CPAN 228 – ONB

WEB APPLICATION DEVELOPMENT

Final Project

BARBARA TOSETTO
N01535342

HUMBER COLLEGE
JULY 2024

INTRODUCTION

This report provides a comprehensive overview of the software architecture implemented in the final project for Web Application Development , which revolves around a web-based restaurant management system. The system is designed to streamline the administration of a restaurant's operations, from managing the menu to handling user authentication and security. The system's backend is developed using Java, with Spring Boot as the primary framework facilitating both the development and the orchestration of the business logic.

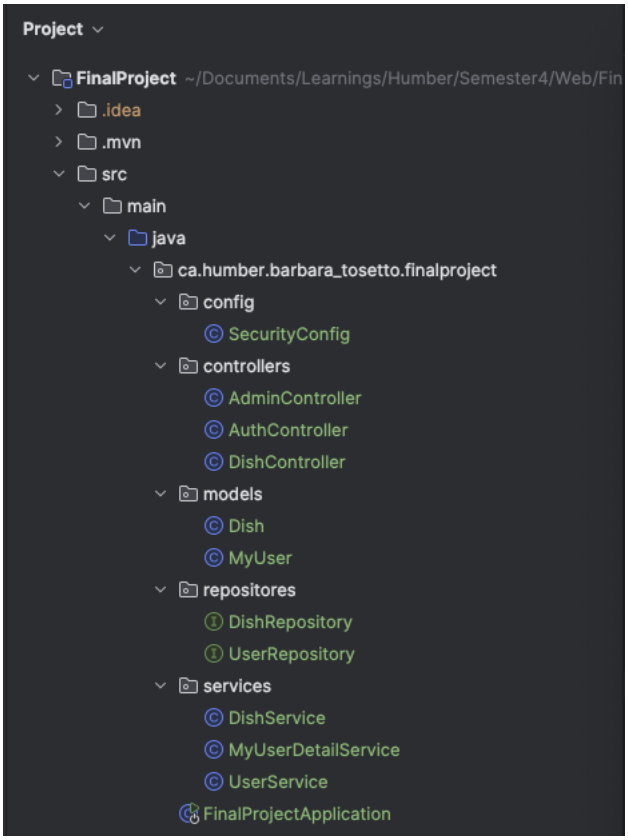
STEP 1

Create a new project on Eclipse with the dependencies required below:

- Thymeleaf
- Spring Data JDBC
- Spring Data JPA
- Spring Web
- H2 Database
- Lombok
- Spring DevTools
- Springboot Starter Web

STEP 2 – Application Architecture

Organizing a Spring Boot application into directories ensures that the application is scalable, maintainable, and robust. Each directory has a specific role that supports the application's structure, promoting clean architecture principles where business logic, data access, and configuration settings are cleanly separated.
This approach not only aids in development but also simplifies troubleshooting and enhances the collaborative aspects of development by allowing team members to focus on specific aspects of the application without interference.
For this project the directories are: Config, Controllers, Models, Repositories and Services, as follow:

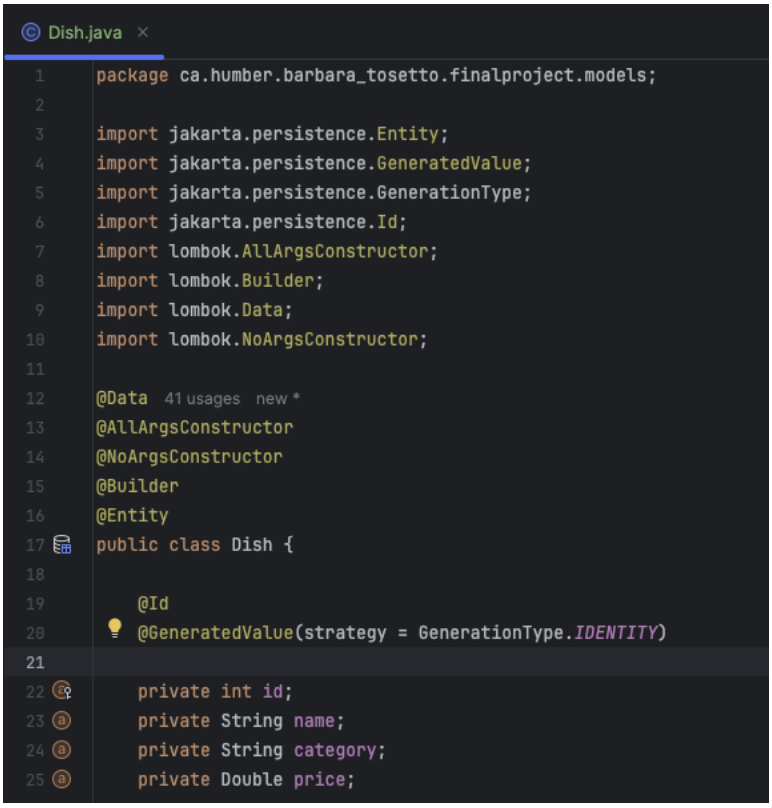


STEP 3 - Models

The `models` package contains entity definitions used within the application, such as `Dish` and `MyUser`. These models are annotated with JPA (Java Persistence API) annotations, allowing them to be easily stored and retrieved from a relational database.
Both the `Dish` and `MyUser` classes are fundamental components of this application's data layer, interacting directly with the database and serving as the primary method of data transport across the application. They ensure that data handling is streamlined and consistent with the application's object-oriented design.
Furthermore, their structured format, enforced through JPA annotations, enhances data integrity and interaction with the Spring Data JPA framework.
This structure supports efficient data operations and integration with other components like services and controllers, ultimately contributing to a robust and maintainable application architecture.

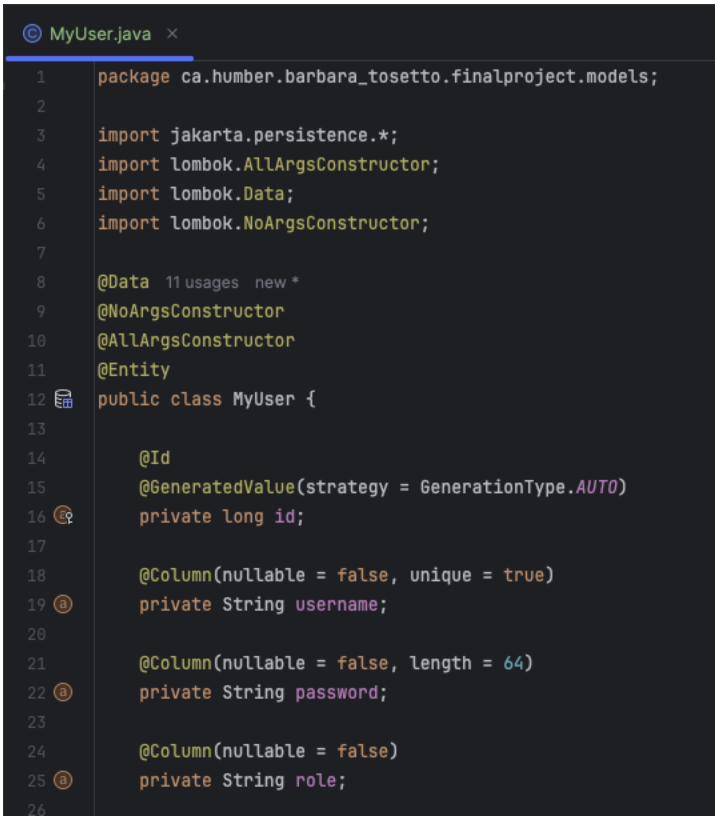
Dish Class

The `Dish class` represents a dish that can be ordered in the restaurant. The attributes are: id, name, category and price.



MyUser Class

The MyUser class represents a user of the system, storing credentials and roles necessary for authentication and authorization. The attributes are id, username, password and role.



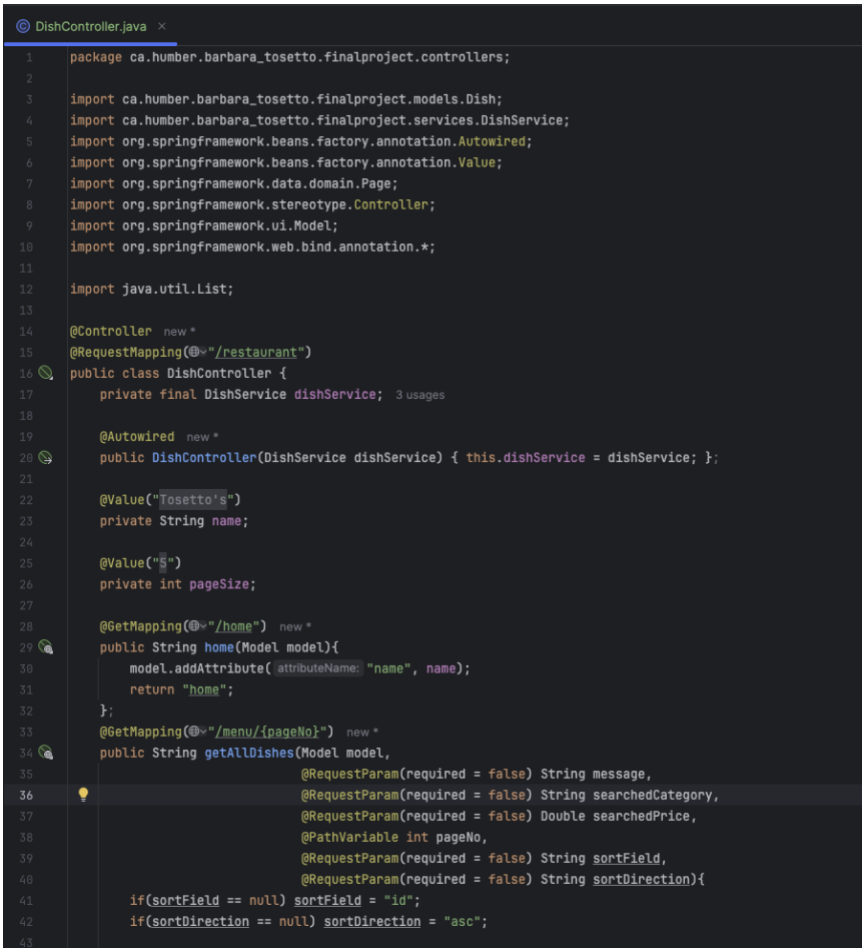
STEP 4 - Controllers

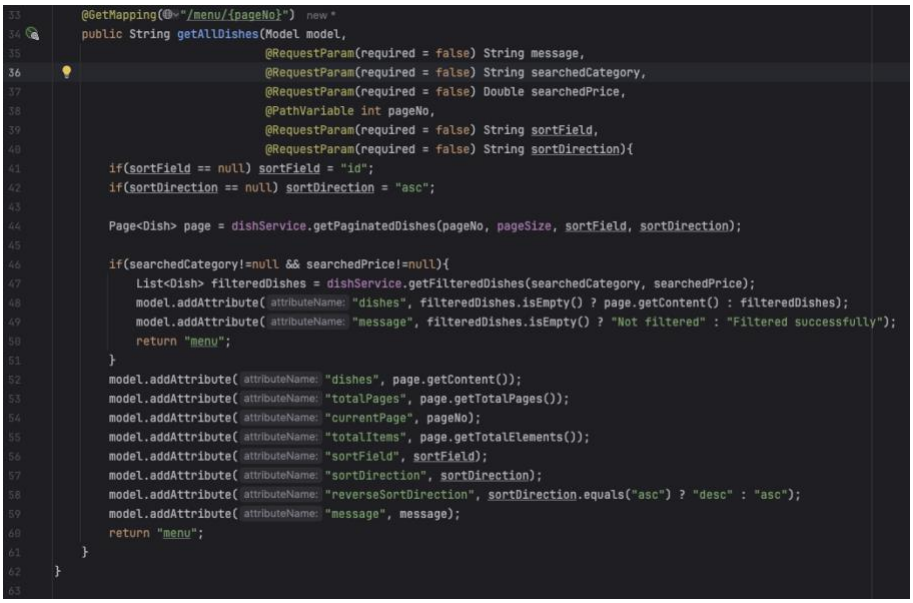
Controller Classes

The `controllers` package includes classes that handle incoming HTTP requests and dispatch responses. The controllers such as `AdminController`, `AuthController`, and `DishController` provide endpoints for tasks such as adding, deleting, and updating dishes, as well as managing user authentication and registration processes.

DishController Class

The `DishController` class in this application plays a crucial role in handling web requests related to dish management. This controller facilitates operations such as displaying dishes, adding new dishes, updating existing dishes, and deleting dishes. Let's delve into the functionalities, methods, and routing handled by `DishController`.





```
@GetMapping("/menu/{pageNo}")  new *
public String getAllDishes(Model model,
                           @RequestParam(required = false) String message,
                           @RequestParam(required = false) String searchedCategory,
                           @RequestParam(required = false) Double searchedPrice,
                           @PathVariable int pageNo,
                           @RequestParam(required = false) String sortField,
                           @RequestParam(required = false) String sortDirection){
    if(sortField == null) sortField = "id";
    if(sortDirection == null) sortDirection = "asc";

    Page<Dish> page = dishService.getPaginatedDishes(pageNo, pageSize, sortField, sortDirection);

    if(searchedCategory!=null && searchedPrice!=null){
        List<Dish> filteredDishes = dishService.getFilteredDishes(searchedCategory, searchedPrice);
        model.addAttribute( attributeName: "dishes", filteredDishes.isEmpty() ? page.getContent() : filteredDishes);
        model.addAttribute( attributeName: "message", filteredDishes.isEmpty() ? "Not filtered" : "Filtered successfully");
        return "menu";
    }
    model.addAttribute( attributeName: "dishes", page.getContent());
    model.addAttribute( attributeName: "totalPages", page.getTotalPages());
    model.addAttribute( attributeName: "currentPage", pageNo);
    model.addAttribute( attributeName: "totalItems", page.getTotalElements());
    model.addAttribute( attributeName: "sortField", sortField);
    model.addAttribute( attributeName: "sortDirection", sortDirection);
    model.addAttribute( attributeName: "reverseSortDirection", sortDirection.equals("asc") ? "desc" : "asc");
    model.addAttribute( attributeName: "message", message);
    return "menu";
}
}
```

AuthController Class

The `AuthController` class in this Spring Boot application is central to managing authentication and authorization functionalities. This controller ensures users can log in, log out, and register, providing a secure and user-friendly way to handle access to various parts of the application.

