

# 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

---

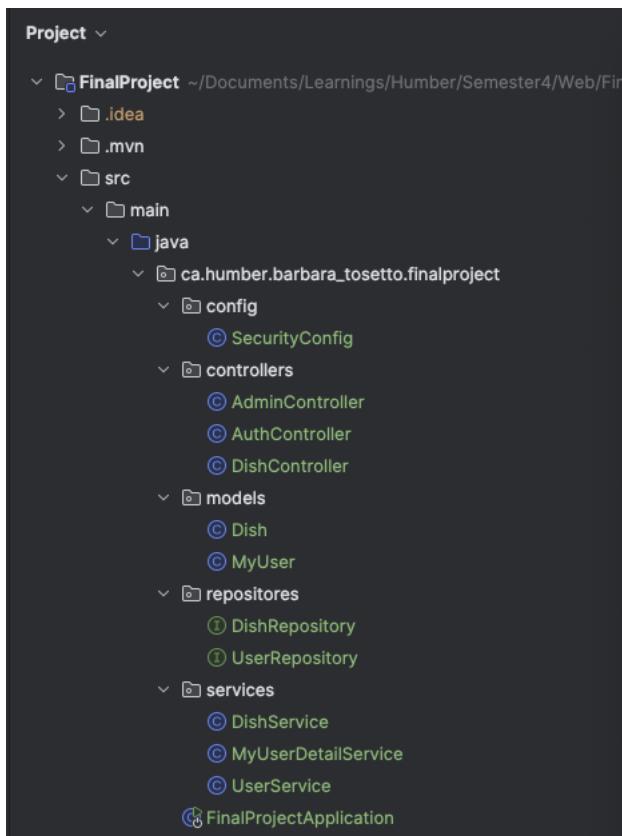
## STEP 2 – Application Architecture

---

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.

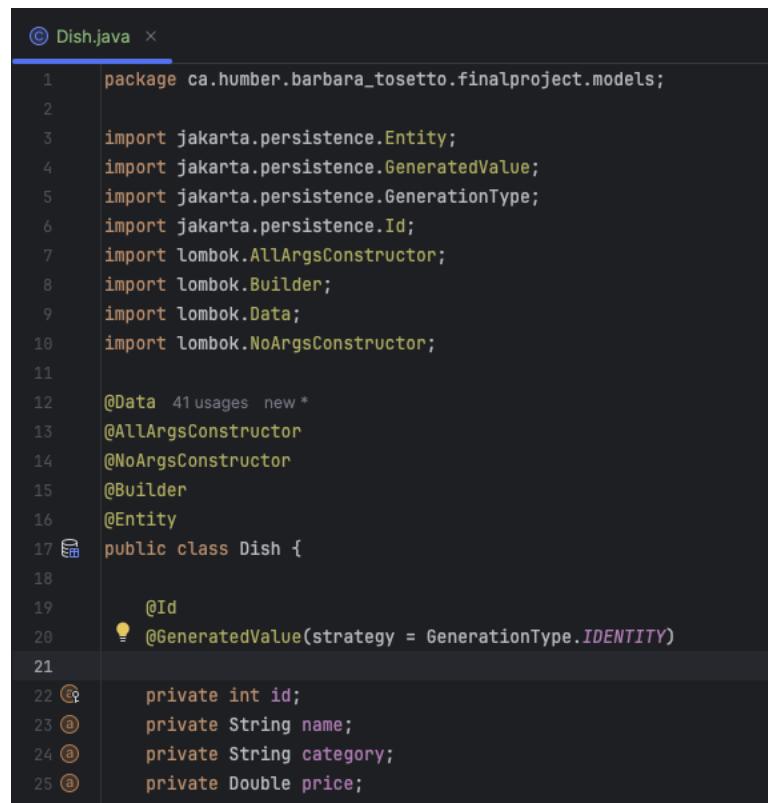
Both the `Dish` and `MyUser` classes are fundamental components of this application's data layer, interacting directly with the database and serving as the primary method of data transport across the application. They ensure that data handling is streamlined and consistent with the application's object-oriented 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.

