Food Box

Phase 5 Capstone Project

Project Documentation

Nia Kelley Jester

December 2022

Instructor: Sonam Soni

ABOUT THIS DOCUMENT

This document represents the conclusion of an incredible learning experience. Over the course of the Simplilearn Full Stack Java Development course, I implemented practical knowledge and learned several new languages. The learning does not stop after I turn in my final project. Thank you for being part of my journey!

- This icon is will take you back to the Table of Contents

CLICKABLE TABLE OF CONTENTS

CLICK ON ANY OF THE ICONS TO JUMP TO A PARTICULAR SECTION OF THIS PRESENTATION

PROJECT OVERVIEW

Food Box

APPLICATION LOGOS

PROJECT MANAGEMENT

USER STORIES

HIGH LEVEL DESIGN

GitHub

GIT REPOSITORY

spring BOOT

BACKEND SAMPLE

FRONTEND SAMPLE

PROJECT CONCLUSION

Project Overview

- **Food Box** is a restaurant chain that delivers food items of different cuisines at affordable prices. They decided to hire a Full Stack Developer to develop an online food delivery web application with a rich and user-friendly interface. The Capstone project is a comprehensive project that aims to deliver end to end functionality of a business application.
- **Required Technical Components – Frontend**

IIIIiDE: Visual Studio Code

Programming Language: Angular

Node.js

Bootstrap

- **Required Technical Components – Backend & Database Management**

IIIIiDE: Eclipse

IIIIFramework: Spring Boot or Spring

Programming Language: Java

Web Server: Apache Tomcat

Database: MySQL

Core Concepts Demonstrated (page 1:3)

- **Angular Framework**
  - Demonstrate component-based architecture to build responsive single page application
  - Interpolation & Dependency Injection
  - Implemented Angular Directives
    - Component
    - Structural (`ngIf, ngFor`)
    - Attribute (`ngClass`)
  - Implemented various types of binding
    - Property Binding
    - Class Binding
    - Style Binding
    - Binding to User Input Events
    - Two-Way Binding (forms)
  - Built-in and Custom Pipes
  - Defined services to implement business logic
  - Implement the Angular routing mechanism
  - Responsive Forms for User input and validation

Core Concepts Demonstrated (page 2:3)

- Implement **Node Package Manager (NPM)** via Node.js
- **Bootstrap**
  - Utilized bootstrap design templates for some interface components
- **HTML**
  - Used to create content for webpages
- **CSS**
  - Used to format and style webpages (still working on developing these skills)
- **Connect Angular Frontend to Spring Boot backend using REST API**

Core Concepts Demonstrated (page 3:3)

- Implement **Node Package Manager (NPM)** via Node.js
- **Bootstrap**
  - Utilized bootstrap design templates for some interface components
- **HTML**
  - Used to create content for webpages
- **CSS**
  - Used to format and style webpages (still working on developing these skills)
- **API**
  - Hardcoded JSON data for each product, tag, and User in the Food Box application

User Roles: Customer

Customer that provides the following features:

- ☑A search form on the home page to allow entry of the food items to be purchased by the customer.
- ☑Based on the item details entered, it will show the available food items with their respective prices.
- ☑Once a person selects an item to purchase, they will be redirected to the list of available items.
- ☑On the next page, they are shown the complete breakout of the order and details of the payment to be made in the payment gateway.
  - ✓ Checkout Button
  - ✓ Products Number Shown
  - ✓ Cart items shown
  - ✓ When payment is made, they are shown a confirmation page with details of the order
- ☑Apply filters and sort results based on different cuisines to get the best deals
- ☑Get a basic order summary details page once the payment is complete
- ☑Fully functional login feature
- ☑Logout Feature
- ☑Persist user login status after page refresh
- ☑Guest User will be able to use the application

Features that were not implemented

(ran out of time)

- Implement ability to update user details and reset password (for Customer user)
- Display most viewed items
- Perform a seamless payment process using PayPal, Stripe, or other payment processing service
- Use email validation for User Registration
- Cart data stored in the database
- Order history stored in the database

User Roles: Admin

Admin with the following features:

- ☑A add a new product to the master list of all the food items available for purchase
- ☑A functionality remove a product form the master list of items
- ☑Add a new product to the application to build a rich product line
- ☑Remove a product to the application
- ☑Add a new tag/category to the application
- ☑Add a new cuisine to the application to build a rich product line
- ☑Remove a cuisine from the application (backend only)
- ☑Remove a tag/category from the application (backend only)
- ☑Edit food item details like name, price, cuisine, description, and offers to keep it aligned to the current prices
- ☑Admin change password page where the admin can change the password after login
- ☑Login/Logout Feature

Features that were not implemented

(ran out of time)

- Enable or disable the food items
- Create frontend forms to allow Admin users to remove a tag/category or cuisine from the backend database

Additional Features

FRONTEND

- ☑Tags for New Product
  - ✓ Fix bug – use ngFor to bind to an Array
- ☑Add “Reset Password” functionality for the Admin user
- ☑Use Observables for User monitoring
- ☑Add HTTP functionality
- ☑Add dynamic appearance of the nav bar, based on user state
- ☑Continue to refine the CSS code to improve the overall look & feel of the application
- ☑Provide API functionality to connect to the backend

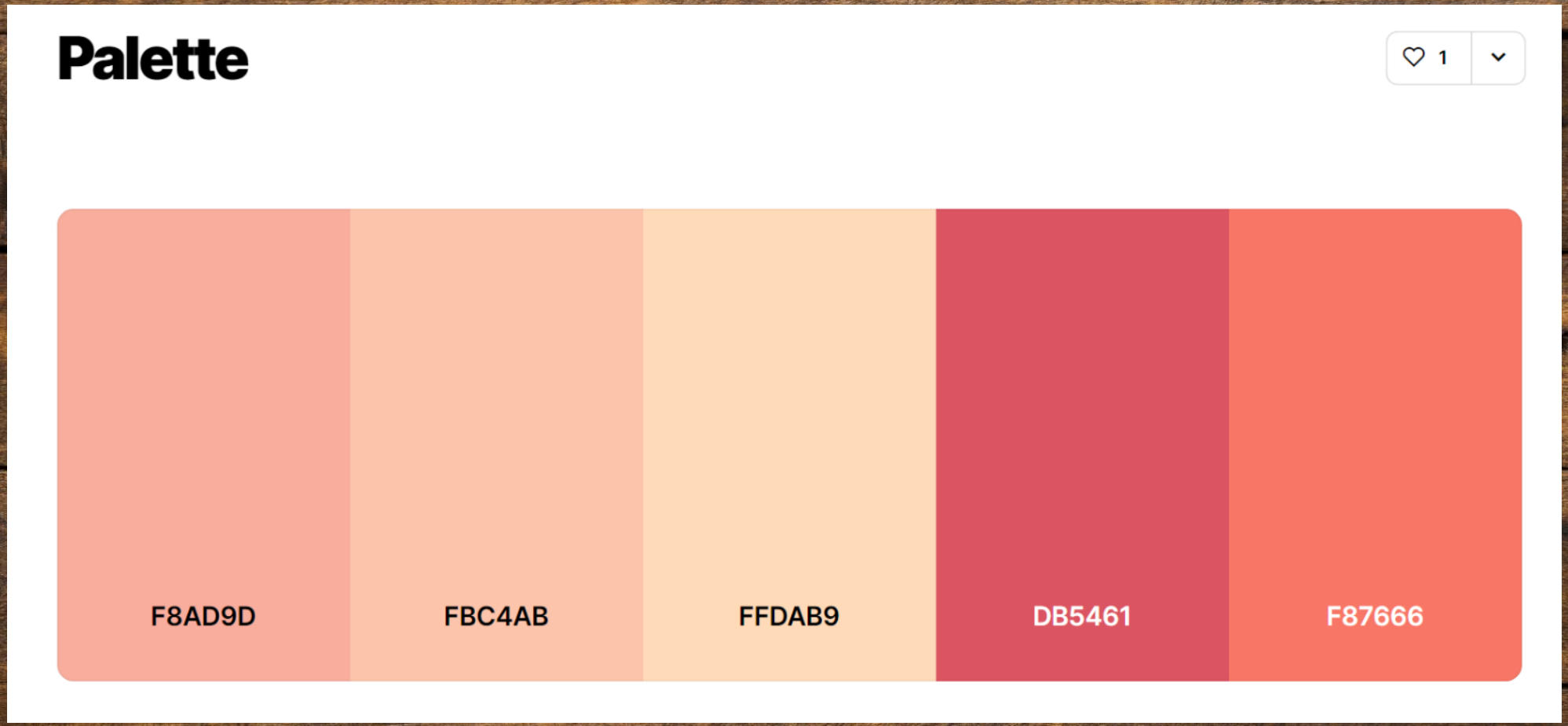
BACKEND

- ☑Create the Spring Boot application
- ☑Establish the REST API
- ☑Enable Swagger Documentation
- ☑Create the MySQL database
  - ✓ Product information
  - ✓ Tags/Categories
  - ✓ Cuisine Information
  - ✓ User data
  - ✓ Initialization script
  - ✓ Populated tables via script and POSTMAN

Food Box

Food Box App Logos

Application Color Palette



https://coolors.co/palette/f8ad9d-fbc4ab-ffdab9-db5461-f87666

Project Management (page 1:2)

Project Overview	
Client	Simplilearn
Consultant	Nia Kelley Jester Full Stack Java Developer
Application Name	Food Box
Application Phase	Prototype
Phase 5 Project Deliverable	E-commerce portal that lets people order food items of different cuisines at affordable prices on the Food Box website.

Requirements Development			
Planned Start Date	Thursday, October 27, 2022	Actual Start Date	Thursday, October 27, 2022
Planned Finish Date	Sunday, October 30, 2022	Actual Finish Date	Sunday, October 30, 2022

Project Management (page 2:2)

Code Development			
Holiday Week/Planned Outage	Monday, November 21, 2022 – Sunday, November 27, 2022		
Planned Project Code Duration	3 Weeks (15 working days)	Actual Project Code Duration	6 Weeks (30 working days)
Planned Start Date	Monday, October 31, 2022	Actual Start Date	Monday, October 31, 2022
Planned Finish Date	Friday, November 18, 2022	Actual Finish Date	Sunday, December 18, 2022
Planned Number of Sprints	3	Actual Number of Sprints	3
Planned Sprint Duration	1 Week (5 working days)	Actual Sprint Duration	2 Weeks (10 working days)
Total Number of Product Backlog Items	63		

Project Documentation			
Planned Start Date	Monday, November 28, 2022	Actual Start Date	Wednesday, October 26, 2022
Planned Finish Date	Friday, December 2, 2022	Actual Finish Date	Sunday, December 18, 2022

Project Summary		
	Planned	**Actual**
Total Project Weeks	5 Weeks	7 Weeks
Total Project Days	25 Days	34 Days
Total Project Hours	125 Hours	160 Hours

Product Backlog by Sprint

Total Product Backlog	63 Items
Sprint 1	25
Sprint 2	9
Sprint 3	23
Moved to next planning cycle	6

Product Backlog by Scenario

Product Backlog ID		Role	Scenario	Backlog Item (User Story)	Story Points	Assigned Sprint
1			Landing Page			header
	1.1	Customer/Admin/Guest	Landing Page	The Food Box landing page feature a welcome message and a registration form.	Small	1
2			New User Registration			header
	2.1	Customer	New User Registration	The Customer can register as a new user using a registration form	Medium	2
	2.2	Customer	New User Registration	The new user will be sent a verification email before adding the user to the database	Large	Next Release
	2.3	Customer	New User Registration	The confirmation email will have time limit expiration	Large	Next Release
	2.4	Customer	New User Registration	The new user will be verified (with verification links) before adding the user to the database	Large	Next Release
	2.5	Customer	New User Registration	The new user will be stored as a user in the database	Medium	2
3			User Management			header
	3.1	Customer/Admin	Users	All Users will be stored in the backend database.	Medium	2
	3.2	Admin	Admin Login	Admin can login to the application	Medium	3
	3.3	Admin	Admin Logout	Admin can logout of the application	Medium	3
	3.4	Customer	Customer Login	The customer can login to the Food Box application using a 'login' link	Medium	3
	3.5	Customer	Customer Login	Once a Customer is logged in, the "Login" navbar should no longer be visible	Medium	3
	3.6	Customer	Customer Logout	User can logout of the application	Small	3
	3.7	Customer	Customer Login	Once a Customer is logged in, a "Logout" link should be visible on the navbar	Medium	3
	3.8	Guest	Proceed without logging in	The customer can proceed as a guest without logging in to the application	Small	3
	3.9	Admin	Update User Details	Admin User can update the details for any user: first name, last name, user name, email adress, phone number, and role type.	Medium	3
	3.10	Admin	Change Password	Admin User can change reset passwords for any user.	Medium	3
	3.11	Admin	Delete User	Admin User can delete a User from the database.	Medium	3

