Livret Technique SmartGraph Frontend

Ce livret technique détaille l'architecture et les composants de l'application **SmartGraph Frontend**, une interface utilisateur Angular conçue pour la visualisation et la gestion de données agricoles connectées (parcelles, capteurs, robots, observations) en communication avec une **API REST backend**.

---

1. Présentation Générale

L'application **SmartGraph Frontend** est une interface web moderne permettant de **monitorer** et de **gérer des infrastructures agricoles connectées**. Elle utilise des **observables RxJS** pour interagir avec l'API backend et implémente une architecture **component-based** pour une **modularité optimale**.

Stack Technique

- **Framework** : Angular 18+
- **Langage** : TypeScript 5+
- **Communication** : HttpClient (REST API)
- **Réactivité** : RxJS (Observables)
- **Routing** : Angular Router
- **Styles** : CSS3 (Responsive Design)
- **Tests** : Jasmine + Karma

Prérequis d'Installation

Angular CLI (OBLIGATOIRE)

Angular CLI est l'outil de ligne de commande officiel pour Angular. Il est **indispensable** pour développer, compiler et déployer l'application.

Installation globale d'Angular CLI

npm install -g @angular/cli

Vérification de l'installation

ng version

Node.js & npm

- Node.js v18+

- npm v9+

node --version

npm --version

---

2. Architecture de l'Application

L'application suit une architecture **Component-Service** en couches :

Presentation Layer (Components)

↓

Business Logic Layer (Services)

↓

HTTP Layer (HttpClient)

↓

Backend API (REST)

Structure des Composants

L'application manipule quatre entités principales via des **composants** dédiés :

- **Dashboard** : Tableau de bord avec statistiques globales
- **Parcelle** : Gestion des zones géographiques de culture (CRUD complet + affichage capteurs)
- **Observations** : Historique et gestion des relevés de capteurs (CRUD complet)
- **Robot** : Gestion des unités mobiles d'intervention (CRUD complet + actions)

---

3. Architecture des Données

Interfaces TypeScript

L'application définit des **interfaces strictes** correspondant aux entités backend :

```
// Parcelle - Zone géographique
interface Parcelle {
  id: string;              // Identifiant unique (PARC_*)
  nom: string;             // Nom de la parcelle
```

```
  superficie: number;          // Surface en hectares
  culture: string;             // Type de culture (Blé, Maïs, etc.)
  type_sol: string;            // Nature du sol (sableux, argileux, limoneux)
  latitude: number;            // Coordonnée GPS
  longitude: number;           // Coordonnée GPS
  statut: string;              // État (active, inactive)
}

// Capteur - Dispositif de mesure (label :SSN_Sensor)
interface Capteur {
  id: string;                  // Identifiant unique (CAP_*)
  parcelle_id: string;         // Référence à la parcelle
  type: string;                // Type de capteur (temperature_sensor, soil_moisture_sensor,
multispectral_camera)
  statut: string;              // État (actif, inactif)
}

// Observation - Donnée temporelle
interface Observation {
  id: string;                  // Identifiant unique (OBS_*)
  capteur_id: string;          // Référence au capteur (madeBySensor)
  valeur: number;              // Valeur numérique (numericValue)
  timestamp: string;           // Horodatage ISO 8601
  qualite: string;             // Qualité de la mesure (bonne, moyenne, mauvaise)
}

// Robot - Unité mobile (labels :Robot:AgriculturalDevice:SAREF_Device:Sensor)
interface Robot {
```

```
  id: string;                // Identifiant unique (ROB_*)
  modele: string;            // Modèle du robot
  type: string;              // Type (pulvérisateur, désherbeur, etc.)
  weed_detection_threshold: number;  // Seuil de détection des mauvaises herbes
  statut: string;            // État (actif, en_mission, maintenance)
}

// Dashboard - Statistiques globales
interface DashboardStats {
  parcelles: number;              // Nombre total de parcelles
  capteurs: number;               // Nombre total de capteurs
  robots: number;                 // Nombre total de robots
  alertes: number;                // Nombre d'alertes actives
  missionsEnCours: number;        // Missions en cours
  capteursDysfonctionne: number;  // Capteurs défaillants
}
```

---

4. Points d'Entrée de l'Application (Routes)

Toutes les routes sont gérées par le **Angular Router** et configurées dans app.routes.ts.

Configuration des Routes

```
export const routes: Routes = [
  {
    path: '',
    redirectTo: '/dashboard',
    pathMatch: 'full'
  },
  {
    path: 'dashboard',
```

```
  component: DashboardComponent
 },
 {
  path: 'parcelle',
  component: ParcelleComponent
 },
 {
  path: 'observations',
  component: ObservationsComponent
 },
 {
  path: 'robot',
  component: RobotComponent
 },
];
```

Navigation

Route	Composant	Description
/	Redirection → /dashboard	Route par défaut
/dashboard	DashboardComponent	Tableau de bord global
/parcelle	ParcelleComponent	Gestion des parcelles et capteurs
/observations	ObservationsComponent	Historique des observations
/robot	RobotComponent	Gestion des robots

---

5. Service de Communication (DataService)

Le **DataService** est le point central de communication avec l'API backend. Il encapsule toutes les requêtes HTTP et retourne des **Observables RxJS**.

