

# kitchenstory.com

e-Commerce Portal

Prototype of the Application

Name : Amit Yadav

GitHub : <https://github.com/amityadav872699/JAVAFSD-Project04-master>

The prototype of the application appears from frontend, and it can also directly start from command prompt. This portal allow user to purchase basic food items online. This prototype is built through various modules that are intertwined with Html & CSS, Components, typescript, models, and Services.

The implementation is done with the help of Node JS and VS Code is done and the source code is written in typescript, Html & CSS.

## **Sprint Planning**

The Implementation is done in five sprints which are mentioned below:

Sprint 1:

- Clarify the specification and requirements.
- Creating sample data for food items containing attributes such as id, food name, food price and food images.
- Creating different packages such as Services, Models and Sample data.
- Finalizing some of the components that will be needed for this prototype.

Sprint 2:

- Creating components named "Homepage".
- Introducing the content in typescript file within home page component for importing food items.
- Updating CSS and Html Content for displaying the appropriate food items with corresponding tags and prices.

Sprint 3:

- Creating food Component.
- Implement functionality in the typescript file that display the specified food selected by the user.
- Implementing styling and elements that will be displayed to the user.
- Implementing another function of the add to cart option in which user has the ability to add multiple food items.

#### Sprint 4:

- Introducing the Cart Component.
- The user can view all food items that user has been selected along with total price.
- Once the user has reviewed his food items, they can visit to the next page "Payment Page".

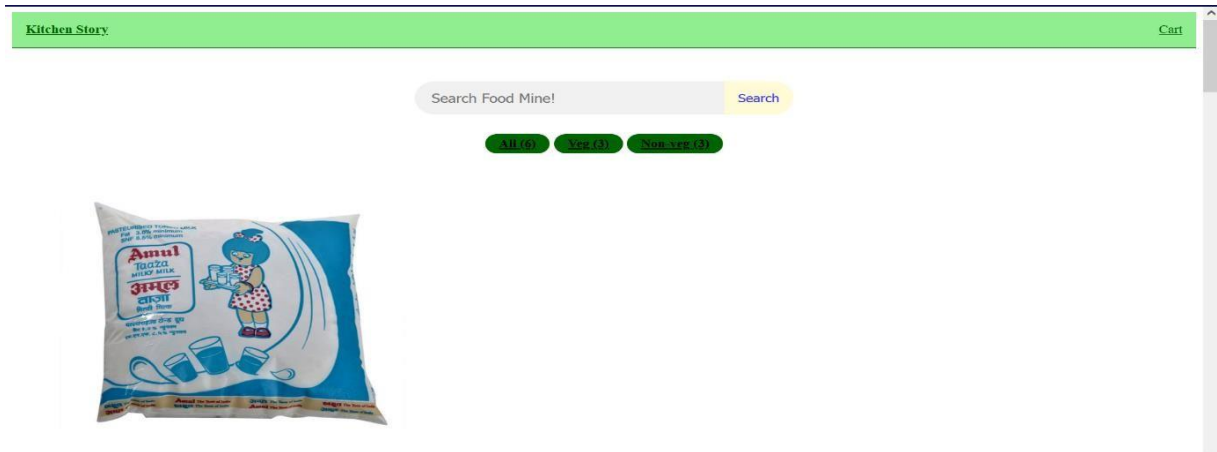
#### Sprint 5:

- Once a user has passed the dummy payment gateway, the system will serve to next page i.e. summary page.
- Here, user can review his confirmed order.
- Documentation

## Documentation of the functionality:

Here are some of the screenshots of testing API through various controllers.

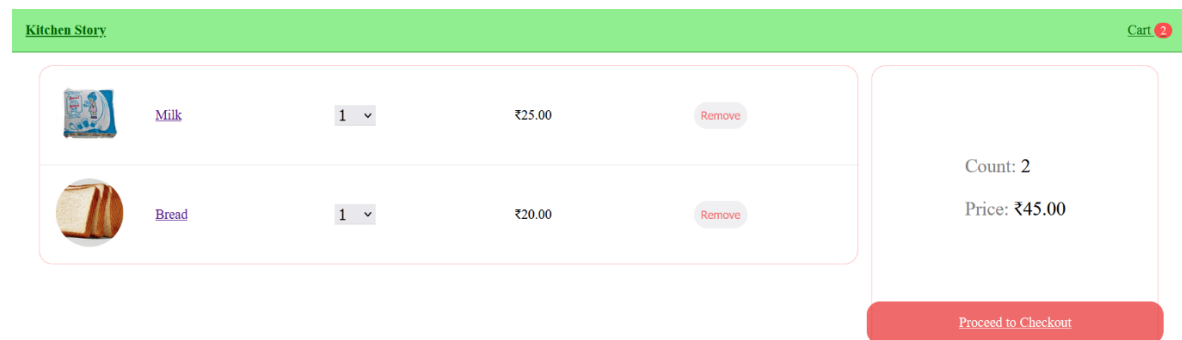
### 1: Home Page



### 2: Food Page



### 3: Cart Page



#### 4: Order Page

[Kitchen Story](#)Cart 2

### Order Register

Name

Ronald Williams



Address

USA

contact

8645544365

Order Items:

	Milk	₹25.00	1	\$25.00
	Bread	₹20.00	1	\$20.00
Total :			₹45.00	

Go To Payment

#### 5: Summary / Payment Page

[Kitchen Story](#)Cart

### Order Summary

**Name:** Ronald Williams

**Address:** USA

**Contact:** 2345632112

Order Items:

	Milk	₹25.00	1	\$25.00
Total :			₹25.00	

Home Page

# Source code

## 1: Sample Data

```
TS datas M X TS home.components search.component.html TS search.components # search.component.css home.compon

src > app > Sample_data > TS datas > ...
1 import { Food } from "../Models/food";
2 import { Tag } from "../Models/tag";
3
4 export const sample_foods: Food[] = [
5   {
6     id: '1',
7     name: 'Milk',
8     price: 25,
9     imageUrl: 'assets/foodItem1.jpg',
10    tags: ['Veg'],
11    company: ''
12  },
13  {
14    id: '2',
15    name: 'Bread',
16    price: 20,
17    imageUrl: 'assets/foodItem2.jpg',
18    tags: ['Veg'],
19    company: ''
20  },
21  {
22    id: '3',
23    name: 'Eggs',
24    price: 10,
25    imageUrl: 'assets/foodItem3.jpg',
26  }
```

## 2: Models

```
order.ts TS carts TS cartItem.ts # tags.component.css TS app-routing.module.ts TS header.components TS tag.ts X U

src > app > Models > TS tags > Tag
1 export class Tag {
2
3   name!: string;
4   count!: number;
5
6 }

TS order.ts X TS carts TS cartItem.ts # tags.component.css TS app-routing.module.ts TS header.components TS tag.ts

src > app > Models > TS orders > Order
1 import { CartItem } from "../cartItem";
2
3 export class Order {
4
5   id!: number;
6   items!: CartItem[];
7   totalPrice!: number;
8   name!: string;
9   address!: string;
10  contact!: number;
11  paymentId!: string;
12  createdAt!: string;
13  status!: string;
14
15 }
```

```
cart.ts    TS cartItem.ts    # tags.component.css    TS app-routing.module.ts    TS header.component.ts    TS tag.ts    TS food.ts    X
```

src > app > Models > TS food.ts > Food > tags

```
1 export class Food {
2
3     id!:string;
4     name!:string;
5     price!:number;
6     tags?: string[];
7     imageUrl!: string;
8     company!: string;
9
10 }
```

```
# payment.component.css    payment.component.html M    TS payment.component.ts    TS order.ts    TS cart.ts    TS cartItem.ts    X
```

src > app > Models > TS cartItem.ts > CartItem

```
1 import { Food } from "../food";
2
3 export class CartItem {
4     constructor(public food: Food) { }
5     quantity: number = 1;
6     price: number = this.food.price;
7 }
```

```
t.component.ts    # payment.component.css    payment.component.html M    TS payment.component.ts    TS order.ts    TS carts    X
```