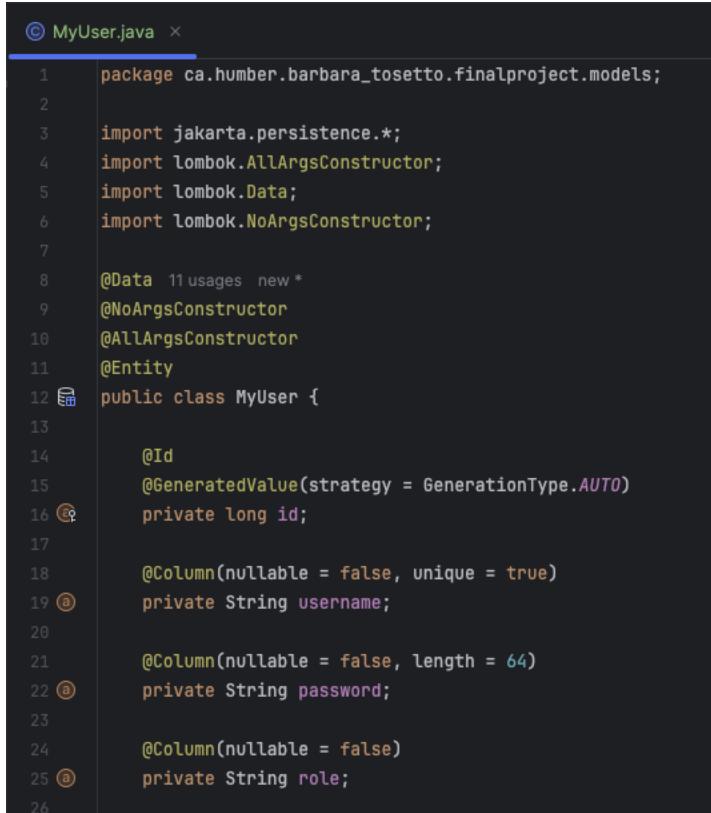


A screenshot of a code editor showing the `Dish.java` file. The code is as follows:

```
① package ca.humber.barbara_tosetto.finalproject.models;
②
③ import jakarta.persistence.Entity;
④ import jakarta.persistence.GeneratedValue;
⑤ import jakarta.persistence.GenerationType;
⑥ import jakarta.persistence.Id;
⑦ import lombok.AllArgsConstructor;
⑧ import lombok.Builder;
⑨ import lombok.Data;
⑩ import lombok.NoArgsConstructor;
⑪
⑫ @Data 41 usages new *
⑬ @AllArgsConstructor
⑭ @NoArgsConstructor
⑮ @Builder
⑯ @Entity
⑰ public class Dish {
⑱
⑲     @Id
⑳     @GeneratedValue(strategy = GenerationType.IDENTITY)
⑲
⑳     private int id;
⑳     private String name;
⑳     private String category;
⑳     private Double 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.



A screenshot of a Java code editor showing the `MyUser.java` file. The code defines a class `MyUser` with three attributes: `id`, `username`, and `password`. The `id` attribute is annotated with `@Id` and `@GeneratedValue(strategy = GenerationType.AUTO)`. The `username` and `password` attributes are annotated with `@Column(nullable = false, unique = true)` and `@Column(nullable = false, length = 64)` respectively. The `role` attribute is annotated with `@Column(nullable = false)`.