Product Backlog by Scenario

Product Backlog ID		Role	Scenario	Backlog Item (User Story)	Story Points	Assigned Sprint
4			Navigation Bar			
	4.1	Customer/Admin/Guest	Navigation Bar	The Food Box Logo will serve as the home page link on the	Small	1
	4.2	Customer/Admin/Guest	Navigation Bar	The Food Box navigation bar will have a Registration link. The link will be visible until an authenticated User logs into the application.	Small	3
	4.3	Customer/Admin/Guest	Navigation Bar	The Food Box navigation bar will have a Login link. The link will be visible until an authenticated User logs into the application	Small	3
	4.4	Customer	Navigation Bar	The navigation bar will change if a Customer User logs into the the application. The menu options will be the User's First Name, Cart link, and a Logout link.	Small	3
	4.5	Admin	Navigation Bar	The navigation bar will change if a Admin User logs into the the application. The menu options will be the User's First Name, Cart link, Admin specific menu, and a Logout link.	Small	3
	4.6	Admin	Admin Menu	The Admin menu lists services that an Admin User can perform: Add a New Product, Remove Product, Edit Products, Edit Users, Add a New Cuisine, and Delete a Cuisine. (implementation logic is separate).	Small	1
	4.7	Admin	Admin Menu	The Admin menu is not visible unless an Admin User is logged in	Medium	3
	4.8	Customer/Admin/Guest	Navigation Bar	The "Cart" link in the navigation bar should dynamically reflect the number of items in the cart.	Small	1
5			Home Page			
	5.1	Customer/Admin/Guest	Home Page	The Home Page will be have 3 sections: Header (which contains the navigation bar), a main body, and a Footer.	Small	1
	5.2	Customer/Admin/Guest	Home Page	The main body will have a search bar, a section for categories, cuisines, and available products.	Small	1
	5.3	Customer/Admin/Guest	Home Page	The available products/search results will show all of the details for a product.	Small	1
	5.4	Customer/Admin/Guest	Home Page	The available products/search results will have links to take the User to the individual product page.	Small	1

Product Backlog by Scenario

Product Backlog ID		Role	Scenario	Backlog Item (User Story)	Story Points	Assigned Sprint
6			Product Management			
	6.1	Customer/Admin/Guest	Product	All products will be stored in backend database. The frontend application will fetch/update/delete the appropriate data as needed.	Medium	2
	6.2	Admin	Add new product	A add a new product to the master list of all the food items available for purchase	Medium	3
	6.3	Admin	Update Product Details	An Admin User can update product details in the application	Medium	3
	6.4	Admin	Delete product	An Admin User can remove a product from the master list of items	Medium	3
7			Search Products			
	7.1	Customer/Admin/Guest	Search Bar	A search form on the home page to allow entry of the food items to be purchased by the customer. The database will search the available products by product name.	small	1
	7.2	Customer/Admin/Guest	Search Bar	The search form will uncover any product matches. The customer has the option to go to the individual product page.	Small	1
	7.3	Customer/Admin/Guest	Search Bar	If the search does not find any matches, the Customer will be notified by a appropriate message.	small	1
8			Product Categories/Tags			
	8.1	Customer/Admin/Guest	Tags	The tags/categories will be stored in the backend database.	Medium	3
	8.2	Customer/Admin/Guest	Tags Feature from the Home Page	The home page will have all of the tags/cagetory and show the total number of products that are in that category	Medium	2
	8.3	Customer/Admin/Guest	Tags Feature from the Individual Product Page	Each product page will have a tags/category button. When pressed, it will filter the product list for the products that belong in the category.	Medium	2
	8.4	Admin	Tags	If an Admin adds a new product to the database, a new tag will be created in the database if one doesn't already exist.	Medium	3

Product Backlog by Scenario

Product Backlog ID		Role	Scenario	Backlog Item (User Story)	Story Points	Assigned Sprint
9			Product Cuisines			
	9.1	Customer/Admin/Guest	Cuisines	The cuisines will be stored in the backend database.	Medium	3
	9.2	Customer/Admin/Guest	Cuisines	The available cuisines will be presented as a list on the landing page.	Small	3
	9.3	Customer/Admin/Guest	Cuisines Feature from the Home Page	The home page will have all of the cuisines avaiable and show the total number of products that are in that category	Medium	3
	9.4	Customer/Admin/Guest	Cuisines Feature from the Individual Product Page	Each product page will show the cuisines listed for that product. It will filter the product list for the products that belong in the cuisine.	Medium	2
	9.5	Admin	Cuisines	An Admin User can add a new cuisine to the backend database.	Medium	2
	9.6	Admin	Cuisines	An Admin User can delete a cuisine from the backend database.	Medium	2
10			Individual Product Page			
	10.1	Customer/Admin/Guest	Individual Product Page	Each individual product page will show relevant product information including the product name, calories, star rating, number of reviews, tags/categories, cuisines, price, detailed product description, and product image.	small	1
	10.2	Customer/Admin/Guest	Individual Product Page	Each individual product page will have a button to add the product to the shopping cart	Small	1
	10.3	Customer/Admin/Guest	Individual Product Page	Each individual product page will have a button to continue shopping by taking the Customer back to the home page	Small	1

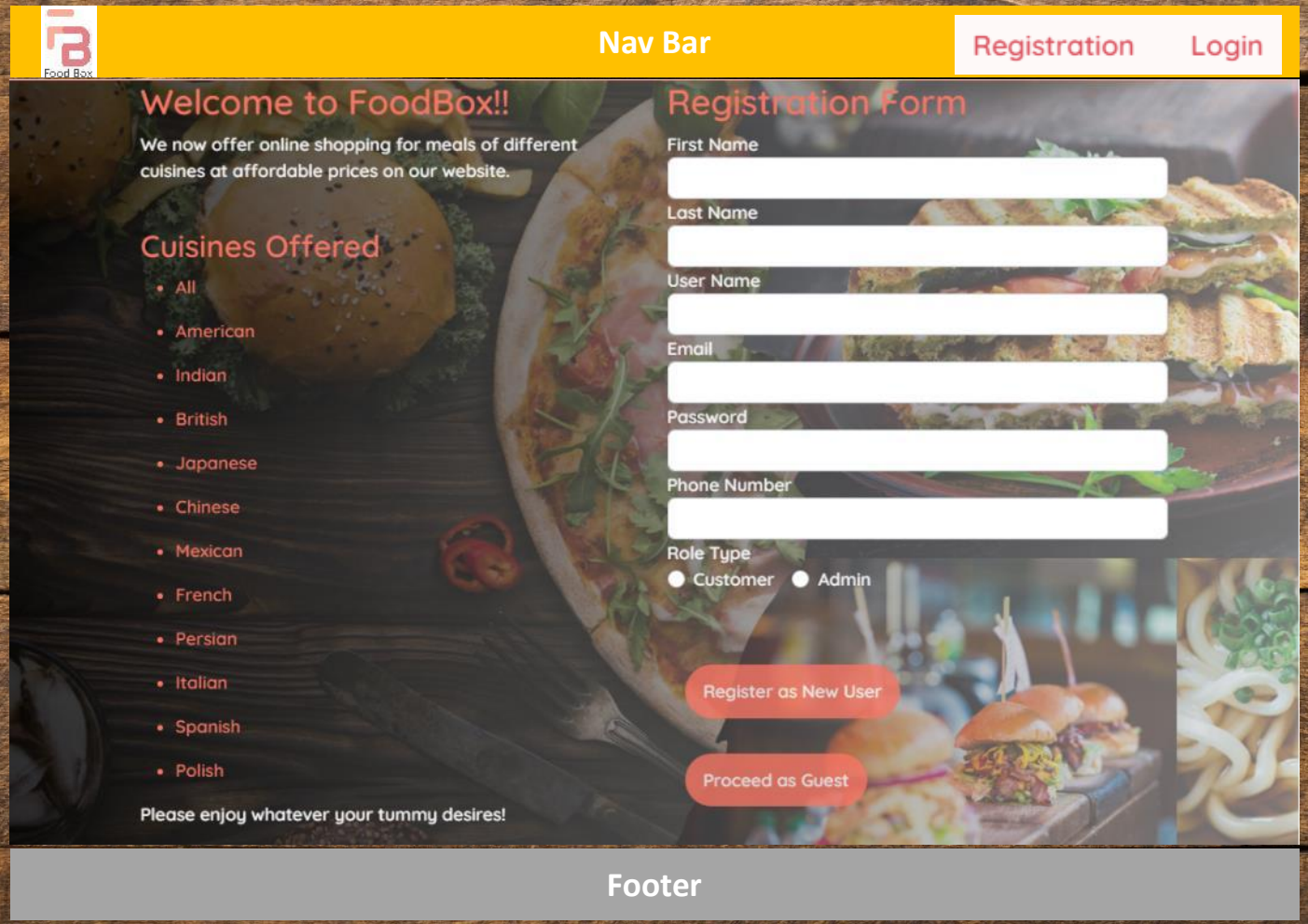
Product Backlog by Scenario

Product Backlog ID		Role	Scenario	Backlog Item (User Story)	Story Points	Assigned Sprint
11			Shopping Cart			
	11.1	Customer/Admin/Guest	Shopping Cart page	The shopping cart page will have the ability to change the quantity of each individual product	Small	1
	11.2	Customer/Admin/Guest	Shopping Cart page	The shopping cart page will have the reflect the total price of cart	Small	1
	11.3	Customer/Admin/Guest	Shopping Cart page	The shopping cart page will have the reflect the total number of items in the cart	Medium	1
	11.4	Customer/Admin/Guest	Shopping Cart page	The shopping cart page will have the ability to remove an item from the cart	Medium	1
	11.5	Customer/Admin/Guest	Shopping Cart page	Each shopping cart page will have a button to proceed to the checkout page	Small	1
	11.6	Customer/Admin/Guest	Shopping Cart page	Each shopping cart page will have a button to continue shopping by taking the Customer back to the home page	Small	1
12			Order Details			
	12.1	Customer/Admin/Guest	Order Details	Customer is shown a order details page, that includes the item count, order total, ordered items, and a button to proceed to payment.	Small	1
	12.2	Customer/Admin/Guest	Order Details	Customer can not update item quantity from the order details page.	Small	1
	12.3	Customer/Admin/Guest	Order Details	The products shown in the order details page are not hyperlinked to the individual product pages.	Small	1

