer en utilisant son token de rafraîchissement.
--	--	--	---

JWTSERVICE:

Le service JwtService est responsable de la génération, de la validation et de la gestion des tokens JWT (JSON Web Tokens) pour l'authentification et l'autorisation des utilisateurs. Il génère des tokens JWT en fonction des informations de l'utilisateur, extrait des informations utiles à partir des tokens, vérifie la validité des tokens en vérifiant leur date d'expiration, et garantit la sécurité des communications entre le client et le serveur en utilisant une clé secrète pour signer et vérifier les tokens

Méthode	Signature	Return	Description
generate	(String username)	String	Génère un nouveau token JWT pour un nom d'utilisateur donné. Cette méthode est essentielle pour créer des tokens lors de la connexion des utilisateurs.
extractUsername	(String token)	String	Extrait le nom d'utilisateur d'un token JWT. Cela est utilisé pour identifier l'utilisateur associé à un token donné.
isTokenExpired	(String token)	bool	Vérifie si un token JWT est expiré. Cette méthode est cruciale pour valider la validité des tokens.

getExpirationDateFromToken (private)	(String token)	Date	Récupère la date d'expiration d'un token JWT. Utilisée en interne pour diverses vérifications et logiques.
getAllClaims (private)	(String token)	Claims	Récupère tous les claims (affirmations/attributs) d'un token JWT. Utilisée pour extraire des informations détaillées d'un token.
generateJwtByUser	(UserDto user)	String	Génère un token JWT pour un objet utilisateur spécifié. Cela permet de créer un token basé sur les détails complets d'un utilisateur.
getKey (private)	()	Key	Récupère la clé utilisée pour signer les tokens JWT. C'est une opération privée cruciale pour la sécurité de la génération des tokens.

User Story

L'Inscription et l'authentification de Bob : Le Début de l'Aventure Sécurisée

Bob visite notre application et choisit de s'inscrire. Il remplit le formulaire d'inscription avec son nom d'utilisateur, son adresse e-mail et son mot de passe.

La demande d'inscription est envoyée à l'endpoint /signup géré par AuthController. Ce contrôleur reçoit les données d'inscription et les transmet à AuthService.

AuthService prend le relais. Il vérifie d'abord que les données de Bob sont valides et que l'utilisateur n'existe pas déjà. Puis, il utilise CryptagePassword pour hacher le mot de passe de Bob avant de le stocker. Cette étape est cruciale pour assurer que même en cas de compromission des données, les mots de passe restent indéchiffrables.

Bob reçoit une confirmation que son compte a été créé avec succès. À ce stade, il est enregistré avec un compte sécurisé.

La Connexion de Bob : Accès Authentifié et Protégé

Bob décide de se connecter et soumet ses informations via le formulaire de connexion à l'endpoint /login.

Spring Security intercepte cette requête. Le AuthenticationManager de Spring Security orchestre le processus d'authentification. Il crée un objet Authentication contenant les détails de Bob et le passe à travers une chaîne d'AuthenticationProvider.

UserDetailsService est appelé pour récupérer les informations de Bob. notre UserService personnalisé, qui implémente UserDetailsService, charge les informations de Bob, y compris le mot de passe haché stocké.

PasswordEncoder vérifie le mot de passe en comparant le mot de passe fourni avec le hachage stocké. Si cela correspond, Bob est considéré comme authentifié.

Un token JWT est généré pour Bob par JwtService. Ce token contient des informations sur Bob et une signature numérique pour vérifier son intégrité. Bob reçoit ce token comme preuve de son authentification.

Accès aux Ressources : La Navigation Sécurisée de Bob

Bob navigue dans l'application et fait des requêtes à des endpoints sécurisés, en incluant son token JWT dans les en-têtes.

JwtFilter intercepte chaque requête pour extraire et valider le token JWT. Il utilise JwtService pour vérifier la validité et l'expiration du token. Si le token est valide, la requête est autorisée à continuer.

