

# Rapport CWA :

## Application simulant le fonctionnement d'une société de négoce en céréales

---

### Groupe 1 :

CHOUMILOFF Sacha  
EL OUALI Ayman  
ESCARBAJAL Quentin  
GUILLEMIN Sébastien  
TRIDARD Jérémy  
VERNEY Mathieu  
ZOUMAROU WALIS Faïzath Jedida

---

## **Sommaire**

### **Introduction**

- 1. Pré-requis**
- 2. Description de l'application**
  - 2.1. Arborescence des dossiers**
  - 2.2. Composants**
  - 2.3. Services**
  - 2.4. Partie modèle**
  - 2.5. NodeJS**

### **Conclusion**

### **Bibliographie**

## **Introduction**

Ce document présente le rapport du projet du demi-module Conception Web Avancée (CWA). L'application réalisée avec Angular se propose de simuler le fonctionnement d'une société de négoce en céréales.

Le travail à faire consistait, en premier lieu, à réaliser cette application et également de pouvoir la tester comme elle apparaîtrait aux yeux d'un utilisateur.

Nous nous proposons de décrire l'application ainsi réalisée.

## 1. Pré-requis

Au cours du demi-module CWA, nous avons appris comment installer les dépendances nécessaires à une application web mais également la génération du squelette de base d'une application en Angular. Par la suite, nous avons utilisé certains frameworks et langages que nous jugions nécessaires :

- **Angular CLI**

Lors de la création de projets Angular, à l'aide de la CLI Angular, un ensemble de composants, de tests ... sont créés.

Chaque fois que nous créons des composants, services ... en utilisant le CLI, en plus de créer le fichier de code principal, il crée également un simple fichier de spécifications Jasmine nommé de la même manière que le fichier de code principal mais se terminant par `.spec.ts`.

- **Utilisation de Protractor**

Protractor est un framework de test E2E développé pour Angular et AngularJS.

Son avantage est qu'il permet d'exécuter ces tests E2E directement dans le navigateur physique de notre choix. Il interagit avec le navigateur naturellement comme tout utilisateur le ferait.

- **Jasmine**

Jasmine décrit les tests dans un format lisible par l'homme afin que les personnes non techniques puissent comprendre ce qui est testé.

- **Karma**

Exécuter des tests Jasmine en rafraîchissant un onglet de navigateur de manière répétée dans différents navigateurs chaque fois que nous modifions un code peut devenir fastidieux.

Karma est un outil qui nous permet de créer des tests de Jasmine à partir de la ligne de commande. Les résultats des tests sont affichés sur la ligne de commande ou dans un navigateur.

Karma peut également surveiller les fichiers de développement pour y détecter des modifications et relancer les tests automatiquement.

- **TypeScript**

De nos jours, il est pratiquement indispensable de se passer du langage JavaScript lors du développement d'une application Web. En effet, ce langage permet d'améliorer l'expérience utilisateur et nous apporte un réel gain de temps. TypeScript quant à lui est entre autres une surcouche de JavaScript. C'est un langage open source qui nous permet d'introduire des fonctionnalités optionnelles tels que le typage ou encore la POO (Programmation Orientée Objet).

De plus, tout code ou fonction en JavaScript peut être également utilisé en TypeScript.

## 2. Description de l'application

Notre application utilise plusieurs composants et services.

Un composant est utilisé pour traiter l'affichage et l'interaction avec l'utilisateur pour une certaine partie du site. Par exemple, un composant s'occupe d'afficher le formulaire de connexion et de récupérer les données fournies par l'utilisateur via ce formulaire.

D'un autre côté, les services sont utilisés pour regrouper dans une seule classe des traitements que peuvent utiliser plusieurs composants.

### 2.1. Arborescence des dossiers

La structure de nos dossiers est la suivante :

