EUROPEAN INSTITUTE OF TECHNOLOGY

Documentation Technique

Projet “Software Development”

PICTSMANAGER

Master of Science 1

Rédigée par
DAMODARANE Jean-Baptiste, ARIBI Mahdi, TALATIZI Kamel et ZHANG Victor

Mai 2024

Table des matières

I. Introduction

PICTSMANAGER est un projet visant à développer une application mobile et un serveur pour gérer les images. Cette documentation fournit une vue d'ensemble de l'architecture, des fonctionnalités clés et des aspects techniques du système.

1. Objectifs du Projet

Notre objectif est de créer une plateforme de gestion d'images robuste, réactive et sécurisée, offrant une expérience utilisateur optimale et évolutive.
Objectifs :

- Créer une plateforme robuste et réactive pour la gestion d'images.
- Offrir une expérience utilisateur optimale.
- Respecter les normes de sécurité et de performance.
- Implémenter une compression d'images efficace.
- Gérer les albums avec des droits d'accès appropriés.
- Assurer une recherche de contenu efficace.
- Concevoir une architecture scalable pour accompagner la croissance des utilisateurs et des données.

2. Portée de la documentation

Cette documentation technique est votre guide complet, que vous soyez développeur, testeur ou administrateur. Elle couvre tout, de l'architecture aux fonctionnalités, en passant par le déploiement et la maintenance, tout en mettant en avant les meilleures pratiques de développement.

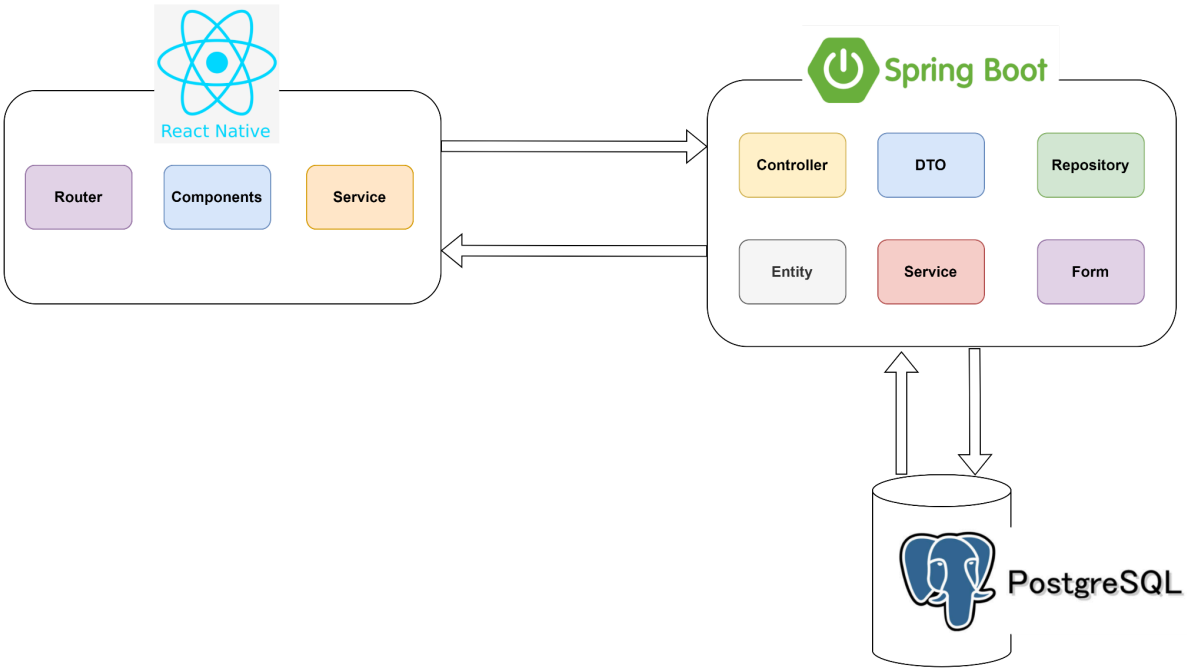
3. Principales fonctionnalitées implémentées dans le projet

Les principales fonctionnalités de l'application PICTSMANAGER comprennent :

- Prise de photos avec compression d'images
- Création, modification et suppression d'albums
- Gestion des droits d'accès aux albums
- Recherche efficace de contenu
- Réactivité et performance optimisées Sécurité des données et gestion des erreurs

II. Architecture Globale de PictsManager

1. Vue d'ensemble de l'architecture du système



Le système de gestion d'images de **Pictsmanager** repose sur une architecture complète, comprenant un backend développé avec **Spring Boot**, un frontend basé sur **React Native**, ainsi qu'une base de données **PostgreSQL** pour stocker les données.

Le rôle central du backend est de gérer la **logique métier**, la **gestion des données**, ainsi que l'**authentification** et l'**autorisation** des utilisateurs. En fournissant une API RESTful au frontend, le backend permet aux utilisateurs de **visualiser**, de **modifier** et de **gérer** facilement les **albums** et les **images** stockés dans la base de données.
Cette architecture robuste assure une séparation claire des préoccupations et permet une évolutivité et une maintenance efficaces de l'application.

2. Description des composants

- **Backend (Spring Boot) :**
  - Responsable de la logique métier et de la gestion des données.
  - Fournir une ***API RESTful*** pour le frontend.
  - Gère ***l'authentification*** et ***l'autorisation*** des utilisateurs.
  - Utilise ***Spring Data JPA*** pour interagir avec la base de données.
  - Implémente des endpoints pour les opérations ***CRUD*** (Create, Read, Update, Delete) sur les albums et les images.

- **Frontend (React Native) :**
  - ***Interface*** utilisateur permettant aux utilisateurs d'interagir avec le système.
  - Affiche les ***albums*** et les ***images*** dans une interface conviviale.
  - Permet la ***création***, la ***modification*** et la ***suppression*** d'albums et d'images.
  - Communique avec le backend via des requêtes ***HTTP*** pour récupérer et mettre à jour les données.

