# Compte rendu : *Projet Sushi-App*

## Tables des matières

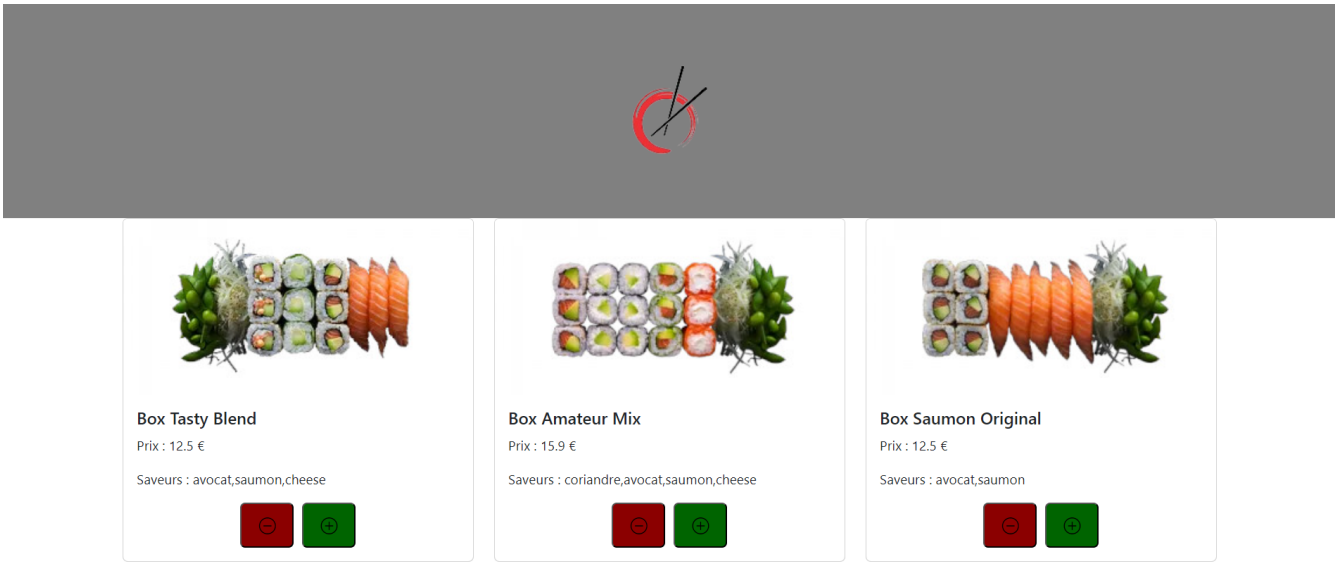

Contexte: . . . 2
Introduction : . . . 2
Préparation du projet . . . 2
- Initialisation de l'API . . . 2
- Initialisation de l'application (CLI) et des dépendances Bootstrap . . . 4

Démarrage du projet . . . 5
- Création des composants . . . 5
- Structure des données . . . 5
- Création des services . . . 6

Difficultés rencontrées . . . 7
Conclusion . . . 7

Projet réalisé par :
**TANDABANY** Adrien

**Du 17/03/2024 au 07/04/2024**




# Contexte:

L'objectif du projet consistait à créer une application commercialisant des boxes de sushi à destinations d'utilisateurs via une borne de commande. Ce projet impliquait l'utilisation de TypesScript et du framework Angular.

# Introduction :

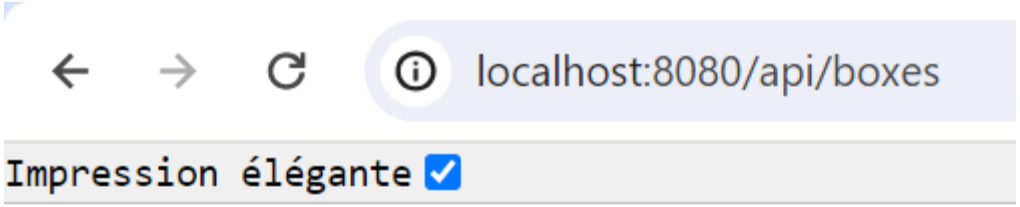
Après avoir défini les objectifs du projet, l'étape suivante consistera à expliquer la création de composants, de services et de méthodes. Par la suite, les difficultés rencontrées seront détaillées, et enfin, une conclusion sera apportée.

| | |
|---|---|
| **IMPORTANT** | Github : https://github.com/atandabany/E5_Sushi-App<br>Application en production : https://atandabany.github.io/E5_Sushi-App/ |

# Préparation du projet

## Initialisation de l'API

Initialement, nous allons lancer l'API Sushi depuis IntelliJ.