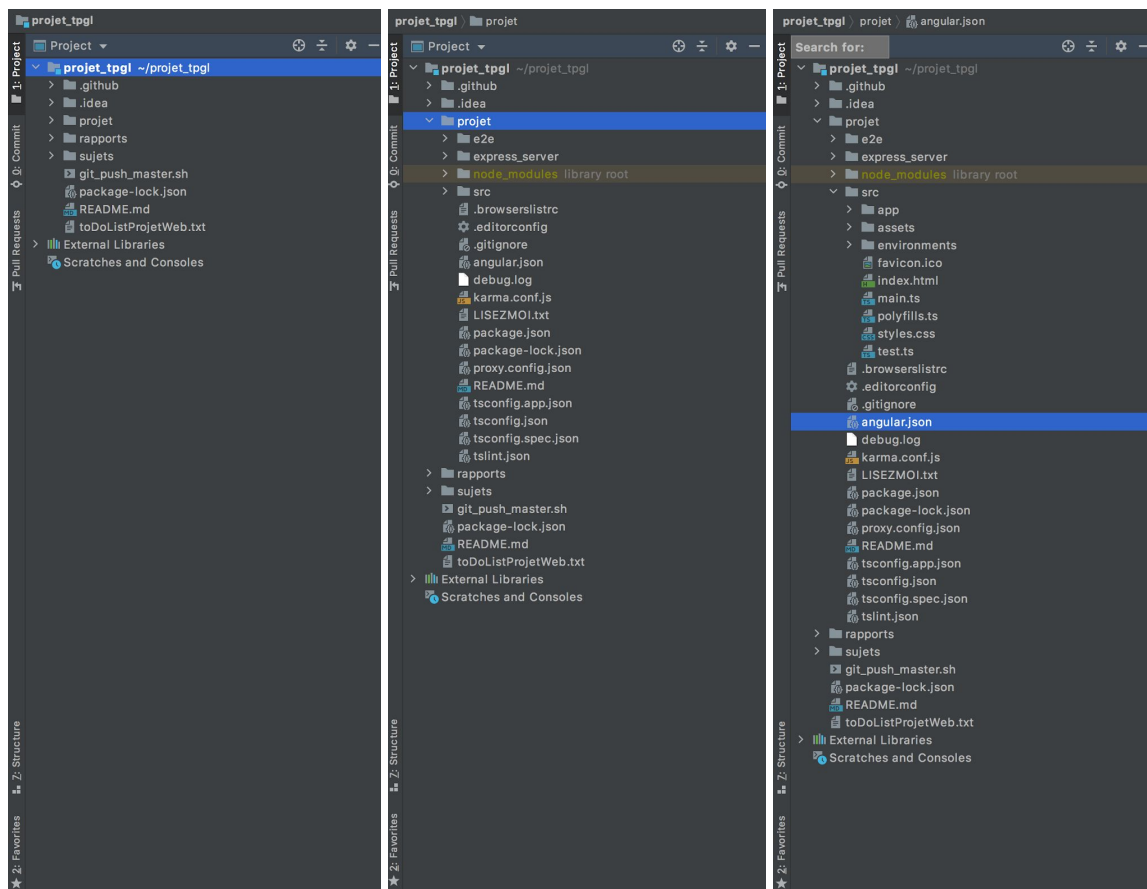


Illustration 1: Arborescence des dossiers

- node\_modules, package.json : les packages et modules des dépendances des framework javascript pour le frontend et la backend.
- tslint.json, tsconfig.json : les fichiers de configuration pour TypeScript.
- e2e : les fichiers nécessaires à l'outil de test Protractor
- src : l'organisation des répertoires du frontend est structurée avec deux principaux répertoires, celui pour le core et celui pour les modules.

## 2.2. Composants

Dans un premier temps, nous allons décrire ce qu'est un composant. Puis nous expliquerons les différents composants que nous avons mis en place afin de mener à bien notre projet.

On peut définir un composant comme une classe qui va gérer l'affichage et les interactions avec l'utilisateur pour une partie de la page. Une page peut être composée de plusieurs composants.

Lors de ce projet, nous en avons créé plusieurs :

- **FournirDonneesComponent**

Le composant **FournirDonneesComponent** a été développé dans le but de permettre à l'administrateur de saisir des informations sur les céréales et sur le matériel.

En effet, un utilisateur possédant les droits administrateurs a la possibilité de remplir un formulaire pour donner des informations sur un lot de céréales ou un autre formulaire pour l'état des matériels. Une fois validées, toutes ces informations sont envoyées au serveur qui s'occupe de les sauvegarder. Il y a un sous-composant pour chaque formulaire : **FournirDonneesCerealesComponent** et **FournirDonneesMaterielComponent**.

Cependant, un utilisateur ne possédant pas les droits administrateurs sera redirigé vers une autre route.