```
① MyUser.java ×
1 package ca.humber.barbara_tosetto.finalproject.models;
2
3 import jakarta.persistence.*;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 @Data 11 usages new *
9 @NoArgsConstructor
10 @AllArgsConstructor
11 @Entity
12 ⑬ public class MyUser {
13
14     @Id
15     @GeneratedValue(strategy = GenerationType.AUTO)
16     ⑭ private long id;
17
18     @Column(nullable = false, unique = true)
19     ⑮ private String username;
20
21     @Column(nullable = false, length = 64)
22     ⑯ private String password;
23
24     @Column(nullable = false)
25     ⑰ private String role;
26
```

---

## STEP 4 - Controllers

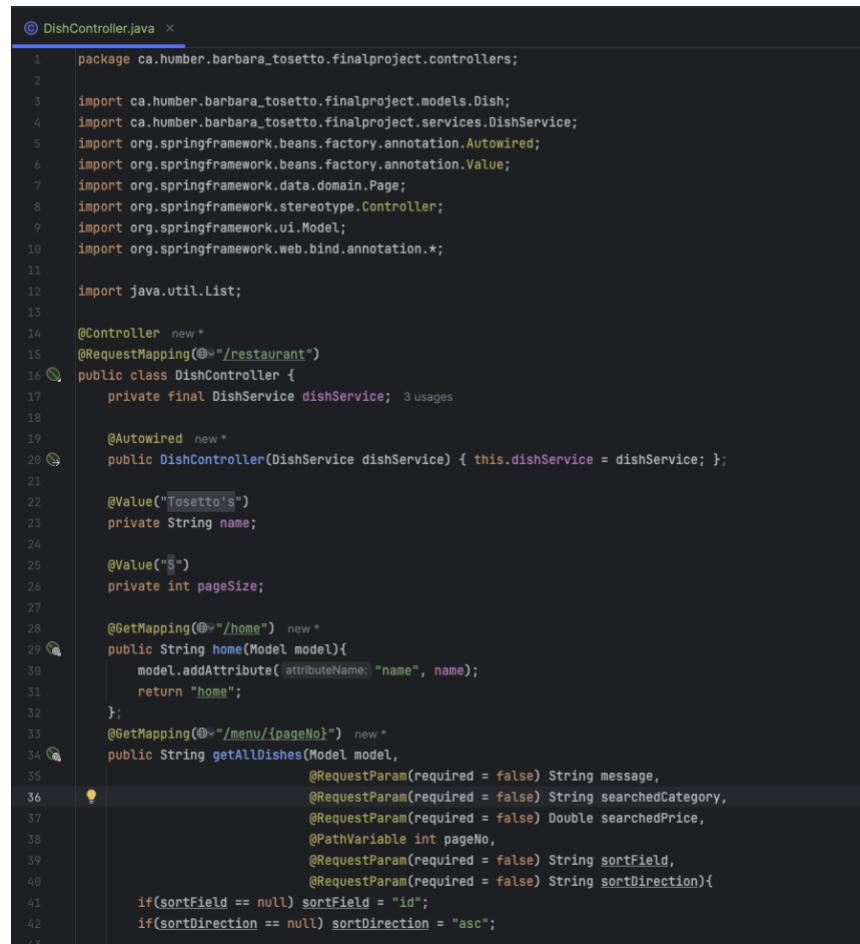
---

### 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`.



```
① package ca.humber.barbara_tosetto.finalproject.controllers;
②
③ import ca.humber.barbara_tosetto.finalproject.models.Dish;
④ import ca.humber.barbara_tosetto.finalproject.services.DishService;
⑤ import org.springframework.beans.factory.annotation.Autowired;
⑥ import org.springframework.beans.factory.annotation.Value;
⑦ import org.springframework.data.domain.Page;
⑧ import org.springframework.stereotype.Controller;
⑨ import org.springframework.ui.Model;
⑩ import org.springframework.web.bind.annotation.*;
⑪
⑫ import java.util.List;
⑬
⑭ @Controller
⑮ @RequestMapping("/restaurant")
⑯ public class DishController {
⑰     private final DishService dishService;  3 usages
⑱
⑲     @Autowired
⑳     public DishController(DishService dishService) { this.dishService = dishService; }
⑳
⑳     @Value("Tosetto's")
⑳     private String name;
⑳
⑳     @Value("5")
⑳     private int pageSize;
⑳
⑳     @GetMapping("/home")  new *
⑳     public String home(Model model){
⑳         model.addAttribute( attributeName: "name", name);
⑳         return "home";
⑳     };
⑳     @GetMapping("/menu/{pageNo}")  new *
⑳     public String getAllDishes(Model model,
⑳         @RequestParam(required = false) String message,
⑳         @RequestParam(required = false) String searchedCategory,
⑳         @RequestParam(required = false) Double searchedPrice,
⑳         @PathVariable int pageNo,
⑳         @RequestParam(required = false) String sortField,
⑳         @RequestParam(required = false) String sortDirection){
⑳             if(sortField == null) sortField = "id";
⑳             if(sortDirection == null) sortDirection = "asc";
⑳         }
⑳     }
⑳ }
```