- **Base de données (PostgreSQL) :**
  - Stocke les données des ***utilisateurs***, des ***albums*** et des ***images***.
  - Utilise un ***modèle relationnel*** avec des tables telles que User, Album et Image.
  - Garantit ***l'intégrité des données*** à l'aide de ***contraintes*** et ***d'index***.
  - Permet des opérations efficaces de ***lecture*** et ***d'écriture*** pour assurer des performances optimales.

III. Backend

Le **backend** de **Pictsmanager** est responsable de la gestion de toutes les opérations serveur, de la logique métier et de l'accès aux données. Il fournit une **API** permettant aux clients frontend d'interagir avec les fonctionnalités de gestion des utilisateurs, des albums et des photos. Le backend assure également la **sécurité** des données et la **gestion** des **autorisations** d'accès aux ressources.

1. Description générale du backend :

Le backend de Pictsmanager suit une architecture basée sur **Spring** Boot, un framework Java utilisé pour le développement d'applications.
Voici une vue d'ensemble des composants de l'architecture :

- **Controllers** : Les contrôleurs définissent les points d'entrée de **l'API** RESTful, reçoivent les requêtes **HTTP** des clients et dirigent le flux de contrôle vers les services appropriés.

- **Services** : Les services contiennent la logique métier de l'application. Ils interagissent avec les repositories pour accéder aux données et effectuer des opérations **CRUD** (Create, Read, Update, Delete) sur les entités.

- **Repositories** : Les repositories fournissent un accès aux données stockées dans la base de données. Ils utilisent **JPA** (Java Persistence API) pour interagir avec le système de gestion de base de données.

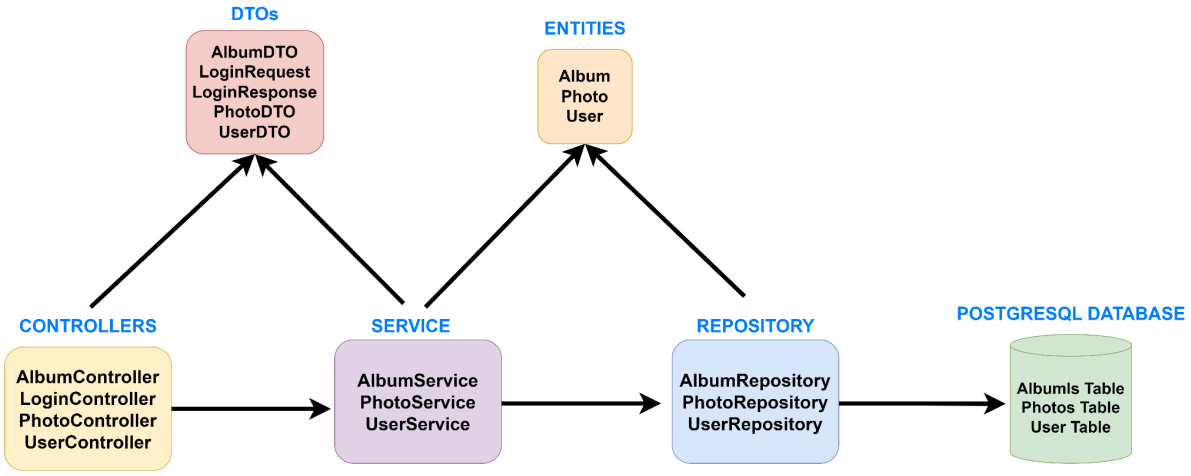
- **DTOs** (Data Transfer Objects) : Les DTOs sont des objets Java simples utilisés pour transférer des données entre les contrôleurs et les services, en séparant les entités de données de la couche de présentation.

- **Entities** : Les entités représentent les objets métier principal de l'application, tels que les utilisateurs, les albums et les photos. Elles sont persistées dans la base de données à l'aide de JPA.

2. Spécification de l'architecture logicielle

La spécification de l'architecture logicielle de Pictsmanager décrit la structure globale et l'organisation du système logiciel.

a. Diagrammes de composants :



Contrôleurs :

- **Rôle** : Les contrôleurs sont responsables de recevoir les requêtes HTTP de l'application et de les acheminer vers la logique métier appropriée.
- **Fonctionnement** : Ils interagissent avec le front (React Native) et retournent généralement des réponses HTTP appropriées, souvent au format JSON.

Contrôleur	Description
LoginController	Contrôleur pour gérer les demandes de connexion des utilisateurs
AlbumController	Classe de contrôleur pour gérer les opérations liées à l'album
PhotoController	Classe de contrôleur pour gérer les opérations liées à la photo
UserController	Classe de contrôleur pour gérer les opérations liées aux utilisateurs

Services :

- **Rôle** : Les services contiennent la logique métier de l'application. Ils encapsulent les opérations complexes ou répétitives qui ne sont pas directement liées à la manipulation des données.
- **Fonctionnement** : Les services sont souvent appelés par les contrôleurs pour effectuer des opérations telles que la validation des données, l'accès à la base de données, la manipulation des données, etc.

Service	Description
AlbumService	Classe de service pour la gestion des opérations liées aux albums
PhotoService	Classe de service pour la gestion des opérations liées à la photo
UserService	Classe de service pour gérer les opérations liées aux utilisateurs

Repositories :

- **Rôle** : Les repositories sont des interfaces qui fournissent un moyen d'accéder aux données persistantes, généralement une base de données. Ils abstraient la logique d'accès aux données de la couche de service.
- **Fonctionnement** : Les méthodes définies dans les interfaces Repository sont implémentées par les frameworks de persistance de données (comme Hibernate pour Java) pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur les entités.

