

1. Contexte et objectif .....	1
2. Données Open Data .....	1
2.1. Source des données .....	1
2.2. Nettoyage et préparation des données .....	2
3. Service Producteur Kafka .....	2
4. Service Consommateur Kafka .....	2
4.1. Fonctionnement .....	2
4.2. Gestion des bases de données .....	3
4.3. API RESTful .....	3
5. Service Kafka Streams .....	3
5.1. Fonctionnalités .....	3
5.2. Technologies utilisées .....	4
6. Application Flutter .....	4
6.1. Fonctionnalités .....	4
6.2. Technologies utilisées .....	4
7. Architecture Globale .....	4
Conclusion .....	5
Annexes .....	6

## Contexte et objectif

Ce projet vise à mettre en place un pipeline complet pour la gestion, le traitement, la transformation et la visualisation de données issues de l'**Open Data**. Les données utilisées proviennent du site [data.gouv.fr](https://data.gouv.fr), et concernent les **stations de taxi**. Le projet repose sur une architecture distribuée utilisant Kafka, des services Spring Boot, un système de base de données, et une application Flutter pour la visualisation sur une carte Google Maps.

## 1. Données Open Data

### 1.1. Source des données

Les données brutes ont été récupérées depuis [data.gouv.fr](https://data.gouv.fr), une plateforme fournissant des jeux de données publiques en France. Ces données contiennent des informations sur les stations de taxi, notamment :

- Identifiant de la station.
- Nom de la station.
- Adresse.
- Code INSEE.
- Nombre d'emplacements.
- Latitude et longitude.
- Statut de la station (active ou inactive).

## 1.2. Nettoyage et préparation des données

Avant leur ingestion dans le pipeline, les données ont été nettoyées et préparées à l'aide d'**OpenRefine** :

- **Correction des erreurs** dans les champs (orthographe, format des adresses, etc.).
- **Suppression des doublons** pour éviter les redondances.
- **Validation des coordonnées GPS** pour garantir une précision sur la carte.
- **Exportation** du fichier final au format **CSV**.

## 2. Service Producteur Kafka

Le **service producteur Kafka** a pour rôle de lire les données nettoyées (au format CSV) et de les envoyer dans un **topic Kafka** nommé taxi-stations.raw. Voici les étapes principales :

1. **Lecture des Données** : Utilisation de la bibliothèque OpenCSV pour parser le fichier CSV.
2. **Transformation en JSON** : Conversion de chaque ligne du CSV en objet JSON.
3. **Envoi vers Kafka** : Les messages JSON sont envoyés au topic taxi-stations.raw via un producteur Kafka.

### ▪ Technologies Utilisées

- Spring Boot.
- Kafka Producer API.
- OpenCSV pour le parsing du fichier.

## 3. Service Consommateur Kafka

Le **service consommateur Kafka** est un composant clé qui consomme les messages JSON envoyés par le producteur et les stocke dans une base de données pour une utilisation ultérieure.

### 3.1. Fonctionnement

1. **Consommation Kafka** : Lecture des messages JSON depuis le topic taxi-stations.raw.

## 2. **Stockage en base de données :**

- **MySQL** est utilisé comme base de données principale.
- En cas d'indisponibilité de MySQL, le service bascule automatiquement vers une base **H2** persistante grâce à une logique dans l'application.

## 3. **Exposition des Données :**

- Une API RESTful expose les données stockées dans la base.

### 3.2. **Gestion des bases de données**

#### 1. **Configuration Multiple :**

- Un fichier application.properties global est utilisé par défaut.
- Des fichiers spécifiques (application-mysql.properties et application-h2.properties) configurent respectivement MySQL et H2.

#### 2. **Détection Automatique :**

- Le service teste la disponibilité de MySQL au démarrage.
- Si MySQL est indisponible, il bascule automatiquement sur H2.

### 3.3. **API RESTful**

L'API RESTful permet :

- **Récupération de toutes les stations.**
- **Recherche par identifiant.**
- **Pagination des résultats** pour optimiser les requêtes.

## 4. **Service Kafka Streams**

Le **service Kafka Streams** est un composant autonome qui traite les données brutes issues du topic taxi-stations.raw. Son rôle est de transformer, regrouper et enrichir les données avant de les produire dans de nouveaux topics Kafka.