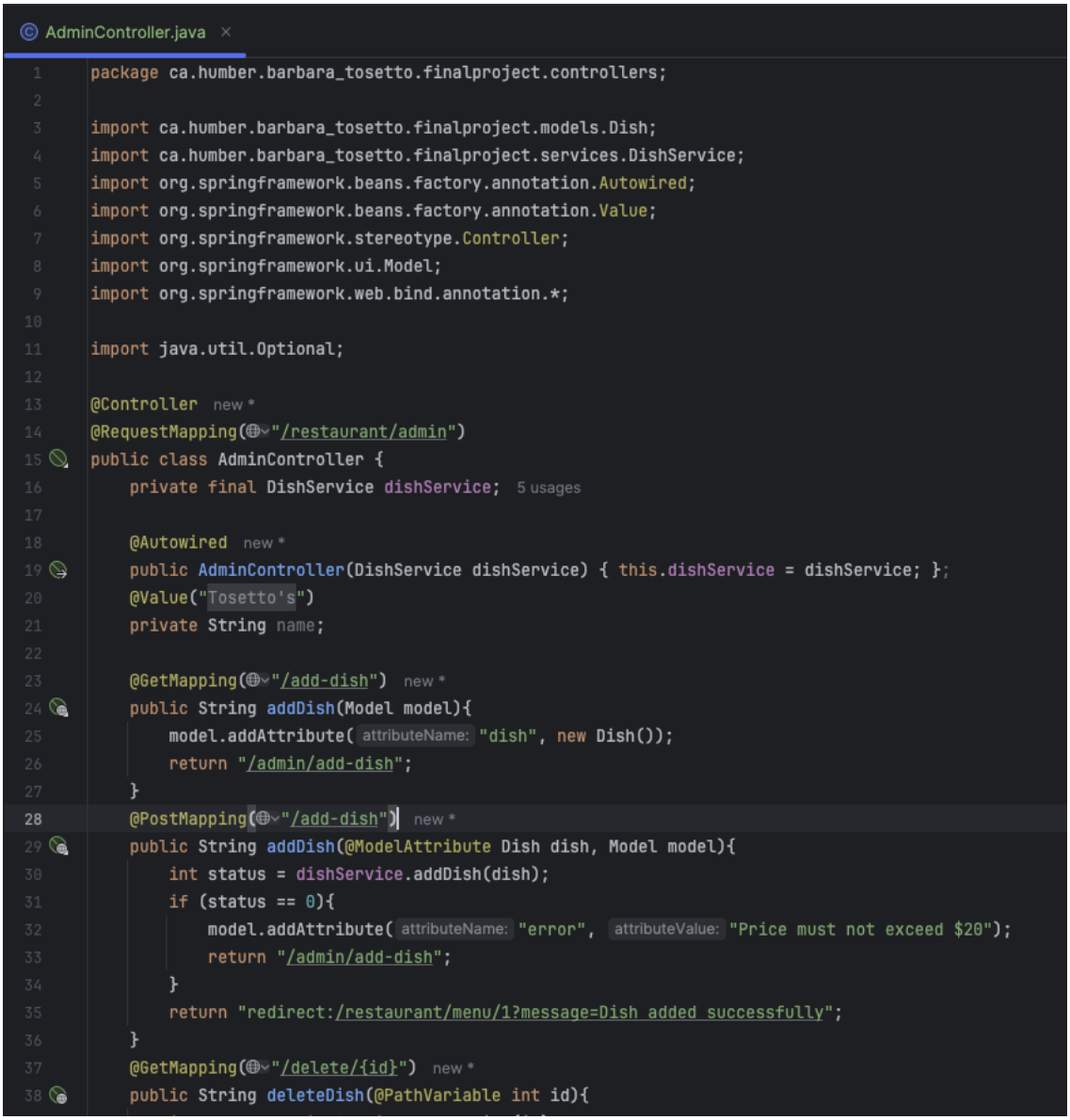
AuthController.java

```java
package ca.humber.barbara_tosetto.finalproject.controllers;

import ca.humber.barbara_tosetto.finalproject.models.MyUser;
import ca.humber.barbara_tosetto.finalproject.services.UserService;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller new *
public class AuthController implements org.springframework.boot.web.servlet.error.ErrorController {

    @Value("${restaurant.name}")
    private String name;
    private UserService userService; 2 usages

    public AuthController(UserService userService){ new *
        this.userService = userService;
    }

    @GetMapping("/error") new *
    public String handleError(){
        return "auth/error";
    }
    @GetMapping("/login") new *
    public String login(Model model, @RequestParam(required = false) String message){
        model.addAttribute( attributeName: "message", message);
        model.addAttribute( attributeName: "restaurantName", name);
        return "auth/login";
    }
    @GetMapping("/logout") new *
    public String logout(HttpServletRequest req,
```

```java
    }
    @GetMapping("/logout") new *
    public String logout(HttpServletRequest req,
                         HttpServletResponse res,
                         Authentication auth){
        new SecurityContextLogoutHandler().logout(req, res, auth);
        return "redirect:/login?message=You have been logged out";
    }
    @GetMapping("/register") new *
    public String register(Model model, @RequestParam(required = false) String message){
        model.addAttribute( attributeName: "user", new MyUser());
        model.addAttribute( attributeName: "message", message);
        return "auth/register";
    }
    @PostMapping("/register") new *
    public String register(@ModelAttribute MyUser user){
        int saveUserCode = userService.createUser(user);
        if(saveUserCode == 0){
            return "redirect:/register?message=User already exists";
        };
        return "redirect:/login?message=Registration successful";
    }
}
```

AdminController Class

The `AdminController` in this Spring Boot project is a crucial component that facilitates the administrative management of the restaurant's offerings, specifically focusing on the operations related to dish management. This controller provides backend functionality for adding, updating, and deleting dishes, essential tasks for maintaining the restaurant's menu



```java
    @GetMapping("/delete/{id}")  new *
    public String deleteDish(@PathVariable int id){
        int status = dishService.deleteDish(id);
        if (status > 0){
            return "redirect:/restaurant/menu/1?message=Dish deleted successfully";
        }
        return "redirect:/restaurant/menu/1?message=Dish does not exist";
    }
    @GetMapping("/update/{id}")  new *
    public String updateDish(Model model, @PathVariable int id){
        Optional<Dish> dishToUpdate = dishService.getDishById(id);
        model.addAttribute( attributeName: "dish", dishToUpdate.orElse( other: null));
        return "/admin/add-dish";
    }
    @PostMapping("/update-dish")  new *
    public String updateDish(@ModelAttribute Dish dish){
        dishService.updatedDish(dish);
        return "redirect:/restaurant/menu/1?message=Dish updated";
    }
}
```

STEP 5 – Services Class

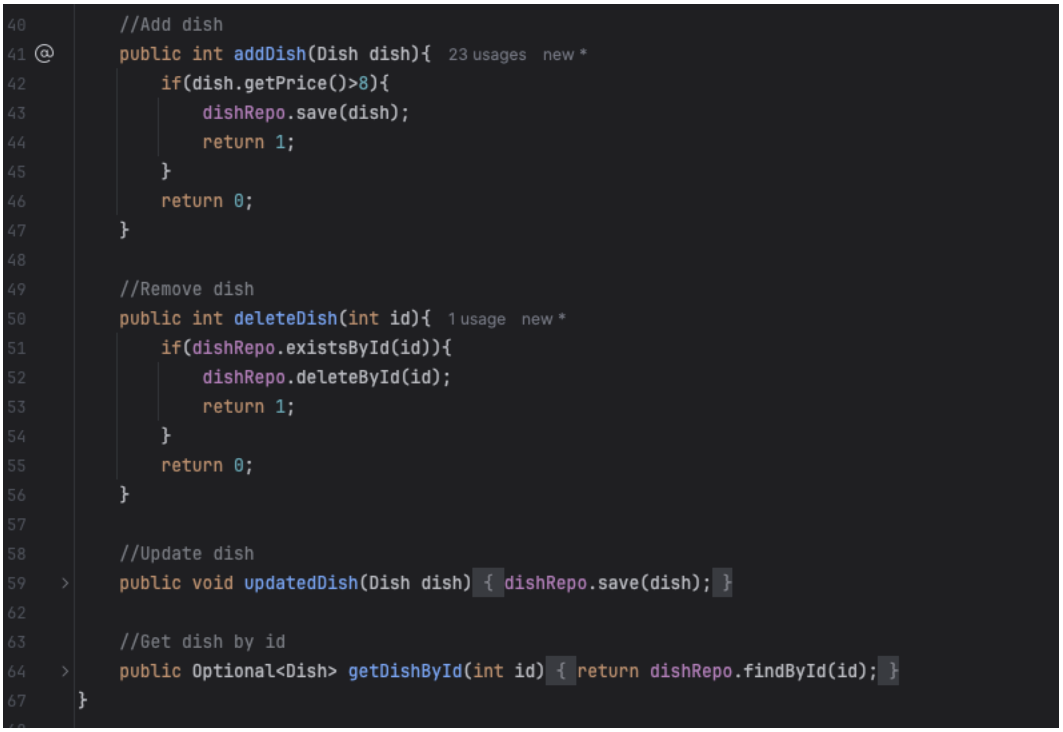
The three service classes—`DishService`, `UserService`, and `MyUserDetailsService`—serve critical roles in encapsulating the business logic, managing data interactions, and integrating with the security framework of the application.
Each of these services is designed to handle specific functionalities that ensure the system operates efficiently, securely, and remains maintainable.

DishService Class

The `DishService` class acts as a core component of the service layer, dealing directly with the business logic related to dish management. This class interfaces with the `DishRepository` , which will be explained later, to execute data access and manipulation tasks, ensuring that business rules and logic are correctly applied before any interactions with the database.

DishService.java

```java
package ca.humber.barbara_tosetto.finalproject.services;

import ca.humber.barbara_tosetto.finalproject.models.Dish;
import ca.humber.barbara_tosetto.finalproject.repositores.DishRepository;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import org.springframework.beans.factory.annotation.Autowired;
import java.util.List;
import java.util.Optional;

@Service 9 usages new *
public class DishService {
    private final DishRepository dishRepo; 9 usages

    @Autowired new *
    public DishService(DishRepository dishRepo) { this.dishRepo = dishRepo; }

    //Get all dishes
    public List<Dish> getAllDishes() { return dishRepo.findAll(); }

    //Get filtered dishes
    public List<Dish> getFilteredDishes(String searchedCategory, Double searchedPrice){ 1 usage new *
        return dishRepo.findByCategoryAndPrice(searchedCategory, searchedPrice);
    }

    //Get paginated dishes
    public Page<Dish> getPaginatedDishes(int pageNo, int pageSize, String sortField, String sortDirection){ 1 usage new *
        Sort sort = sortDirection.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortField).ascending() : Sort.by(sortField).descending();
        Pageable pageable = PageRequest.of( pageNumber: pageNo-1, pageSize, sort);
        return dishRepo.findAll(pageable);
    }
```



```
    //Add dish
    public int addDish(Dish dish){  23 usages  new *
        if(dish.getPrice()>8){
            dishRepo.save(dish);
            return 1;
        }
        return 0;
    }

    //Remove dish
    public int deleteDish(int id){  1 usage  new *
        if(dishRepo.existsById(id)){
            dishRepo.deleteById(id);
            return 1;
        }
        return 0;
    }

    //Update dish
    public void updatedDish(Dish dish) { dishRepo.save(dish); }

    //Get dish by id
    public Optional<Dish> getDishById(int id) { return dishRepo.findById(id); }
}
```

MyUserDetailService Class

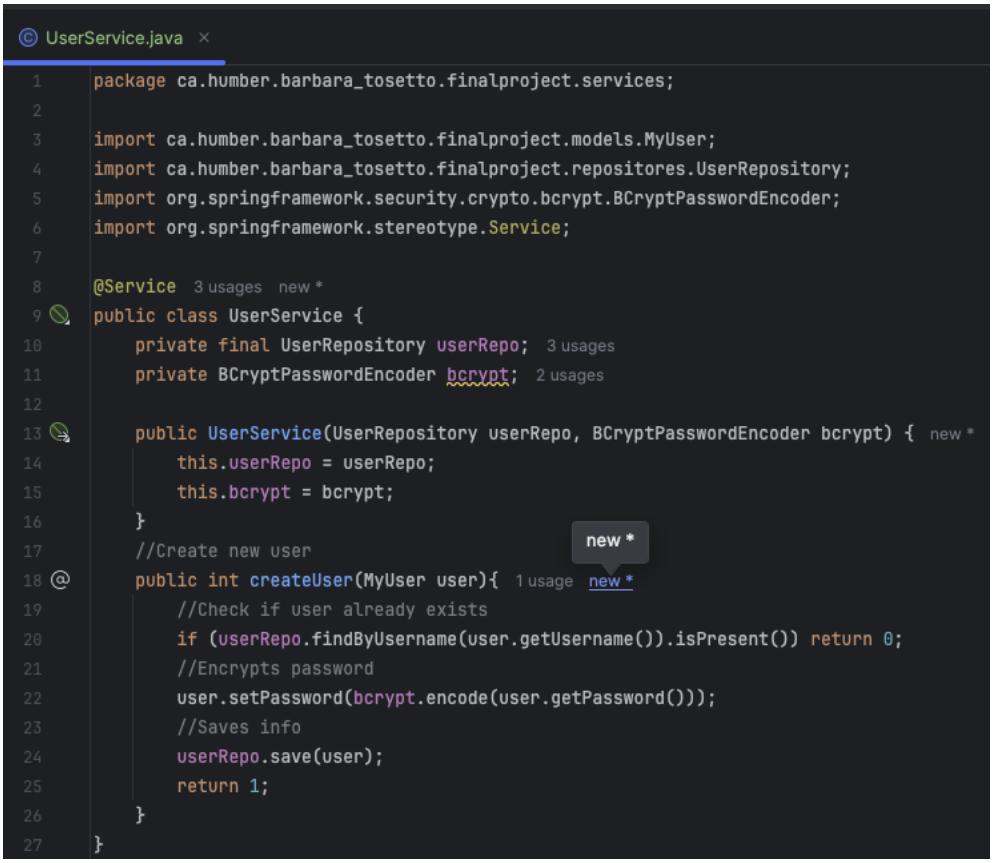
The `MyUserDetailService` in this application is essential for integrating user authentication with Spring Security. It implements the `UserDetailsService` interface, specifically the `loadUserByUsername` method. This method retrieves user details from the database using the username, constructs a `UserDetails` object with credentials and roles, and then hands it off to Spring Security for authentication. This ensures that user logins are secure and managed in line with the stored credentials and roles.