Repository	Description
AlbumRepository	Interface de référentiel pour gérer les entités d'album
PhotoRepository	Interface de référentiel pour la gestion des entités photo
UserRepository	Interface de référentiel pour gérer les données utilisateur dans la base de données

Data Transfer Objects (DTO) :

- **Rôle** : Les DTO sont des objets utilisés pour transférer des données entre les différentes couches de l'application, souvent entre la couche de présentation et la couche de service.
- **Fonctionnement** : Les DTO encapsulent généralement des données spécifiques nécessaires à une opération ou à une réponse particulière. Ils sont souvent utilisés pour éviter de transmettre des entités de domaine complexes avec plus de données que nécessaire.

DTO	Description
AlbumDTO	Objet de transfert de données (DTO) pour représenter les informations de l'album
PhotoDTO	Objet de transfert de données (DTO) pour représenter les informations sur les photos
UserDTO	Classe DTO (Data Transfer Object) pour l'entité Utilisateur
LoginRequest	Classe DTO représentant une demande de connexion
LoginResponse	Classe DTO représentant une réponse de connexion

Entities :

- **Rôle** : Les entités représentent généralement les objets métier principal de l'application. Ce sont des objets qui ont une identité propre et qui sont persistés dans la base de données.
- **Fonctionnement** : Les entités sont généralement annotées avec des métadonnées spécifiques au framework de persistance (comme JPA pour Java) pour les mapper aux tables de la base de données. Elles représentent souvent des objets comme les utilisateurs, les albums, les photos, etc.

Entity	Description
Album	Classe d'entité représentant un album
Photo	Classe d'entité représentant une photo
User	Classe d'entité représentant un utilisateur dans l'application

b. Spécification de l'interface :

Cette partie détaille les **interactions** entre les différents composants du système, y compris les **API**, les **formats de données** et les **protocoles de communication**.
Les interactions entre le frontend et le backend se font principalement via des **API** RESTful.

Application Program Interface (API)

API RESTful L'architecture de Pictsmanager utilise une approche RESTful pour faciliter la communication entre le frontend et le backend. Les endpoints sont conçus pour être intuitifs et suivent les principes REST, permettant une manipulation efficace des ressources.

Voici les endpoints de l'API :

Photos:

Endpoints	Rôle
/api/photo/upload (POST)	Télécharge une nouvelle photo.
/api/photo/user/{userId}/album/{albumId} (GET)	Récupère les photos d'un utilisateur dans un album spécifique.
/api/photo/user/{userId} (GET)	Récupère les photos d'un utilisateur.
/api/photo/user/{userId}/visibility/{visibility} (GET)	Récupère les photos d'un utilisateur avec une visibilité spécifique.
/api/photo/public/photos (GET)	Récupère toutes les photos publiques.
/api/photo/image/{filename} (GET)	Récupère une image par le nom de ce fichier.
/api/photo/delete/{photoId} (DELETE)	Supprime une photo spécifique par son ID.
/api/photo/album/{albumId}/last (GET)	Récupère la dernière photo d'un album spécifique.
/api/photo/visibility/{photoId}	Met à jour l'accès à des photos

Users :

Endpoints	Rôle
/api/users/adduser (POST)	Ajoute un nouvel utilisateur à la base de données.
/api/users/getusers (GET)	Récupère la liste de tous les utilisateurs enregistrés.
/api/users/{id} (GET)	Récupère les détails d'un utilisateur spécifique par son ID.
/api/users/update/user/{id} (PUT)	Met à jour les informations d'un utilisateur existant.
/api/users/getuser (GET)	Récupère les détails de l'utilisateur à partir du token JWT.

Albums :

Endpoints	Rôle
/api/album/create (POST)	Crée un nouvel album.
/api/album/all (GET)	Récupére la liste de tous les albums.
/api/album/{albumId} (GET)	Récupère les détails d'un album spécifique par son ID.
/api/album/min-id/user/{userId}	Récupère l'id minimum de l'album associé à l'id de l'utilisateur.
/api/album/{albumId} (DELETE)	Supprime un album spécifique par son ID.

Formats de données

Les **données échangées** entre le **frontend** et le **backend** sont généralement au format **JSON** (JavaScript Object Notation), qui est largement utilisé et facilement parsable par les applications.

Exemple de **JSON** pour se connecter (Login) :

POST http://localhost:8080/api/login

JSON renvoyé au serveur :

```
{
  "email":"john@gmail.com",
  "password":"jojo"
}
```

Protocole de Communication

Les communications :

- **Type de protocole** : HTTP
- **Description** : Utilisé pour le transfert de données entre le frontend et le backend.
- **Sécurité** : Utilisation de HTTP au lieu de HTTPS, donc les données ne sont pas chiffrées.
- **Recommandation** : Bien que HTTP soit utilisé, l'adoption de HTTPS est recommandée pour garantir la confidentialité et l'intégrité des données.
- **Avantages** : Offre une communication standardisée et largement acceptée sur le web.
- **Considérations** : Moins sécurisé que HTTPS, ce qui peut poser des risques en termes de confidentialité des données.

3. Attributs de qualité :

Pour Pictsmanager, nous avons accordé une attention particulière aux attributs de qualité suivants :

- **Performance** :
  - **Description** : Assurer des temps de réponse rapides et une expérience utilisateur fluide.
  - **Approc he** : Optimisation des requêtes SQL, utilisation d'algorithmes de compression efficaces.

- **Sécurité** :
  - **Description** : Protection des données et des utilisateurs contre les accès non autorisés.
  - **Approche** : Authentification des utilisateurs, gestion rigoureuse des autorisations, sécurité des échanges de données.

- **Fiabilité** :
  - **Description** : Garantir un fonctionnement prévisible et une disponibilité élevée.
  - **Approche** : Réduction des points de défaillance unique, tests rigoureux, mécanismes de sauvegarde.

