AREA

Documentation technique

ADELE DE PREMONVILLE NOAH LE VEVE LUCAS HOLCZINGER HUGO PATTEIN LOUIS CHAMBON

SOMMAIRE

Présentation AREA

AREA (**Action REAction**) est une plateforme d'automatisation conçue pour automatiser des interactions entre divers services.

Inspirée par des plateformes comme **IFTTT** (**If This Then That**) et **Zapier**, **AREA** vise à fournir aux utilisateurs un moyen d'automatiser des tâches répétitives.

Objectifs Principaux :

- Fournir une interface utilisateur **intuitive** permettant de **créer**, **gérer** des automatisations.
- Intégrer une variété de **services populaires** et permettre des **interactions fluides** entre eux.

Fonctionnalités Clés :

- **Création d'Automatisations** : Les utilisateurs peuvent créer des automatisations en liant une action d'un service à une réaction d'un autre service.
- **Gestion des Utilisateurs** : Les utilisateurs peuvent s'inscrire, se connecter et gérer leurs profils.
- **Gestion des Services** : Les utilisateurs peuvent connecter et gérer différents services à leur compte AREA.

Architecture Générale :

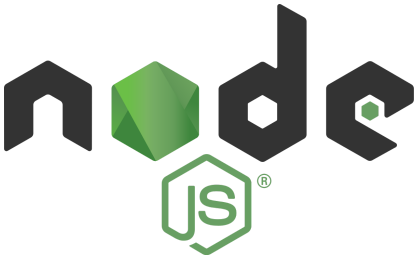
- **Serveur** : Le cœur de AREA, gère les utilisateurs, les services, et les automatisations.
- **Client Web** : Une interface web permettant aux utilisateurs d'interagir avec la plateforme.
- **Client Mobile** : Une application mobile permettant une interaction sur les appareils mobiles.
- **Base de Données** : Stocke les données des utilisateurs, les configurations des services et les automatisations.

Technologies utilisées

Dans cette section, nous décrivons les **technologies clés utilisées** pour construire et déployer **AREA**, en couvrant la **base de données**, le **serveur**, les clients **mobile** et **web**, ainsi que l'environnement de **containerisation**.

Base de Données :

- **MongoDB** : Base de données NoSQL choisie pour sa flexibilité et sa capacité à gérer des données structurées et non structurées (stockage des données dans des documents au format BSON).
- **Mongo Compass** : Utilisé pour une représentation visuelle et une interaction facile avec la base de données MongoDB.

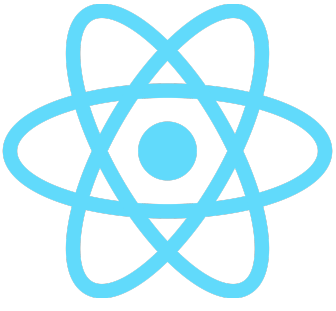


Serveur :

- **Node.js** : Environnement d'exécution côté serveur basé sur JavaScript, choisi pour son écosystème riche en librairies.

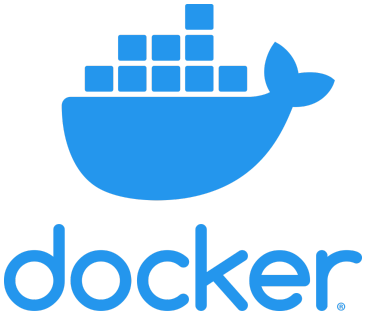
Client Mobile :

- **Flutter** : SDK de développement mobile de Google, permettant de créer des applications natives de haute qualité pour iOS et Android à partir d'une seule base de code.



Client Web :

- **React** : Bibliothèque JavaScript pour construire des interfaces utilisateur, choisie pour sa modularité.



Containerisation et Déploiement :

- **Docker** : Utilisé pour containeriser l'application et assurer la consistance des environnements de déploiement.
  - **Docker Compose** : Utilisé pour définir et gérer les services multi-conteneurs de Docker.

Librairies utilisées

Serveur :

- **Axios**: Utilisé pour les requêtes HTTP.
- **Bcrypt et Bcryptjs**: Utilisés pour le hachage des mots de passe.
- **Express**: Cadre de serveur pour gérer les routes et les requêtes.
- **Mongoose**: Outil pour interagir avec MongoDB.
- **Passport et ses stratégies**: Authentification avec OAuth pour le service Google.
- **Nodemailer**: Envoi d'emails.
- **Jsonwebtoken**: Gestion des tokens JWT pour l'authentification.
- **Swagger-jsdoc et Swagger-ui-express**: Documentation API.