MyUserDetailService.java

```java
package ca.humber.barbara_tosetto.finalproject.services;

import ca.humber.barbara_tosetto.finalproject.models.MyUser;
import ca.humber.barbara_tosetto.finalproject.repositores.UserRepository;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service  new *
public class MyUserDetailService implements UserDetailsService {

    private UserRepository userRepo;  2 usages

    public MyUserDetailService(UserRepository userRepo) { this.userRepo = userRepo; }

    @Override  new *
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Optional<MyUser> userOp = userRepo.findByUsername(username);

        if (userOp.isPresent()){
            MyUser user = userOp.get();
            return User.builder()
                    .username(user.getUsername())
                    .password(user.getPassword())
                    .roles(user.getRole())
                    .build();
        } else{
            throw new UsernameNotFoundException("User not found");
        }
    }
}
```

UserService Class

The `UserService` class is pivotal for managing user-related operations such as registration, password encryption, and the persistence of user data. It interacts with the `UserRepository` to perform CRUD operations on user data. When a new user registers, the `UserService` checks for the uniqueness of the username, encrypts the password using BCrypt for security, and then stores the user details in the database. This service centralizes user management tasks, ensuring data integrity and enforcing security measures crucial for protecting user information.

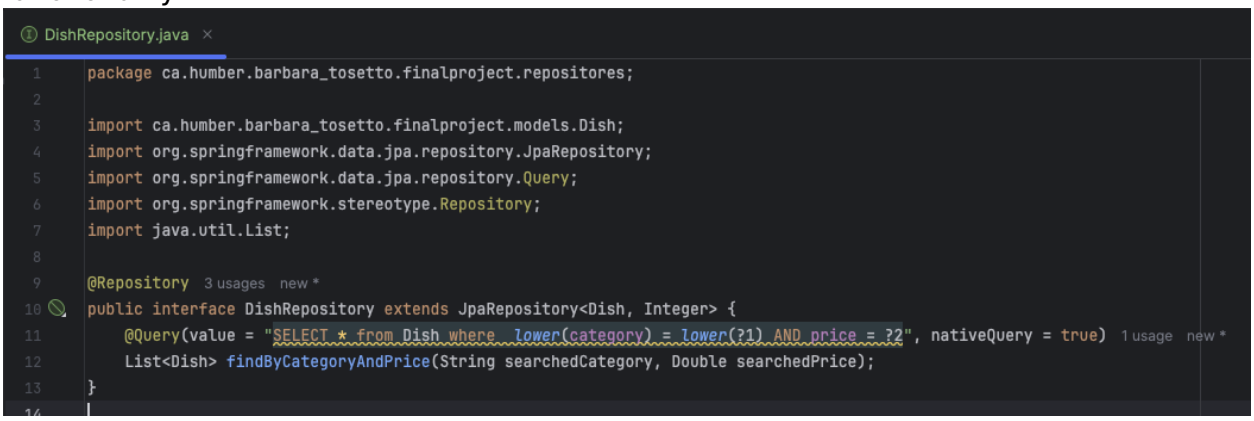

UserService.java
package ca.humber.barbara_tosetto.finalproject.services;
import ca.humber.barbara_tosetto.finalproject.models.MyUser;
import ca.humber.barbara_tosetto.finalproject.repositores.UserRepository;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
@Service 3 usages new *
public class UserService {
private final UserRepository userRepo; 3 usages
private BCryptPasswordEncoder bcrypt; 2 usages
public UserService(UserRepository userRepo, BCryptPasswordEncoder bcrypt) { new *
this.userRepo = userRepo;
this.bcrypt = bcrypt;
}
//Create new user
public int createUser(MyUser user){ 1 usage new *
new *
//Check if user already exists
if (userRepo.findByUsername(user.getUsername()).isPresent()) return 0;
//Encrypts password
user.setPassword(bcrypt.encode(user.getPassword()));
//Saves info
userRepo.save(user);
return 1;
}
}

STEP 6 - Repositories

The repositories in this Spring Boot application, `DishRepository` and `UserRepository`, act as essential data access layers, extending `JpaRepository` to facilitate CRUD operations and specialized queries like `findByUsername` and `findByCategoryAndPrice`. These repositories streamline database interactions, simplifying the management of dish and user data with robust, easy-to-maintain methods that support the application's core functionalities.

DishRepository

The `DishRepository` acts as the data access layer for managing the `Dish` entity, extending `JpaRepository` to provide comprehensive CRUD operations and custom queries like `findByCategoryAndPrice`. This repository simplifies complex database interactions, enabling efficient and targeted data retrieval and management, crucial for the application's functionality.



UserRepository

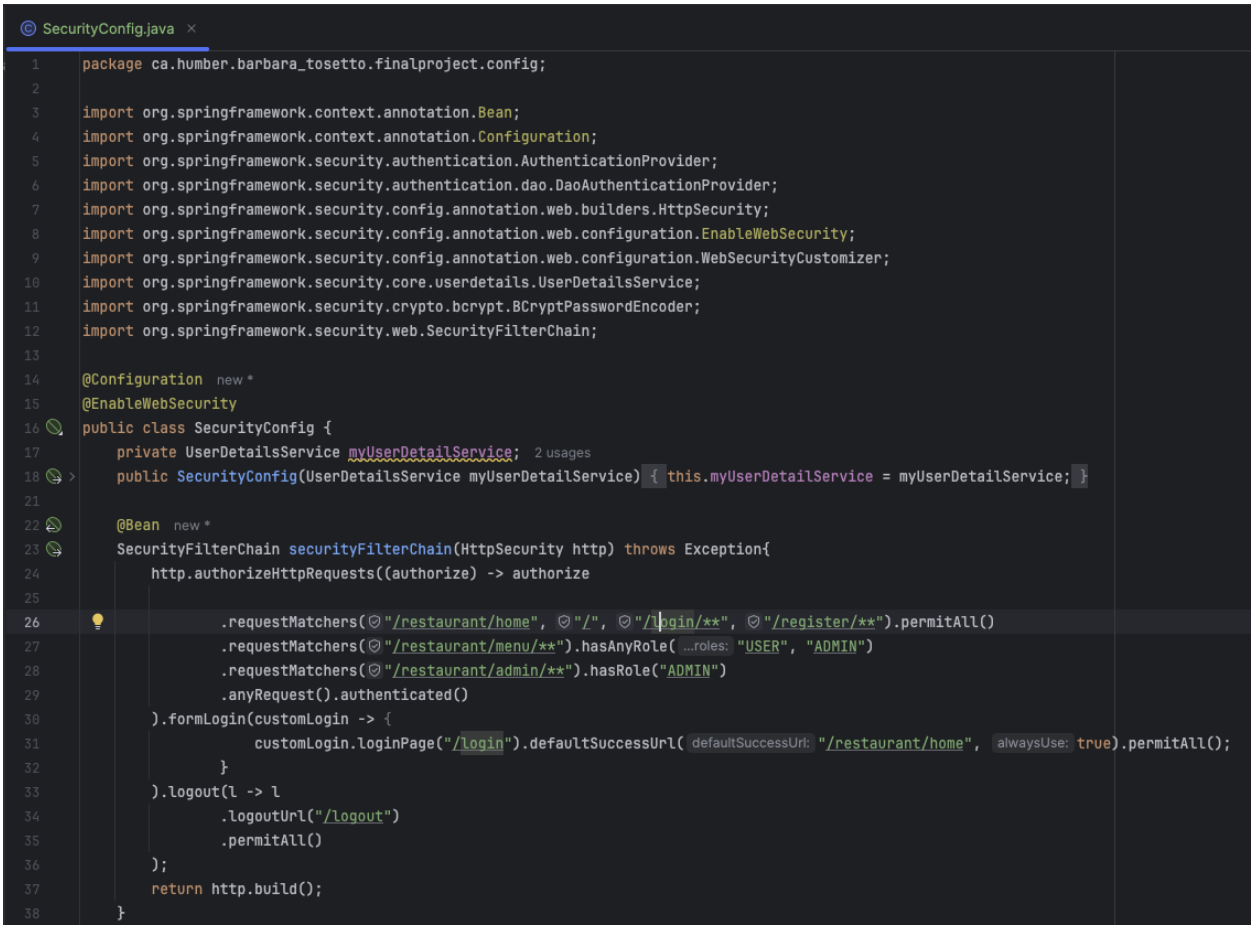
The `UserRepository` is crucial for accessing and managing user data stored in the database. It extends `JpaRepository`, offering built-in methods for CRUD operations, and includes a custom query method, `findByUsername`, which allows for fetching user details based on their username. This repository simplifies user data management, facilitating efficient and secure user authentication and profile management within the application.

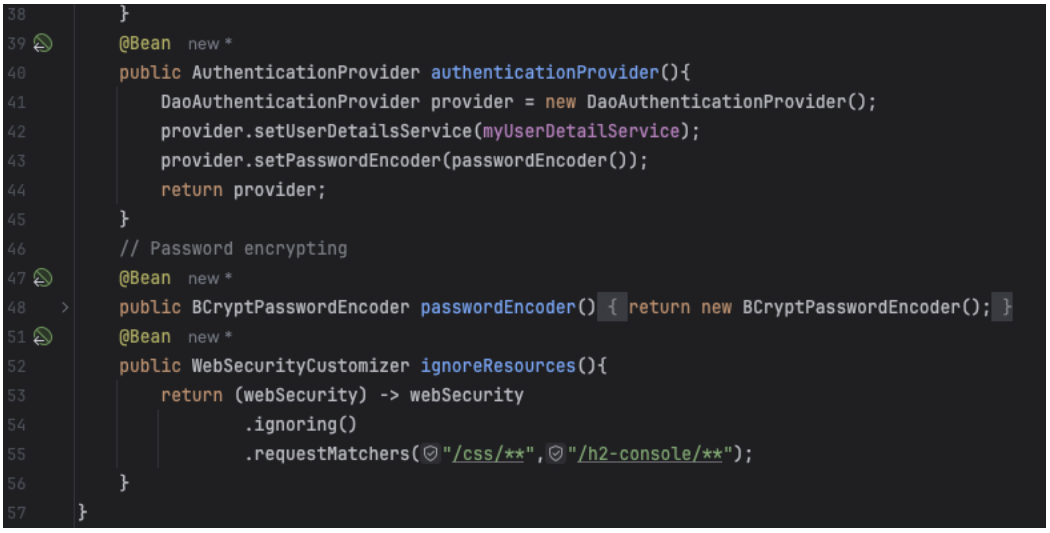
UserRepository.java

```java
package ca.humber.barbara_tosetto.finalproject.repositores;

import ca.humber.barbara_tosetto.finalproject.models.MyUser;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface UserRepository extends JpaRepository<MyUser, Long> {  6 usages  new *
    public Optional<MyUser> findByUsername(String username);  2 usages  new *

}
```

STEP 7 - Security

The `SecurityConfig` class configures security settings to manage authentication and authorization across the application. It defines rules for securing endpoints, specifies URL access permissions based on user roles, and sets up form-based authentication and logout functionalities. Utilizing Spring Security's `HttpSecurity`, the class ensures that only authenticated users can access certain resources, enhancing the overall security posture of the application. This class is central to enforcing security measures that protect sensitive data and user interactions within the system.

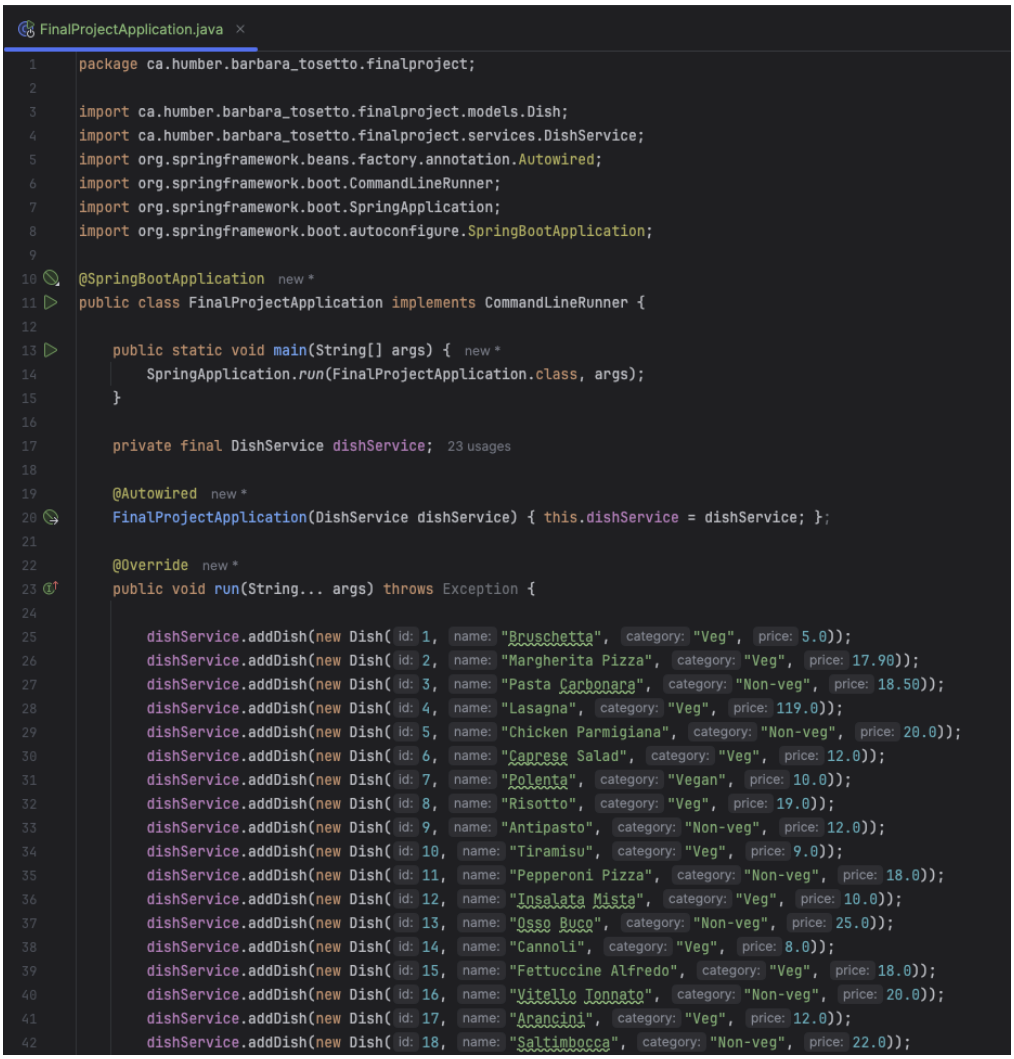




```
    }
    @Bean  new *
    public AuthenticationProvider authenticationProvider(){
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(myUserDetailService);
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }
    // Password encrypting
    @Bean  new *
    public BCryptPasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
    @Bean  new *
    public WebSecurityCustomizer ignoreResources(){
        return (webSecurity) -> webSecurity
                .ignoring()
                .requestMatchers("/css/**","/h2-console/**");
    }
}
```