src > app > Models > TS carts > Cart

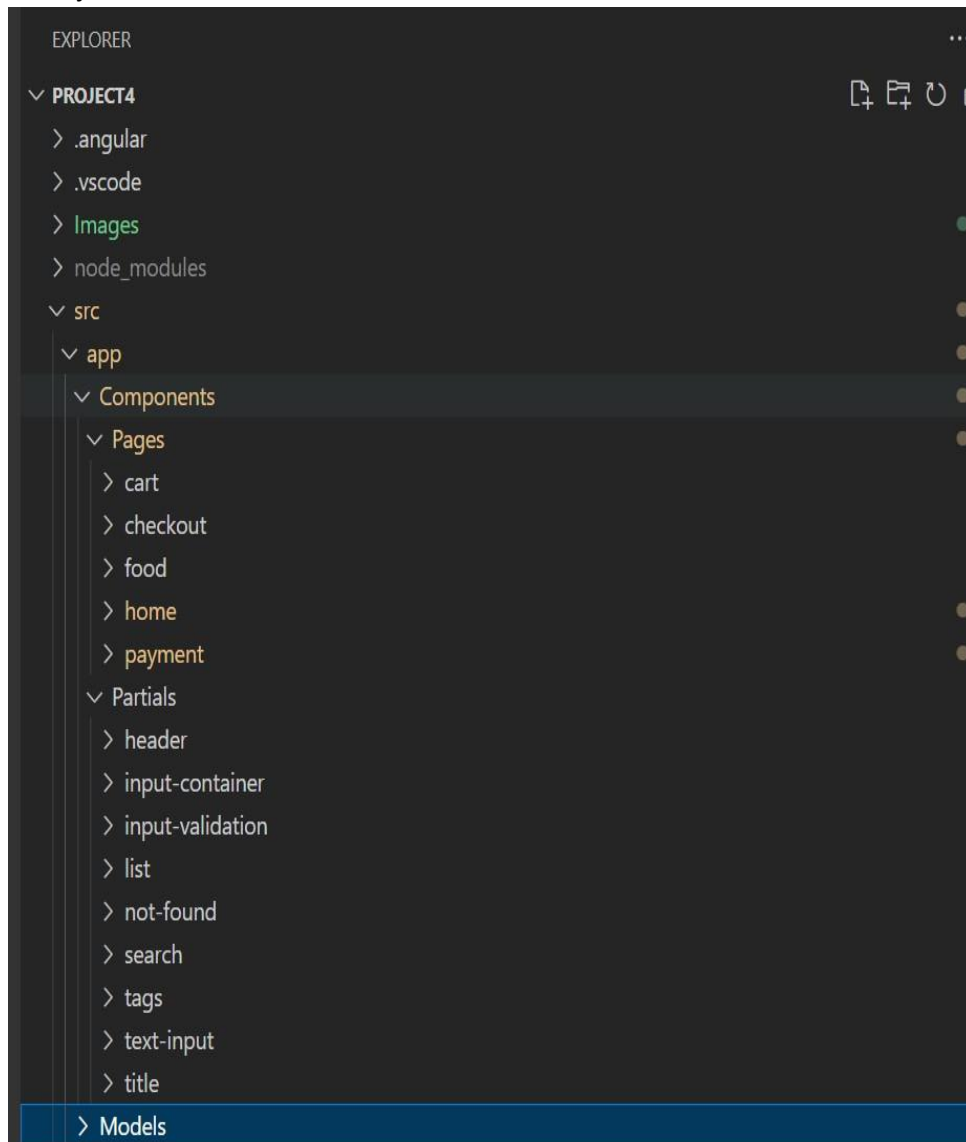
```
1 import { CartItem } from "../cartItem";
2
3 export class Cart {
4     items: CartItem[] = [];
5     totalPrice: number = 0;
6     totalCount: number = 0;
7 }
```