- **Évolutivité** :
  - **Description** : Capacité à gérer efficacement une augmentation de la charge de travail.
  - **Approche** : Architecture modulaire et extensible, déploiement sur des infrastructures cloud, techniques de mise en cache et de répartition de charge.

4. Qualification du logiciel

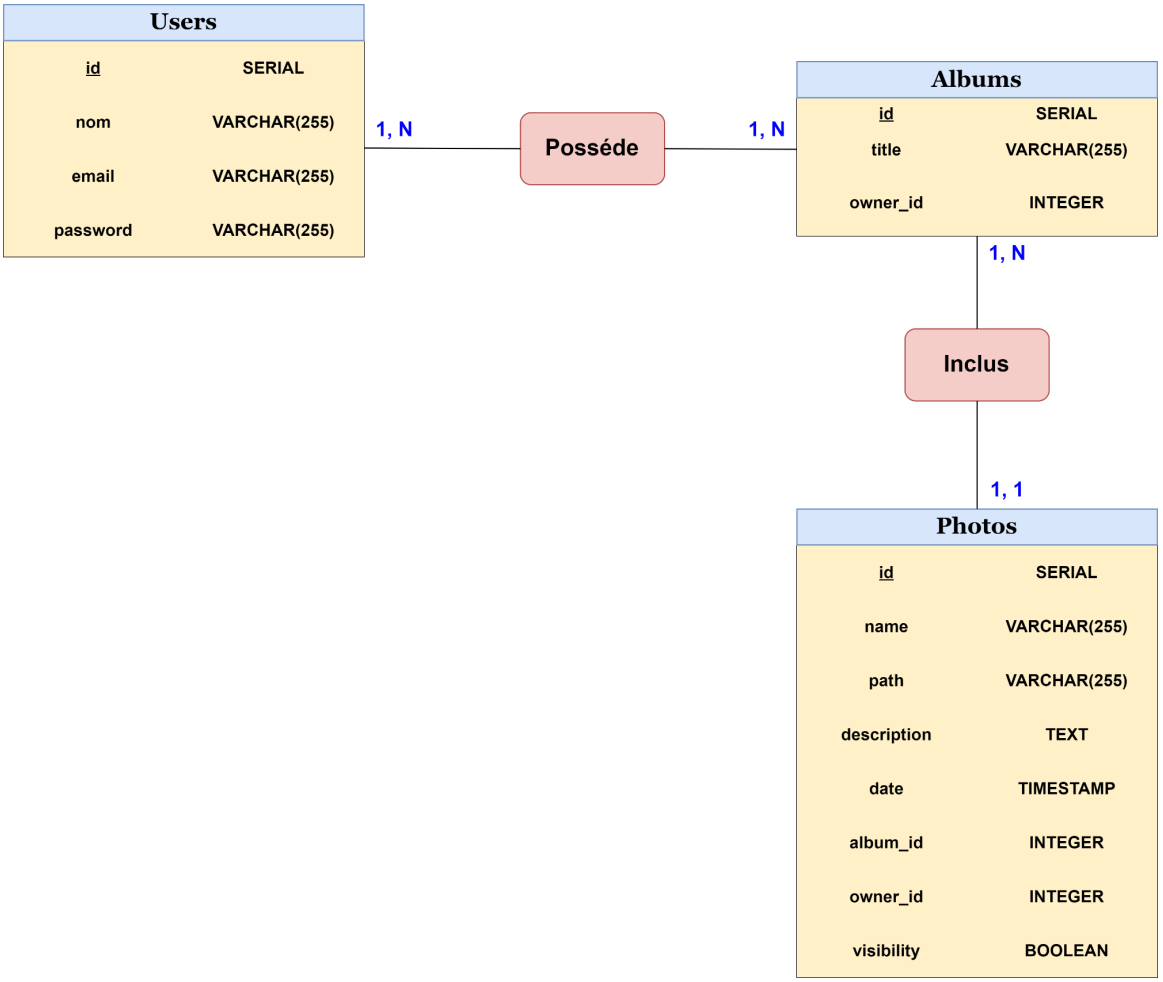
La qualification de Pictsmanager vise à garantir la conformité aux exigences fonctionnelles et non fonctionnelles, ainsi qu'à assurer la qualité globale du système.

- **Tests Fonctionnels**
  - Pour vérifier que toutes les fonctionnalités du système, telles que la création d'albums, l'ajout de photos, etc..
- **Tests de Performance**
  - Pour évaluer la réactivité du système, la vitesse de chargement des pages et la gestion de la charge.
- **Tests de Sécurité**
  - Pour identifier et corriger les vulnérabilités potentielles, notamment en matière d'authentification, d'autorisation et de protection des données.
- **Tests d'Intégration**
  - Pour garantir que tous les composants du système fonctionnent ensemble de manière harmonieuse et sans conflit.
- **Tests d'Utilisabilité**
  - Pour évaluer la convivialité de l'interface utilisateur et la facilité d'utilisation globale du système, en prenant en compte les retours des utilisateurs.
- **Tests de Régression**
  - Pour s'assurer qu'aucun nouveau code n'a introduit de régressions dans les fonctionnalités existantes.