Voir les lots de céréales Voir état du matériel **Fournir données** Déconnexion

## Fournir les données

### Fournir données CEREALES

Numéro du lot

Type de céréales

Poids (en kg)

Qualité

Acheminement (séparer les étapes de l'acheminement par des virgules)

Valider

### Fournir données MATERIEL

Nom du matériel










- **EtatMaterielComponent:**

Ce composant a deux fonctionnalités :

- Visualiser les informations relatives à chaque matériel, à proprement dit : le nom du matériel ainsi que son état, où chaque matériel à des états différents.
- Permettre la modification de l'état de chaque matériel qui sera directement mis à jour dans notre base de données en cliquant sur le bouton présent à côté de chaque matériel.

Voir les lots de céréales Voir état du matériel **Fournir données** Déconnexion

## Voir état du matériel

<b>Silo</b> Etat : Plein  Vider	<b>Cellule1</b> Etat : Pleine  Vider	<b>Cellule2</b> Etat : Vide  Remplir
<b>Cellule3</b> Etat : Vide  Remplir	<b>Cellule4</b> Etat : Vide  Remplir	<b>Cellule5</b> Etat : Vide  Remplir
<b>Cellule6</b> Etat : Vide  Remplir	<b>Cellule7</b> Etat : Vide  Remplir	<b>Cellule8</b> Etat : Vide  Remplir



- **CerealesComponent:**

Le composant **CerealesComponent** a été implémenté pour visualiser la description de chaque lot de céréale.

L’affichage nous permet d’observer les différents lots de céréales ainsi que leurs attributs ( numéro du lot, poids, etc ) qui seront issus de notre base de données.



- **ConnectionComponent**

Lors de l'accès à l'application, le composant afficher est ce composant car il est obligatoire de se connecter pour avoir accès aux différentes fonctionnalités.

Nous avons créé deux rôles pour les utilisateur:

- administrateur (admin) : Dans ce cas, il aura accès à toutes les pages et toutes les fonctionnalités sans aucune restriction. Par défaut le nom d'utilisateur est 'admin' et le mot de passe est 'admin'.
- utilisateur (user) : Dans ce cas, les fonctionnalités sont les mêmes mise à part l'accès à la page fournir données, en voulant accéder à cette page il sera automatiquement redirigé vers le composant permettant de voir les lots de céréales. Par défaut le nom d'utilisateur est 'user' et le mot de passe est 'user'.

- **DeconnexionComponent**

Le composant **DeconnexionComponent** se résume en un simple bouton **Déconnexion** présent dans le header de la page permettant à l'utilisateur de se déconnecter de l'application Web.

### 2.3. Services

Nous utilisons des services afin de ne pas avoir de code redondant à divers endroits de l'application. Par exemple, il est possible que plusieurs composants aient à vérifier si l'utilisateur est connecté. Plutôt que d'écrire la même méthode dans chaque composant, ils utilisent un service qui se charge d'interroger le serveur.

Les services sont situés dans le dossier **app/shared/services** :

- **CerealesService** : permet d'envoyer et de récupérer les données concernant les lots de céréales depuis le serveur.
- **MaterielService** : permet d'envoyer et de récupérer les données concernant les matériels depuis le serveur.
- **ConnectionService** : permet d'interroger le serveur pour savoir si l'utilisateur est connecté (utilisation de sessions), d'envoyer les données de connexions au serveur (pour savoir si le nom d'utilisateur et le mot de passe sont corrects), d'indiquer au serveur que l'utilisateur doit être déconnecté (fermeture de la session), de récupérer l'utilisateur connecté et de le rediriger en fonction de l'état de l'utilisateur (utilisateur connecté et/ou non autorisé à accéder à une page).
- **FormService** : permet de générer un formulaire à partir d'une classe héritant de l'interface **ModelFormInterface** (voir ci-dessous). Une fois le formulaire rempli, le service va récupérer les informations fournies par l'utilisateur, les vérifier et, si tout est correct, *hydrater* (affecter des valeurs aux attributs de manière automatique) l'instance de la classe implémentant l'interface.
- **MaterielFactoryService** : permet de créer une nouvelle instance d'une classe de matériel (Silo, Cellule ...) en fonction d'un type (passé par appel de fonction).

Afin d'utiliser les services dans les composants, nous utilisons le système d'injection de dépendances d'Angular qui permet de passer un service directement à un constructeur.