Accès autorisé ou refusé : Si le token est valide, Bob accède à la ressource demandée. Si non, il est informé que son token est invalide ou expiré et peut être invité à se reconnecter.

Lorsque Bob, après s'être connecté avec succès, fait des requêtes sur notre site, Spring Security utilise plusieurs mécanismes pour vérifier s'il est authentifié et autorisé à accéder aux ressources demandées. Voici comment cela fonctionne étape par étape :

1. Bob envoie une requête :

Après avoir reçu son token JWT lors de la connexion, Bob envoie une requête à notre application pour accéder à une ressource spécifique. Il inclut son token JWT, généralement dans l'en-tête Authorization de la requête HTTP.

2. Interception par Spring Security :

Filtres de Sécurité : Spring Security a un ensemble de filtres qui interceptent toutes les requêtes entrantes. Parmi ces filtres, un filtre personnalisé qui est JwtFilter, est configuré pour traiter les tokens JWT.

3. Extraction et Validation du Token :

Extraction du Token : JwtFilter extrait le token JWT de l'en-tête Authorization.

Validation du Token : Le filtre utilise ensuite JwtService pour valider le token. Les validations incluent :

Signature : Vérification que le token a été signé par notre serveur.

Expiration : Vérification que le token n'est pas expiré.

Claims : Vérification d'autres claims (comme le rôle de l'utilisateur) si nécessaire.

4. Décision d'Accès :

Si Bob est authentifié (token valide) :

Extraction des Détails : Les informations de Bob sont extraites du token.

Contexte de Sécurité : Spring Security met à jour le contexte de sécurité avec l'identité de Bob. Cela signifie que notre application sait qui est Bob.

Accès aux Ressources : Si Bob a les bons rôles ou permissions (selon la configuration de sécurité de l'endpoint demandé), il accède à la ressource. Sinon, il reçoit une erreur d'autorisation.

Si Bob n'est pas authentifié (token invalide ou absent) :

Erreur d'Authentification : Bob reçoit une réponse indiquant une erreur d'authentification. Cela est un code d'état HTTP 401 (Non autorisé).

Aucun Accès : Bob ne peut pas accéder à la ressource demandée tant qu'il ne fournit pas un token valide.

5. Gestion des Erreurs :

Lorsque Bob fait des requêtes et que des erreurs surviennent, nous avons implémenté un mécanisme fort et puissant pour gérer les exceptions dans notre application Spring, en définissant un conseiller de contrôleur global utilisé pour gérer les exceptions au niveau global pour tous les contrôleurs de notre application. Voici comment les différentes exceptions sont gérées :

BadCredentialsException :

Quand cela survient ? Quand les identifiants de Bob sont incorrects.

Réponse : Bob reçoit une réponse avec le statut HTTP 401 (Non autorisé) indiquant que ses identifiants ne sont pas valides.

SignatureException et MalformedJwtException :

Quand cela survient ? Quand le token JWT de Bob est mal formé ou à une signature invalide.

Réponse : Bob reçoit une réponse avec le statut HTTP 401 indiquant que son token n'est pas valide.

ExpiredJwtException :

Quand cela survient ? Quand le token JWT de Bob a expiré.

Réponse : Bob reçoit une réponse avec le statut HTTP 401 lui signalant que son token n'est plus valide.

AccessDeniedException :

Quand cela survient ? Quand Bob essaie d'accéder à une ressource pour laquelle il n'a pas les bonnes autorisations.

Réponse : Bob reçoit une réponse avec le statut HTTP 403 (Interdit) lui informant qu'il n'a pas les droits nécessaires pour effectuer cette action.

InstaException :

Quand cela survient ? Pour les conflits ou autres problèmes spécifiques à notre application.

Réponse : Bob reçoit une réponse avec le statut HTTP 409 (Conflit) avec des détails sur le problème rencontré.

ImageNotFoundException :

Quand cela survient ? Quand Bob demande une image ou une ressource qui n'existe pas.