STEP 8 – Set up and configuration

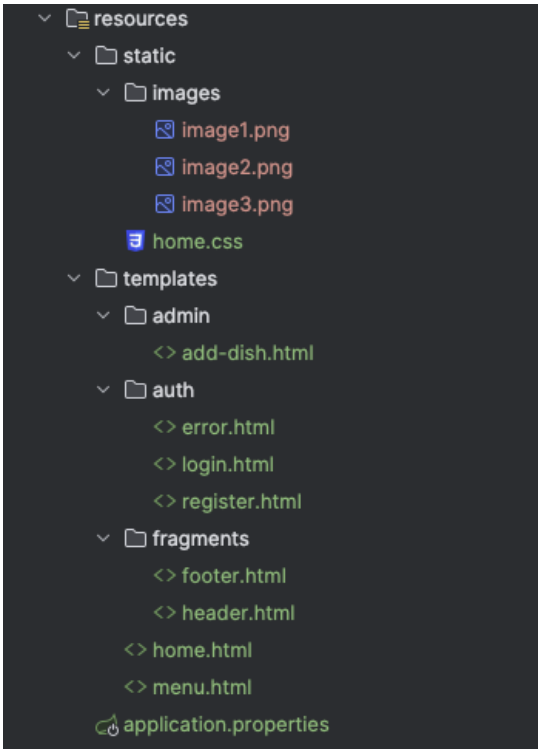
The `FinalProjectApplication` class in this Spring Boot project is the main entry point, configured with `@SpringBootApplication` to facilitate auto-configuration and component scanning. It implements `CommandLineRunner` to execute initial data loading after the application starts. Within its `run` method, it uses `DishService` to pre-load the database with a set of dish entities. This automatic data population is crucial for initializing the application with default values for immediate functionality upon startup, enhancing testing and development workflows.



```java
package ca.humber.barbara_tosetto.finalproject;

import ca.humber.barbara_tosetto.finalproject.models.Dish;
import ca.humber.barbara_tosetto.finalproject.services.DishService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication  new *
public class FinalProjectApplication implements CommandLineRunner {

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

    private final DishService dishService;  23 usages

    @Autowired  new *
    FinalProjectApplication(DishService dishService) { this.dishService = dishService; };

    @Override  new *
    public void run(String... args) throws Exception {

        dishService.addDish(new Dish( id: 1,  name: "Bruschetta",  category: "Veg",  price: 5.0));
        dishService.addDish(new Dish( id: 2,  name: "Margherita Pizza",  category: "Veg",  price: 17.90));
        dishService.addDish(new Dish( id: 3,  name: "Pasta Carbonara",  category: "Non-veg",  price: 18.50));
        dishService.addDish(new Dish( id: 4,  name: "Lasagna",  category: "Veg",  price: 119.0));
        dishService.addDish(new Dish( id: 5,  name: "Chicken Parmigiana",  category: "Non-veg",  price: 20.0));
        dishService.addDish(new Dish( id: 6,  name: "Caprese Salad",  category: "Veg",  price: 12.0));
        dishService.addDish(new Dish( id: 7,  name: "Polenta",  category: "Vegan",  price: 10.0));
        dishService.addDish(new Dish( id: 8,  name: "Risotto",  category: "Veg",  price: 19.0));
        dishService.addDish(new Dish( id: 9,  name: "Antipasto",  category: "Non-veg",  price: 12.0));
        dishService.addDish(new Dish( id: 10,  name: "Tiramisu",  category: "Veg",  price: 9.0));
        dishService.addDish(new Dish( id: 11,  name: "Pepperoni Pizza",  category: "Non-veg",  price: 18.0));
        dishService.addDish(new Dish( id: 12,  name: "Insalata Mista",  category: "Veg",  price: 10.0));
        dishService.addDish(new Dish( id: 13,  name: "Osso Buco",  category: "Non-veg",  price: 25.0));
        dishService.addDish(new Dish( id: 14,  name: "Cannoli",  category: "Veg",  price: 8.0));
        dishService.addDish(new Dish( id: 15,  name: "Fettuccine Alfredo",  category: "Veg",  price: 18.0));
        dishService.addDish(new Dish( id: 16,  name: "Vitello Tonnato",  category: "Non-veg",  price: 20.0));
        dishService.addDish(new Dish( id: 17,  name: "Arancini",  category: "Veg",  price: 12.0));
        dishService.addDish(new Dish( id: 18,  name: "Saltimbocca",  category: "Non-veg",  price: 22.0));
```

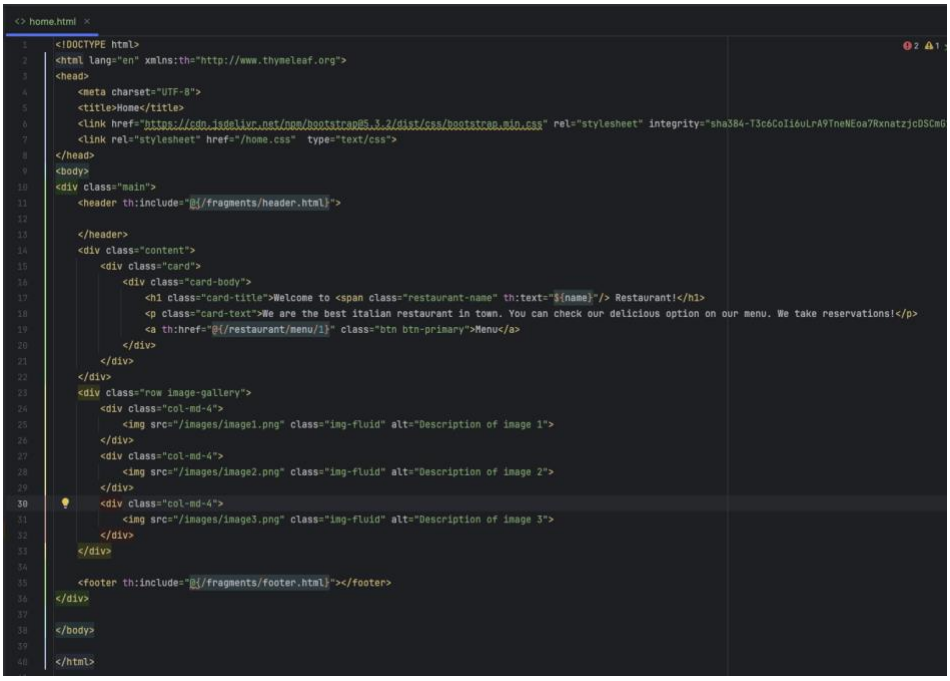
STEP 9 – Front-end Pages

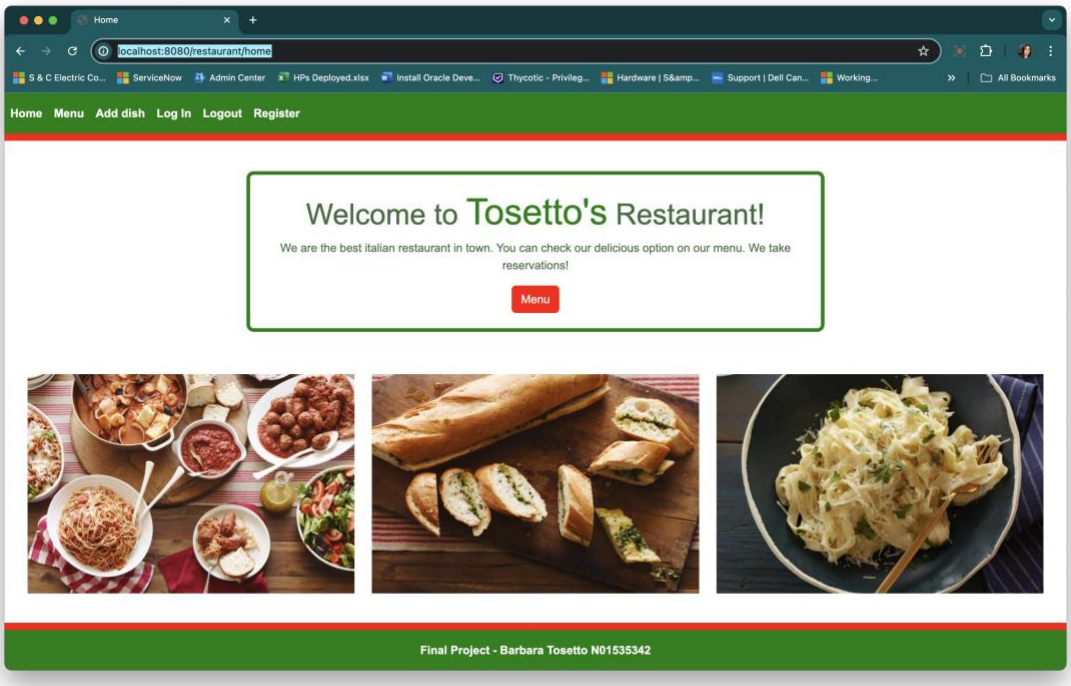
The front-end of this Spring Boot application is structured to provide a dynamic and user-friendly interface, primarily managed through Thymeleaf templates located in the `templates` directory. These templates are HTML files enriched with Thymeleaf syntax, allowing for server-side rendering of dynamic content.

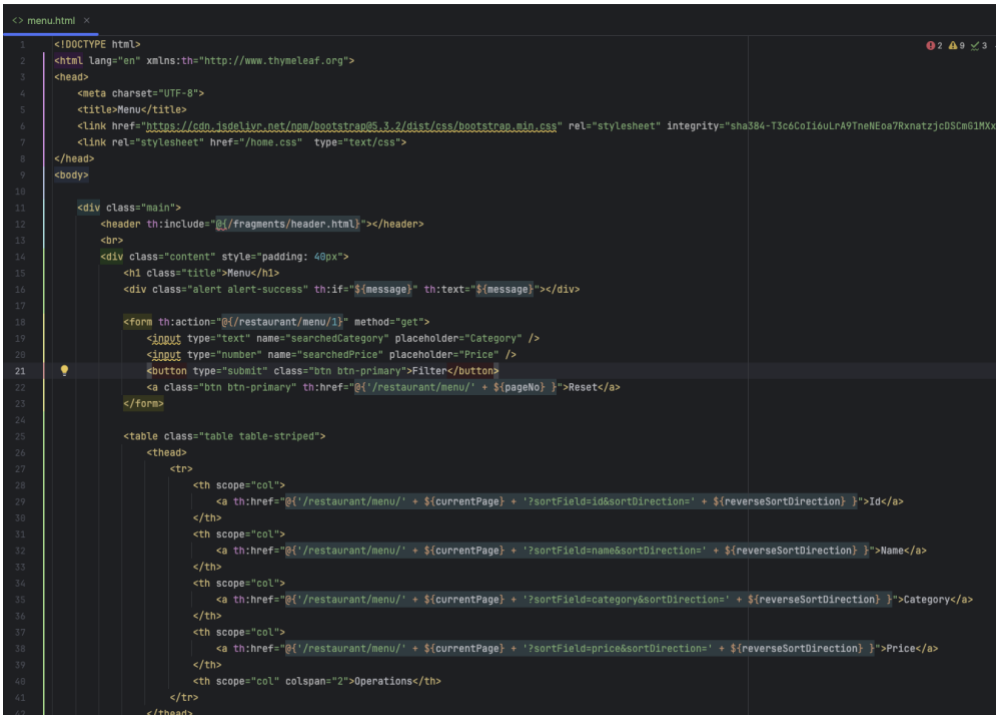


Main Templates

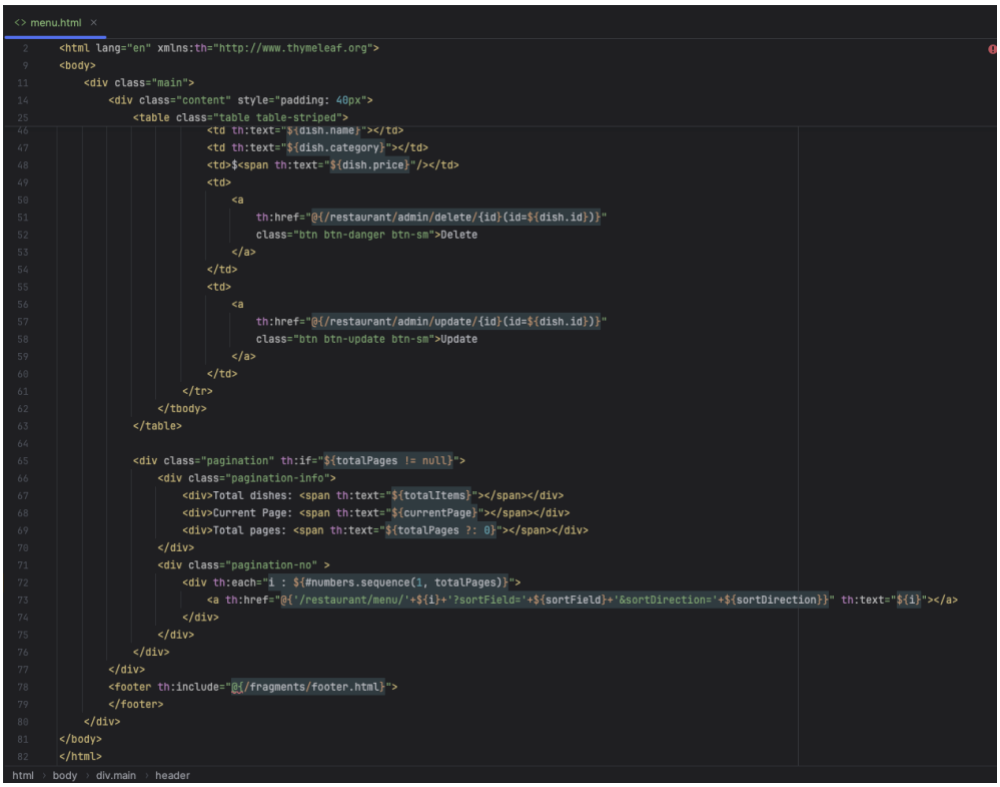
Including key pages like `home.html`, `menu.html`, and others, which serve as the primary interaction points for users, displaying content like menus and home page information.