```

53     @GetMapping("menu/{pageNo}")
54     public String getAllDishes(Model model,
55         @RequestParam(required = false) String message,
56         @RequestParam(required = false) String searchedCategory,
57         @RequestParam(required = false) Double searchedPrice,
58         @PathVariable int pageNo,
59         @RequestParam(required = false) String sortField,
60         @RequestParam(required = false) String sortDirection){
61         if(sortField == null) sortField = "id";
62         if(sortDirection == null) sortDirection = "asc";
63
64         Page<Dish> page = dishService.getPaginatedDishes(pageNo, pageSize, sortField, sortDirection);
65
66         if(searchedCategory!=null & searchedPrice!=null){
67             List<Dish> filteredDishes = dishService.getFilteredDishes(searchedCategory, searchedPrice);
68             model.addAttribute( attributeName: "dishes", filteredDishes.isEmpty() ? page.getContent() : filteredDishes);
69             model.addAttribute( attributeName: "message", filteredDishes.isEmpty() ? "Not filtered" : "Filtered successfully");
70             return "menu";
71         }
72         model.addAttribute( attributeName: "dishes", page.getContent());
73         model.addAttribute( attributeName: "totalPages", page.getTotalPages());
74         model.addAttribute( attributeName: "currentPage", pageNo);
75         model.addAttribute( attributeName: "totalItems", page.getTotalElements());
76         model.addAttribute( attributeName: "sortField", sortField);
77         model.addAttribute( attributeName: "sortDirection", sortDirection);
78         model.addAttribute( attributeName: "reverseSortDirection", sortDirection.equals("asc") ? "desc" : "asc");
79         model.addAttribute( attributeName: "message", message);
80         return "menu";
81     }
82 }

```

## 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.

```

⑧ AuthController.java ×
1 package ca.humber.barbara_tosetto.finalproject.controllers;
2
3 import ca.humber.barbara_tosetto.finalproject.models.MyUser;
4 import ca.humber.barbara_tosetto.finalproject.services.UserService;
5 import jakarta.servlet.http.HttpServletRequest;
6 import jakarta.servlet.http.HttpServletResponse;
7 import org.springframework.beans.factory.annotation.Value;
8 import org.springframework.security.core.Authentication;
9 import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;
10 import org.springframework.stereotype.Controller;
11 import org.springframework.ui.Model;
12 import org.springframework.web.bind.annotation.GetMapping;
13 import org.springframework.web.bind.annotation.ModelAttribute;
14 import org.springframework.web.bind.annotation.PostMapping;
15 import org.springframework.web.bind.annotation.RequestParam;
16
17 @Controller@v new *
18 public class AuthController implements org.springframework.boot.web.servlet.error.ErrorController {
19
20     @Value("${restaurant.name}")
21     private String name;
22     private UserService userService; 2 usages
23
24 public AuthController(UserService userService){ new *
25     this.userService = userService;
26 }
27
28 @GetMapping(@v"/error") new *
29 public String handleError(){
30     return "auth/error";
31 }
32 @GetMapping(@v"/login") new *
33 public String login(Model model, @RequestParam(required = false) String message){
34     model.addAttribute( attributeName: "message", message);
35     model.addAttribute( attributeName: "restaurantName", name);
36     return "auth/login";
37 }
38 @GetMapping(@v"/logout") new *
39 public String logout(HttpServletRequest req,
        HttpServletResponse res, Authentication auth){
40     new SecurityContextLogoutHandler().logout(req, res, auth);
41     return "redirect:/login?message=You have been logged out";
42 }
43 @GetMapping(@v"/register") new *
44 public String register(Model model, @RequestParam(required = false) String message){
45     model.addAttribute( attributeName: "user", new MyUser());
46     model.addAttribute( attributeName: "message", message);
47     return "auth/register";
48 }
49 @PostMapping(@v"/register") new *
50 public String register(@ModelAttribute MyUser user){
51     int saveUserCode = userService.createUser(user);
52     if(saveUserCode == 0){
53         return "redirect:/register?message=User already exists";
54     };
55     return "redirect:/login?message=Registration successful";
56 }
57 }
58 }
```

```

57 }
58 }
59 }
```

## 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.