Réponse : Bob reçoit une réponse avec le statut HTTP 404 (Non trouvé) indiquant que la ressource demandée est introuvable.

RefreshTokenException:

Quand cela survient ? Quand Bob demande de réfléchir le token avec un refresh token invalide ou expiré .

Réponse : Bob reçoit une réponse avec le statut HTTP 401 lui signalant que son refresh token n'est pas valide.

SignupException:

Cela survient lorsqu'un utilisateur met un mot de passe non sécurisé.

Réponse : Bob reçoit une réponse avec le statut 400 BAD REQUEST

Utilisation des Mappers dans notre Application Spring Boot

1. Introduction aux Mappers

Dans le développement moderne d'applications, séparer les préoccupations et les responsabilités est crucial pour créer des systèmes maintenables et évolutifs. Les mappers jouent un rôle essentiel dans cette séparation en agissant comme des intermédiaires qui transforment les données entre les modèles de domaine (entités) et les modèles de transfert de données (DTOs). Dans notre application, les mappers facilitent la communication entre la couche de présentation et la couche de logique métier, assurant une architecture propre et modulaire.

2. Raison d'Utilisation des Mappers

Isolation des Couches : Les mappers renforcent l'isolation en séparant la logique métier des modèles de domaine de la présentation et du transfert des données. Cette séparation permet de modifier un modèle sans affecter l'autre, réduisant ainsi le risque d'erreurs et simplifiant les mises à jour et la maintenance.

Sécurité : En utilisant des DTOs, nous contrôlons précisément les données exposées aux utilisateurs finaux. Cela évite l'exposition accidentelle de données sensibles et garantit que seules les informations pertinentes sont transmises.

Flexibilité et Maintenance : Les mappers nous permettent de modifier facilement la structure des données en réponse à l'évolution des exigences, sans perturber la logique métier sous-jacente ni l'interface utilisateur.

3. Implémentation dans Notre Application

Dans notre application, les mappers tels que IMapper et UserMapper sont utilisés pour transformer les ImageEntity et UserEntity en ImageResponse et UserResponseBody respectivement, et vice versa. Par exemple :

Lorsqu'une nouvelle image est téléchargée, IMapper convertit un ImageRequest en ImageEntity pour le traitement et la persistance.

Lors de la récupération des détails de l'utilisateur, UserMapper transforme les UserEntity en UserResponseBody pour envoyer les données à l'utilisateur de manière cohérente et sécurisée.

Ces transformations garantissent que les opérations de données sont gérées de manière efficace, cohérente et sécurisée.

4. Avantages de l'Utilisation des Mappers

Performance : Les mappers nous permettent d'optimiser les performances en chargeant et en transférant uniquement les données nécessaires, évitant ainsi une surcharge inutile.

Testabilité : Avec des mappers bien définis, nous pouvons tester séparément les transformations de données, assurant que chaque couche de l'application fonctionne comme prévu indépendamment des autres.