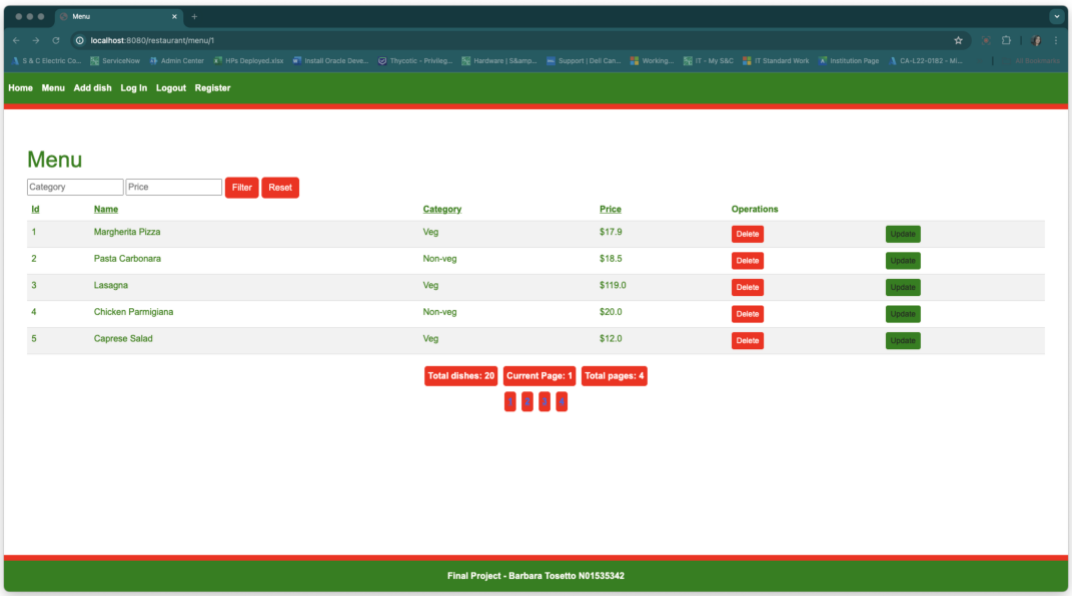




```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Menu</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXx
    <link rel="stylesheet" href="/home.css"  type="text/css">
</head>
<body>

    <div class="main">
        <header th:include="@{/fragments/header.html}"></header>
        <br>
        <div class="content" style="padding: 40px">
            <h1 class="title">Menu</h1>
            <div class="alert alert-success" th:if="${message}" th:text="${message}"></div>

            <form th:action="@{/restaurant/menu/1}" method="get">
                <input type="text" name="searchedCategory" placeholder="Category" />
                <input type="number" name="searchedPrice" placeholder="Price" />
                <button type="submit" class="btn btn-primary">Filter</button>
                <a class="btn btn-primary" th:href="@{'/restaurant/menu/' + ${pageNo} }">Reset</a>
            </form>

            <table class="table table-striped">
                <thead>
                    <tr>
                        <th scope="col">
                            <a th:href="@{'/restaurant/menu/' + ${currentPage} + '?sortField=id&sortDirection=' + ${reverseSortDirection} }">Id</a>
                        </th>
                        <th scope="col">
                            <a th:href="@{'/restaurant/menu/' + ${currentPage} + '?sortField=name&sortDirection=' + ${reverseSortDirection} }">Name</a>
                        </th>
                        <th scope="col">
                            <a th:href="@{'/restaurant/menu/' + ${currentPage} + '?sortField=category&sortDirection=' + ${reverseSortDirection} }">Category</a>
                        </th>
                        <th scope="col">
                            <a th:href="@{'/restaurant/menu/' + ${currentPage} + '?sortField=price&sortDirection=' + ${reverseSortDirection} }">Price</a>
                        </th>
                        <th scope="col" colspan="2">Operations</th>
                    </tr>
                </thead>
```

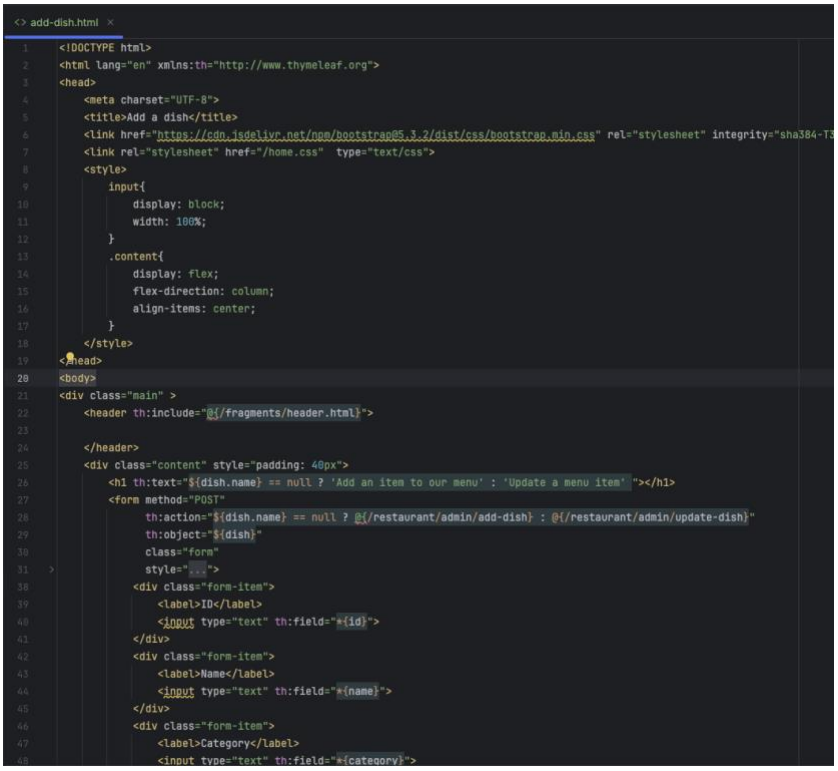


```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<body>
    <div class="main">
        <div class="content" style="padding: 40px">
            <table class="table table-striped">
                        <td th:text="${dish.name}"></td>
                        <td th:text="${dish.category}"></td>
                        <td>$<span th:text="${dish.price}"/></td>
                        <td>
                            <a
                                th:href="@{/restaurant/admin/delete/{id}(id=${dish.id})}"
                                class="btn btn-danger btn-sm">Delete
                            </a>
                        </td>
                        <td>
                            <a
                                th:href="@{/restaurant/admin/update/{id}(id=${dish.id})}"
                                class="btn btn-update btn-sm">Update
                            </a>
                        </td>
                    </tr>
                </tbody>
            </table>

            <div class="pagination" th:if="${totalPages != null}">
                <div class="pagination-info">
                    <div>Total dishes: <span th:text="${totalItems}"></span></div>
                    <div>Current Page: <span th:text="${currentPage}"></span></div>
                    <div>Total pages: <span th:text="${totalPages ?: 0}"></span></div>
                </div>
                <div class="pagination-no" >
                    <div th:each="i : ${#numbers.sequence(1, totalPages)}">
                        <a th:href="@{'/restaurant/menu/'+${i}+'?sortField='+${sortField}+'&sortDirection='+${sortDirection}}" th:text="${i}"></a>
                    </div>
                </div>
            </div>
        </div>
        <footer th:include="@{/fragments/footer.html}">
        </footer>
    </div>
</body>
</html>
```


Menu
localhost:8080/restaurant/menu/1
Home Menu Add dish Log In Logout Register
Menu
Category
Price
Filter
Reset
Id Name Category Price Operations
1 Margherita Pizza Veg $17.9 Delete Update
2 Pasta Carbonara Non-veg $18.5 Delete Update
3 Lasagna Veg $119.0 Delete Update
4 Chicken Parmigiana Non-veg $20.0 Delete Update
5 Caprese Salad Veg $12.0 Delete Update
Total dishes: 20
Current Page: 1
Total pages: 4
1 2 3 4
Final Project - Barbara Tosetto N01535342

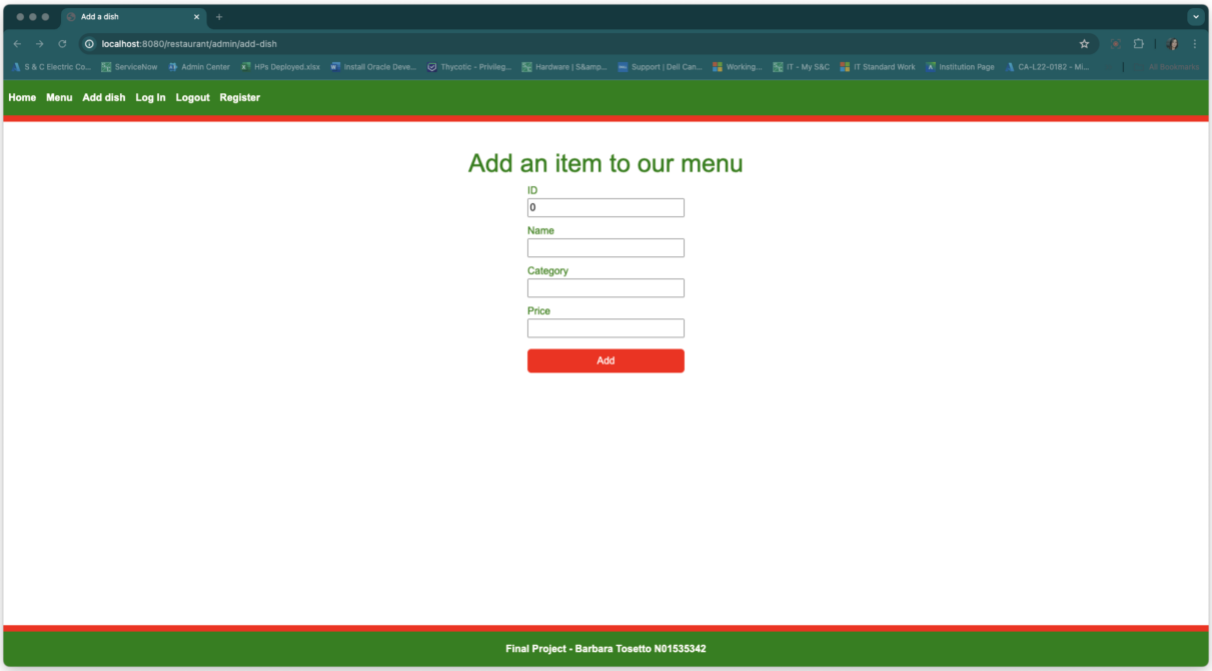
Admin Section

Contains templates such as `add-dish.html` under the `admin` folder, enabling administrative tasks like adding or updating dishes.



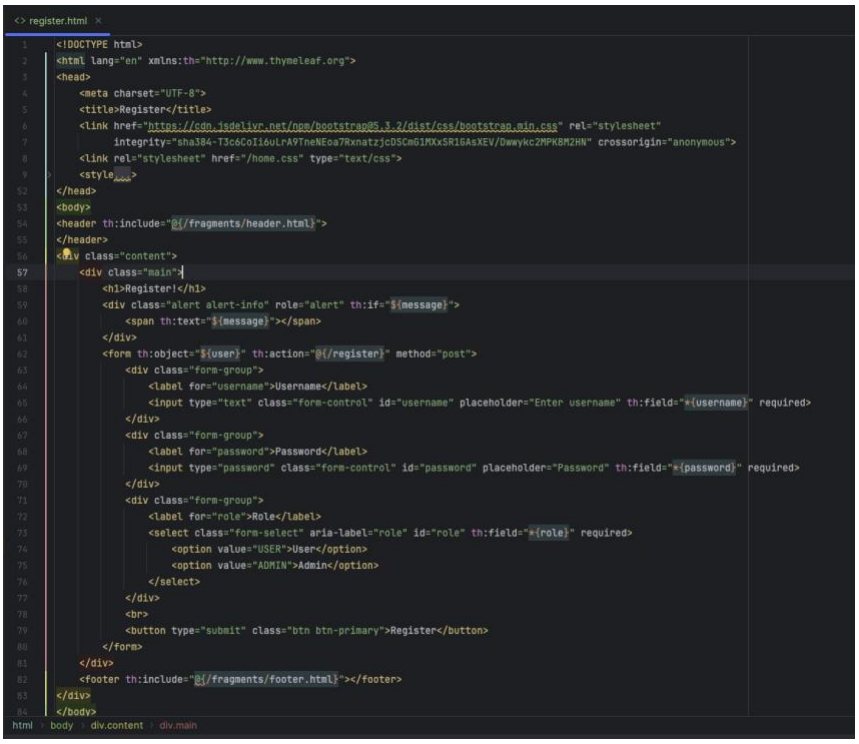
```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Add a dish</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3
    <link rel="stylesheet" href="/home.css"  type="text/css">
    <style>
        input{
            display: block;
            width: 100%;
        }
        .content{
            display: flex;
            flex-direction: column;
            align-items: center;
        }
    </style>
</head>
<body>
<div class="main" >
    <header th:include="@{/fragments/header.html}">

    </header>
    <div class="content" style="padding: 40px">
        <h1 th:text="${dish.name} == null ? 'Add an item to our menu' : 'Update a menu item' "></h1>
        <form method="POST"
              th:action="${dish.name} == null ? @{/restaurant/admin/add-dish} : @{/restaurant/admin/update-dish}"
              th:object="${dish}"
              class="form"
              style="...">
            <div class="form-item">
                <label>ID</label>
                <input type="text" th:field="*{id}">
            </div>
            <div class="form-item">
                <label>Name</label>
                <input type="text" th:field="*{name}">
            </div>
            <div class="form-item">
                <label>Category</label>
                <input type="text" th:field="*{category}">
```