```
② AdminController.java ×

1 package ca.humber.barbara_tosetto.finalproject.controllers;
2
3 import ca.humber.barbara_tosetto.finalproject.models.Dish;
4 import ca.humber.barbara_tosetto.finalproject.services.DishService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.beans.factory.annotation.Value;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.ui.Model;
9 import org.springframework.web.bind.annotation.*;
10
11 import java.util.Optional;
12
13 @Controller new *
14 @RequestMapping(@v"/restaurant/admin")
15 ④ public class AdminController {
16     private final DishService dishService;  5 usages
17
18     @Autowired new *
19     ④ public AdminController(DishService dishService) { this.dishService = dishService; }
20     @Value("Tosetto's")
21     private String name;
22
23     @GetMapping(@v"/add-dish") new *
24     ④ public String addDish(Model model){
25         model.addAttribute( attributeName: "dish", new Dish());
26         return "/admin/add-dish";
27     }
28     @PostMapping(@v"/add-dish") new *
29     ④ public String addDish(@ModelAttribute Dish dish, Model model){
30         int status = dishService.addDish(dish);
31         if (status == 0){
32             model.addAttribute( attributeName: "error", attributeValue: "Price must not exceed $20");
33             return "/admin/add-dish";
34         }
35         return "redirect:/restaurant/menu/1?message=Dish added successfully";
36     }
37     @GetMapping(@v"/delete/{id}") new *
38     ④ public String deleteDish(@PathVariable int id){
```

```
37 @GetMapping("/{id}") new *
38 public String deleteDish(@PathVariable int id){
39     int status = dishService.deleteDish(id);
40     if (status > 0){
41         return "redirect:/restaurant/menu/1?message=Dish deleted successfully";
42     }
43     return "redirect:/restaurant/menu/1?message=Dish does not exist";
44 }
45 @GetMapping("/{id}") new *
46 public String updateDish(Model model, @PathVariable int id){
47     Optional<Dish> dishToUpdate = dishService.getDishById(id);
48     model.addAttribute( attributeName: "dish", dishToUpdate.orElse( other: null));
49     return "/admin/add-dish";
50 }
51 @PostMapping("/update-dish") new *
52 public String updateDish(@ModelAttribute Dish dish){
53     dishService.updatedDish(dish);
54     return "redirect:/restaurant/menu/1?message=Dish updated";
55 }
56 }
```

---

## 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.

```
② DishService.java ×
1 package ca.humber.barbara_tosetto.finalproject.services;
2
3 import ca.humber.barbara_tosetto.finalproject.models.Dish;
4 import ca.humber.barbara_tosetto.finalproject.repositories.DishRepository;
5 import org.springframework.data.domain.Page;
6 import org.springframework.data.domain.PageRequest;
7 import org.springframework.data.domain.Pageable;
8 import org.springframework.data.domain.Sort;
9 import org.springframework.stereotype.Service;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import java.util.List;
12 import java.util.Optional;
13
14 @Service " 9 usages new "
15 public class DishService {
16     private final DishRepository dishRepo; " 9 usages
17
18     @Autowired " new "
19     public DishService(DishRepository dishRepo) { this.dishRepo = dishRepo; }
20
21     //Get all dishes
22     public List<Dish> getAllDishes() { return dishRepo.findAll(); }
23
24     //Get filtered dishes
25     public List<Dish> getFilteredDishes(String searchedCategory, Double searchedPrice){ " 1 usage new "
26         return dishRepo.findByCategoryAndPrice(searchedCategory, searchedPrice);
27     }
28
29     //Get paginated dishes
30     public Page<Dish> getPaginatedDishes(int pageNo, int pageSize, String sortField, String sortDirection){ " 1 usage new "
31         Sort sort = sortDirection.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortField).ascending() : Sort.by(sortField).descending();
32         Pageable pageable = PageRequest.of( pageNumber: pageNo-1, pageSize, sort);
33         return dishRepo.findAll(pageable);
34     }
35
36
37
38 }
```