5. Modèle de données :

Dans le modèle de données de PICTSMANAGER, chaque table incarne un pilier essentiel de l'écosystème photographique, depuis la gestion des utilisateurs jusqu'à l'organisation minutieuse des albums et des clichés, offrant ainsi une structure robuste pour une expérience utilisateur immersive et fluide.

a. Schéma MCD (Modèle Conceptuel des Données)

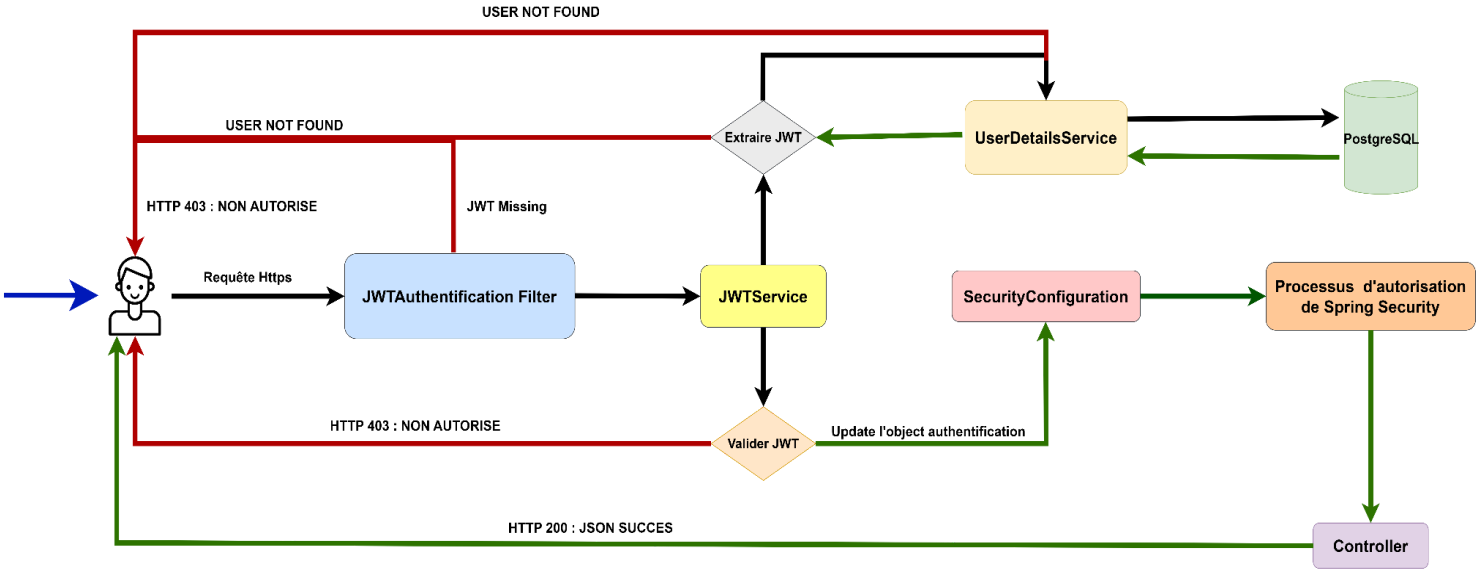


La table "**users**" stocke les informations sur les utilisateurs, telles que leur nom, leur adresse e-mail et leur mot de passe.

La table "**albums**" contient les détails sur les albums créés par les utilisateurs, avec notamment leur titre et l'identifiant de leur propriétaire.

La table "**photos**" enregistre les informations sur les photos téléchargées, telles que leur nom, leur chemin d'accès, leur description, leur date de création, ainsi que l'identifiant de l'album auquel elles appartiennent et l'identifiant de leur propriétaire. De plus, elle indique la visibilité de la photo (publique ou non).