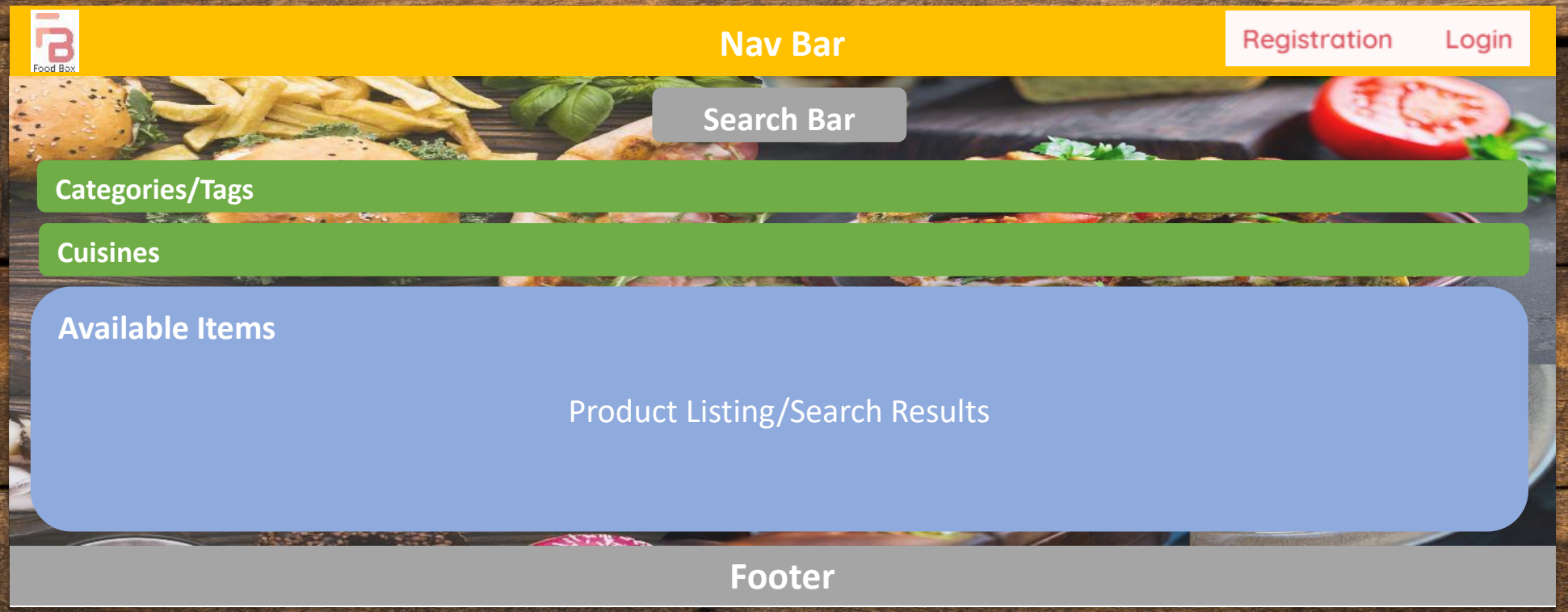
Product Backlog by Scenario

Product Backlog ID		Role	Scenario	Backlog Item (User Story)	Story Points	Assigned Sprint
13			Payment			
	13.1	Customer/Admin/Guest	Payment	Once the order is paid for, the Customer will receive a message in the application stating that the order is confirmed and payment has been processed.	Small	1
	13.2	Customer/Admin/Guest	Payment	Once payment has been processed, the Cart is cleared	Small	1
	13.3	Customer/Admin/Guest	Payment	Enhance the Food Box application by implementing a payment processing feature.	Epic	Next Release
14			Security			
	14.1			The Food Box Application will have secure endpoints implemented by Spring Security	Large	Next Release
	14.2			The Food Box Application frontend will have protected routes to prevent unauthorized access.	Medium	Next Release

High Level App Design: Landing Page



High Level App Design: Home Page



High Level Program

Single Page Application

Frontend

Register Page

Login Page

Home Page

Checkout

Payment

User

- Get All Users
- Create Single User
- Update Single User
- Delete Single User

Search

- Search by term (product name only)
- Filter by cuisine
- Filter by category

Product

- Get All Products
- Add Product
- Edit Product
- Remove Product
- Get Product by Cuisine
- Get Product by ID

Custom Pipes

- Phone Number Format
- Truncate Product Description

Cart

- Add Product to Cart
- Remove Product from Cart
- Show all products in Cart

Backend

spring BOOT

MySQL

FRONTEND CLIENT

ANGULAR COMPONENTS
Will have access to client services

ANGULAR SERVICES
(HTTP Requests)

- Client Services will have access to the backend
- Will essentially mirror the endpoints in the backend controller

UI
What the User is going to see on their screen

BACKEND SERVER

REST CONTROLLER

REST SERVICES

REST REPOSITORIES

REST MODELS

MySQL

API DESIGN

Food Box

- Built the server, controller, access to the database, and the relevant endpoints
- The backend is library agnostic, meaning it can communicate with any tool that has the capability of sending HTTP requests (i.e. POSTMAN, Angular, etc.)
- Reminder: the REST Controller is the part of the backend (server) that will be exposed to the outside world.

Git Repository

All code for this project can be found at the following link:
https://github.com/niakelleyjester/simplilearn-projects

Spring Boot Backend Code Sample

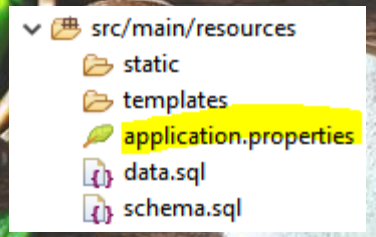
spring BOOT

MySQL


DATABASE
CONFIGURATION

Database Connectivity

Database Connectivity: Create the application.properties file to setup connectivity to the database

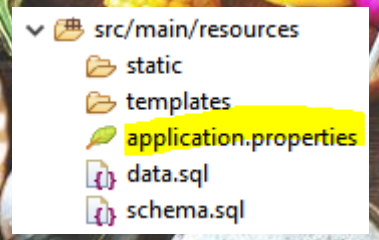



```
#MySQL Configuration
spring.datasource.username=root
spring.datasource.password=DesignerNia27!
spring.datasource.url=jdbc:mysql://localhost:3306/foodboxdb
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.sql.init.platform = mysql
spring.sql.init.schema-locations = classpath:schema.sql
spring.sql.init.data-locations = classpath:data.sql
#instructs Spring Boot to initialize the database schema
spring.sql.init.mode = always
server.port=8080
#For Swagger Documentation
spring.mvc.pathmatch.matching-strategy=ANT_PATH_MATCHER

```

Database Creation: Create the database (sql -> foodboxdb)

```
mysql> create database foodboxdb;
```

Database Connectivity



application.properties

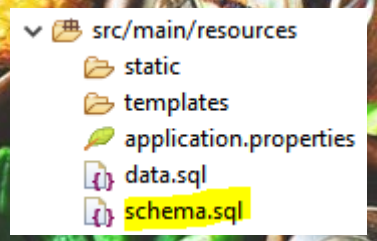
```
#MySQL Configuration
spring.datasource.username=root
spring.datasource.password=DesignerNia27!
spring.datasource.url=jdbc:mysql://localhost:3306/foodboxdb

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

#To disable the Hibernate automatic creation and use script-based intialzation, set to none
#spring.jpa.hibernate.ddl-auto=auto
#spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.hbm2ddl.auto: update
#spring.jpa.properties.hibernate.hbm2ddl.auto: none
spring.jpa.show-sql=true

spring.sql.init.platform = mysql
#Database specific schema and data files
spring.sql.init.schema-locations = classpath:schema.sql
#spring.sql.init.data-locations = classpath:data.sql
#spring.sql.init.schema-locations = classpath:schema-mysql.sql
#spring.sql.init.data-locations = classpath:data-mysql.sql
#instructs Spring Boot to initialize the database schema from a script file
#spring.sql.init.mode = always
server.port=8080

#For Swagger Documentation
spring.mvc.pathmatch.matching-strategy=ANT_PATH_MATCHER

```

Note: Couldn't get this to work completely. Had to ignore the data.sql file, had to ddl to update, and ignore the init.mode setting.
Filled the tables by using POSTMAN and MySQLWorkbench

At this point, you can start the Spring Boot application. Hibernate will create the associated table(s) that you specified as @Entity components.

```
FoodBoxBackend - FoodBoxBackendApplication [Spring Boot App] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (Nov 18, 2022, 7:24:02 PM) [pid: 40768]
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.5)

2022-11-18 19:24:04.525  INFO 40768 --- [           main] c.f.r.FoodBoxBackendApplication          : Starting FoodBoxBackendApplication using Java 17.0.1 on Nia-Laptop wi
2022-11-18 19:24:04.528  INFO 40768 --- [           main] c.f.r.FoodBoxBackendApplication          : No active profile set, falling back to 1 default profile: "default"
2022-11-18 19:24:05.454  INFO 40768 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-11-18 19:24:05.529  INFO 40768 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 59 ms. Found 1 JPA reposi
2022-11-18 19:24:06.254  INFO 40768 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-11-18 19:24:06.268  INFO 40768 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-11-18 19:24:06.269  INFO 40768 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.68]
2022-11-18 19:24:06.469  INFO 40768 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-11-18 19:24:06.469  INFO 40768 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1886 ms
2022-11-18 19:24:06.781  INFO 40768 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-11-18 19:24:06.863  INFO 40768 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.12.Final
2022-11-18 19:24:07.143  INFO 40768 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-11-18 19:24:07.310  INFO 40768 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-11-18 19:24:07.762  INFO 40768 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-11-18 19:24:07.791  INFO 40768 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
Hibernate: create table user (id bigint not null, date_created datetime(6), email varchar(255), first_name varchar(255), last_name varchar(255), password varchar(255), ph
2022-11-18 19:24:08.695  INFO 40768 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.tr
2022-11-18 19:24:08.708  INFO 40768 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-11-18 19:24:09.227  WARN 40768 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database qu
2022-11-18 19:24:09.569  INFO 40768 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-11-18 19:24:09.578  INFO 40768 --- [           main] c.f.r.FoodBoxBackendApplication          : Started FoodBoxBackendApplication in 5.426 seconds (JVM running for 6
```

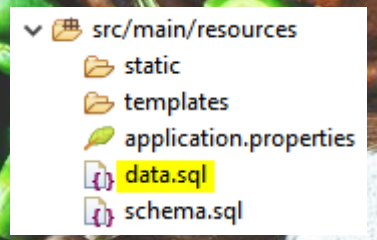
SQL Schema File: schema.sql



Note: Couldn't get this to work completely. Had to ignore the data.sql file, had to ddl to update, and ignore the init.mode setting.
Filled the tables by using POSTMAN and MySQLWorkbench

```
-- MySQL (x86_64)
--
-- Host: localhost    Database: foodboxdb
-- ------------------------------------------------------
-- Port: 3306

-- ########   TABLES WITH NO FOREIGN KEYS ###### --

CREATE DATABASE IF NOT EXISTS foodboxdb;
USE foodboxdb;

--
-- Table: cuisines
--
DROP TABLE IF EXISTS cuisines;
CREATE TABLE cuisines
(
cuisineid bigint NOT NULL AUTO_INCREMENT,
cuisineName varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
PRIMARY KEY (cuisineid)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

--
-- Table: tags
--
DROP TABLE IF EXISTS tags;
CREATE TABLE tags
(
tagid bigint NOT NULL AUTO_INCREMENT,
tagname varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
productcount int DEFAULT 0,
PRIMARY KEY (tagid)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

--
-- Table: userRoles
--
DROP TABLE IF EXISTS userRoles;
CREATE TABLE userRoles
(
roleid bigint NOT NULL AUTO_INCREMENT,
roletype varchar(50) COLLATE utf8mb4_unicode_ci,
PRIMARY KEY (roleid)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

--
-- Table: products
--
DROP TABLE IF EXISTS products;
CREATE TABLE products
(
    productId bigint NOT NULL AUTO_INCREMENT,
    productImageUrl varchar(255) COLLATE utf8mb4_unicode_ci
    productName varchar(255) COLLATE utf8mb4_unicode_ci DEFA
    price decimal(19,2) DEFAULT NULL,
    calories int DEFAULT 0,
    rating float DEFAULT NULL,
    numOfReviews float DEFAULT NULL,
    description longtext DEFAULT NULL,
    tags tinyblob DEFAULT NULL,
    cuisines tinyblob DEFAULT NULL,
    dateCreated datetime(6) default NULL,
    PRIMARY KEY (productId)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unico

-- ########   TABLES WITH FOREIGN KEYS ###### --

--
-- Table: users
--
DROP TABLE IF EXISTS users;
CREATE TABLE users
(
userid bigint NOT NULL AUTO_INCREMENT,
roleid bigint,
firstname varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
lastname varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
username varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
email varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
password varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
phoneNumber varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
date_created timestamp default NULL,
PRIMARY KEY (userid),
FOREIGN KEY (roleid) REFERENCES userroles(roleid)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uni
```