```

40     //Add dish
41     @ public int addDish(Dish dish){ 23 usages new *
42         if(dish.getPrice()>8){
43             dishRepo.save(dish);
44             return 1;
45         }
46         return 0;
47     }
48
49     //Remove dish
50     public int deleteDish(int id){ 1 usage new *
51         if(dishRepo.existsById(id)){
52             dishRepo.deleteById(id);
53             return 1;
54         }
55         return 0;
56     }
57
58     //Update dish
59     > public void updateDish(Dish dish) { dishRepo.save(dish); }
60
61     //Get dish by id
62     > public Optional<Dish> getDishById(int id) { return dishRepo.findById(id); }
63
64
65 }
```

## MyUserDetailsService Class

The `MyUserDetailsService` 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 ×

1 package ca.humber.barbara_tosetto.finalproject.services;
2
3 import ca.humber.barbara_tosetto.finalproject.models.MyUser;
4 import ca.humber.barbara_tosetto.finalproject.repositories.UserRepository;
5 import org.springframework.security.core.userdetails.User;
6 import org.springframework.security.core.userdetails.UserDetails;
7 import org.springframework.security.core.userdetails.UserDetailsService;
8 import org.springframework.security.core.userdetails.UsernameNotFoundException;
9 import org.springframework.stereotype.Service;
10
11 import java.util.Optional;
12
13 @Service new*
14 public class MyUserDetailService implements UserDetailsService {
15
16     private UserRepository userRepo; 2 usages
17
18     public MyUserDetailService(UserRepository userRepo) { this.userRepo = userRepo; }
19
20     @Override new*
21     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
22         Optional<MyUser> userOp = userRepo.findByUsername(username);
23
24         if (userOp.isPresent()){
25             MyUser user = userOp.get();
26             return User.builder()
27                 .username(user.getUsername())
28                 .password(user.getPassword())
29                 .roles(user.getRole())
30                 .build();
31         } else{
32             throw new UsernameNotFoundException("User not found");
33         }
34     }
35 }
36 }
37 }

```

## 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 ×

1 package ca.humber.barbara_tosetto.finalproject.services;
2
3 import ca.humber.barbara_tosetto.finalproject.models.MyUser;
4 import ca.humber.barbara_tosetto.finalproject.repositories.UserRepository;
5 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
6 import org.springframework.stereotype.Service;
7
8 @Service 3 usages new *
9 public class UserService {
10     private final UserRepository userRepo; 3 usages
11     private BCryptPasswordEncoder bcrypt; 2 usages
12
13     public UserService(UserRepository userRepo, BCryptPasswordEncoder bcrypt) { new *
14         this.userRepo = userRepo;
15         this.bcrypt = bcrypt;
16     }
17     //Create new user
18     @
19     public int createUser(MyUser user){ 1 usage new *
20         //Check if user already exists
21         if (userRepo.findByUsername(user.getUsername()).isPresent()) return 0;
22         //Encrypts password
23         user.setPassword(bcrypt.encode(user.getPassword()));
24         //Saves info
25         userRepo.save(user);
26         return 1;
27     }
}
```

---

## 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.

```
① DishRepository.java ×
1 package ca.humber.barbara_tosetto.finalproject.repositories;
2
3 import ca.humber.barbara_tosetto.finalproject.models.Dish;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.stereotype.Repository;
7 import java.util.List;
8
9 @Repository 3 usages new *
10 public interface DishRepository extends JpaRepository<Dish, Integer> {
11     @Query(value = "SELECT * from Dish where lower(category) = lower(?1) AND price = ?2", nativeQuery = true) 1 usage new *
12     List<Dish> findByCategoryAndPrice(String searchedCategory, Double searchedPrice);
13 }
14 |
```

### 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

```
1 package ca.humber.barbara_tosetto.finalproject.repositorie;
2
3 import ca.humber.barbara_tosetto.finalproject.models.MyUser;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 import java.util.Optional;
7
8  public interface UserRepository extends JpaRepository<MyUser, Long> { 6 usages new *
9     public Optional<MyUser> findByUsername(String username); 2 usages new *
10 }
11 }
```

---

## 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.