Client Web :

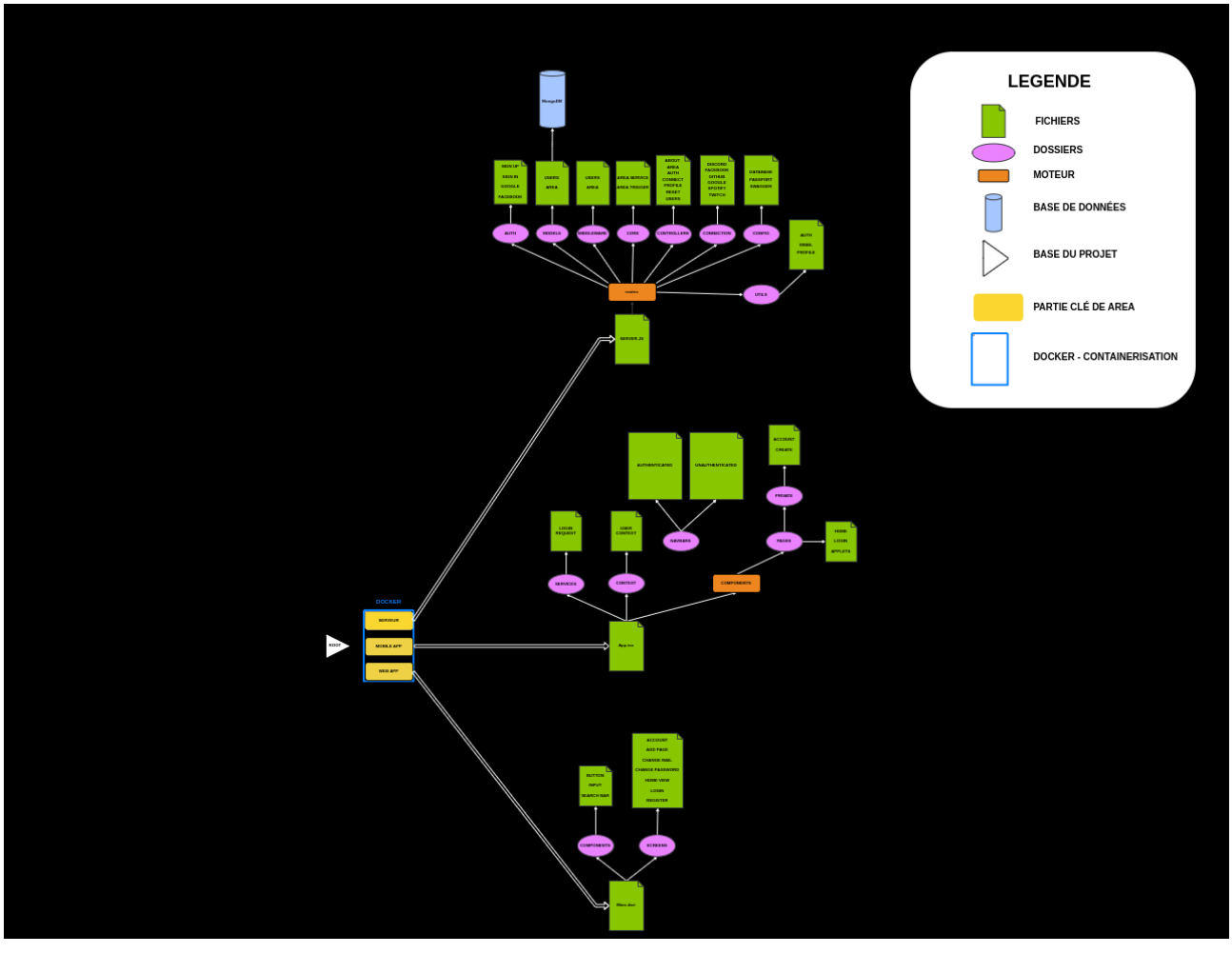
- **React**: Bibliothèque pour construire l'interface utilisateur.
- **Axios**: Utilisé pour les requêtes HTTP.
- **React-Icons**: Bibliothèque d'icônes.
- **React-Router-Dom**: Gestion des routes dans l'application.

Client Mobile (Flutter) :

- **HTTP**: Utilisé pour les requêtes HTTP.
- **Shared Preferences**: Stockage de données simples de manière persistante.
- **Cupertino Icons**: Bibliothèque d'icônes de style iOS.
- **Flutter Lints**: Règles d'analyse statique pour encourager les bonnes pratiques de codage.