### 4.1. **Fonctionnalités**

#### 1. **Regroupement par Statut :**

- Compte le nombre de stations en fonction de leur statut (active ou inactive).
- Produit les résultats dans le topic stations.grouped.by.status.

#### 2. **Filtrage pour Paris :**

- Filtre les stations situées à Paris à partir des champs address et latitude/longitude.
- Produit les résultats filtrés dans le topic paris.stations.

### 3. Création automatique des topics :

- Les topics nécessaires (taxi-stations.raw, stations.grouped.by.status, paris.stations) sont créés dynamiquement au démarrage grâce à l'API Kafka Admin.

## 4.2. Technologies utilisées

- Kafka Streams API.
- Jackson pour le parsing JSON.
- Kafka AdminClient pour la gestion des topics.

## 5. Application Flutter

Une **application mobile Flutter** a été développée pour visualiser les données exposées par le consommateur via son API RESTful. Cette application affiche les stations de taxi sur une carte interactive.

### 5.1. Fonctionnalités

1. **Récupération des Données :**
  - L'application consomme les endpoints de l'API pour récupérer les données des stations.
2. **Affichage sur Google Maps :**
  - Les stations sont affichées sur une carte Google Maps avec des marqueurs interactifs.
3. **Recherche :**
  - L'utilisateur peut rechercher une station spécifique par son nom ou son adresse.
4. **Navigation :**
  - Possibilité d'obtenir des itinéraires vers une station sélectionnée.

### 5.2. Technologies utilisées

- Flutter pour le développement mobile.
- Google Maps API pour l'intégration cartographique.
- http pour consommer les endpoints REST.

## 6. Architecture Globale

Schéma :

- **Producteur Kafka :** Topic d'entrée : taxi-stations.raw.
- **Kafka Streams :** Lit taxi-stations.raw et produit dans :
  - stations.grouped.by.status (regroupement par statut).
  - paris.stations (stations filtrées pour Paris).
- **Consommateur Kafka :**

- Lit stations.grouped.by.status et paris.stations.
- Stocke dans MySQL (ou H2 si indisponible).
- Expose une API RESTful.
- **Application Flutter :**
  - Consomme l'API pour afficher les données sur Google Maps.