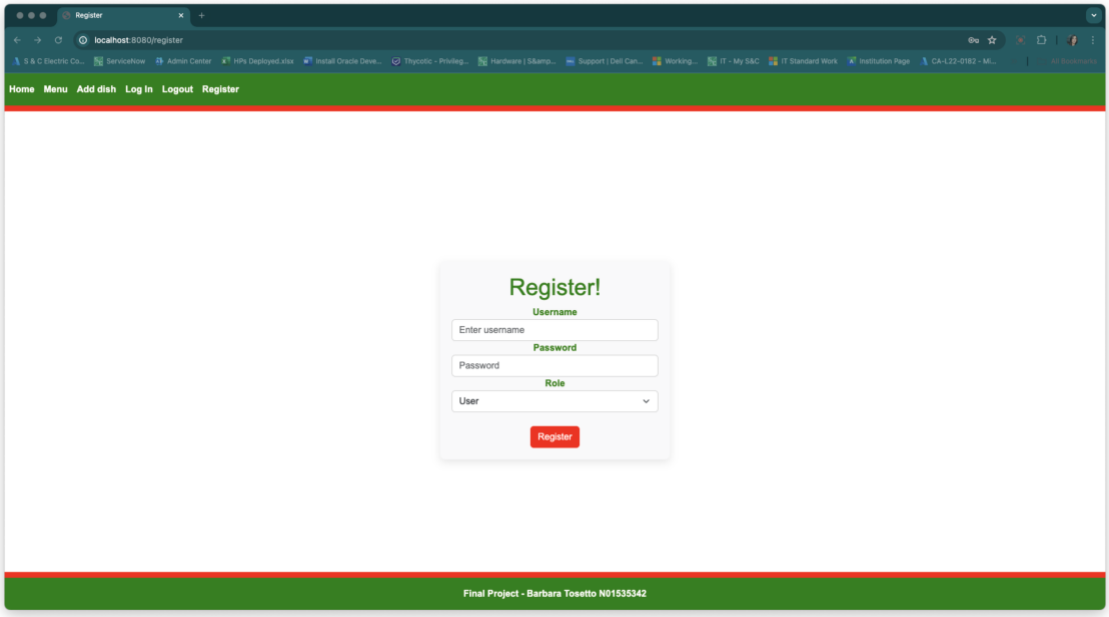
```
                <label>Category</label>
                <input type="text" th:field="*{category}">
            </div>
            <div class="form-item">
                <label>Price</label>
                <input type="number" th:field="*{price}">
            </div>
            <button type="submit" class="btn btn-primary mt-2"
                    th:text="${dish.name} == null ? 'Add':'Update'">
            </button>
        </form>
        <div th:if="${error}" class="alert alert-danger mt-3">
            <span th:text="${error}"></span>
        </div>
    </div>
    <footer th:include="@{/fragments/footer.html}">

    </footer>
</div>
</body>
</html>
```


Add a dish
localhost:8080/restaurant/admin/add-dish
Home Menu Add dish Log In Logout Register
Add an item to our menu
ID
0
Name
Category
Price
Add
Final Project - Barbara Tosetto N01535342

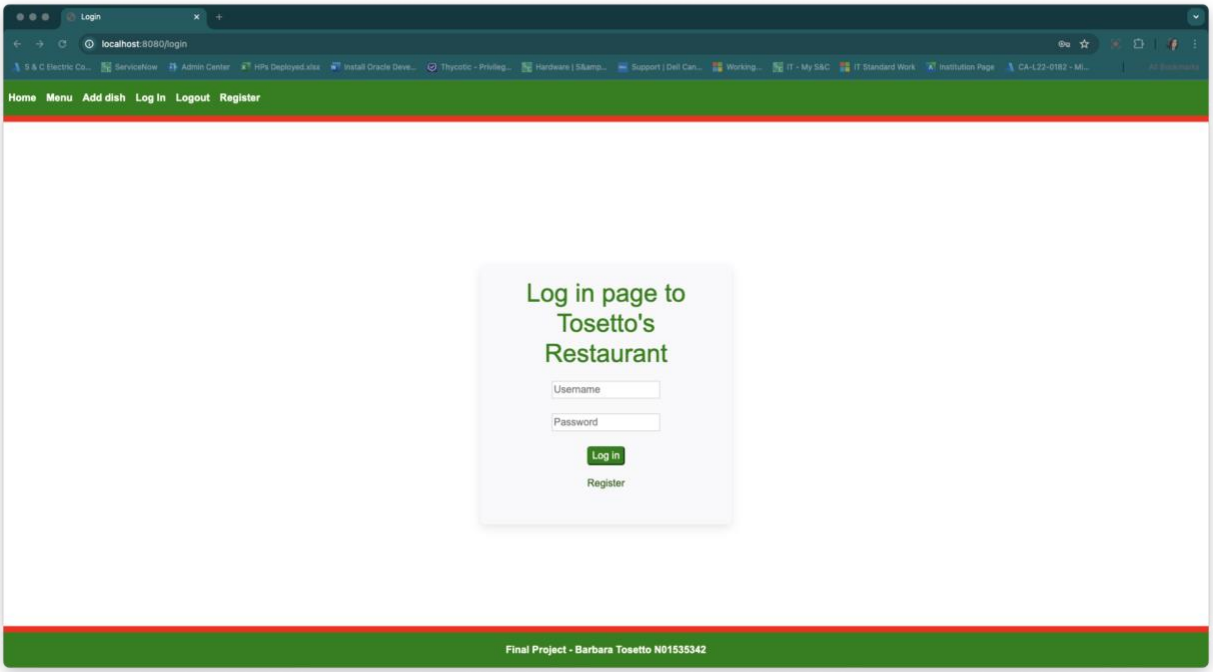
Authentication Pages

Found under the `auth` folder, with templates for login (`login.html`), registration (`register.html`), and error handling (`error.html`), facilitating user authentication and authorization processes.



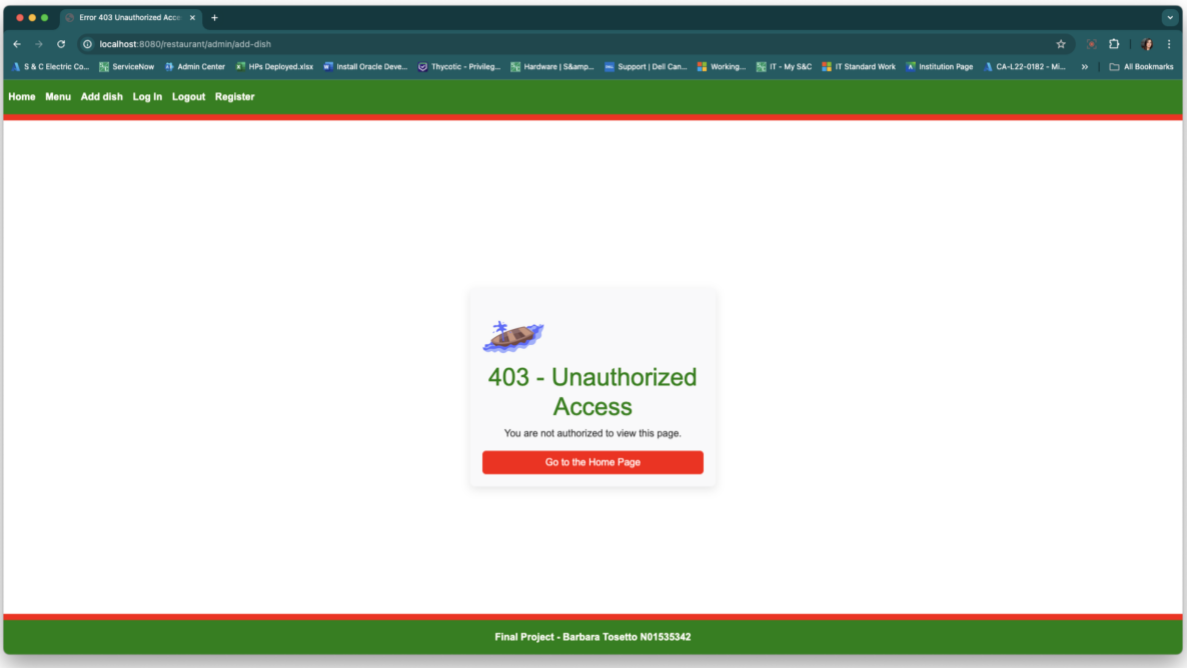


```
<> login.html ×
1   <!DOCTYPE html>
2   <html lang="en" xmlns:th="http://www.thymeleaf.org">
3   <head>
4       <meta charset="UTF-8">
5       <title>Login</title>
6       <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
7             integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">
8       <link rel="stylesheet" href="/home.css" type="text/css">
9   ›   <style...>
59  </head>
60  <body>
61  <header th:include="@{/fragments/header.html}"></header>
62  <div class="content">
63  <div class="main" >
64      <h1>Log in page to <span th:text="${restaurantName}"> </span> Restaurant</h1>
65      <div class="alert alert-info" role="alert" th:if="${message}">
66          <span th:text="${message}"></span>
67      </div>
68      <div th:if="${param.error}" class="alert alert-danger">
69          Invalid username and password.
70      </div>
71      <div th:if="${param.logout}" class="alert alert-warning">
72          You have been logged out.
73      </div>
74      <form th:action="@{/login}" method="post">
75          <div>
76              <input type="text" name="username" placeholder="Username"/>
77          </div>
78          <br>
79          <div>
80              <input type="password" name="password" placeholder="Password"/>
81          </div>
82          <br>
83          <input type="submit" value="Log in" />
84      </form>
85      <div class="alert">
86          <a class="nav-link" th:href="@{/register}">Register</a>
87      </div>
88      <footer th:include="@{/fragments/footer.html}"></footer>
89  </div>
90  </div>
91  </body>
html › body › div.content
```



<> error.html ×

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Error 403 Unauthorized Access</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
          integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">
    <link rel="stylesheet" href="/home.css" type="text/css">
    <style...>
</head>
<body>
<header th:include="@{/fragments/header.html}">
</header>
<div class="content">
    <div class="main">
        <img src="https://i.imgur.com/U3vTGjX.png" alt="Error 403 Image" style="width: 100px;">
        <h1>403 - Unauthorized Access</h1>
        <p>You are not authorized to view this page.</p>
        <a href="/restaurant/home" class="btn btn-primary">Go to the Home Page</a>
    </div>
</div>
<footer th:include="@{/fragments/footer.html}"></footer>
</body>
</html>
```



Reusable Fragments

Such as `header.html` and `footer.html` in the `fragments` folder, which are incorporated across various templates to ensure a consistent layout and reduce redundancy.

```
<> footer.html ×
<div class="bg-grey d-flex justify-content-around pt-3 pb-3" style="color: #ffffff;
    background-color: #008000;
    border-top: 10px solid red;
    font-weight: bold;
    text-align: center;
    position: fixed;
    bottom: 0;
    left: 0;
    padding: 10px 20px;
    width: 100%;">
    <span>Final Project - Barbara Tosetto N01535342</span>
</div>
```

```
<> header.html ×
<nav class="navbar navbar-expand-lg" style="background-color: #008000; /* white */ border-bottom: 10px solid red; /* red */">
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class="nav-link" th:href="@{/restaurant/home}" style="color: #ffffff; font-weight: bold;">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" th:href="@{/restaurant/menu/1}" style="color: #ffffff; font-weight: bold;">Menu</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" th:href="@{/restaurant/admin/add-dish}" style="color: #ffffff; font-weight: bold;">Add dish</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" th:href="@{/login}" style="color: #ffffff; font-weight: bold;">Log In</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" th:href="@{/logout}" style="color: #ffffff; font-weight: bold;">Logout</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" th:href="@{/register}" style="color: #ffffff; font-weight: bold;">Register</a>
            </li>
        </ul>
    </div>
</nav>
```

Static Resource

Housed in the `static` folder, including CSS files and images that help in styling and theming the pages, enhancing the overall look and feel of the application.

home.css

```css
body {
    background-color: #ffffff; /* white */
    color: #008000; /* black */
    font-family: Arial, sans-serif;

    margin: 0;
}

div.main {
    display: flex;
    flex-direction: column;

}

div.content {
    flex: 1;
}

.content {
    margin: 2px;
    min-height: 65vh;
}

.card {
    width: 800px;
    text-align: center;
    margin: 40px auto;
    border: 5px solid #008000; /* green */
    padding: 5px;
    background-color: #ffffff; /* white */
    border-radius: 10px;
}

.card-title, .card-text {
    color: #3f6c3f; /* black */
}

.restaurant-name {
    color: #008000; /* green */
    font-size: 50px;
}
```

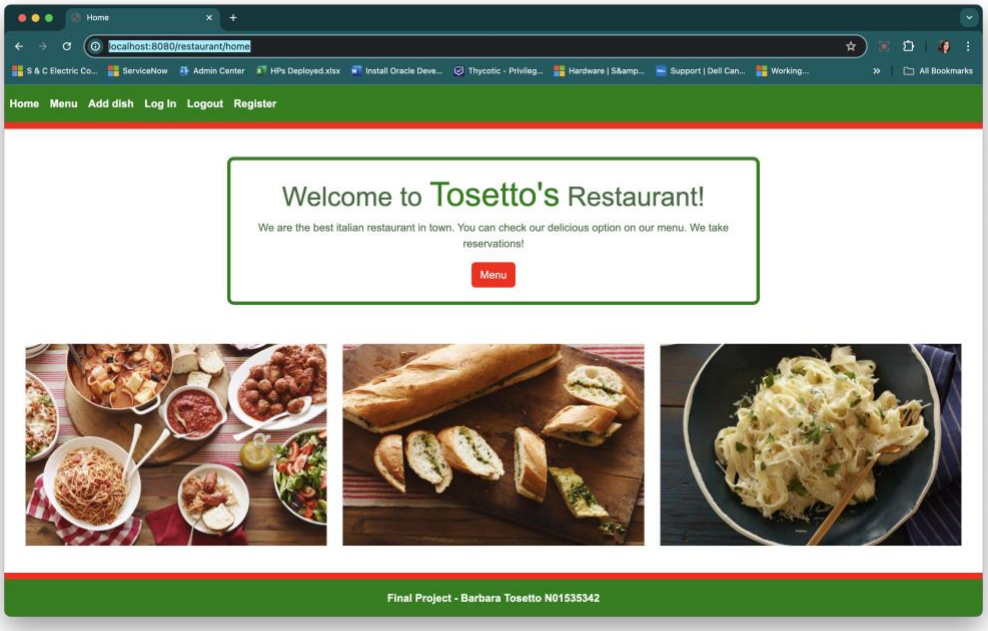
home.css

```css
.image-gallery {
    margin-top: -200px;
    margin-left: 20px;
    margin-right: 20px;
    margin-bottom: 10px;
}

.pagination {
    text-align: center;
    display: flex;
    flex-direction: column;
    justify-content: center;

    width: 100%;
}

.pagination-info, .pagination-no {
    display: flex;
    justify-content: center;
    flex-wrap: wrap;
    border: #cccccc;
}

.pagination-info div, .pagination-no div {
    margin: 5px;
    padding: 5px;
    border: 1px solid #ff0000; /* green */
    background-color: #ff0000; /* red */
    color: white;
    border-radius: 5px;
    cursor: pointer;
    font-weight: bold;
}