SQL Data File: data.sql




```sql
/* Filling the cuisines table */
INSERT INTO cuisines (cuisineName) VALUES ('American');
INSERT INTO cuisines (cuisineName) VALUES ('Indian');
INSERT INTO cuisines (cuisineName) VALUES ('British');
INSERT INTO cuisines (cuisineName) VALUES ('Japanese');
INSERT INTO cuisines (cuisineName) VALUES ('Chinese');
INSERT INTO cuisines (cuisineName) VALUES ('Mexican');
INSERT INTO cuisines (cuisineName) VALUES ('French');
INSERT INTO cuisines (cuisineName) VALUES ('Persian');
INSERT INTO cuisines (cuisineName) VALUES ('Italian');
INSERT INTO cuisines (cuisineName) VALUES ('Spanish');
COMMIT;

/* Filling the tags table */
INSERT INTO tags (tagName) VALUES ('all');
INSERT INTO tags (tagName) VALUES ('pasta');
INSERT INTO tags (tagName) VALUES ('vegetarian');
INSERT INTO tags (tagName) VALUES ('seafood');
INSERT INTO tags (tagName) VALUES ('meat');
INSERT INTO tags (tagName) VALUES ('rice');
INSERT INTO tags (tagName) VALUES ('chicken');
INSERT INTO tags (tagName) VALUES ('bread');
INSERT INTO tags (tagName) VALUES ('sandwich');
INSERT INTO tags (tagName) VALUES ('salad');
INSERT INTO tags (tagName) VALUES ('beef');
INSERT INTO tags (tagName) VALUES ('noodles');
INSERT INTO tags (tagName) VALUES ('soup');
INSERT INTO tags (tagName) VALUES ('stew');
INSERT INTO tags (tagName) VALUES ('side');
COMMIT;

```

Note: Couldn't get this to work completely. Had to ignore the data.sql file, had to ddl to update, and ignore the init.mode setting.
Filled the tables by using POSTMAN and MySQLWorkbench


CORS
CONFIGURATION

CORS

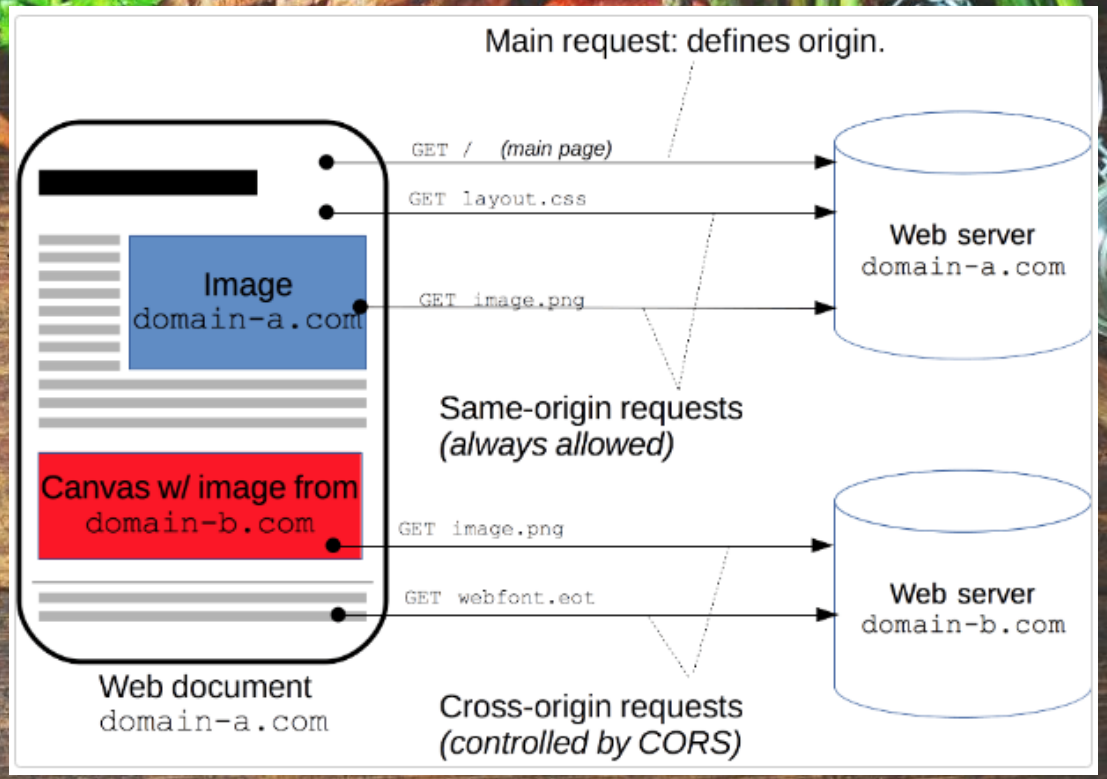
https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Cross-Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

An example of a cross-origin request: the front-end JavaScript code served from `https://domain-a.com` uses `XMLHttpRequest` to make a request for `https://domain-b.com/data.json`.

For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts. For example, `XMLHttpRequest` and the Fetch API follow the same-origin policy. This means that a web application using those APIs can only request resources from the same origin the application was loaded from unless the response from other origins includes the right CORS headers.



Update Backend for CORS Configuration

FoodBoxBackendApplication.java

FoodBoxBackend ▸ src/main/java ▸ com.foodbox.restbackend ▸ FoodBoxBackendApplication ▸

```java
package com.foodbox.restbackend;

import java.util.Arrays;

@SpringBootApplication
public class FoodBoxBackendApplication {

    public static void main(String[] args) {
        SpringApplication.run(FoodBoxBackendApplication.class, args);
    }

    //For CORS Configuration - pretty much reusable code
        @Bean
        public CorsFilter corsFilter() {
            CorsConfiguration corsConfiguration = new CorsConfiguration();
            corsConfiguration.setAllowCredentials(true);
            corsConfiguration.setAllowedOrigins(Arrays.asList("http://localhost:4200"));
            corsConfiguration.setAllowedHeaders(Arrays.asList("Origin", "Access-Control-Allow-Origin", "Content-Type",
                    "Accept", "Authorization", "Origin, Accept", "X-Requested-With",
                    "Access-Control-Request-Method", "Access-Control-Request-Headers"));
            corsConfiguration.setExposedHeaders(Arrays.asList("Origin", "Content-Type", "Accept", "Authorization",
                    "Access-Control-Allow-Origin", "Access-Control-Allow-Origin", "Access-Control-Allow-Credentials"));
            corsConfiguration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
            UrlBasedCorsConfigurationSource urlBasedCorsConfigurationSource = new UrlBasedCorsConfigurationSource();
            urlBasedCorsConfigurationSource.registerCorsConfiguration("/**", corsConfiguration);
            return new CorsFilter(urlBasedCorsConfigurationSource);
        }

}//end class
```


SWAGGER2
CONFIGURATION

Update application.properties file

```
#MySQL Configuration
spring.datasource.username=
spring.datasource.password=
spring.datasource.url=jdbc:mysql://localhost:3306/foodboxdb
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
server.port=8080
#For Swagger Documentation
spring.mvc.pathmatch.matching-strategy=ANT_PATH_MATCHER
```

Update pom.xml

Swagger Dependencies: Update pom.xml to include the dependencies for Swagger2 and SwaggerUI

```
39      <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
40      <dependency>
41          <groupId>io.springfox</groupId>
42          <artifactId>springfox-swagger2</artifactId>
43          <version>2.7.0</version>
44      </dependency>
45      <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
46      <dependency>
47          <groupId>io.springfox</groupId>
48          <artifactId>springfox-swagger-ui</artifactId>
49          <version>2.7.0</version>
50      </dependency>
```

Notes:

1. I couldn't get it to work with the 3.0.0 version of Swagger2 and Swagger-UI. I had to switch to the 2.7.0 version (which Ms. Sonam showed us during class).
2. I found the following article helpful: https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api

> It's important to mention that the latest version of Swagger specification, now known as OpenAPI 3.0, is better supported by the Springdoc project and should be used for documenting Spring REST API.

Updated main application

Swagger Dependencies: Update FoodBoxBackendApplication.java to include annotations and an explicit Bean for Swagger2

```java
package com.foodbox.restbackend;

import java.util.Arrays;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;

import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@SpringBootApplication
@EnableSwagger2
public class FoodBoxBackendApplicat
```

```java

    //For Swagger documentation
    @Bean
    public Docket allApis() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select().apis(RequestHandlerSelectors.basePackage("com.foodbox.restbackend"))
                .build();
    }

}//end class
```

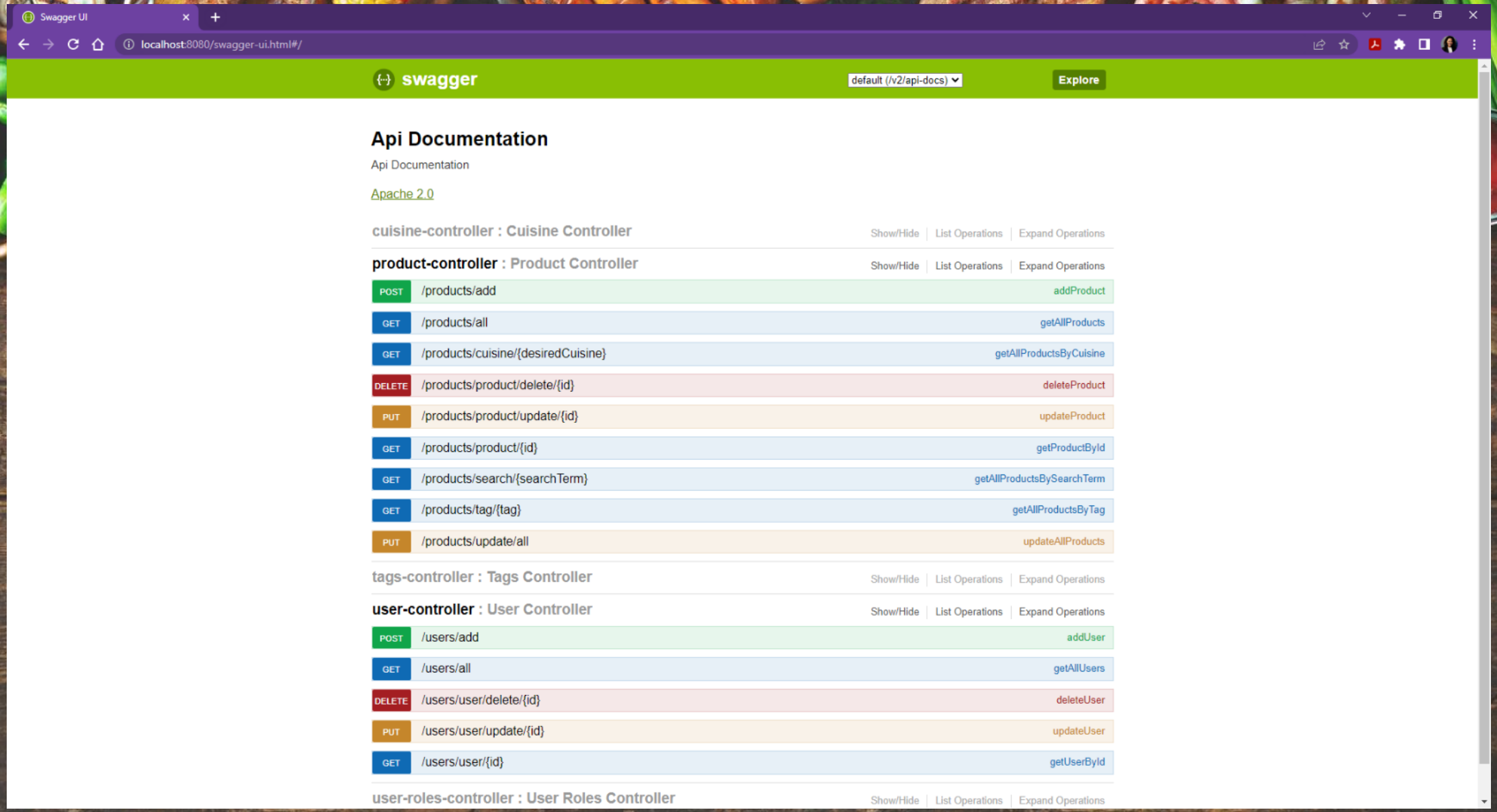
Relaunched the Application

Spring Console

```
FoodBoxBackend - FoodBoxBackendApplication [Spring Boot App] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (Nov 20, 2022, 11:33:12 AM) [pid: 15140]
2022-11-20 11:33:15.590  INFO 15140 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-11-20 11:33:15.591  INFO 15140 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.68]
2022-11-20 11:33:15.744  INFO 15140 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-11-20 11:33:15.744  INFO 15140 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1476 ms
2022-11-20 11:33:15.937  INFO 15140 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-11-20 11:33:15.987  INFO 15140 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.12.Final
2022-11-20 11:33:16.159  INFO 15140 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-11-20 11:33:16.260  INFO 15140 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-11-20 11:33:16.576  INFO 15140 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-11-20 11:33:16.599  INFO 15140 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
2022-11-20 11:33:17.272  INFO 15140 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.tr
2022-11-20 11:33:17.282  INFO 15140 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-11-20 11:33:17.796  WARN 15140 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database qu
2022-11-20 11:33:17.913  INFO 15140 --- [           main] pertySourcedRequestMappingHandlerMapping : Mapped URL path [/v2/api-docs] onto method [springfox.documentation.s
2022-11-20 11:33:18.194  INFO 15140 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-11-20 11:33:18.195  INFO 15140 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Context refreshed
2022-11-20 11:33:18.212  INFO 15140 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Found 1 custom documentation plugin(s)
2022-11-20 11:33:18.241  INFO 15140 --- [           main] s.d.s.w.s.ApiListingReferenceScanner     : Scanning for api listing references
2022-11-20 11:33:18.421  INFO 15140 --- [           main] c.f.r.FoodBoxBackendApplication          : Started FoodBoxBackendApplication in 4.583 seconds (JVM running for 5
2022-11-20 11:33:51.127  INFO 15140 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-11-20 11:33:51.127  INFO 15140 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2022-11-20 11:33:51.128  INFO 15140 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
```