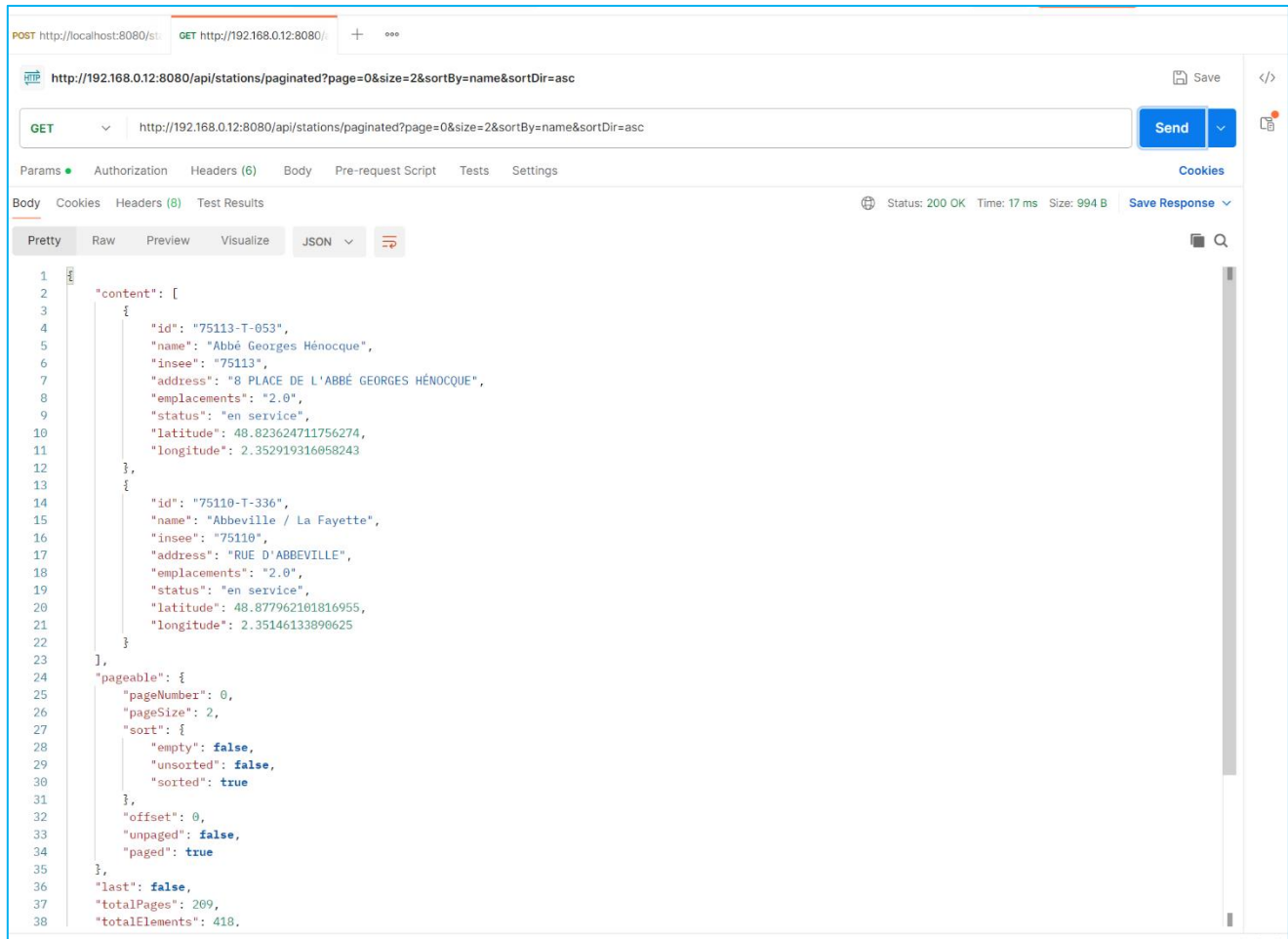
## Conclusion

Ce projet illustre un pipeline complet, allant de l'ingestion et de la transformation des données brutes à leur visualisation interactive. Les principaux atouts du projet incluent :

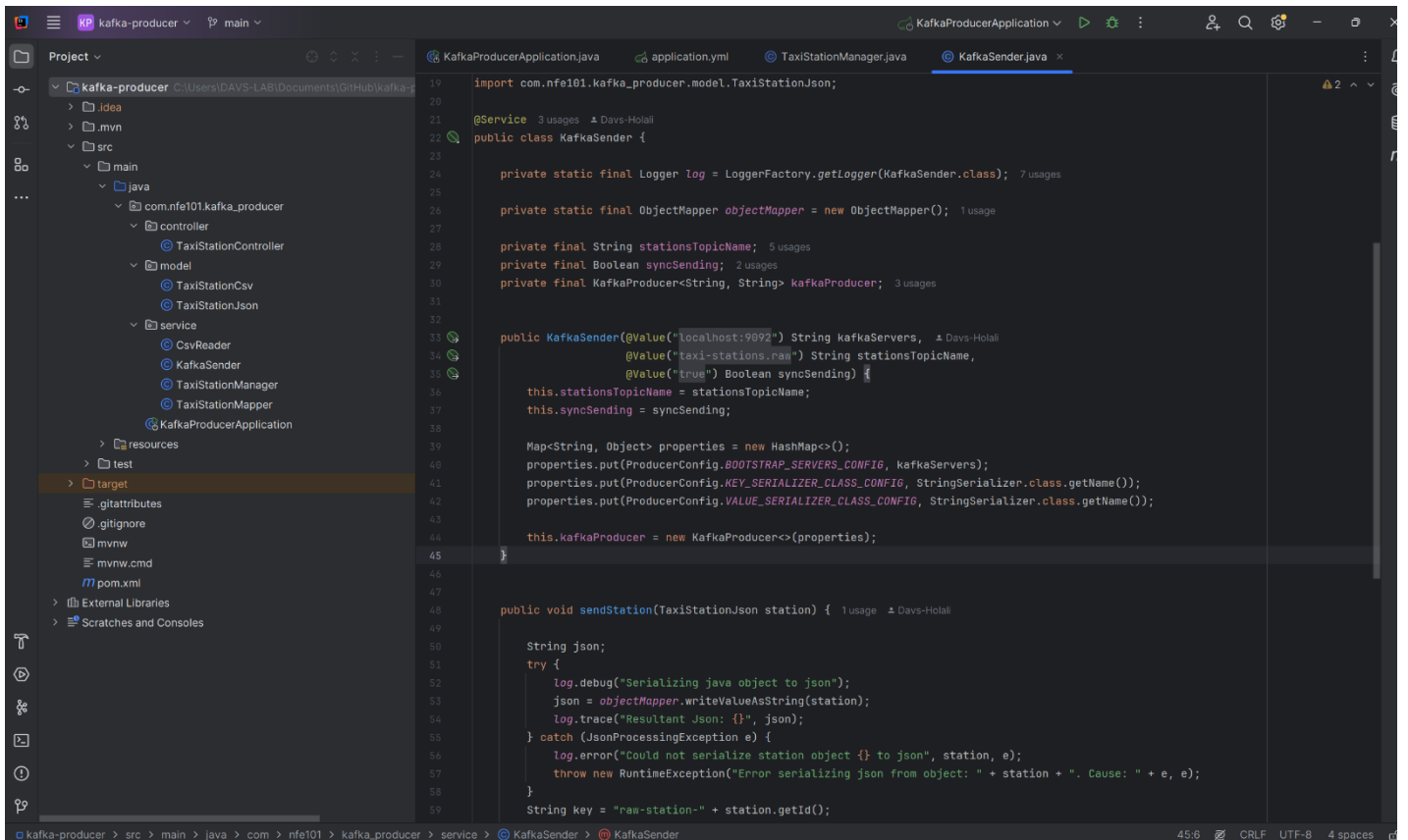
- **Utilisation de données publiques Open Data.**
- **Architecture distribuée** basée sur Kafka.
- **Gestion de la persistance** avec basculement automatique entre MySQL et H2.
- **Flexibilité et scalabilité** grâce à Kafka Streams.
- **Accessibilité** via une application mobile intuitive.