Impression élégante ☑


```
[
  {
    "id": 1,
    "pieces": 12,
    "nom": "Tasty Blend",
    "image": "tasty-blend",
    "prix": 12.5,
    "saveurs": [
      "avocat",
      "saumon",
      "cheese"
    ],
    "aliments": [
      {
        "nom": "California Saumon Avocat",
        "quantite": 3
      },
      {
        "nom": "Sushi Saumon",
        "quantite": 3
      },
      {
        "nom": "Spring Avocat Cheese",
        "quantite": 3
      },
      {
        "nom": "California pacific",
        "quantite": 3
      },
      {
        "nom": "Edamame/Salade de chou",
        "quantite": 1
      }
    ]
  },
```

| NOTE | Pour confirmer le bon fonctionnement de l'API, nous pouvons vérifier la présence de la structure des données en accédant à l'URL suivante : http://localhost:8080/api/boxes. |
|---|---|

## Initialisation de l'application (CLI) et des dépendances Bootstrap

Dans le terminal, dans le dossier contenant les projets Angular, nous saisissons les commandes suivantes :

- ng new sushi-app --routing --defaults --standalone=false
- ng generate environments
- npm install --save bootstrap
- npm install --save bootstrap-icons
- ng add @ng-bootstrap/ng-bootstrap

Puis, nous ajoutons les dépendances dans le fichier angular.json

```
"styles": [
            "node_modules/bootstrap/dist/css/bootstrap.min.css",
            "node_modules/bootstrap-icons/font/bootstrap-icons.css",
            "src/styles.css"
          ],
          "scripts": [
            "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
          ]
```

Enfin, nous configurons le fichier environments.development.ts afin de pouvoir indiquer l'API utilisée dans le projet.

```
export const environment = {
    production: false,
    apiBaseUrl: 'http://localhost:8080/api/boxes'
};
```

# Démarrage du projet

## Création des composants

Tout d'abord, nous créons les composants nécessaires à l'évolution de l'application. Nous utilisons la commande suivante : **ng g component/[le nom du composant]**

Puis, dans le fichier **app-routing.modules.ts**, nous définissons la route par défaut de chaque composant créé.

```
  const routes: Routes = [
  {path: '', component: CarteComponent},
  {path: 'rgpd', component:RgpdComponent},
  {path: 'panier', component:PanierComponent}
];
    }
```

| | |
|---|---|
| **NOTE** | il n'est pas nécessaire d'indiquer le footer et le header dans le fichier **app-routing.module.ts**, car ces composants font déjà partie du contenu principal de l'application et sont inclus directement dans le **app.component.html**. |

## Structure des données

A présent nous créons un dossier **models** dans notre projet et nous créons les 3 fichiers suivants :

- Box.ts
- Commande.ts
- LignePanier.ts

Et nous y définissons les classes;

Cela permet de faciliter la communication des données entre les différents composants et services.

**Exemple :**

```
import { LignePanier } from "./LignePanier";

export class Commande {
    commande: Array<LignePanier>

    constructor(commande:Array<LignePanier>) {
        this.commande = commande
    }
}
```

## Création des services

Nous utilisons des services dans Angular pour organiser les fonctionnalités et créer des méthodes réutilisables qui peuvent être injectées dans plusieurs composants.
Pour ce faire nous saisissons la commande suivante dans le terminal : **ng generate service service/[le nom du service]**.

Voici ci-dessous la méthode **addCommande()** (permet de valider les commandes en cliquant sur le bouton "Mon Panier") dans le service **historiquecommande.service.ts**.

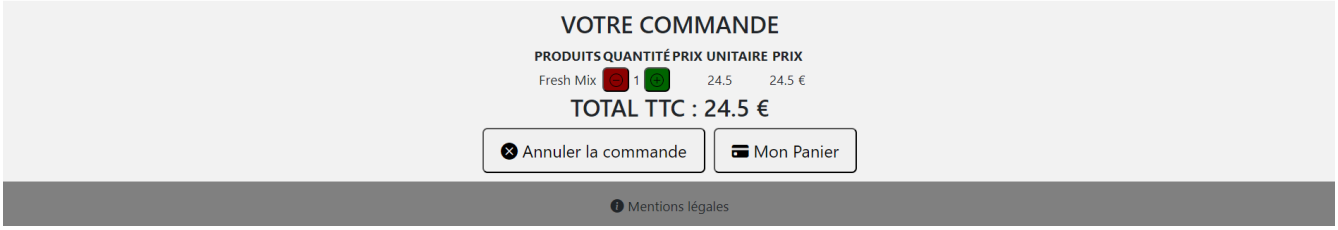
```
 public addCommande(lesLignes: Array<LignePanier>) {
    if (lesLignes.length === 0) {
      return;
    }
```

Enfin, nous récupérons la méthode **ajouterCommande()** dans le composant **commande.component.ts** en l'appelant depuis le service **historiquecommande.service.ts**.

```
 ajouterCommande(){
    console.log(this.getPanier())
     this.historiqueService.addCommande(this.getPanier())
     location.href="/panier"
```

Puis dans le fichier **commande.component.html** nous insérons la méthdoe **ajouterCommande()** dans un bouton afin de déclencher la méthode.

```
 <button (click)="ajouterCommande()" class="bi bi-credit-card-fill rounded" data-bs-
toggle="modal"
        data-bs-target="#staticBackdrop"> Mon Panier
      </button>
```




**NOTE** Nous procédons de la même manière pour l'ensemble des méthodes créées pour l'évolution de l'application.

# Difficultés rencontrées

Le projet s'est avéré difficile à appréhender, principalement en raison de ma difficulté à saisir l'importance de la création de plusieurs services. De plus, j'ai été déconcerté par le nombre important de fichiers, ce qui m'a désorienté quant à l'endroit où je devais coder. Malgré l'intérêt du projet, j'ai eu du mal à m'engager pleinement. La mise en place de l'historique des commandes précédentes s'est avérée très complexe pour moi.

# Conclusion

Le projet a été intéressant car il m'a incité à me mettre à plusieurs reprises à la place de l'utilisateur pour utiliser l'application, ce qui m'a demandé beaucoup de réflexion pour adopter la meilleure logique possible quant à l'utilisation de l'application.