Running API Documentation

http://localhost:8080/swagger-ui.html




USER
COMPONENTS

User Model Class

- Properties
- Default Constructor
- Parameterized Constructor
- Getters & setters (could have used Lombok to get the boilerplate code)
- Override toString()

User.java

FoodBoxBackend ▸ src/main/java ▸ com.foodbox.restbackend.model ▸ User ▸

```
1  package com.foodbox.restbackend.model;
2
3  import java.io.Serializable;
7
8  @Entity
9  @Table(name="users")
10 public class User implements Serializable{
11
12     //Properties
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     @Column(name="userid")
16     private Long id;
17     @Column(name="roleType")
18     private String roleType;
19     @Column(name="firstname")
20     private String firstName;
21     @Column(name="lastname")
22     private String lastName;
23     @Column(name="username")
24     private String userName;
25     @Column(name="email")
26     private String email;
27     @Column(name="password")
28     private String password;
29     @Column(name="phoneNumber")
30     private String phoneNumber;
31     @Column(name="dateCreated")
32     private Timestamp dateCreated;
33
34     //Default Constructor
35     public User() {}
```

User Repository

Repository Class: Create the `User Repository` to use the `JpaRepository` library for CRUD operations.

- This is important because we want to have a way to save a new `User` from the frontend to the database on the backend.
- We created an interface called `UserRepository` that extends `JpaRepository`. That way we can take advantage of the work that Spring Boot has already done for us by creating the CRUD methods. We can inject the object created from the `UserRepository` interface and call one of the CRUD methods.

UserRepository.java

FoodBoxBackend ▸ src/main/java ▸ com.foodbox.restbackend.repo ▸ UserRepository ▸

```
1  package com.foodbox.restbackend.repo;
2
3  import java.util.Optional;
7
8
9  public interface UserRepository extends JpaRepository<User, Long>  {
10     void deleteUserById(Long id);
11
12     Optional<User> findUserById(Long id);
13
14 }//end interface
15
```

Service Class

- **Service Class:** Create a `UserService` service class that will be used in our controller. Add the relevant CRUD methods, using the dependency on the repository object.
  - Create a method to add a new User. The method will use the userRepository.save(user) method to do the actual saving in the database. If you drill into the JpaRepository code, you will see the actual code to make the SQL calls to the database. That way, you don't have to do this manually (unless you need to).
  - Create method to return a list of all the users in the database.
  - Create a method to update a specific User.
  - Create a method to delete a specific User by userId. Perhaps you will create your own method for the delete function. If so, declare it in the repository interface.
  - Create a method to find a User by userId. You can denote the method as Optional<>.

Code: UserService.java

```java
package com.foodbox.restbackend.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.foodbox.restbackend.exception.UserNotFoundException;
import com.foodbox.restbackend.model.User;
import com.foodbox.restbackend.repo.UserRepository;

@Service
@Transactional
public class UserService {

    //Properties
    private final UserRepository userRepo;

    //Constructor with Dependency Injection of the User Repository
    @Autowired
    public UserService(UserRepository userRepo) {
        this.userRepo = userRepo;
    }

    //CRUD Methods
    public User addUser(User user) {
        return userRepo.save(user);
    }

    public List<User> findAllUsers(){
        return userRepo.findAll();
    }

    public User updateUser(User user) {
        return userRepo.save(user);
    }

    public User findUserById(Long id) {
        return userRepo.findUserById(id)
                .orElseThrow(() -> new UserNotFoundException("User by id " + id + " was not found"));
    }

    //must have @Transactional annotation for this to work properly
    public void deleteUser(Long id) {
        userRepo.deleteUserById(id);
    }


}//end class
```

Example: UserNotFoundException.java

Exception Handling: You can create custom exceptions. Ideally, you would want to another package dedicated to exceptions.

UserNotFoundException.java

FoodBoxBackend ▸ src/main/java ▸ com.foodbox.restbackend.exception ▸ UserNotFoundException ▸

```
1  package com.foodbox.restbackend.exception;
2
3  public class UserNotFoundException extends RuntimeException {
4      public UserNotFoundException(String message) {
5          super(message);
6      }
7  }
```

Exposing the API

Controllers: Create a `REST Controller class` with a base URL. These will be the endpoints that we will be exposed. Essentially, we should have controller methods to implement/match the service methods.

user-controller : User Controller — Show/Hide | List Operations | Expand Operations

POST	/users/add	addUser
GET	/users/all	getAllUsers
DELETE	/users/user/delete/{id}	deleteUser
PUT	/users/user/update/{id}	updateUser
GET	/users/user/{id}	getUserById

TESTING THE BACKEND

with POSTMAN

Testing Endpoints:

`@PostMapping("/add")`

POST localhost:8080/users/

localhost:8080/users/add

POST | localhost:8080/users/add

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{
    "roleType": "customer",
"firstName": "Kimberly",
"lastName": "Smith",
"userName": "kimberly",
"email": "kimberly@designerNia.com",
"password": "abc123",
"phoneNumber": "2145678932"
}
```

Body | Cookies | Headers (8) | Test Results

Pretty | Raw | Preview | Visualize | JSON

```
{
    "id": 27,
    "roleType": "customer",
    "firstName": "Kimberly",
    "lastName": "Smith",
    "userName": "kimberly",
    "email": "kimberly@designerNia.com",
    "password": "abc123",
    "phoneNumber": "2145678932",
    "dateCreated": "2022-12-19T06:52:51.136+00:00"
}
```

userid	date_created	email	firstname	lastname	password	phone_number	role_type	us
2	2022-12-03 14:56:39.234753	nia@designerNia.com	Nia	Kelley Jester	admin123	2145678932	admin	nia
3	2022-12-03 14:58:52.459637	jasen@designerNia.com	Jasen	Jester	besthubby	2145678932	customer	jas
4	2022-12-03 14:59:19.891000	christine@designerNia.com	Christine	Smith-Johnson	test123	2145678932	customer	ch
6	2022-12-03 14:59:51.170875	patricia@designerNia.com	Patricia	Weissnat	abc123	2145678932	customer	pa
7	2022-12-03 15:00:09.293619	Julianne@designerNia.com	Julianne	Dietrich	abc123	2145678932	customer	jul
8	2022-12-06 02:13:47.250679	timb@home.net	Tim	Bradford	abc123	1234567890	customer	tim
9	2022-12-06 02:16:04.877000	lucy@designerNia.com	Lucy	Chen-Bradford	abc123	2145678932	customer	luc
11	2022-12-10 11:28:11.003991	Monica@designerNia.com	Monica	Sharp	abc123	2145678932	customer	mo
12	2022-12-10 13:00:49.407000	john@designerNia.com	John	Nolan	abc123	2145678932	customer	joh
15	2022-12-11 22:43:38.346789	linda@gmail.com	Linda	Kelley	abc123	1234567890	customer	lin
16	2022-12-12 02:36:59.936000	kalonji@gmail.com	Kalonji	Kelley	abc123	1112223333	customer	ka
17	2022-12-12 12:47:28.466233	terry@home.com	Terry	Mickens	abc123	1234567890	customer	ter
18	2022-12-12 12:48:47.876921	jackson@designerNia.com	Jackson	West	abc123	1234567890	customer	jac
19	2022-12-17 10:12:07.609960	carrie@home.com	Carrie	Bradshaw	test1234	1234567896	customer	ca
25	2022-12-17 21:29:21.051162	jess@designerNia.com	Jess	Keller	abc123	2145678932	custom	je
27	2022-12-19 00:52:51.136256	kimberly@designerNia.com	Kimberly	Smith	abc123	2145678932	custom	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	

Testing Endpoints:

`@GetMapping("/all")`

GET localhost:8080/users/a

localhost:8080/users/all

GET | localhost:8080/users/all

Params | Authorization | Headers (9) | Body | Pre-request Script | Te

none | form-data | x-www-form-urlencoded | raw | binary

```
{
    "roleType": "customer",
    "firstName": "Kimberly",
    "lastName": "Smith",
    "userName": "kimberly",
    "email": "kimberly@designerNia.com",
    "password": "abc123",
    "phoneNumber": "2145678932"
}
```

Body | Cookies | Headers (8) | Test Results

Pretty | Raw | Preview | Visualize | JSON

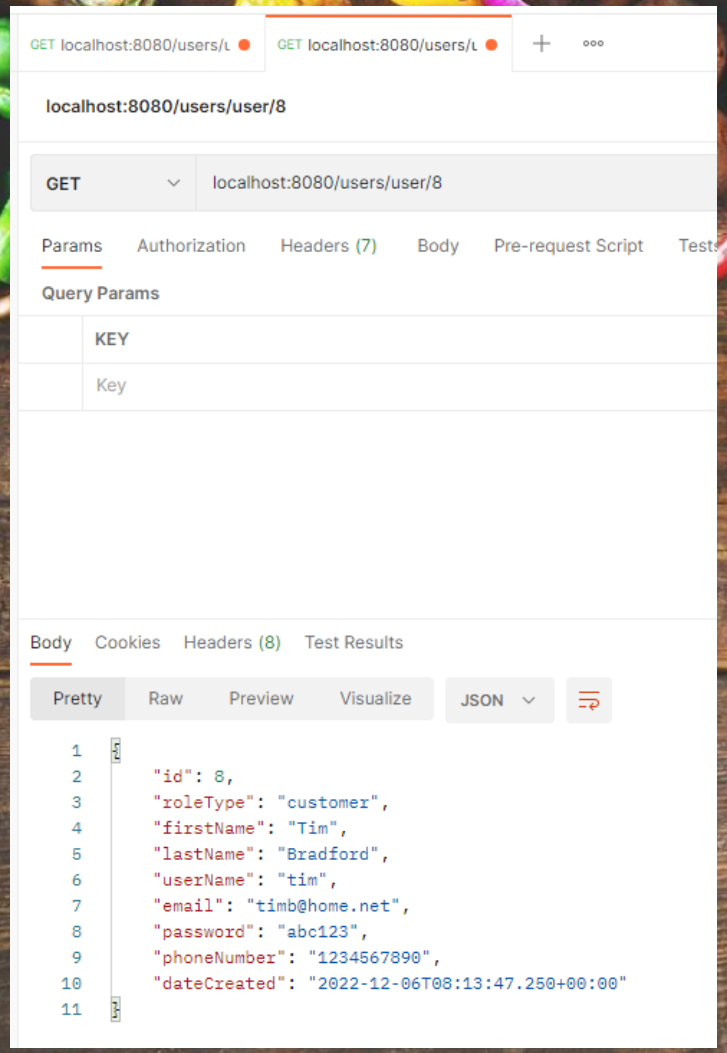