```
① package ca.humber.barbara_tosetto.finalproject.config;
②
③ import org.springframework.context.annotation.Bean;
④ import org.springframework.context.annotation.Configuration;
⑤ import org.springframework.security.authentication.AuthenticationProvider;
⑥ import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
⑦ import org.springframework.security.config.annotation.web.builders.HttpSecurity;
⑧ import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
⑨ import org.springframework.security.config.annotation.web.configuration.WebSecurityCustomizer;
⑩ import org.springframework.security.core.userdetails.UserDetailsService;
⑪ import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
⑫ import org.springframework.security.web.SecurityFilterChain;
⑬
⑭ @Configuration new *
⑮ @EnableWebSecurity
⑯ public class SecurityConfig {
⑰     private UserDetailsService myUserDetailsService;  2 usages
⑱     public SecurityConfig(UserDetailsService myUserDetailsService) { this.myUserDetailsService = myUserDetailsService; }
⑲
⑳     @Bean new *
⑳     SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
⑳         http.authorizeHttpRequests((authorize) -> authorize
⑳
⑳             .requestMatchers(④"/restaurant/home", ④"/", ④"/login/**", ④"/register/**").permitAll()
⑳             .requestMatchers(④"/restaurant/menu/**").hasAnyRole( ...roles: "USER", "ADMIN")
⑳             .requestMatchers(④"/restaurant/admin/**").hasRole("ADMIN")
⑳             .anyRequest().authenticated()
⑳         ).formLogin(customLogin -> {
⑳             customLogin.loginPage("/login").defaultSuccessUrl( defaultSuccessUrl: "/restaurant/home", alwaysUse: true).permitAll();
⑳         }
⑳     ).logout(l -> l
⑳         .logoutUrl("/logout")
⑳         .permitAll()
⑳     );
⑳     return http.build();
⑳ }
```

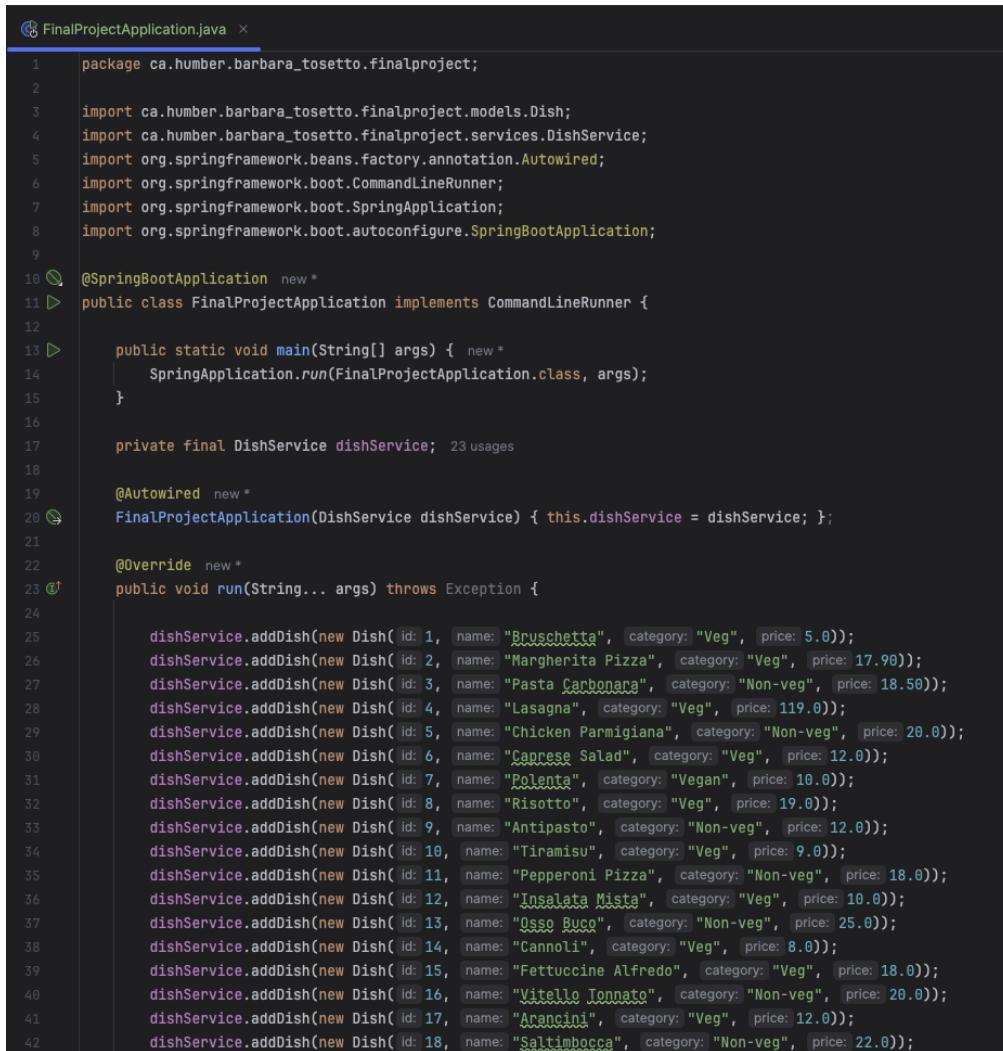
```
38     }
39     @Bean new *
40     public AuthenticationProvider authenticationProvider(){
41         DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
42         provider.setUserDetailsService(myUserDetailsService);