.pagination-info div:hover, .pagination-no div:hover {
    background-color: #ff0000;
    color: #fff;
}
```

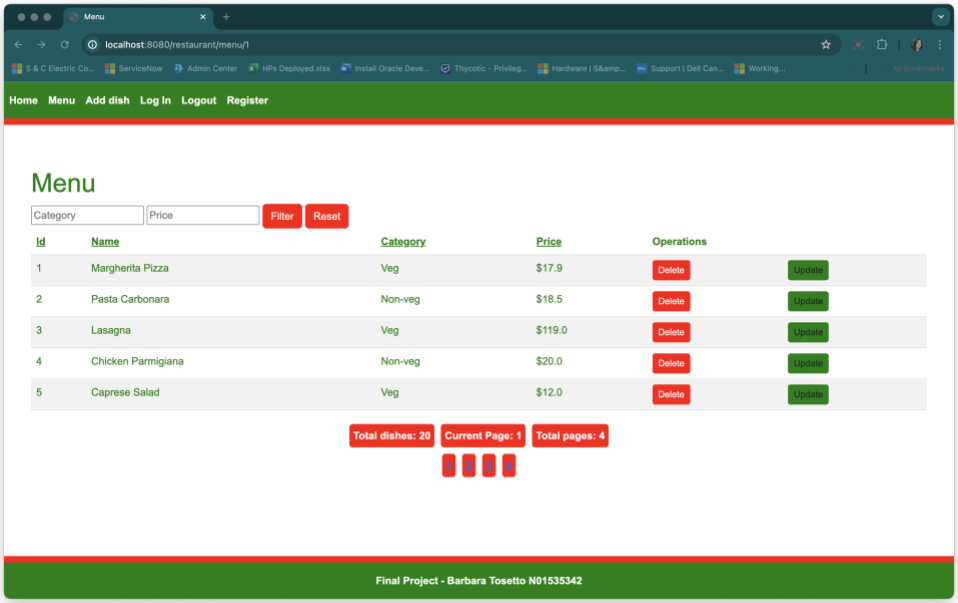
home.css

```css

.table {
    background-color: #ffffff; /* white */
    color: #008000; /* black */
}

.table th, .table td {
    background-color: #ffffff; /* white */
    color: #008000; /* black */
    border-color: #dddddd; /* light grey */
}

.table th a {
    color: #008000; /* green */
}

.btn-danger, .btn-primary {
    background-color: #ff0000; /* red */
    border-color: #ff0000; /* red */
}

.btn-danger:hover, .btn-primary:hover {
    background-color: #cc0000; /* darker red */
    border-color: #cc0000; /* darker red */
}

.btn-update {
    background-color: #008000; /* green */
    border-color: #008000; /* green */
}