## 2.4. Partie modèle

Notre application utilise un ensemble de classes métier constituant la **partie modèle** de l'application. Ces classes sont utilisées afin de représenter des objets qui vont être sauvegardés en base de données (fichiers JSON) par le serveur. Ces classes se trouvent dans le dossier **src/app/shared/model**. Leur code est plutôt simple, il s'agit d'un ensemble d'attributs, de constructeurs, d'accesseurs et de méthodes effectuant des traitements propres à chaque classe.

Par exemple, la classe **Matériel** a pour attributs un nom, un état, un type, un ensemble d'états et un ensemble de labels. Cette classe possède des méthodes permettant d'afficher et changer d'état, de récupérer le label à afficher pour un bouton (sur la page de visualisation des différents matériels)...

Plusieurs classes héritent de la classe **Matériel** (une classe par type de matériel) afin de définir quels sont les états et les labels des boutons pour chacune de ces classes. Ces classes sont dans le dossier **src/app/shared/model/materiels**.

Nous avons défini une interface **ModelFormInterface** obligeant l'implémentation de la méthode **getFormFields()**.

Cette méthode doit retourner une liste de chaînes de caractères. L'interface est implémentée par les classes métiers dont les informations peuvent être saisies via un formulaire.

Par exemple, la classe **User** (représentant un utilisateur) peut avoir son nom d'utilisateur et son mot de passe saisis par l'utilisateur. Pour générer le formulaire correspondant, le service **FormService**, qui a un attribut de type **ModelFormInterface**, appelle la méthode **getFormFields()** (d'une instance de la classe **User**) pour connaître l'ensemble des champs requis pour la création du formulaire (les champs sont tous de type *text*) et le générer.

Une fois le formulaire rempli, le service le traite comme vu plus haut.

## 2.5. NodeJS

La partie serveur est implémentée à l'aide de **NodeJS** et du module **Express**. Elle fait principalement trois choses : enregistrer des données dans des fichiers JSON (mise à jour et insertion), envoyer des données stockées dans ces fichiers à la partie cliente et gérer les sessions.

Les modules que nous avons développés se trouvent dans le dossier **express\_serveur**. Ce dossier contient un fichier **server.ts** qui permet de mettre en place un serveur *express* afin de capter les requêtes http et de les traiter suivant la route, **/api/connection** par exemple, et selon le type **GET** ou **POST**.

Lorsqu'un service client envoie au serveur des données, au format JSON, ou que des données sont demandées, le module **ManagerDB** (du dossier **Service**) est utilisé. Ce module sert d'interface entre la partie serveur qui traite les requêtes et la partie Data Access Object (DAO). En effet, il va utiliser des sous-modules en fonction du type de données à lire/écrire dans les fichiers JSON.

Il y a 3 sous-modules qui sont :

- **CerealeBD** : permet de lire ou d'écrire des données concernant les céréales.
- **MaterielDB** : permet de lire ou d'écrire des données concernant les matériels.
- **UserDB** : permet de lire ou d'écrire des données concernant les utilisateurs.

Ces sous-modules possèdent des méthodes d'insertion et de lecture de données dans les fichiers JSON correspondants. Ces fichiers JSON (*cereale.json*, *materiel.json* et *user.json*) se trouvent dans le répertoire **express\_serveur/database**.

Le module **UserConnectionService** permet de gérer les connexions et déconnexions des utilisateurs. Par exemple, il utilise les sessions pour savoir si un utilisateur est connecté ou non.

Pour finir le module **Util** a pour objectif de regrouper des classes utilitaires utilisées par plusieurs autres modules et classes.

## Conclusion

La réalisation de ce projet portant sur la modélisation d'une application en Angular ainsi que l'utilisation d'une solution d'automatisation de test nous a permis d'approfondir les notions vues en cours sur Node.js, Angular, TypeScript. En d'autres termes, nous en avons appris davantage sur certaines tendances technologiques et utiliser des outils pour tester des scénarios de tests de bout en bout d'une application Web en intégrant l'approche **Test-driven development** (TDD) pour la préparation des cas de tests et en utilisant le langage Typescript pour la rédaction des scripts de test.

## Bibliographie

- ❑ Cours Magistraux du demi-module CWA
- ❑ Angular , URL : <https://angular.io/docs> (consulté sur toute la période du projet)
- ❑ Agira Innovation Engineered, URL : <https://www.agiratech.com/blog/front-end-architecture-angular-applications> (consulté le 27/12/2020)