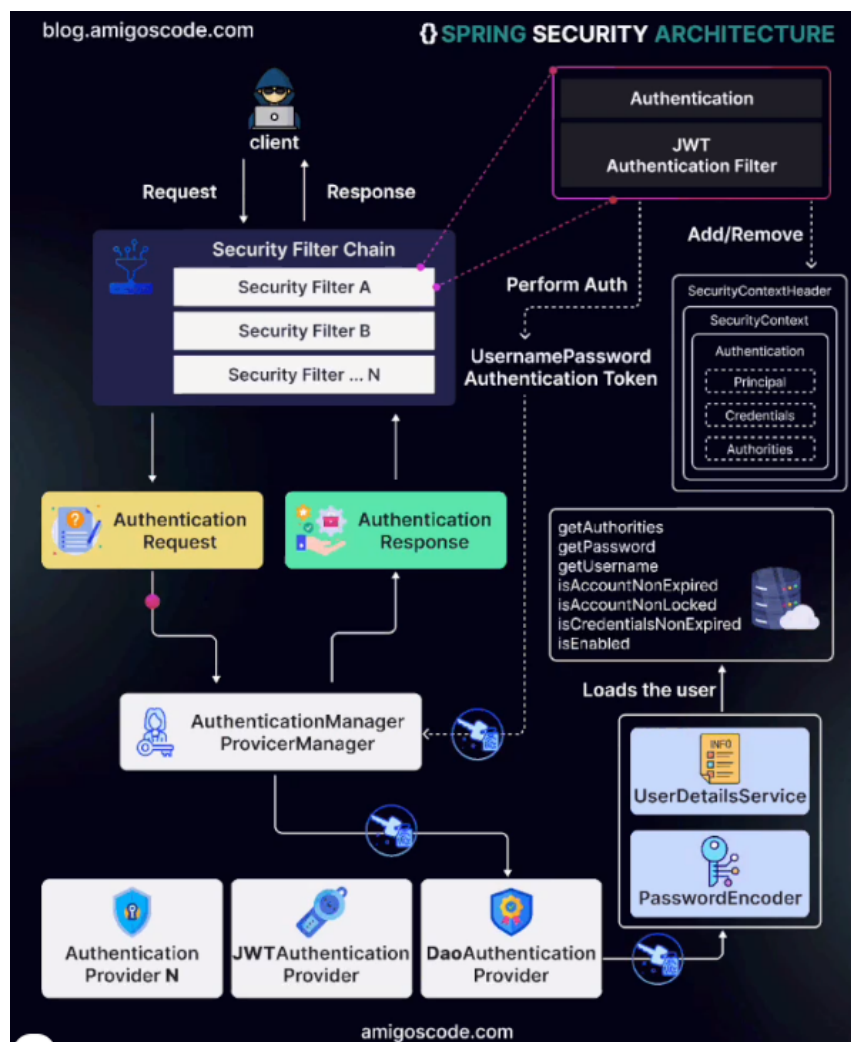
Cohérence : En centralisant la logique de transformation, les mappers garantissent que toutes les transformations de données suivent les mêmes règles et formats, conduisant à une plus grande cohérence dans toute l'application.

5. Conclusion : Justification de Notre Approche

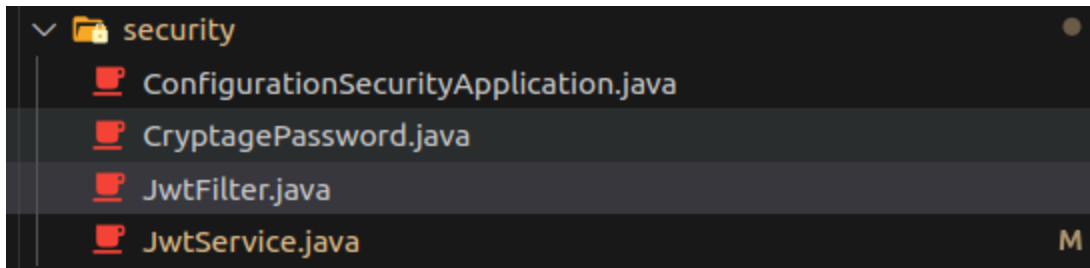
L'utilisation de mappers dans notre application Spring Boot n'est pas seulement une question de préférence; c'est une décision alignée sur les meilleures pratiques de l'industrie pour assurer la clarté, la sécurité et la maintenabilité du code. En adoptant cette approche, nous avons créé une application qui non seulement répond aux besoins actuels de manière efficace mais est également prête à évoluer et à s'adapter aux exigences futures. Les mappers sont un témoignage de notre engagement envers un développement de qualité et une architecture réfléchie, démontrant notre expertise et notre capacité à construire des solutions robustes et sécurisées.

SPRING SECURITY :

L'implémentation de Spring Security dans notre application a été un choix judicieux, car elle nous a permis de garantir la confidentialité des données, d'assurer l'intégrité des transactions et de fournir un contrôle d'accès précis aux différentes parties de notre système. En outre, la flexibilité de Spring Security nous a offert la possibilité d'adapter ces fonctionnalités en fonction des besoins spécifiques de notre application, contribuant ainsi à la création d'un environnement sécurisé et fiable.