.btn-update:hover {
    background-color: #006400; /* dark green */
    border-color: #006400; /* dark green */
}
```

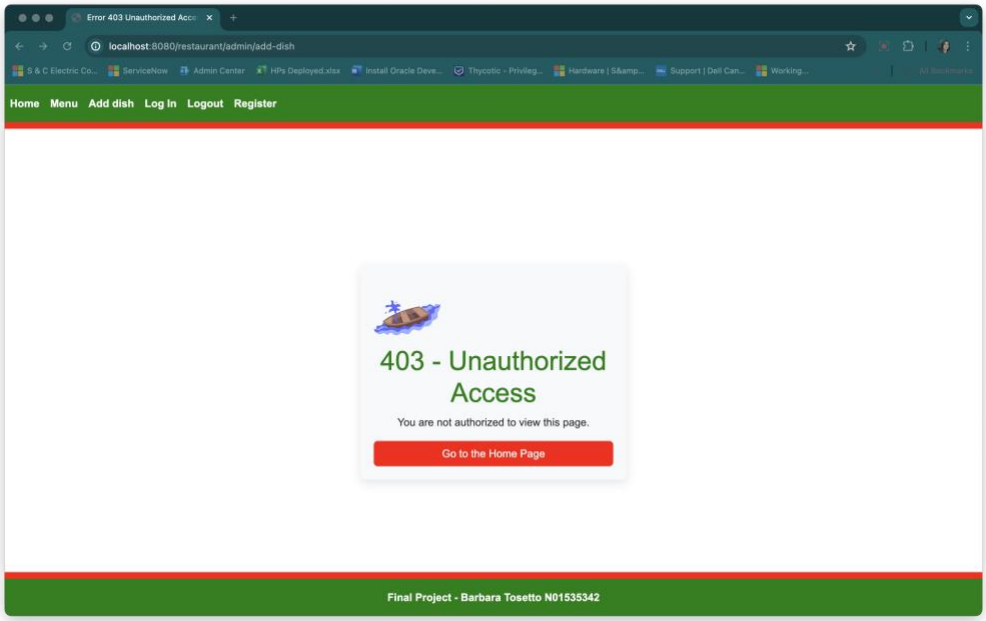
Even if the user is not logged in, the home page is available, also the menu page.
Home page: http://localhost:8080/restaurant/home



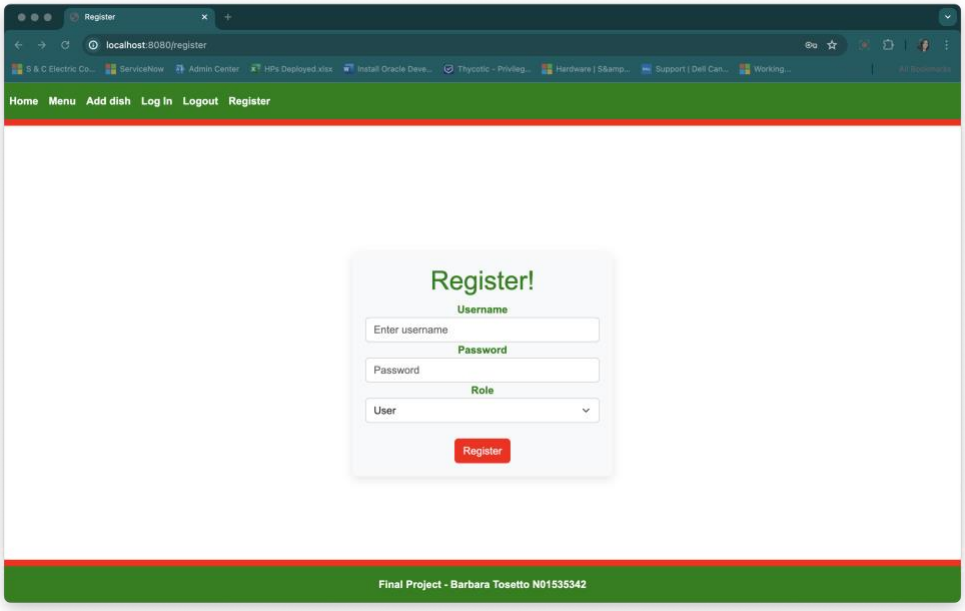
Menu: http://localhost:8080/restaurant/menu/1

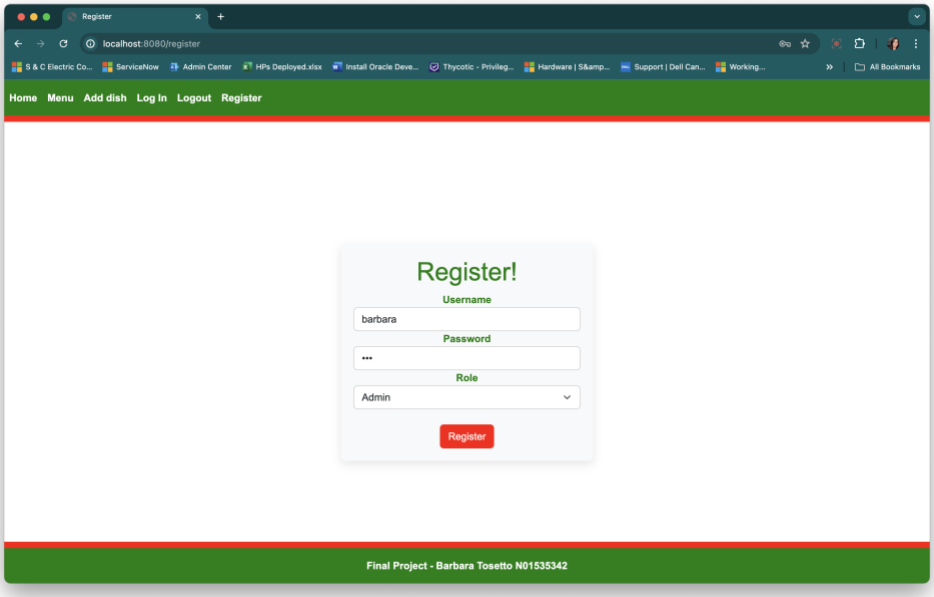


But if the user is not logged in and navigates to “Add Dish” page, or tries to perform “Delete” or “Update”, they will be redirected to an error page:



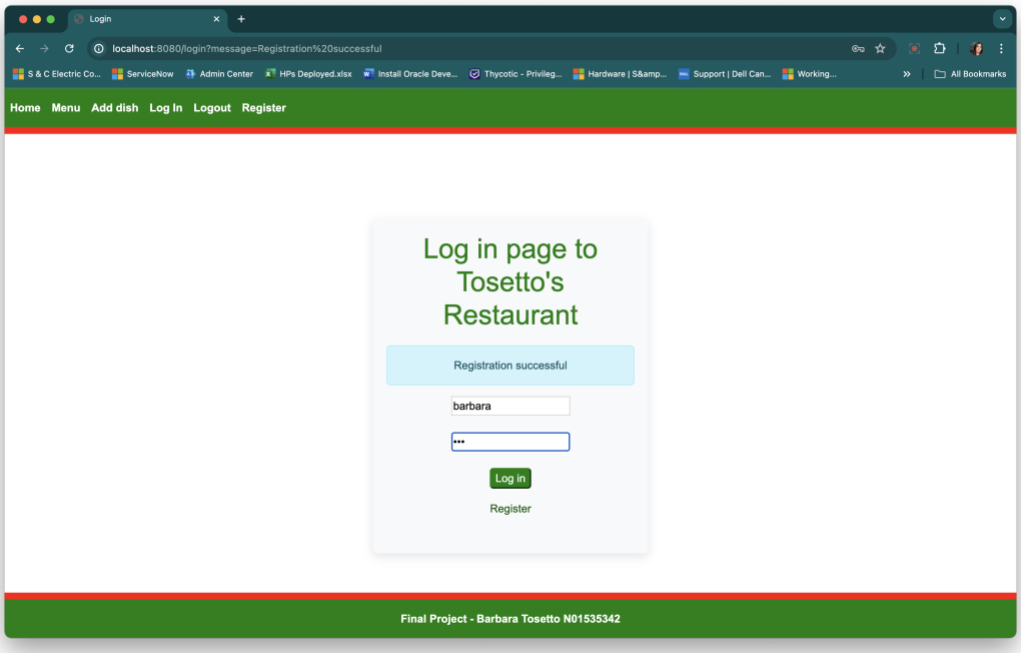
The user can Register as “Admin”, or “User” in order to have access to all features of the application.
Register Page: http://localhost:8080/register





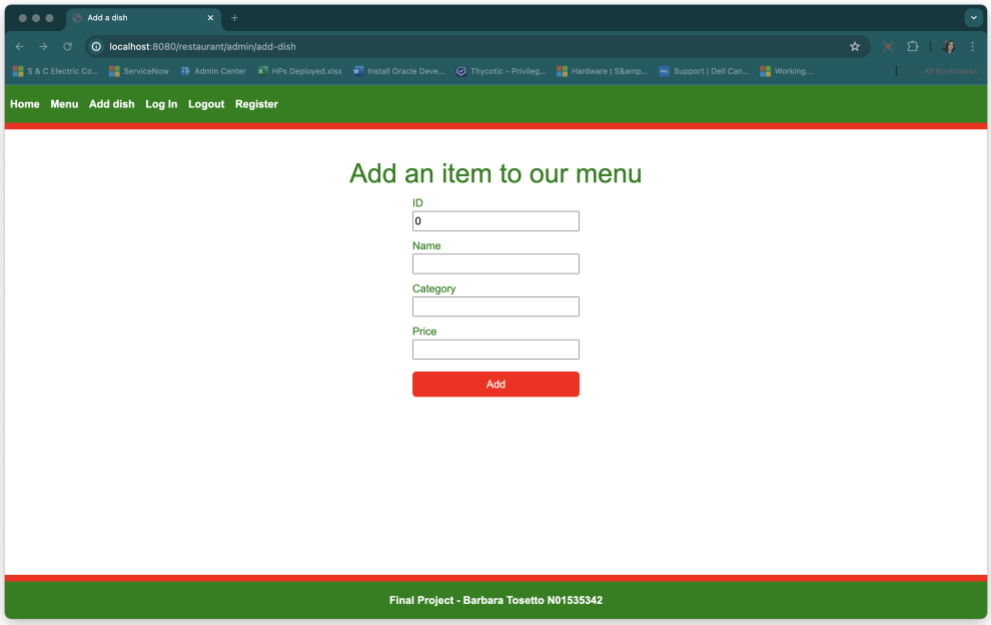
After registering, the user is headed to the Log In page (http://localhost:8080/login?message=Registration%20successful). Finally, the user will be redirected to the Restaurant's home page.

Log In page: http://localhost:8080/login

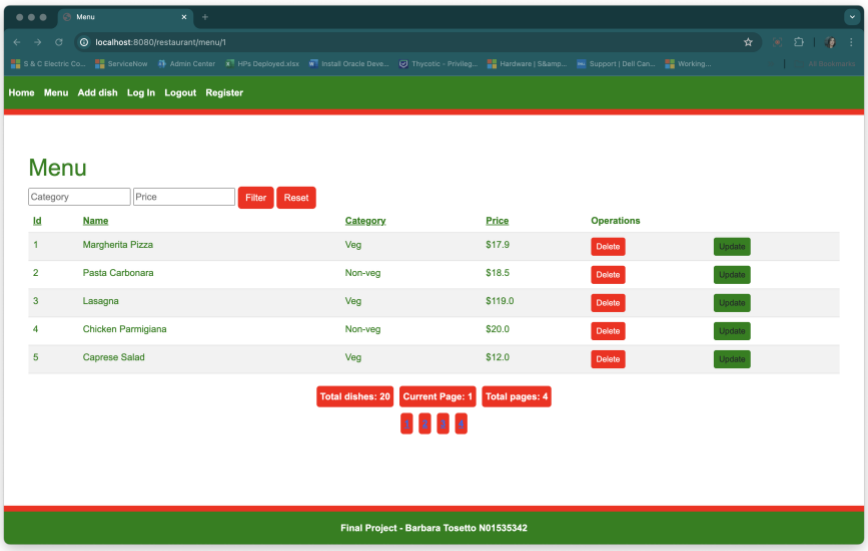


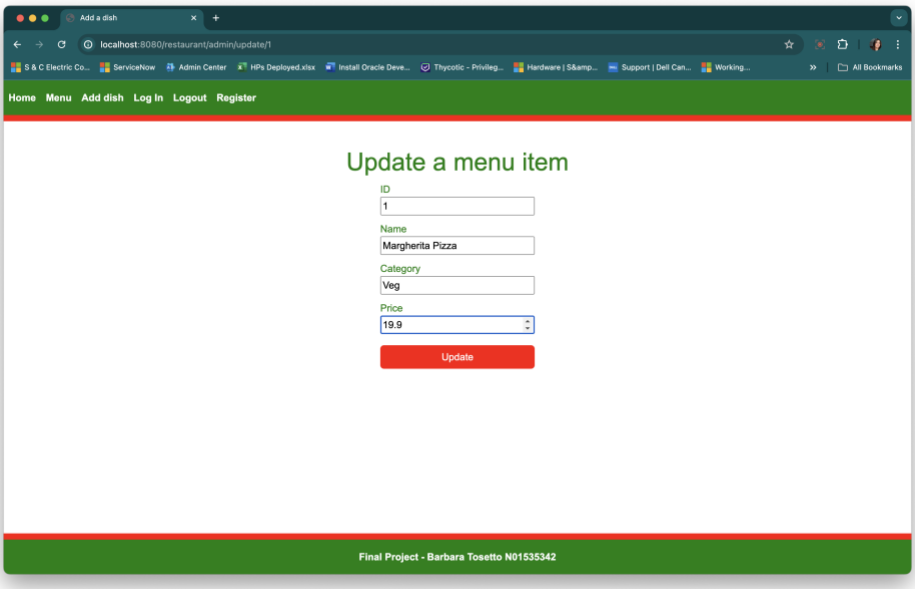
Now the feature to add a dish is available:

Add-Dish: http://localhost:8080/restaurant/admin/add-dish

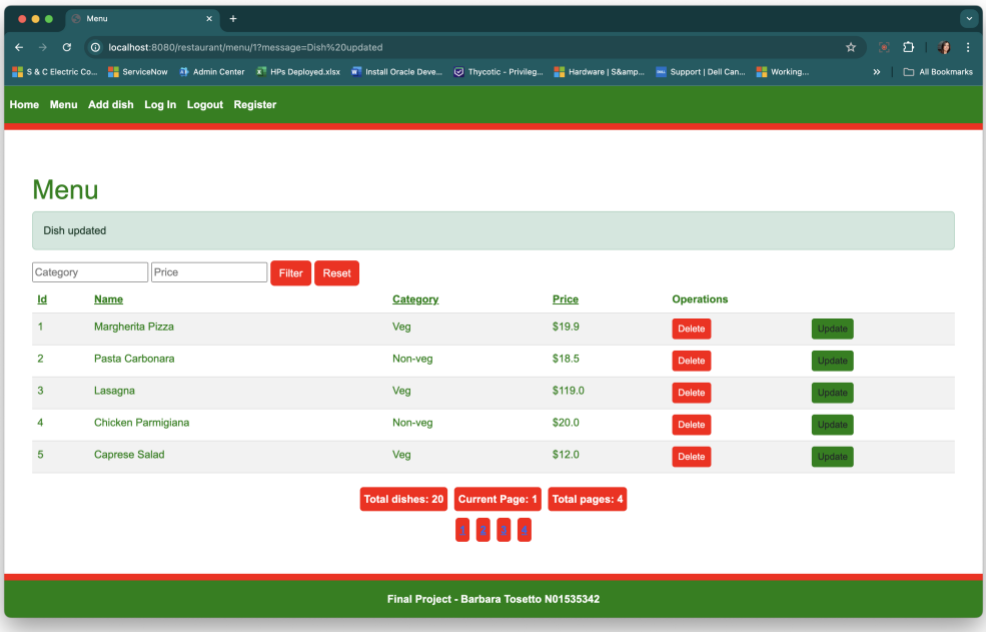


Also, the feature to update and dele dishes is available if the user is logged as “ADMIN”

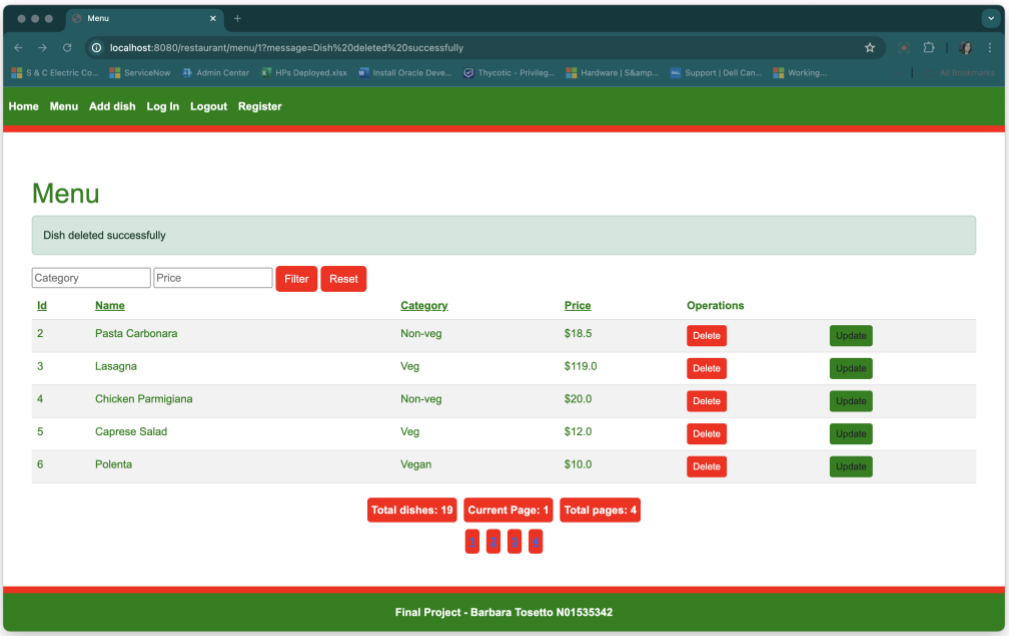




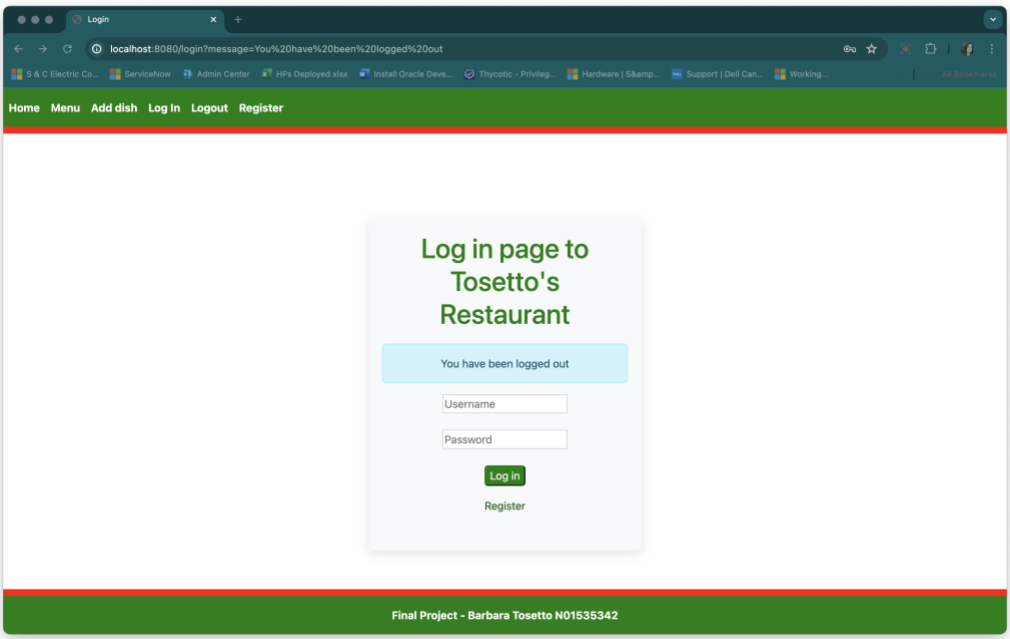
The price of the first item of the list was updated:



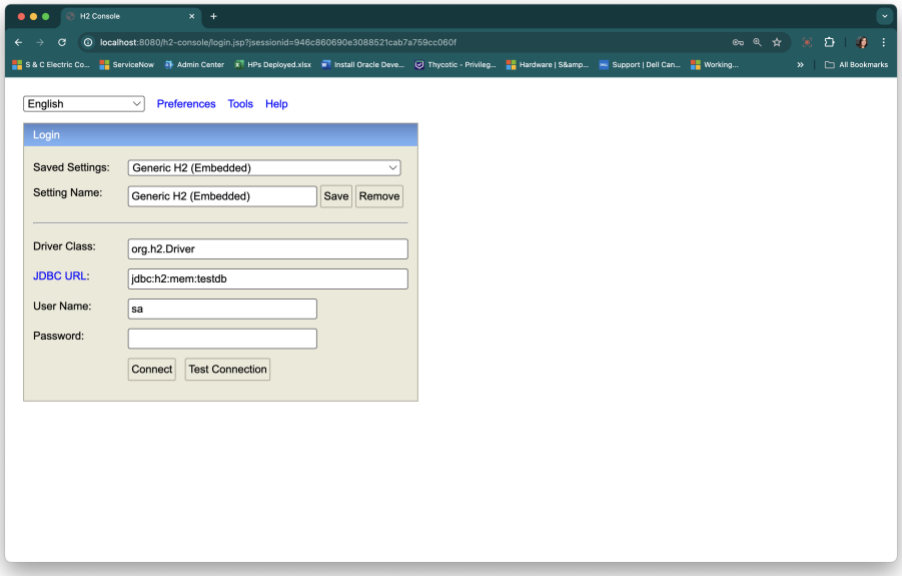
And then the first item was deleted:

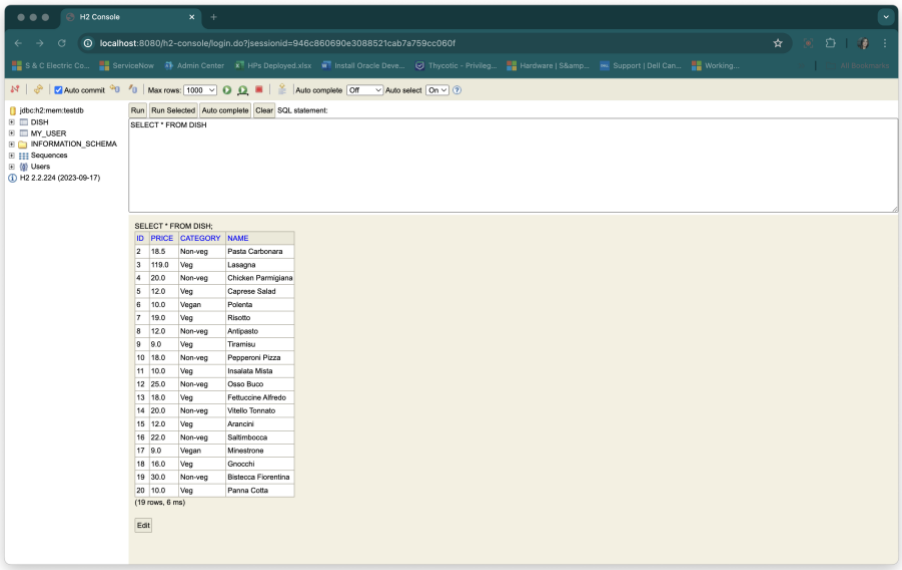


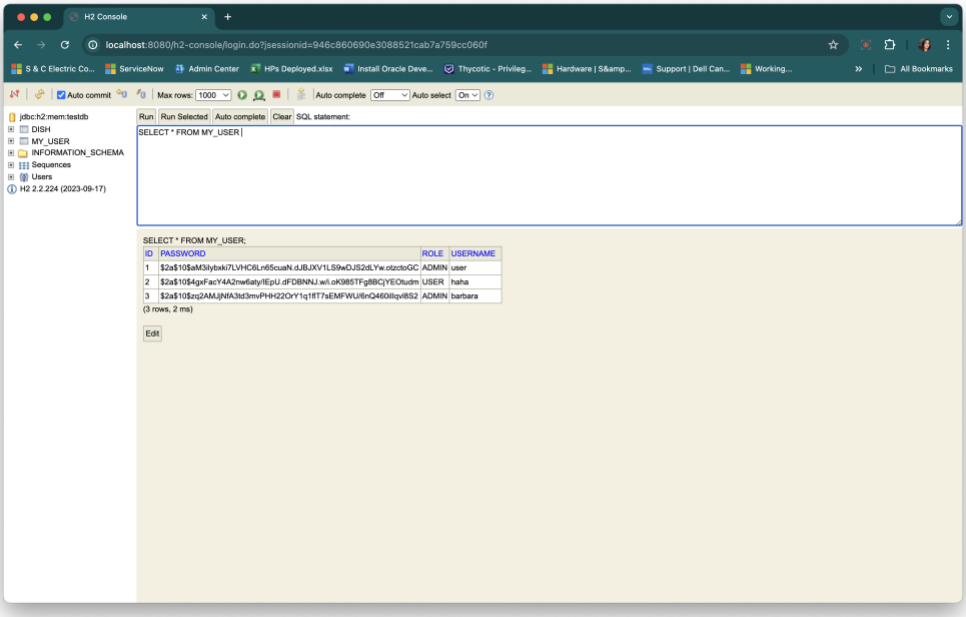
The Logout link on the navbar disconnects the user and displays a successful on the log in page:



Access to H2-console:






H2 Console
localhost:8080/h2-console/login.do?jsessionid=946c860690e3088521cab7a759cc060f
Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:mem:testdb
DISH
MY_USER
INFORMATION_SCHEMA
Sequences
Users
H2 2.2.224 (2023-09-17)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM MY_USER
SELECT * FROM MY_USER;
ID PASSWORD ROLE USERNAME
1 $2a$10$aM3lybxki7LVHC6Ln65cuaN.dJBJXV1LS9wOJS2dLYw.otzctoGC ADMIN user
2 $2a$10$4gxFacY4A2nw6atylIEpU.dFDBNNJ.w/i.oK9l65TFg8BCjYEOtudm USER haha
3 $2a$10$zq2AMJjNfA3td3mvPHH22OrY1q1fT7sEMFWU/6nQ460IlqvI8S2 ADMIN barbara
(3 rows, 2 ms)
Edit