6. Authentification et autorisations :

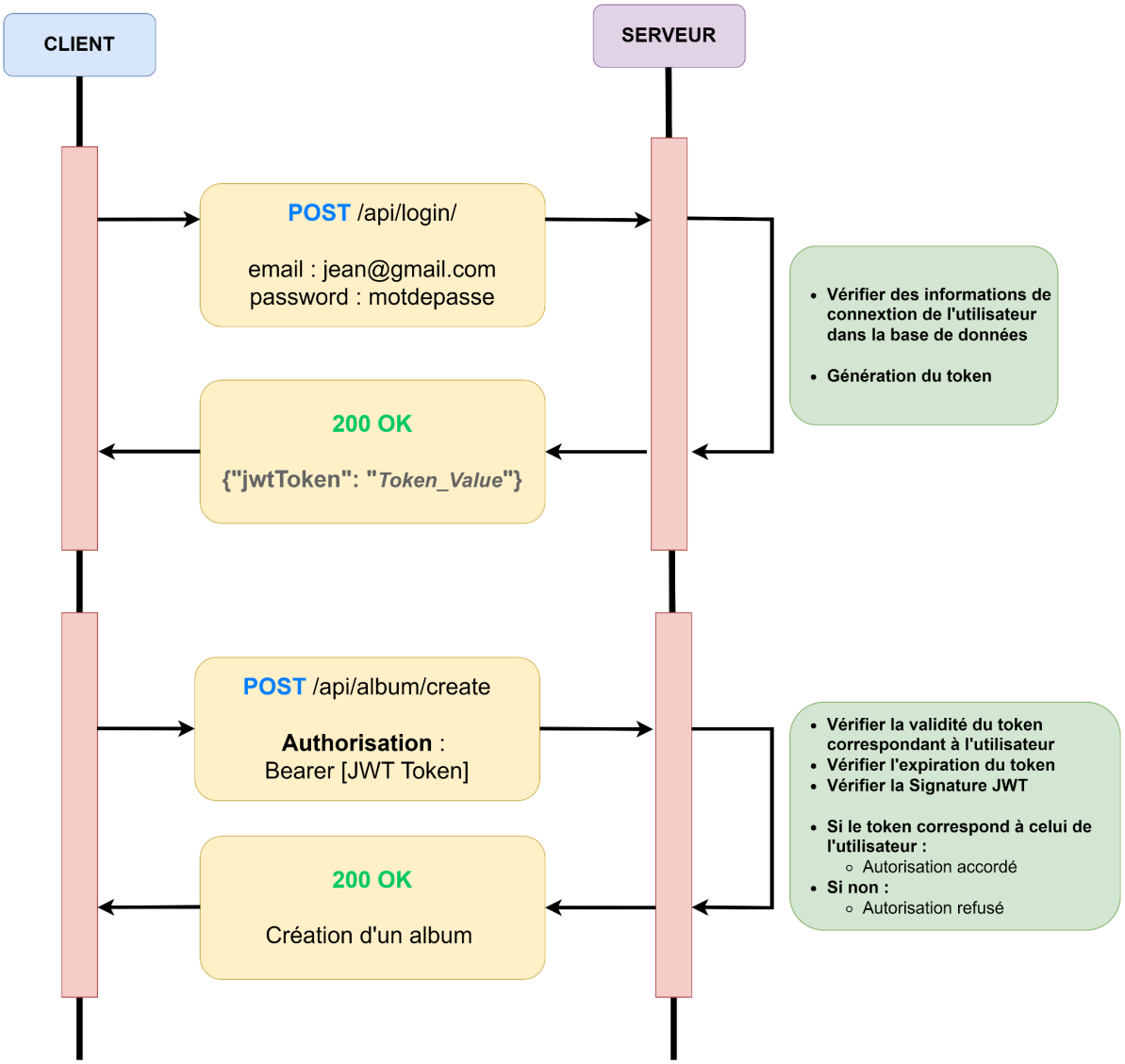


Authentification :

Lorsqu'un utilisateur se connecte avec succès à l'application en fournissant son adresse e-mail et son mot de passe, le backend vérifie ces informations par rapport à celles stockées dans la base de données. Si les informations sont valides, un **JWT** est généré et envoyé au client.

Lorsqu'un nouvel utilisateur est enregistré, son **mot de passe** est **haché** à l'aide de l'encodeur de mot de passe (**PasswordEncoder**) et enregistré dans la base de données. Une fois l'utilisateur enregistré, un JWT est généré à l'aide de la classe JwtUtil et renvoyé au client.

Autorisation :



Une fois authentifié, l'utilisateur envoie ce **JWT** dans l'en-tête Authorization de chaque requête ultérieure. Le backend vérifie alors la validité et l'intégrité du **token**, puis extrait les informations d'identification de l'utilisateur. Sur la base de ces informations, le backend détermine si l'utilisateur est autorisé à accéder à la ressource demandée.

Le **JWT** envoyé par l'utilisateur est extrait et validé dans les **filtres de sécurité** configurés dans la classe **SecurityConfig**. Ces filtres vérifient la signature du **JWT**, extraient les informations d'identification de l'utilisateur et les rendent disponibles pour les autres parties du code pour effectuer des vérifications d'autorisation appropriées.
Si le token **JWT** est valide, alors l'utilisateur est autorisé à utiliser des services front end faisant aux endpoints du back end.

7. Configuration Backend de PictsManager :

Pour configurer l'application Pictsmanager, suivre attentivement ces étapes pour garantir le bon fonctionnement de l'application :

Fichiers de propriétés :

Dans le dossier **src/main/resources**, vous trouverez le fichier **application.properties**. Ce fichier contient plusieurs paramètres de configuration pour votre application Spring Boot, y compris la configuration de la base de données et d'autres paramètres spécifiques à l'application. Vous pouvez personnaliser ces paramètres selon vos besoins.

Base de données :

Assurez-vous d'avoir une base de données **PostgreSQL** installée et configurée. Vous pouvez spécifier les détails de connexion à la base de données dans le fichier application.properties. Voici un exemple de configuration de base de données dans ce fichier :

```
spring.datasource.url= jdbc:postgresql://localhost:5432/pictsmanager
spring.datasource.username= postgres
spring.datasource.password= postgres
spring.datasource.driver-class-name= org.postgresql.Driver
```

Assurez-vous de remplacer username et password par vos informations d'identification de base de données.

Dépendances Maven :

Assurez-vous que votre fichier pom.xml contient toutes les dépendances Maven nécessaires pour Spring Boot, y compris **Spring Security** et **JWT**. Veillez à ce que les versions des dépendances soient compatibles entre elles et avec Spring Boot.

Sécurité :

La classe **SecurityConfig** gère la configuration de sécurité, en définissant les filtres pour l'authentification et l'autorisation des utilisateurs. Assurez-vous que cette classe est correctement configurée pour répondre aux exigences de sécurité de votre application. Adaptez la configuration de sécurité selon vos besoins spécifiques, en ajoutant des filtres d'authentification et en configurant des stratégies d'autorisation.

8. Gestion des erreurs :

Dans l'API Pictsmanager, les erreurs sont gérées de manière à renvoyer des réponses **HTTP** appropriées avec des informations sur l'erreur. Voici les erreurs les plus courantes possibles de PictsManager :

Code HTTP	Erreur	Explication
400	Bad Request	La requête envoyée par le client est incorrecte ou mal formée.
401	Unauthorized	L'utilisateur n'est pas authentifié ou n'a pas les autorisations nécessaires pour accéder à la ressource demandée.
403	Forbidden	L'utilisateur authentifié n'a pas les autorisations nécessaires pour accéder à la ressource demandée.
404	Not Found	La ressource demandée n'a pas été trouvée sur le serveur.
409	Conflict	Un conflit est survenu lors de la tentative de traitement de la requête, souvent en raison de données en conflit.
500	Internal Server Error	Une erreur interne du serveur s'est produite, souvent en raison d'un dysfonctionnement ou d'une exception inattendue.

9. Tests :

Dans le cadre du développement de Pictsmanager, les tests sont importants pour garantir la fiabilité et la qualité du logiciel.

Pour cela, nous avons utilisé **JUnit**, un framework de test unitaire pour Java, et **Mockito**, une bibliothèque de mocking, en conjonction avec Spring Boot.