```
[
    {
        "id": 2,
        "roleType": "admin",
        "firstName": "Nia",
        "lastName": "Kelley Jester",
        "userName": "niajester",
        "email": "nia@designerNia.com",
        "password": "admin123",
        "phoneNumber": "2145678932",
        "dateCreated": "2022-12-03T20:56:39.234+00:00"
    },
    {
        "id": 3,
        "roleType": "customer",
        "firstName": "Jasen",
        "lastName": "Jester",
        "userName": "jasenjester",
        "email": "jasen@designerNia.com",
        "password": "besthubby",
        "phoneNumber": "2145678932",
```

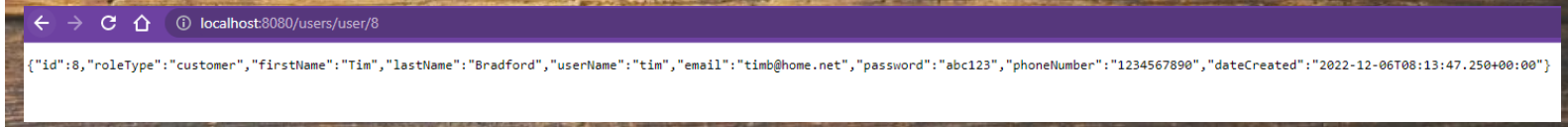

localhost:8080/users/all


```
[{"id":2,"roleType":"admin","firstName":"Nia","lastName":"Kelley Jester","userName":"niajester","email":"nia@designerNia.com","password":"admin123","phoneNumber":"2145678932","dateCreated":"2022-12-03T20:56:39.234+00:00"},
{"id":3,"roleType":"customer","firstName":"Jasen","lastName":"Jester","userName":"jasenjester","email":"jasen@designerNia.com","password":"besthubby","phoneNumber":"2145678932","dateCreated":"2022-12-03T20:58:52.459+00:00"},
{"id":4,"roleType":"customer","firstName":"Christine","lastName":"Smith-Johnson","userName":"christine","email":"christine@designerNia.com","password":"test123","phoneNumber":"2145678932","dateCreated":"2022-12-03T20:59:19.891+00:00"},
{"id":6,"roleType":"customer","firstName":"Patricia","lastName":"Weissnat","userName":"patriciaw","email":"patricia@designerNia.com","password":"abc123","phoneNumber":"2145678932","dateCreated":"2022-12-03T20:59:51.170+00:00"},
{"id":7,"roleType":"customer","firstName":"Julianne","lastName":"Dietrich","userName":"julianne","email":"Julianne@designerNia.com","password":"abc123","phoneNumber":"2145678932","dateCreated":"2022-12-03T21:00:09.293+00:00"},
{"id":8,"roleType":"customer","firstName":"Tim","lastName":"Bradford","userName":"tim","email":"timb@home.net","password":"abc123","phoneNumber":"1234567890","dateCreated":"2022-12-06T08:13:47.250+00:00"},
{"id":9,"roleType":"customer","firstName":"Lucy","lastName":"Chen-Bradford","userName":"lucy","email":"lucy@designerNia.com","password":"abc123","phoneNumber":"2145678932","dateCreated":"2022-12-06T08:16:04.877+00:00"},
{"id":11,"roleType":"customer","firstName":"Monica","lastName":"Sharp","userName":"monica","email":"Monica@designerNia.com","password":"abc123","phoneNumber":"2145678932","dateCreated":"2022-12-10T17:28:11.003+00:00"},
{"id":12,"roleType":"customer","firstName":"John","lastName":"Nolan","userName":"john","email":"john@designerNia.com","password":"abc123","phoneNumber":"2145678932","dateCreated":"2022-12-10T19:00:49.407+00:00"},
{"id":15,"roleType":"customer","firstName":"Linda","lastName":"Kelley","userName":"linda","email":"linda@gmail.com","password":"abc123","phoneNumber":"1234567890","dateCreated":"2022-12-12T04:43:38.346+00:00"},
{"id":16,"roleType":"customer","firstName":"Kalonji","lastName":"Kelley","userName":"kalonji","email":"kalonji@gmail.com","password":"abc123","phoneNumber":"1112223333","dateCreated":"2022-12-12T08:36:59.936+00:00"},
{"id":17,"roleType":"customer","firstName":"Terry","lastName":"Hickens","userName":"terry","email":"terry@home.com","password":"abc123","phoneNumber":"1234567890","dateCreated":"2022-12-12T18:47:28.466+00:00"},
{"id":18,"roleType":"customer","firstName":"Jackson","lastName":"West","userName":"jackson","email":"jackson@designerNia.com","password":"abc123","phoneNumber":"1234567890","dateCreated":"2022-12-12T18:48:47.876+00:00"},
{"id":19,"roleType":"customer","firstName":"Carrie","lastName":"Bradshaw","userName":"carrie","email":"carrie@home.com","password":"test1234","phoneNumber":"1234567896","dateCreated":"2022-12-17T16:12:07.609+00:00"},
{"id":25,"roleType":"customer","firstName":"Jess","lastName":"Keller","userName":"jess","email":"jess@designerNia.com","password":"abc123","phoneNumber":"2145678932","dateCreated":"2022-12-18T03:29:21.051+00:00"},
{"id":27,"roleType":"customer","firstName":"Kimberly","lastName":"Smith","userName":"kimberly","email":"kimberly@designerNia.com","password":"abc123","phoneNumber":"2145678932","dateCreated":"2022-12-19T06:52:51.136+00:00"}]
```

Testing Endpoints:

```
@GetMapping("/user/{id}")
```





Testing Endpoints:

```
@DeleteMapping("/user/delete/{id}")
```

GET localhost:8080/users/u ● GET localhost:8080/users/u ● DEL localhost:8080/users/u ●

localhost:8080/users/user/delete/27

DELETE | localhost:8080/users/user/delete/27

Params | Authorization | Headers (7) | Body | Pre-request Script | Tests | Se

Query Params

KEY	Value
Key	

Body | Cookies | Headers (7) | Test Results

Pretty | Raw | Preview | Visualize | Text

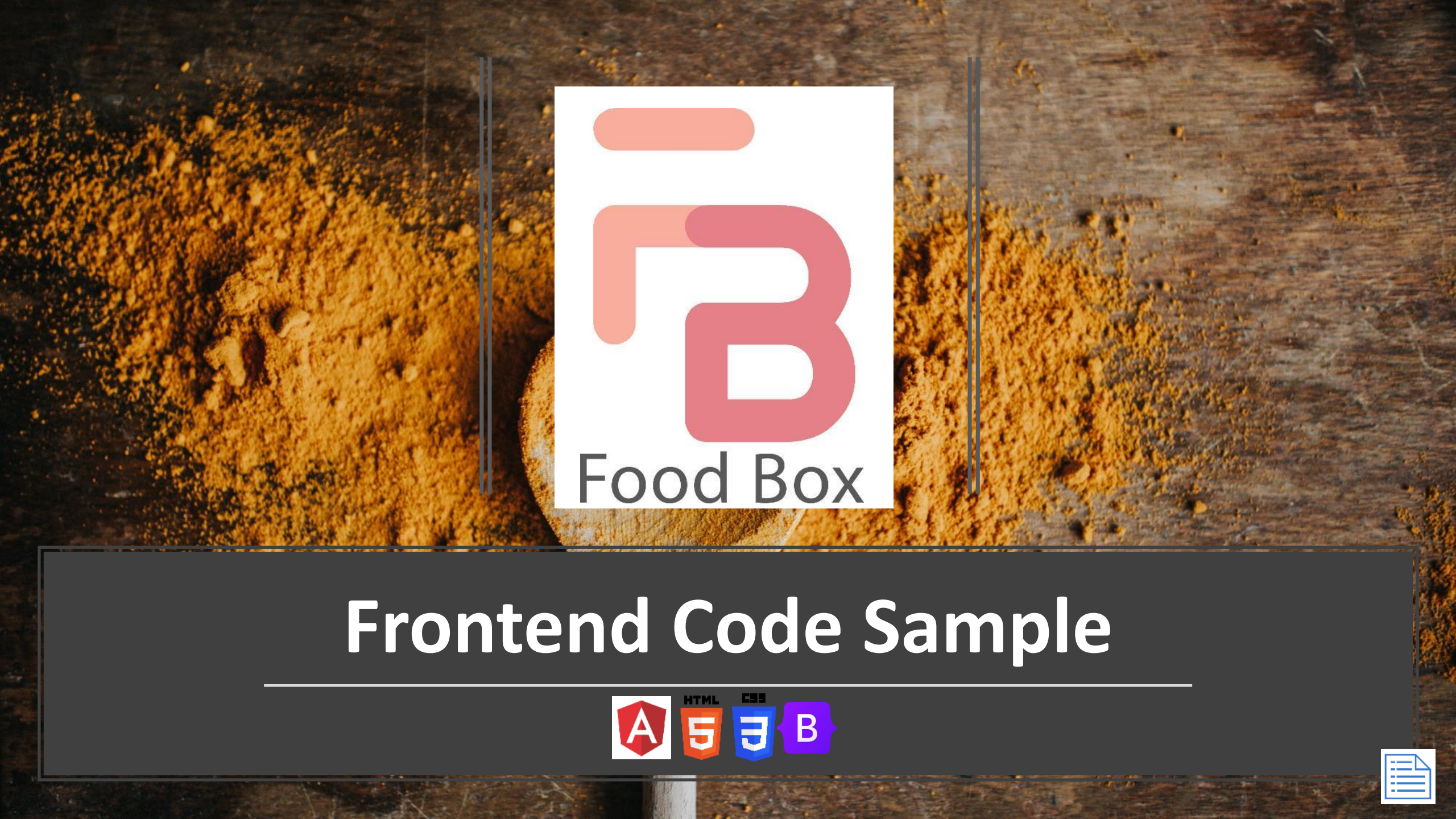
```
1
```

BEFORE

userid	date_created	email	firstname	lastname	password	phone_number	role_type	us
2	39.234753	nia@designerNia.com	Nia	Kelley Jester	admin123	2145678932	admin	nia
3	52.459637	jasen@designerNia.com	Jasen	Jester	besthubby	2145678932	customer	jas
4	2022-12-03 14:59:19.891000	christine@designerNia.com	Christine	Smith-Johnson	test123	2145678932	customer	ch
6	2022-12-03 14:59:51.170875	patricia@designerNia.com	Patricia	Weissnat	abc123	2145678932	customer	pa
7	2022-12-03 15:00:09.293619	Julianne@designerNia.com	Julianne	Dietrich	abc123	2145678932	customer	jul
8	2022-12-06 02:13:47.250679	timb@home.net	Tim	Bradford	abc123	1234567890	customer	tim
9	2022-12-06 02:16:04.877000	lucy@designerNia.com	Lucy	Chen-Bradford	abc123	2145678932	customer	luc
11	2022-12-10 11:28:11.003991	Monica@designerNia.com	Monica	Sharp	abc123	2145678932	customer	mo
12	2022-12-10 13:00:49.407000	john@designerNia.com	John	Nolan	abc123	2145678932	customer	jol
15	2022-12-11 22:43:38.346789	linda@gmail.com	Linda	Kelley	abc123	1234567890	customer	lin
16	2022-12-12 02:36:59.936000	kalonji@gmail.com	Kalonji	Kelley	abc123	1112223333	customer	ka
17	2022-12-12 12:47:28.466233	terry@home.com	Terry	Mickens	abc123	1234567890	customer	ter
18	2022-12-12 12:48:47.876921	jackson@designerNia.com	Jackson	West	abc123	1234567890	customer	jac
19	2022-12-17 10:12:07.609960	carrie@home.com	Carrie	Bradshaw	test1234	1234567896	customer	ca
25	2022-12-17 21:29:21.051162	jess@designerNia.com	Jess	Keller	abc123	2145678932	customer	jes
27	2022-12-19 00:52:51.136256	kimberly@designerNia.com	Kimberly	Smith	abc123	2145678932	customer	kim

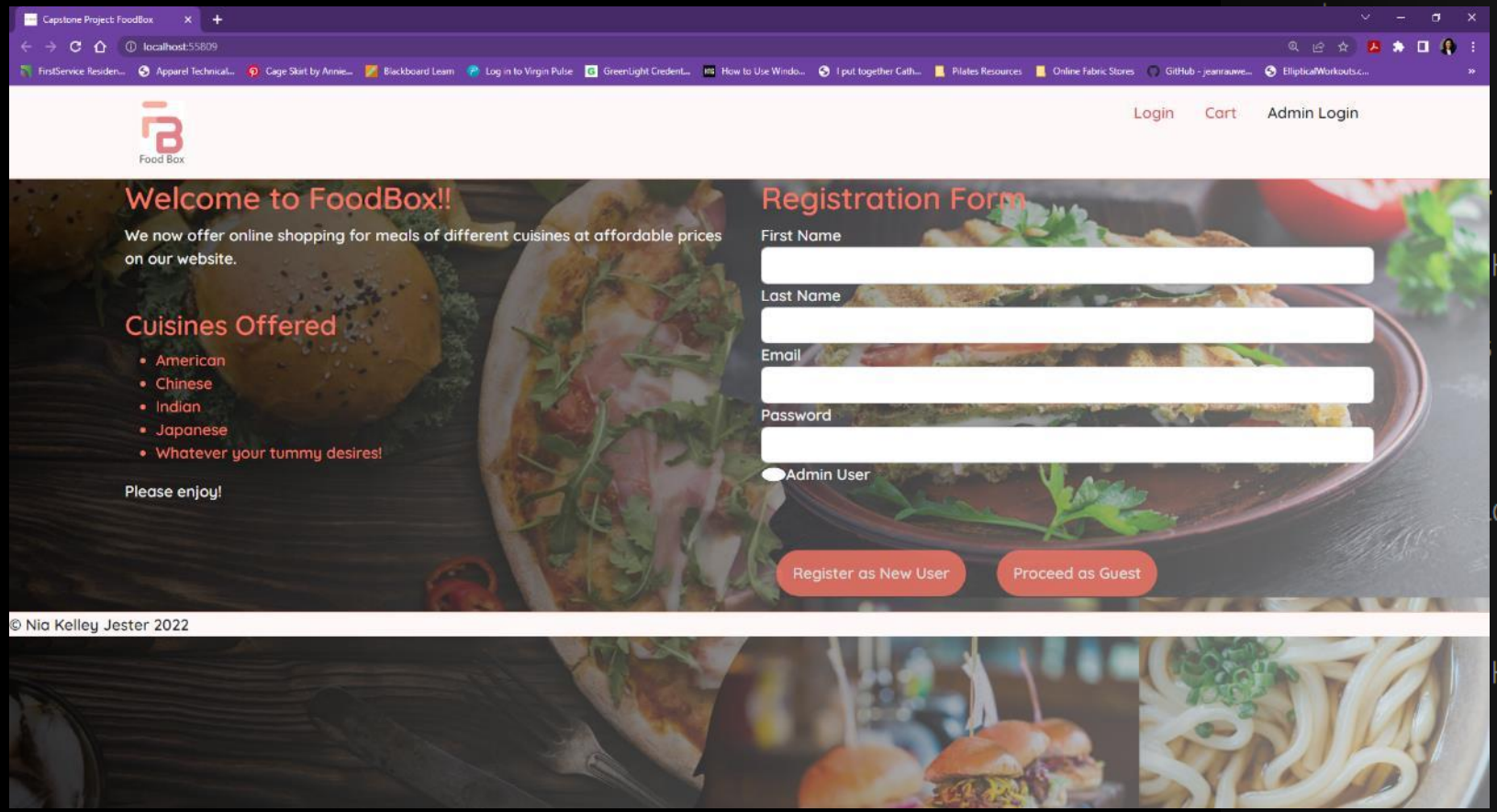
AFTER

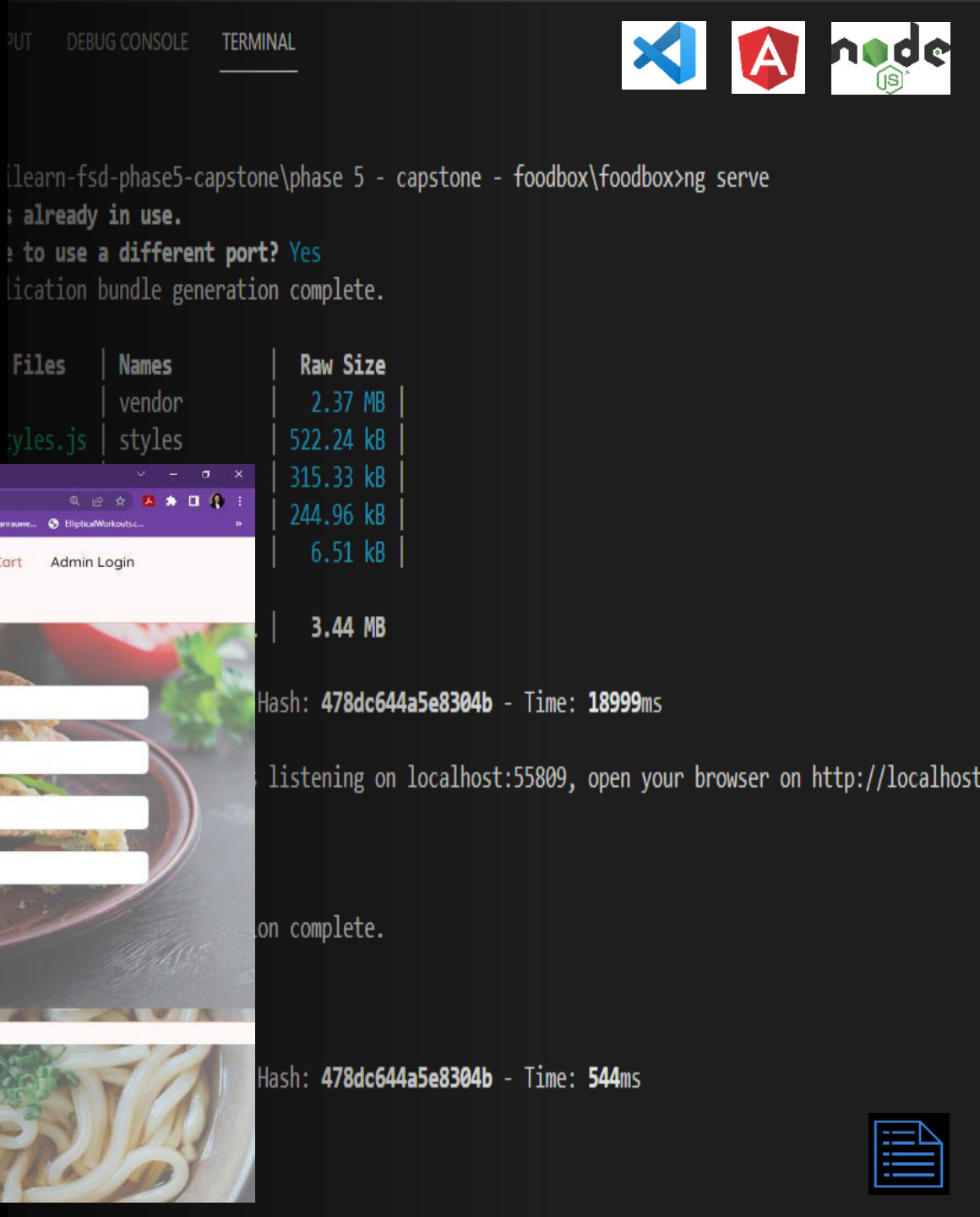
userid	date_created	email	firstname	lastname	password	phone_number	role_type	us
2	2022-12-03 14:56:39.234753	nia@designerNia.com	Nia	Kelley Jester	admin123	2145678932	admin	nia
3	2022-12-03 14:58:52.459637	jasen@designerNia.com	Jasen	Jester	besthubby	2145678932	customer	jas
4	2022-12-03 14:59:19.891000	christine@designerNia.com	Christine	Smith-Johnson	test123	2145678932	customer	ch
6	2022-12-03 14:59:51.170875	patricia@designerNia.com	Patricia	Weissnat	abc123	2145678932	customer	pa
7	2022-12-03 15:00:09.293619	Julianne@designerNia.com	Julianne	Dietrich	abc123	2145	er	jul
8	2022-12-06 02:13:47.250679	timb@home.net	Tim	Bradford	abc123	1234	er	tim
9	2022-12-06 02:16:04.877000	lucy@designerNia.com	Lucy	Chen-Bradford	abc123	2145678932	customer	luc
11	2022-12-10 11:28:11.003991	Monica@designerNia.com	Monica	Sharp	abc123	2145678932	customer	mo
12	2022-12-10 13:00:49.407000	john@designerNia.com	John	Nolan	abc123	2145678932	customer	jol
15	2022-12-11 22:43:38.346789	linda@gmail.com	Linda	Kelley	abc123	1234567890	customer	lin
16	2022-12-12 02:36:59.936000	kalonji@gmail.com	Kalonji	Kelley	abc123	1112223333	customer	ka
17	2022-12-12 12:47:28.466233	terry@home.com	Terry	Mickens	abc123	1234567890	customer	ter
18	2022-12-12 12:48:47.876921	jackson@designerNia.com	Jackson	West	abc123	1234567890	customer	jac
19	2022-12-17 10:12:07.609960	carrie@home.com	Carrie	Bradshaw	test1234	1234567896	customer	ca
25	2022-12-17 21:29:21.051162	jess@designerNia.com	Jess	Keller	abc123	2145678932	customer	j



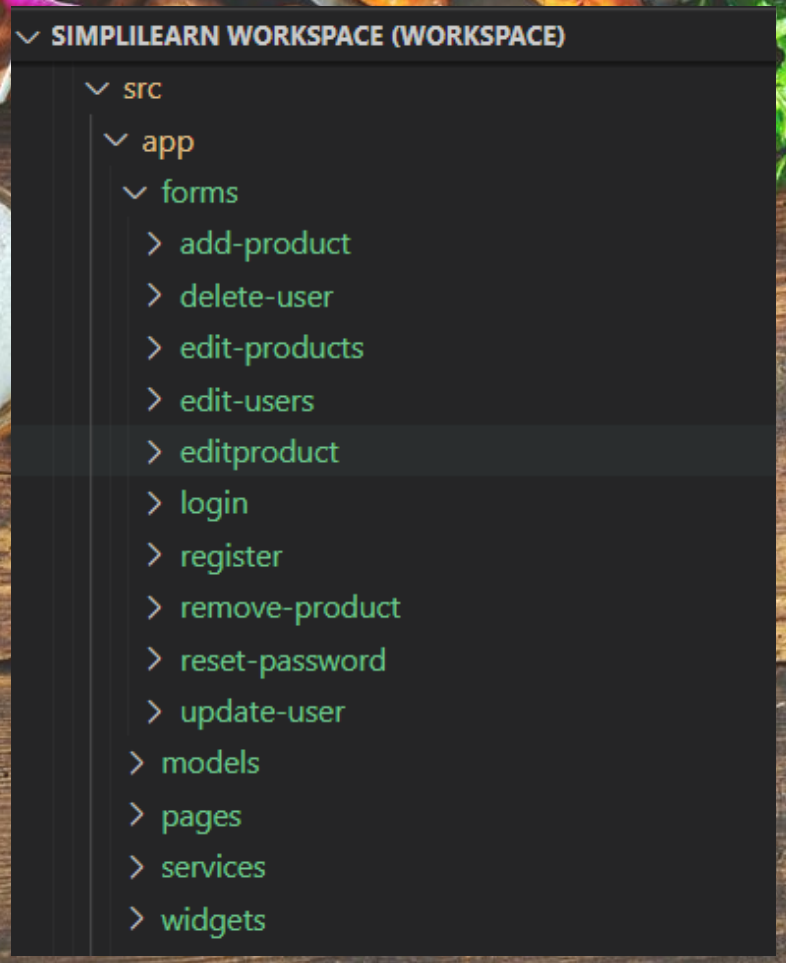
Frontend Code Sample

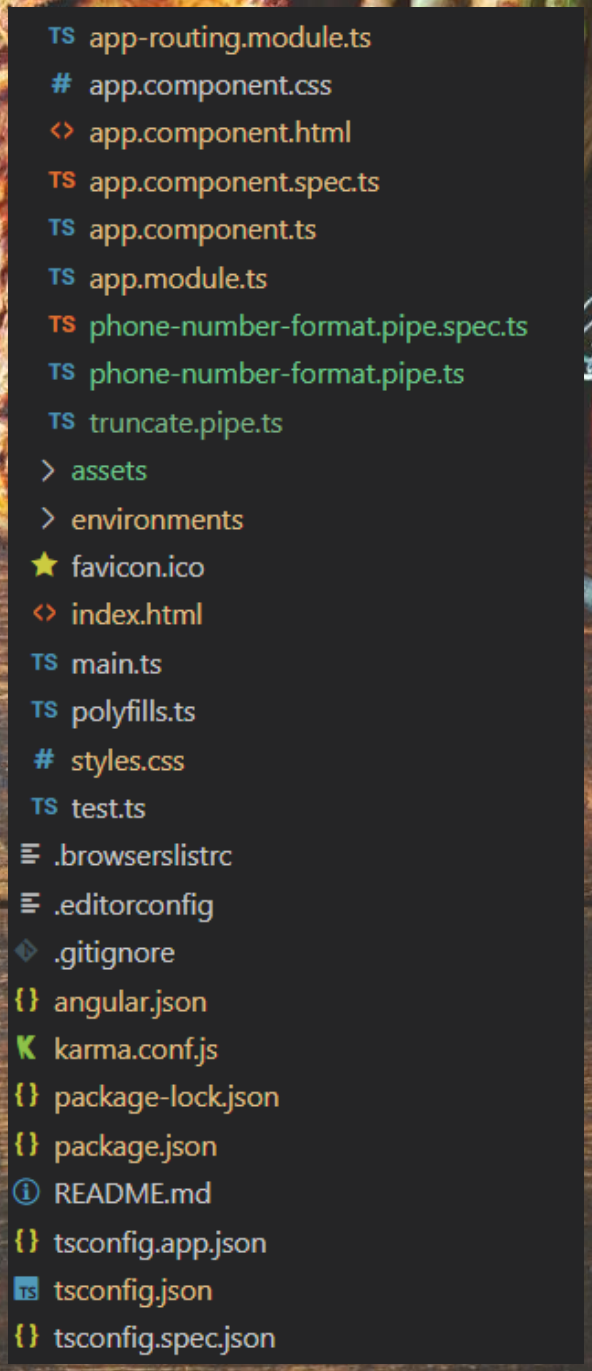
Food Box used the Kitchen Story project as it base.





SPA Directory Layout





Angular Components & Services

- **Angular Components**
  - Should be light-weight
  - Mainly rendering views through their template supported by application logic for better user experience.
- **Angular Services**
  - Provide their functionality to be consumed by components
  - Components don’t fetch data from the server or validate user input. It delegates such tasks to services.

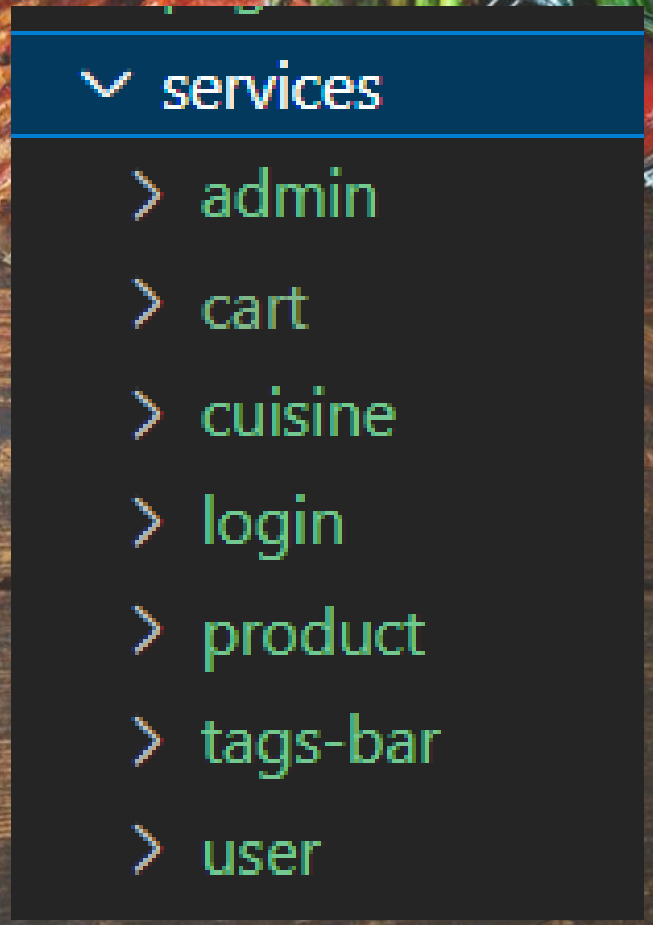
Food Box Angular Components

- Components are the basic building blocks of Angular applications. Food Box has **24 components**, which are declared in the `app.module.ts` file.
- A component controls a portion of the screen via a view. The view is comprised of an associated HTML template and CSS file.
- Components should only contain user interface related logic related to the view.
- Every Angular application has at least one component: the root component named `AppComponent` in `app.component.ts`.
- The `*.component.spec.ts` file allows you to unit test your app with other testing libraries.
  - We didn't cover unit testing of Angular in our course, so I didn't implement direct unit testing of the code.