Source : amigoscode



Cette fonction génère un jeton JWT pour un utilisateur spécifique en incorporant son adresse e-mail, son nom complet, l'heure d'émission, et l'heure d'expiration. Le jeton est ensuite signé et retourné sous forme de chaîne de caractères.

```
public String generateJwtByUser(UserDto user) {  
    final long currentTime = System.currentTimeMillis();  
    final long expirationTime = currentTime + 300 * 600 * 1000;  
  
    final Map<String, Object> claims = Map.of(  
        "fullname", user.getFullname(),  
        Claims.EXPIRATION, new Date(expirationTime),  
        Claims.SUBJECT, user.getMail()  
    );  
  
    final String bearer = Jwts.builder()  
        .setIssuedAt(new Date(currentTime))  
        .setExpiration(new Date(expirationTime))  
        .setSubject(user.getMail())  
        .setClaims(claims)  
        .signWith(getKey(), SignatureAlgorithm.HS256)  
        .compact();  
  
    return bearer;    You, 5 days ago • login & signup & user management  
}  
  
private Key getKey() {  
    final byte[] decoder = Decoders.BASE64.decode(Constant.ENCRYPTION_KEY);  
    return Keys.hmacShaKeyFor(decoder);  
}
```

Cette fonction est un filtre de sécurité pour les requêtes HTTP, vérifiant la présence et la validité des tokens JWT dans les en-têtes des requêtes pour authentifier les utilisateurs avant de les laisser accéder à des ressources sécurisées. Elle gère également les exceptions et assure que la chaîne de filtres continue de traiter la requête selon les besoins.

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
    try {
        String token = null, username = null, authorization = request.getHeader(Constant.AUTHORIZATION);
        Boolean isTokenExpired = true;
        if (authorization != null && authorization.startsWith(Constant.BEARER)) {
            token = authorization.substring(7);
            isTokenExpired = jwtService.isTokenExpired(token);
            username = jwtService.extractUsername(token);
        }
        //si le username !=null et que aucune authentication n'est en cours
        if (isTokenExpired == false && username != null
            && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails user = userService.loadUserByUsername(username);
            UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(user, null, user.
                getAuthorities());
            SecurityContextHolder.getContext().setAuthentication(authenticationToken);
        }
        filterChain.doFilter(request, response);
    } catch (Exception exception){
        log.info("hahhhadisiocunc");
        handlerExceptionResolver.resolveException(request, response, null, exception);
    }
}
```

La classe **CryptagePassword** est une configuration Spring qui fournit un bean de type **BCryptPasswordEncoder**. Elle utilise l'annotation **@Configuration** pour signaler à Spring qu'elle contient des définitions de beans, et l'annotation **@Bean** pour indiquer que la méthode **passwordEncoder()** retourne un bean géré par Spring.

BCryptPasswordEncoder est une méthode de cryptage de mots de passe robuste et sécurisée qui utilise l'algorithme BCrypt. Cette classe est utile dans les applications Spring Security pour crypter les mots de passe avant leur stockage et pour vérifier les mots de passe lors de l'authentification des utilisateurs.

```

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
public class CryptagePassword {

    @Bean
    public BCryptPasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
}

```

Cette méthode configure la sécurité HTTP de l'application, en définissant la manière dont différentes requêtes doivent être traitées en termes d'authentification et d'autorisation, en gérant la politique de session, et en intégrant un filtre personnalisé pour traiter les tokens JWT. Elle rend les endpoints spécifiques accessibles à tous tout en sécurisant les autres, et configure le système pour être sans état, ce qui est idéal pour les API REST