Le projet démontre une intégration réussie de technologies modernes et répond efficacement aux besoins de gestion et de visualisation de données en temps réel.

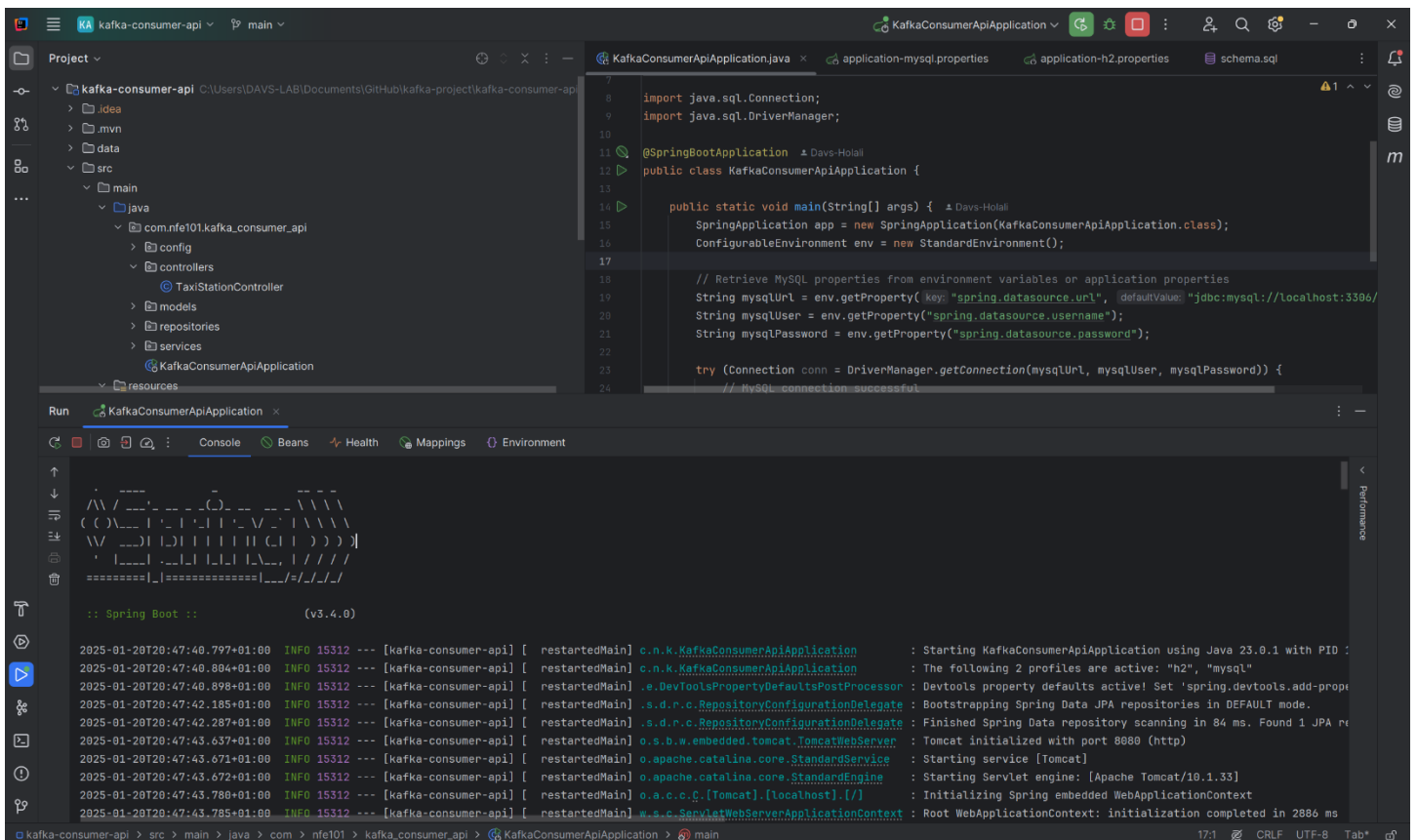
## Annexe 1 : Requête API via Postman



## Annexe 2 : Service Kafka Producer



## Annexe 3 : Service Kafka consumer and API



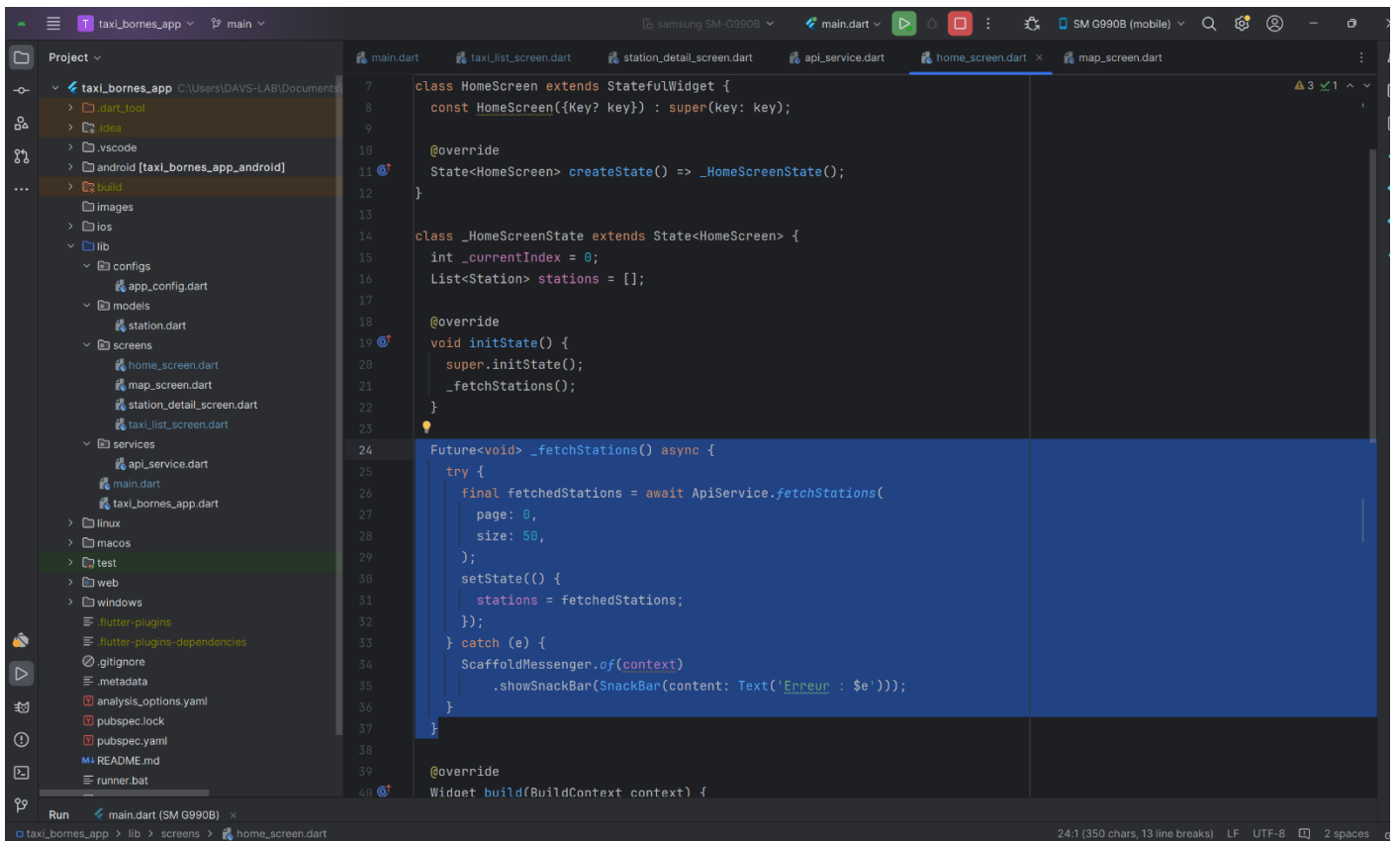
The screenshot shows an IDE with the project 'kafka-consumer-api' open. The file 'KafkaConsumerApiApplication.java' is selected, showing the following code:

```
7
8 import java.sql.Connection;
9 import java.sql.DriverManager;
10
11 @SpringBootApplication
12 public class KafkaConsumerApiApplication {
13
14     public static void main(String[] args) {
15         SpringApplication app = new SpringApplication(KafkaConsumerApiApplication.class);
16         ConfigurableEnvironment env = new StandardEnvironment();
17
18         // Retrieve MySQL properties from environment variables or application properties
19         String mysqlUrl = env.getProperty(key: "spring.datasource.url", defaultValue: "jdbc:mysql://localhost:3306/");
20         String mysqlUser = env.getProperty("spring.datasource.username");
21         String mysqlPassword = env.getProperty("spring.datasource.password");
22
23         try (Connection conn = DriverManager.getConnection(mysqlUrl, mysqlUser, mysqlPassword)) {
24             // MySQL connection successful
25         }
26     }
27 }
```