Ces librairies ont été sélectionnées pour faciliter le développement et améliorer l'expérience utilisateur sur chaque plateforme.

Diagramme AREA



https://drive.google.com/file/d/1jzOh1jPY-B9QYK48_Vk2jUxgHF_oNVVn/view?usp=drive_link

API et services externes

API

- **Endpoints /about** :
  - **GET** /about/about.json : Récupère des informations sur le serveur et ses services.
- **Endpoints /areas** :
  - **GET** /areas : Liste toutes les areas disponibles.
  - **POST** /areas : Crée une nouvelle area en spécifiant les services, les noms, et les paramètres pour l'action et la réaction.
  - **GET** /areas/{id} : Obtient une area spécifique par son ID.
  - **PUT** /areas/{id} : Met à jour une area par son ID.
  - **DELETE** /areas/{id} : Supprime une area par son ID.
- **Endpoints /auth** :
  - **POST** /auth/sign-up : Inscription d'un nouvel utilisateur.
  - **GET** /auth/confirm/{token} : Confirmation de l'inscription d'un utilisateur.
  - **GET** /auth/confirm-email-change/{token} : Confirmation du changement d'email d'un utilisateur.
  - **POST** /auth/sign-in : Connexion d'un utilisateur.
  - **GET** /auth/google et GET /auth/google/callback : Authentification et callback de Google.
- **Endpoints /connect** :
  - Endpoints pour connecter et gérer le callback de divers services tels que Google (Gmail, Youtube) , GitHub, Spotify, et Twitch.
- **Endpoints /profile** :
  - **GET** /profile : Récupère le profil de l'utilisateur authentifié.
  - **PUT** /profile/update : Met à jour le profil de l'utilisateur authentifié.
- **Endpoints /reset** :
  - **POST** /reset/request-password-reset : Envoie un lien de réinitialisation du mot de passe à l'email de l'utilisateur.
  - **GET** /reset/password/{token} : Affiche le formulaire (ou la page) pour que l'utilisateur définisse un nouveau mot de passe.
  - **POST** /reset/password : Met à jour le mot de passe de l'utilisateur en utilisant le token fourni et le nouveau mot de passe.
- **Endpoints /users** :
  - **DELETE** /users/delete : Supprime l'utilisateur connecté.

Ces **endpoints** permettent une interaction fluide entre le serveur, les clients web et mobile, et les services externes.

API compléte disponible ici: http://localhost:8080/api-docs/

Services Externes Intégrés

Google: Authentification & Connexion OAuth.

GitHub: Connexion pour les utilisateurs GitHub.

Spotify: Connexion pour les utilisateurs Spotify.

Twitch: Connexion pour les utilisateurs Twitch.

Ces services externes sont intégrés dans AREA, permettant aux utilisateurs de lier leurs comptes de ces plateformes avec leur profil AREA, facilitant ainsi l'automatisation des interactions entre divers services.

Architecture serveur

Structure des Fichiers et Répertoires :

L'architecture du serveur est organisée en plusieurs répertoires et fichiers pour une meilleure séparation des préoccupations et une gestion simplifiée. Voici une description sommaire de la structure :

Racine du Serveur :

- **docker-compose.yml** : Fichier de configuration pour Docker Compose.
- **Dockerfile** : Fichier pour construire l'image Docker du serveur.
- **package.json & package-lock.json** : Gestion des dépendances.
- **server.js** : Fichier principal pour démarrer le serveur.

Répertoire src :

- **area** :
  - Scripts spécifiques aux areas.
- **auth** :
  - Stratégies d'authentification.
- **config** :
  - Configurations pour la base de données, Passport et Swagger.
- **connection** :
  - Stratégies de connexion pour divers services externes comme GitHub, Google, Spotify et Twitch.
- **controllers** :
  - Contrôleurs pour gérer les logiques métier des différentes routes.
- **core** :
  - Services et déclencheurs pour la logique centrale des areas.
- **middleware** :
  - Middleware personnalisé pour traiter les requêtes.
- **models** :
  - Modèles pour les schémas de la base de données.
- **routes** :
  - Définition des routes pour chaque contrôleur.
- **routes.js** : Fichier centralisant l'ensemble des routes.
- **utils** :
  - Utilitaires pour l'authentification, l'email et la gestion des profils.

Cette organisation permet une **maintenance** et une **évolution** facilitées du serveur.