```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    return
        httpSecurity
            .cors(Customizer.withDefaults())
            .csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(
                authorize ->
                    authorize
                        .requestMatchers(POST, "/inscription").permitAll()
                        .requestMatchers(POST, "/connexion").permitAll()
                        .requestMatchers(POST, "/refresh-token").permitAll()
                        .requestMatchers(GET, "/images").permitAll()
                        .anyRequest().authenticated()
                    )
            .sessionManagement(httpSecuritySessionManagementConfigurer ->
                httpSecuritySessionManagementConfigurer.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            )
            .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)
            .build();
}

@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration) throws Exception {
    return authenticationConfiguration.getAuthenticationManager();
}

```

Difficultés rencontrés :

Ajout d'image et affichage

Solution : Ajout d'image en formdata et recevoir l'image en Byte et la convertir en Base64 pour l'afficher sur notre interface.

JAR TO WAR

Un peu compliquée, notre projet est en JAR on a voulu le convertir en WAR mais après avoir tout fait on avait un peu peur de tout casser 😞

Manque d'xp dans quelques technologies

Solution : Formation rapide

État d'avancement :

Nous considérons avoir satisfait toutes les fonctionnalités requises pour l'application.

Amélioration et review ??

Nous avons acquis de nombreuses connaissances grâce à ce projet et avons cherché à réaliser des tâches plus avancées pour approfondir nos compétences. Cependant, le programme du Master 2 est assez chargé. 😊

Setup et lancement de l'application

Lancer la base de données :

```
docker compose up
```

Installer les dépendances et lancer le serveur de spring:

```
cd backend
```

```
mvn clean install
```

```
mvn spring-boot:run
```

Installer les dépendances et lancer le serveur de react :

```
cd front
```

```
npm install
```


npm start

Création d'un administrateur :

Avec une simple requête vers cet url avec les données ci-dessous :

<http://localhost:8082/inscription>

```
{  
  "mail": "admin@gmail.fr",  
  "fullname": "admin",  
  "password": "Password123+",  
  "role": "ROLE_ADMINISTRATEUR"  
}
```

Nos interfaces

Gestion des publications (admin):

Accueil

Publications

Gestion des publications

Gestion des utilisateurs

Gestion des publications

Dans cette page vous auriez la possibilité de modifier des publications

Ajouter

Titre	Visibilité	Actions
simple title	Public	 



Se déconnecter

Accueil
Publications
Gestion des publications
Gestion des utilisateurs
Se déconnecter

Gestion des publications

Dans cette page vous auriez la possibilité de modifier des publications

Ajouter

Titre	Visibilité	Actions
simple title	Public	 

Ajouter une publication

Parcourir une image

Entrer un titre

☐ Private

Add Post









Gestion des utilisateurs (admin)

Accueil
Publications
Gestion des publications
Gestion des utilisateurs
Se déconnecter

Gestion des utilisateurs

Dans cette page vous auriez la possibilité de modifier un utilisateur

Ajouter

Nom & Prénom	Pseudo	Email	Rôle	Privilège	Actions
admin	N/A	admdin@admin.frr	ROLE_ADMINISTRATEUR	Non	
Ilies	Ilies	ilies@gmail.com	ROLE_UTILISATEUR	Non	 
Younes	Youes	younes@gmail.com	ROLE_UTILISATEUR	Oui	 
Fouad	foufou	foufou@gmail.com	ROLE_UTILISATEUR	Non	 
sofiane	soso	sofinaels@gmail.com	ROLE_UTILISATEUR	Oui	 

1

Formulaire d'ajout :

Ajouter un utilisateur

Full Name:

Pseudo:

Email:

Mot de passe:

Role:

Utilisateur Normal



Un utilisateur avec des privilege ☐

Ajouter

Publication (pour tous)



admin