The console output shows the application starting successfully:

```
2025-01-20T20:47:40.797+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] c.n.k.KafkaConsumerApiApplication : Starting KafkaConsumerApiApplication using Java 23.0.1 with PID :
2025-01-20T20:47:40.804+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] c.n.k.KafkaConsumerApiApplication : The following 2 profiles are active: "h2", "mysql"
2025-01-20T20:47:40.898+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-propri
2025-01-20T20:47:42.185+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-01-20T20:47:42.287+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 84 ms. Found 1 JPA re
2025-01-20T20:47:43.637+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-01-20T20:47:43.671+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-01-20T20:47:43.672+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.33]
2025-01-20T20:47:43.780+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] o.a.c.g.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-01-20T20:47:43.785+01:00 INFO 15312 --- [kafka-consumer-api] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2886 ms
```

## Annexe 4 : Application flutter (extrait de codes)



The screenshot shows an IDE with the project 'taxi\_bornes\_app' open. The file 'home\_screen.dart' is selected, showing the following code:

```
7 class HomeScreen extends StatefulWidget {
8     const HomeScreen({Key? key}) : super(key: key);
9
10     @override
11     State<HomeScreen> createState() => _HomeScreenState();
12 }
13
14 class _HomeScreenState extends State<HomeScreen> {
15     int _currentIndex = 0;
16     List<Station> stations = [];
17
18     @override
19     void initState() {
20         super.initState();
21         _fetchStations();
22     }
23
24     Future<void> _fetchStations() async {
25         try {
26             final fetchedStations = await ApiService.fetchStations(
27                 page: 0,
28                 size: 50,
29             );
30             setState() {
31                 stations = fetchedStations;
32             };
33         } catch (e) {
34             ScaffoldMessenger.of(context)
35                 .showSnackBar(SnackBar(content: Text('Erreur : $e')));
36         }
37     }
38
39     @override
40     Widget build(BuildContext context) {
41         // ...
42     }
43 }
```



Annexe 5 : Interface UI de l'application mobile flutter