```
40  @NgModule({
41    declarations: [
42      AppComponent,
43      HomeComponent,
44      HeaderComponent,
45      ProductComponent,
46      FooterComponent,
47      TruncatePipe,
48      SearchComponent,
49      TagsBarComponent,
50      ProductPageComponent,
51      CartPageComponent,
52      NotFoundComponent,
53      RegisterComponent,
54      CheckoutPageComponent,
55      OrderConfirmationComponent,
56      LoginComponent,
57      AddProductComponent,
58      RemoveProductComponent,
59      ResetPasswordComponent,
60      CuisinesBarComponent,
61      EditUsersComponent,
62      PhoneNumberFormatPipe,
63      EditProductsComponent,
64      DeleteUserComponent,
65      UpdateUserComponent,
66      EditproductComponent
67    ],
```

Food Box Angular Services

- Business logic to get data from a server should not be in a component but instead be encapsulated in a separate class called a service.
- Food Box has **7 services** to process the necessary logic for operation.
  1. Admin
  2. Cart
  3. Cuisine
  4. Login
  5. Product
  6. Tags-Bar
  7. User



Backend Server Communication

environment.ts M

java-simplilearn-fsd-phase5-capstone > phase 5 - capstone - foodbox > foodbox > src > environments > environment.ts > ...

```ts
// This file can be replaced during build by using the `fileReplacements` array.
// `ng build` replaces `environment.ts` with `environment.prod.ts`.
// The list of file replacements can be found in `angular.json`.

export const environment = {
  production: false,
  restUrl: 'http://localhost:8080'
};

/*
 * For easier debugging in development mode, you can import the following file
 * to ignore zone related error stack frames such as `zone.run`, `zoneDelegate.invokeTask`.
 *
 * This import should be commented out in production mode because it will have a negative impact
 * on performance if an error is thrown.
 */
// import 'zone.js/plugins/zone-error';  // Included with Angular CLI.

```

Front End Service

product.service.ts

```typescript
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import Product from 'src/app/models/Product';
import { Tag } from 'src/app/models/Tag';
import { environment } from 'src/environments/environment';
import { PRODUCTS_URL  } from 'src/app/constants/urls';

/* We mark our Product service as avaiable for dependency injection by decorating it with
    the @Injectable() annotation. */
@Injectable({
  providedIn: 'root'
})

//The Frontend ProductService class should mirror the corresponding backend Product controller class
export class ProductService {

  //Properties
  private restUrl:string = environment.restUrl; //obtained from the development env settings

  //Constructor
  //whenever we generate a ProductService object, it will have the ability tc
  //Note: HttpClient will not send the raw data. It will send an Observable t
  constructor(private http:HttpClient) {
  }
```

```typescript
//HTTP Methods - should mirror the ProductController on the backend

/* Notes: For Angular to connect to backend servers, we need Observables (from Reactive Extension library) ···

 /* ****************************************
 * Method Name: getAllProducts()
 * Access Type: public
 * Input Parameters: none
 * Return Type: Observable Product[] array
 * Purpose: Calls the backend /products/all endpoint (HTTP GET)
 * ****************************************/
 public getAllProducts():Observable<Product[]>{
  return this.http.get<Product[]>(`${this.restUrl}/products/all`);
 }

 /* ****************************************
 * Method Name: getProductById()
 * Access Type: public
 * Input Parameters: Product ID number
 * Return Type: Observable Product object
 * Purpose: Calls the backend /products/product/id endpoint (HTTP GET)
 * ****************************************/
 public getProductById(id:number):Observable<Product>{
     return this.http.get<Product>(`${this.restUrl}/products/product/` + id);
 }
```

Front End Service

product.service.ts

```
62
63    /* ****************************************
64     * Method Name: addProduct()
65     * Access Type: public
66     * Input Parameters: Product Object
67     * Return Type: Observable of any Product object
68     * Purpose: Calls the backend /products/add endpoint (HTTP POST)
69     * **************************************** */
70    public addProduct(product:Product):Observable<Product>{
71      return this.http.post<Product>(`${this.restUrl}/products/add`, product);
72    }
73
74    /* ****************************************
75     * Method Name: updateProduct()
76     * Access Type: public
77     * Input Parameters: Single Product object
78     * Return Type: Observable of Product
79     * Purpose: Calls the backend /products/product/update/id endpoint (HTTP PUT)
80     * **************************************** */
81    public updateProduct(product: Product, id:number): Observable<Product> {
82      return this.http.put<Product>(`${this.restUrl}/products/product/update/${id}`, product);
83    }
84
85    /* ****************************************
86     * Method Name: deleteProductById()
87     * Access Type: public
88     * Input Parameters: Product ID to be deleted
89     * Return Type: void Observable
90     * Purpose: Calls the backend /products/product/delete/id endpoint (HTTP DELETE)
91     * **************************************** */
92    public deleteProductById(id: number):Observable<Product> {
93      console.log("Made it to the delete method...");
94      return this.http.delete<Product>(`${this.restUrl}/products/product/delete/${id}`);
95    }
96
```

```
97    /* ****************************************
98     * Method Name: getAllProductsBySearchTerm()
99     * Access Type: public
100    * Input Parameters: search term string
101    * Return Type: Observable Product[] array
102    * Purpose: Calls the backend endpoint (HTTP GET)
103    * **************************************** */
104   getAllProductsBySearchTerm(searchTerm:string):Observable<Product[]>{
105     return this.http.get<Product[]>(`${this.restUrl}/products/search/${searchTerm}`);
106   }
107
108   /* ****************************************
109    * Method Name: getAllProductsByTag()
110    * Access Type: public
111    * Input Parameters: search term string
112    * Return Type: Observable Product[] array
113    * Purpose: Calls the backend endpoint (HTTP GET)
114    * **************************************** */
115   getAllProductsByTag(tag: string): Observable<Product[]> {
116     return tag.toLowerCase() === "all" ?
117       this.getAllProducts() :
118       this.http.get<Product[]>(`${this.restUrl}/products/tag/${tag}`);
119   }
120
121   /* ****************************************
122    * Method Name: getAllProductsByCuisine()
123    * Access Type: public
124    * Input Parameters: cuisine string
125    * Return Type: Observable Product[] array
126    * Purpose: Calls the backend endpoint (HTTP GET)
127    * **************************************** */
128   getAllProductsByCuisine(desiredCuisine:string):Observable<Product[]>{
129     return this.http.get<Product[]>(`${this.restUrl}/products/cuisine/${desiredCuisine}`);
130   }
131
132 }//end class
```

Conclusion

Food Box Application

For the Next Release

Frontend

- Angular
  - Implement Route Guards
  - Make sure event listeners are closed
  - Fix user state persistence when the application is reloaded
- CSS
  - Fix alignment issue for the navigation bar

Backend

- Implement Spring Security
  - User tokens
  - Role Based Authentication
  - Email Validation
- Fix the initialization & data population scripts for MySQL
- Implement a Junit Test Suite
- Fix backend bugs:
  - When adding/removing products, the product counts in the Tags & Categories tables are not being updated properly (the front end logic is fine)

REFERENCES

Technical Books