```
version: '3.0'
services:
  front:
    build:
      context: ./front
      dockerfile: Dockerfile
    container name: deliverygofront
    restart: always
    ports:
      - "4200:3000"
    volumes:
      - type: bind
        source: ./front
        target: /front
    networks:
      - deliverygonetwork
  back:
    build:
      context: ./back
      dockerfile: Dockerfile
    restart: always
    container name: deliverygoback
    ports:
      - "8080:8080"
    volumes:
      - type: bind
        source: ./back
        target: /back
    depends_on:
      - mongo
    links:
      - mongo
    networks:
      - deliverygonetwork
  environments:
    - MONGO_URL=mongo
    - PORT=8080
```



HomePage :

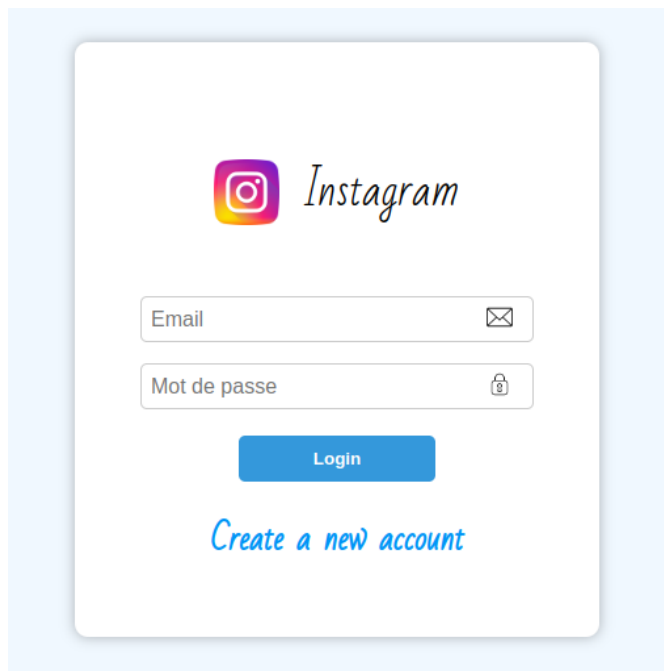


Amstagram

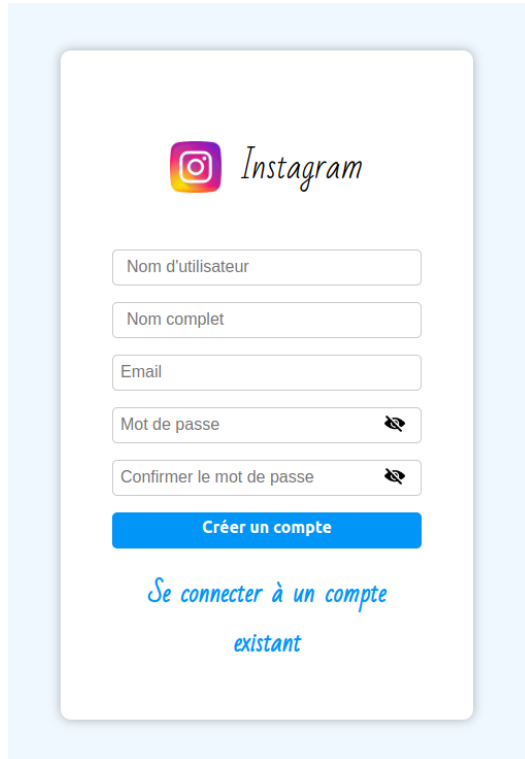
your new photo-sharing hub! Capture and share life's moments effortlessly. Explore a vibrant feed, connect with friends, and express yourself with likes and comments.

Get Started


Login :



Signup :




The image shows a digital form for creating an Instagram account. At the top, the Instagram logo (a camera icon) is followed by the word "Instagram" in its signature script font. Below this, there are five input fields: "Nom d'utilisateur", "Nom complet", "Email", "Mot de passe", and "Confirmer le mot de passe". The password fields include small eye icons to toggle visibility. A blue button labeled "Créer un compte" is positioned below the input fields. At the bottom, there is a link that says "Se connecter à un compte existant" in a blue, italicized font.


 *Instagram*

Nom d'utilisateur

Nom complet

Email

Mot de passe 

Confirmer le mot de passe 

Créer un compte

*Se connecter à un compte
existant*