Configuration Backend

```
// src/app/services/data.service.ts

private apiUrl = 'http://localhost:8080/api/v1';
```

Méthodes du Service

A. Gestion des Parcelles

Méthode	Endpoint Backend	Description
getParcelles()	GET /parcelles	Liste toutes les parcelles
getParcelleById(id)	GET /parcelles/{id}	Détails d'une parcelle
createParcelle(data)	POST /parcelles	Créer une parcelle
updateParcelle(id, data)	PUT /parcelles/{id}	Modifier une parcelle
deleteParcelle(id)	DELETE /parcelles/{id}	Supprimer une parcelle

```
// Exemple d'utilisation

getParcelles(): Observable<Parcelle[]> {

 return this.http.get<Parcelle[]>(`${this.apiUrl}/parcelles`);

}
```

B. Gestion des Capteurs

Méthode	Endpoint Backend	Description
getCapteurs()	GET /capteurs	Liste tous les capteurs
getCapteursByParcelleId(id)	Filtrage local	Capteurs d'une parcelle (filtre côté frontend)

Note : Le backend expose /capteurs global. Le filtrage par parcelle_id est effectué côté frontend via RxJS map().

```
getCapteursByParcelleId(parcelleId: string): Observable<Capteur[]> {

 return this.getCapteurs().pipe(

  map(capteurs => capteurs.filter(c => c.parcelle_id === parcelleId))

 );

}
```

C. Gestion des Observations

Méthode	Endpoint Backend	Description
getObservations()	GET /observations	Liste toutes les observations
createObservation(data)	POST /observations	Enregistrer une observation
updateObservation(id, data)	PUT /observations/{id}	Modifier une observation
deleteObservation(id)	DELETE /observations/{id}	Supprimer une observation

D. Gestion des Robots

Méthode	Endpoint Backend	Description
getRobots()	GET /robots	Liste tous les robots
createRobot(data)	POST /robots	Enregistrer un robot
updateRobot(id, data)	PUT /robots/{id}	Modifier un robot
deleteRobot(id)	DELETE /robots/{id}	Supprimer un robot
changeRobotStatus(id, status)	PUT /robots/{id}	Changer le statut (actif, en_mission, maintenance)

E. Dashboard & Statistiques

Méthode	Endpoint Backend	Description
getDashboardStats()	GET /dashboard/stats	Résumé global du système

---

6. Composants de l'Application

Les **composants** suivants gèrent les différentes fonctionnalités du frontend :

- **DashboardComponent** : Affiche les statistiques globales.
- **ParcelleComponent** : Gère les parcelles agricoles (CRUD) et les capteurs associés.
- **ObservationsComponent** : Affiche l'historique des observations des capteurs.
- **RobotComponent** : Gère les robots agricoles et leurs actions.

---

7. Configuration & Sécurité

Le backend expose un filtre CORS pour permettre les requêtes de http://localhost:4200 (serveur de développement Angular).

Headers autorisés :

- origin
- content-type
- accept
- authorization

Méthodes autorisées :

- GET, POST, PUT, DELETE, OPTIONS, HEAD

L'application utilise des **Observables RxJS** pour toutes les opérations asynchrones, garantissant une gestion réactive des données.

---

8. Installation & Démarrage

Pour démarrer l'application localement, voici les étapes à suivre :

1. Installer Angular CLI (une seule fois)

npm install -g @angular/cli

2. Cloner le repository

git clone https://github.com/cakanyac/Front-end-SG.git

cd smartgraph-frontend

3. Installer les dépendances

npm install

Pour démarrer le serveur de développement :

ng serve --open

Les fichiers optimisés pour la production sont générés via ng build --prod.

---

9. Tests

Exécute les tests unitaires avec :

ng test

Exécute les tests de bout en bout avec :

ng e2e

---

10. Déploiement

L'application prend en charge plusieurs environnements via les fichiers src/environments/.

Pour un déploiement en production :

ng build --configuration production

Les fichiers optimisés sont disponibles dans dist/smartgraph-frontend/.

---

11. Commandes Utiles

ng serve --open # Serveur dev + navigateur

ng build # Build dev

ng build --prod # Build prod

ng test # Tests unitaires

---

12. Dépannage

Problème	**Solution**
ng: command not found	Installer Angular CLI : npm install -g @angular/cli
Port 4200 occupé	Utiliser ng serve --port 4201
Erreurs CORS	Vérifier que le backend autorise localhost:4200

---

13. Ressources

- **Documentation Angular** : https://angular.io/docs
- **Angular CLI** : https://angular.io/cli