### 3: Services

```
TS food.service.ts M TS tags.component.ts tags.component.html TS app.module.ts TS order.service.ts M X food.component.html
src > app > Services > TS order.service.ts > OrderService
1 import { Injectable } from '@angular/core';
2
3 import { Order } from '../Models/order';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class OrderService {
9
10  constructor() { }
11
12  order!: Order;
13
14  create(order : Order){
15    return this.order = order;
16  }
17
18  getNewOrder(): Order {
19    return this.order;
20  }
21
22 }
23

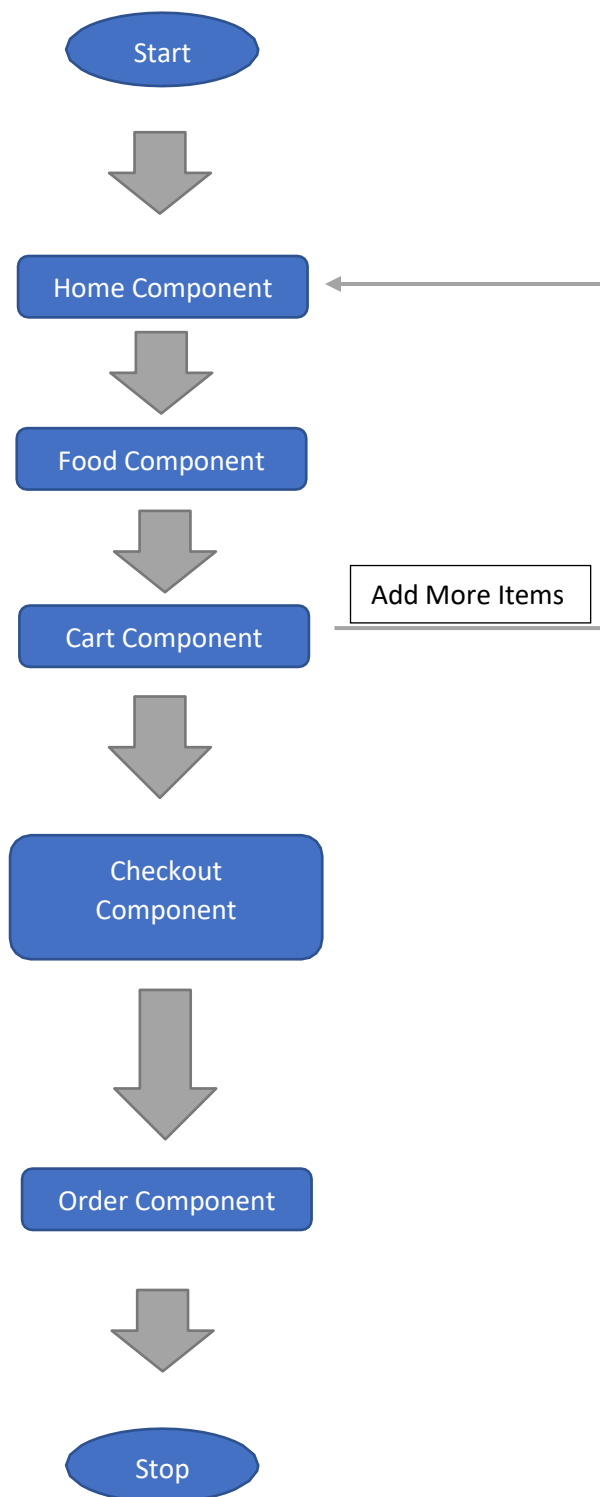
TS food.service.ts M X TS tags.component.ts tags.component.html TS app.module.ts TS order.service.ts food.component.html
src > app > Services > TS food.service.ts > FoodService
1 import { Injectable } from '@angular/core';
2
3 import { Food } from '../Models/food';
4 import { Tag } from '../Models/tag';
5 import { sample_foods, sample_tags } from '../Sample_data/data';
6
7 @Injectable({
8   providedIn: 'root'
9 })
10 export class FoodService {
11
12  constructor() { }
13
14  getAll(): Food[] {
15    return sample_foods;
16  }
17
18  getAllTags(): Tag[] {
19    return sample_tags;
20  }
21
22  getAllFoodsByTag(tag: string) : Food[] {
23    return tag == "All" ? this.getAll() : this.getAll().filter(food => food.tags?.includes(tag));
24  }
25
26  getAllFoodsBySearchTerm(searchTerm: string) {
27
28  }
29
30 }
31
32 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
✓ Compiled successfully.
PC C:\Users\mushad\OneDrive - Cisco\Desktop\Training\Java_FSD\Java_Projects\ProjectA
TS cart.service.ts M X # checkout.component.css checkout.component.html TS title.component.ts title.component.html # title.c
src > app > Services > TS cart.service.ts > ...
1 import { Injectable } from '@angular/core';
2 import { BehaviorSubject, Observable } from 'rxjs';
3
4 import { Cart } from '../Models/cart';
5 import { CartItem } from '../Models/cartItem';
6 import { Food } from '../Models/food';
7
8 @Injectable({
9   providedIn: 'root'
10 })
11 export class CartService {
12
13  private cart: Cart = new Cart();
14  private cartSubject: BehaviorSubject<Cart> = new BehaviorSubject(this.cart);
15
16  constructor() { }
17
18  addToCart(food: Food): void {
19    let cartItem = this.cart.items.find(item => item.food.id === food.id);
20    if (cartItem)
21      return;
22
23    this.cart.items.push(new CartItem(food));
24    this.setCartToLocalStorage();
25  }
26
27 }
```

#### 4: Project Structure





## Flow Diagram



❖ Core Concepts used in this project are mostly typescript, html & CSS.

## **Algorithm**

Step 1> Start.

Step 2> The user can interact with the available food list from two tags.

Step 3> The user can also search for the food item in the search filter.

Step 4> Once a user has selected food item, they can progress it to "Add to Cart".

Step 5> The user can also add multiple food items and can add it to the same cart.

Step 6> Once a user has confirmed their items and total price to be paid. They will be forwarded to a dummy payment gateway.

Step 7> Once a dummy payment gateway has been confirmed, it will move to a next page showing the summary of the user details and food order list that need to be delivered.

Step 8> Stop

## **Conclusion**

1: The prototype is robust and platform independent.

2: User can easily use the prototype and safely exit out of it.

3: As a developer, we can enhance it by introducing several new features such as authguards, interceptors, API, Database, custom validators and can have more user-friendly by adding styling (CSS, Bootstrap).

4: Though this prototype is tightly connected and has been minimized by cookies and cache, it is still stateless (Data cannot be save) project.

5: This prototype can also be implemented with Lazy Loading Module and standalone to save space and time and enable better performance.

6: And lastly, this prototype can be upgraded by creating an admin backend with full access of food items and is able to change or modify.