- **Utilisation de JUnit :**
  - Test Unitaires : Avec **JUnit**, nous avons écrit des tests unitaires pour chaque composant de notre application, y compris les contrôleurs, les services et les repositories.
  - Annotations Spring : Nous avons intégré les annotations de Spring dans nos tests pour bénéficier de fonctionnalités telles que l'injection de dépendances et la gestion des contextes d'application.

- **Utilisation de Mockito :**
  - Mocking des Dépendances : **Mockito** nous a permis de créer des mocks pour simuler le comportement des dépendances externes, telles que les repositories ou les services.
  - Facilité d'Utilisation : Grâce à Mockito, nous avons pu simplifier nos tests en remplaçant les dépendances réelles par des mocks contrôlables, ce qui nous a permis de simuler différents scénarios et de tester des cas limites.

- **Intégration avec Spring Boot :**
  - Annotations Spring Boot Test : Nous avons utilisé les annotations de test fournies par Spring Boot, telles que **@SpringBootTest** et **@MockBean**, pour créer et configurer automatiquement les contextes d'application nécessaires à nos tests.
  - Tests d'Intégration : En utilisant ces annotations, nous avons pu écrire des tests d'intégration pour vérifier le bon fonctionnement de nos composants dans un environnement similaire à celui de production.

IV. Frontend

A. Description générale du frontend

a. Aperçu de l'interface utilisateur

PictsManager est développée avec React Native. Elle offre une interface utilisateur intuitive et conviviale pour la gestion et le partage de photos. L'interface utilisateur est conçue pour être esthétique et facile à naviguer, avec une attention particulière portée à la simplicité et à la cohérence du design.

b. Fonctionnement général de l'application

L'application permet aux utilisateurs de visualiser leurs albums photo, d'ajouter de nouvelles photos, de créer de nouveaux albums, de partager des photos avec d'autres utilisateurs, et plus encore. Voici quelques fonctionnalités clés :

- **Authentification** : Les utilisateurs peuvent se connecter à leur compte ou créer un nouveau compte s'ils n'en ont pas déjà un.
- **Navigation** : La navigation entre les différentes sections de l'application est fluide et intuitive, facilitée par un système de navigation bien conçu.
- **Gestion des albums** : Les utilisateurs peuvent créer, afficher et supprimer des albums photo, ainsi que ajouter et supprimer des photos à/from ces albums.
- **Partage de photos** : Les utilisateurs peuvent partager leurs photos avec d'autres utilisateurs de l'application, en leur donnant accès à des albums spécifiques.

c. Technologies utilisées

- **React Native** : Framework JavaScript pour le développement d'applications mobiles multiplateformes.
- **React Navigation** : Bibliothèque de navigation pour React Native, facilitant la navigation entre les écrans.
- **Axios** : Bibliothèque HTTP pour effectuer des requêtes réseau depuis l'application.
- **AsyncStorage** : API pour le stockage local des données de l'application, comme les jetons d'authentification.

B. Architecture

a. Composants principaux

L'architecture du frontend de PictsManager repose sur des composants principaux qui facilitent la modularité, la réutilisabilité et la maintenance du code.

Voici les principaux composants utilisés dans l'application :

- **Screens** :
  Les screens correspondent à chaque vue distincte de l'application. Voici les écrans principaux de l'application PictsManager :
  - **EditUser.js** :
    - Écran de modification des informations utilisateur.
  - ForgotPassword.js :
    - Écran permettant aux utilisateurs de réinitialiser leur mot de passe.
  - **Home.js** :
    - Écran principal de l'application affichant les albums et les photos.
  - **Login.js** :
    - Écran de connexion pour les utilisateurs.
  - **Logout.js** :
    - Écran permettant aux utilisateurs de se déconnecter de l'application.
  - **Signup.js** :
    - Écran d'inscription pour les nouveaux utilisateurs.
  - **Users.js** :
    - Écran affichant la liste des utilisateurs et leurs photos.

- **Components** :
  Les composants sont des éléments d'interface utilisateur réutilisables qui peuvent être utilisés dans plusieurs écrans ou dans d'autres composants. Voici quelques exemples de composants utilisés dans PictsManager :
  - **Custom Button.js** :
    - Composant de boutons personnalisés pour une utilisation dans toute l'application.
  - **Album.js** :
    - Composant représentant un album avec ses détails.
  - **Photo.js** :
    - Composant représentant une photo avec ses détails.
  - **User.js** :
    - Composant représentant un utilisateur avec ses détails.

- **Navigation** :
  La navigation entre les écrans est gérée par le fichier AppNavigation.js, qui utilise React Navigation pour configurer les différentes méthodes de navigation telles que les piles, les onglets, etc.

b. Gestion de l'Etat

Dans l'application PictsManager, nous utilisons principalement le Context API de React pour gérer l'état de l'application. Voici comment cela fonctionne de manière concise :

- **Context API de React :**
  - Le Context API est utilisé pour gérer l'état global de l'application, tel que les données utilisateur, les informations sur les albums et les photos, etc.
  - Il permet de créer des contextes isolés pour différents types de données, offrant ainsi une organisation claire et modulaire.

- **Hooks :**
  - Nous utilisons principalement les hooks *useState* et *useEffect* pour gérer l'état local et les effets secondaires dans les composants fonctionnels.
  - *useState* est utilisé pour déclarer et mettre à jour l'état local d'un composant.
  - *useEffect* est utilisé pour exécuter des actions en réponse à des changements d'état ou de cycle de vie du composant, comme les appels à des API.

- **Communication entre composants :**
  - Les données sont transmises entre les composants via les propriétés (props) et le contexte.
  - Les composants peuvent accéder aux données du contexte en utilisant le hook *useContext*, simplifiant ainsi la communication entre les composants.

c. Navigation

PictsManager utilise React Navigation pour gérer la navigation entre les écrans.

- **React Navigation :**
  - React Navigation est une bibliothèque de navigation pour React Native, permettant de définir la structure de navigation de manière déclarative.
  - Elle prend en charge un seul type de navigation (normal).