Architecture mobile

Structure des Fichiers et Répertoires :

L'architecture de l'application mobile est bien organisée pour faciliter le développement et la maintenance. Voici une description sommaire de la structure :

Répertoire components :

- Contient les composants réutilisables de l'interface utilisateur.
- **my_button.dart** : Composant pour les boutons.
- **my_input.dart** : Composant pour les champs de saisie.
- **search_bar.dart** : Composant pour la barre de recherche.

Répertoire screens :

- Contient les **différentes pages ou écrans** de l'application.
- **account_page.dart** : Page de gestion du compte utilisateur.
- **add_page.dart** : Page pour ajouter de nouvelles fonctionnalités ou informations.
- **change_mail_page.dart** : Page pour changer l'adresse email.
- **change_password_page.dart** : Page pour changer le mot de passe.
- **home_view.dart** : Page d'accueil de l'application.
- **login_screen.dart** : Écran de connexion.
- **register_screen.dart** : Écran d'inscription.

Fichiers à la Racine :

- **main_container.dart** : Container principal qui peut être utilisé pour gérer l'état global ou les dépendances de l'application.
- **main.dart** : Point d'entrée de l'application où la fonction main() est définie, et où l'application Flutter est initialisée.

Cette structure permet une **séparation** claire entre les **composants réutilisables**, les écrans de l'application, et le point d'entrée de l'application. Elle facilite également la navigation et la compréhension du code pour les développeurs.

Architecture web

Structure des Fichiers et Répertoires :

L'architecture du site web est structurée de manière à séparer clairement les différentes parties du code, facilitant ainsi son développement et sa maintenance. Voici une description sommaire de cette structure :

Répertoire assets :

- Contient les ressources statiques comme les images, les polices, etc.

Répertoire public :

- Contient les fichiers statiques de base de l'application web comme l'icône de la favicone, le HTML d'index, et les logos.

Répertoire src :

- Le code source de l'application.
- **App.tsx**, **App.css** : Point d'entrée principal de l'application et son fichier CSS associé.
- **index.tsx**, **index.css** : Fichier d'initialisation de React et son fichier CSS associé.
- **Sous-répertoire components** :
  - Contient les composants réutilisables.
  - **Button.tsx**, **Button.css** : Composant de bouton et son style.
  - **Sous-répertoire NavBars** :
    - Contient les composants de barres de navigation.
  - **Sous-répertoire pages** :
    - Contient les composants de page.
    - **Sous-répertoire Private** :
      - Contient les pages qui nécessitent une authentification.
- **Sous-répertoire context** :
  - Contient le contexte de l'utilisateur qui permet de partager l'état de l'utilisateur dans toute l'application.
- **Sous-répertoire services** :
  - Contient les services, comme les requêtes d'authentification.
- **Sous-répertoire utils** :
  - Contient les fichiers d'utilitaires.

Fichiers à la Racine :

- **Dockerfile** : Fichier Docker pour la configuration de l'environnement de l'application.
- **package.json**, **package-lock.json** : Fichiers de configuration de Node.js, contenant les dépendances et scripts.
- **tsconfig.json** : Configuration TypeScript pour le projet.

Cette structure permet une **organisation claire et modulaire** du code. Elle sépare bien les différentes parties de l'application, rendant le code plus lisible et plus facile à maintenir.

Mise en marche AREA

Pour la mise en marche d'AREA, il est crucial de suivre une séquence ordonnée pour garantir que tous les composants et services sont correctement initialisés. Voici un guide étape par étape pour démarrer AREA :

Prérequis :

Assurez-vous d'avoir installé Docker et Docker Compose.

Clonez le dépôt Git du projet AREA sur votre machine locale.

Naviguez vers le répertoire racine du projet AREA dans votre terminal.

Étapes de Mise en Marche :

Construction des Images Docker :

- Exécutez la commande suivante pour construire les images Docker pour le serveur, le client web et le client mobile :

→ **“docker-compose build”**

Démarrage des Services :

- Utilisez la commande suivante pour démarrer tous les services définis dans le fichier docker-compose.yml :

→ **“docker-compose up”**

Vérification des Services :

- Une fois les services démarrés, vous pouvez vérifier leur état en accédant aux URL suivantes dans votre navigateur :
  - Serveur : http://localhost:8080
  - Client Web : http://localhost:8081

ADELE DE PREMONVILLE NOAH LE VEVE LUCAS HOLCZINGER HUGO PATTEIN LOUIS CHAMBON