- **Configuration de la navigation :**
  - La configuration de la navigation se fait généralement dans le fichier App.js.
  - Dans ce fichier, les différents écrans de l'application sont importés et configurés avec leurs options de navigation.

- Voici un **exemple** de configuration de la navigation par pile (stack) dans App.js:

```
<NavigationContainer>
    <Stack.Navigator>
     <Stack.Screen
      name="Login"
      component={Login}
      options={{ headerShown: false }}
     />
     <Stack.Screen
      name="HomeTabs"
      component={AppNavigator}
      options={{ headerShown: false }}
     />

     <Stack.Screen
      name="PhotoList"
      component={PhotoList}
      options={{ headerShown: true }}
     />
     <Stack.Screen
      name="Signup"
      component={Signup}
      options={{ headerShown: false }}
     />
     <Stack.Screen
      name="Logout"
      component={Logout}
      options={{
       headerTitle: "",
       headerBackVisible: true,
       headerStyle: { backgroundColor: "black" },
       headerTintColor: "white",
      }}
     />
    </Stack.Navigator>
 </NavigationContainer>
```

Cette configuration utilise React Navigation avec un conteneur de navigation <NavigationContainer> et un conteneur de pile <Stack.Navigator>. Chaque écran est défini avec <Stack.Screen> et associé à un nom et un composant. Les options spécifiques, comme la visibilité de l'en-tête et le style, sont définies pour chaque écran. Par exemple, "Login" et "HomeTabs" ont l'en-tête masqué, tandis que "PhotoList", "Signup", "ForgotPassword", et "Logout" ont des configurations spécifiques pour l'en-tête.

- Navigation entre écrans :

  Pour naviguer entre les écrans, nous utilisons les méthodes fournies par React Navigation, telles que **navigate**, **push**, **goBack**, etc.
  Par exemple, pour naviguer de LoginScreen à HomeScreen, nous utilisons `navigation.navigate('Home')`.

d. Appels API

Les appels API depuis le frontend sont effectués principalement à l'aide de la bibliothèque Axios, qui simplifie l'envoi de requêtes HTTP.

Schéma du fonctionnement d'appels d'API PictsManager :

Voici un aperçu de la gestion des appels API dans l'application :

- **Configuration Axios** : Une **URL** de base est définie pour simplifier les appels **API**.
- **Effectuer une requête** : Les requêtes sont envoyées à l'aide d'une fonction asynchrone qui utilise Axios pour envoyer une requête à l'URL appropriée avec les paramètres nécessaires.
- **Gestion des réponses** : Si la réponse est réussie (statut **HTTP 200**), les données sont extraites et traitées. En cas d'échec, une erreur est gérée et un message approprié est affiché.
- **Indicateurs de chargement et messages d'erreur** : Des indicateurs de chargement sont affichés pendant le chargement de la requête, et des messages d'erreur sont affichés en cas d'échec de la requête.

C. Styles

- **Feuilles de style StyleSheet** : Les styles sont définis dans des objets StyleSheet.create, ce qui permet de garantir des performances optimales en évitant les calculs inutiles à chaque rendu.

- **Utilisation des propriétés de style** : Les propriétés de style telles que backgroundColor, color, fontSize, etc., sont utilisées pour définir l'apparence des composants.

- **Réutilisation de composants** : Les composants de style, tels que CustomTabButton, sont créés pour encapsuler le style et le comportement répétitifs, favorisant ainsi la réutilisation du code.

- **Intégration avec d'autres bibliothèques** : Des bibliothèques tierces comme react-native-vector-icons sont intégrées pour ajouter des icônes stylisées aux composants.

- **Gestion des états** : Les styles peuvent également être conditionnels, par exemple, la couleur de fond d'un bouton peut changer en fonction de son état (isLoading dans l'exemple donné).

- Exemple d'implémentation du style dans le frontend :

```
const styles = StyleSheet.create({
  button: {
    flex: 1,
    alignItems: "center",
    justifyContent: "center",
    backgroundColor: "#000",
  },
  buttonText: {
    color: "#FFF",
    fontSize: 9,
  },
});
```

D. Configuration du frontend

Pour déployer et configurer correctement le frontend de PictsManager, suivre ces étapes:

- **Sur téléphone :**
    - Activer le mode développeur sur votre téléphone.
    - Activer le débogage USB dans les paramètres développeur de votre téléphone.
- **Sur le terminal du projet :**
    - S’assurer que votre téléphone est correctement connecté à votre ordinateur via USB.
    - Ouvrir un terminal et exécutez la commande ***adb devices*** pour vous assurer que votre téléphone est détecté par votre ordinateur.
    - Ensuite, exécuter la commande ***npx react-native run-android*** dans le répertoire racine de votre projet pour lancer l'application sur votre téléphone Android.
- **Pour configurer l'API :**
    - Sur votre ordinateur, ouvrez une invite de commande ou un terminal.
    - Exécutez la commande ***ipconfig*** dans l'invite de commande pour afficher les informations de configuration du réseau.
    - Recherchez votre adresse IPv4 dans les résultats affichés.
    - Utilisez cette adresse IPv4 comme URL de base pour accéder à votre API depuis l'application frontend.

V. Conclusion

En conclusion, PictsManager est une application complète de gestion d'images, offrant une expérience utilisateur optimale grâce à une interface intuitive et conviviale. L'architecture robuste et évolutive repose sur un backend Spring Boot, un frontend React Native et une base de données PostgreSQL. Les fonctionnalités clés incluent la prise de photos avec compression, la gestion d'albums, les droits d'accès, la recherche efficace et la sécurité des données.

Cette documentation technique fournit une vue d'ensemble détaillée de l'architecture, des composants et des fonctionnalités de PictsManager, servant de guide complet pour les développeurs, testeurs et administrateurs.