43         provider.setPasswordEncoder(passwordEncoder());
44         return provider;
45     }
46     // Password encrypting
47     @Bean new *
48     > public BCryptPasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
49     @Bean new *
50     public WebSecurityCustomizer ignoreResources(){
51         return (webSecurity) -> webSecurity
52             .ignoring()
53             .requestMatchers(@"/css/**", @"/h2-console/**");
54     }
55 }
```

---

## 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.



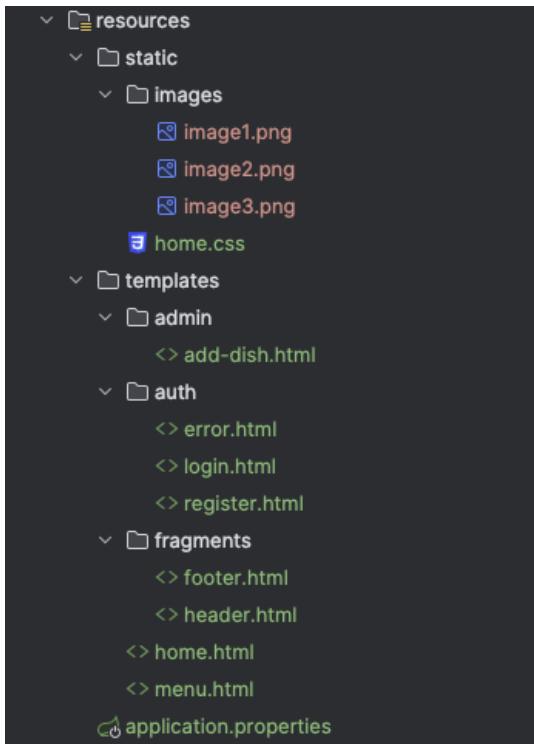
```
⑥ FinalProjectApplication.java ×
1 package ca.humber.barbara_tosetto.finalproject;
2
3 import ca.humber.barbara_tosetto.finalproject.models.Dish;
4 import ca.humber.barbara_tosetto.finalproject.services.DishService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.CommandLineRunner;
7 import org.springframework.boot.SpringApplication;
8 import org.springframework.boot.autoconfigure.SpringBootApplication;
9
10 @SpringBootApplication new *
11 public class FinalProjectApplication implements CommandLineRunner {
12
13     public static void main(String[] args) { new *
14         SpringApplication.run(FinalProjectApplication.class, args);
15     }
16
17     private final DishService dishService; 23 usages
18
19     @Autowired new *
20     FinalProjectApplication(DishService dishService) { this.dishService = dishService; }
21
22     @Override new *
23     public void run(String... args) throws Exception {
24
25         dishService.addDish(new Dish( id: 1, name: "Bruschetta", category: "Veg", price: 5.0));
26         dishService.addDish(new Dish( id: 2, name: "Margherita Pizza", category: "Veg", price: 17.90));
27         dishService.addDish(new Dish( id: 3, name: "Pasta Carbonara", category: "Non-veg", price: 18.50));
28         dishService.addDish(new Dish( id: 4, name: "Lasagna", category: "Veg", price: 119.0));
29         dishService.addDish(new Dish( id: 5, name: "Chicken Parmigiana", category: "Non-veg", price: 20.0));
30         dishService.addDish(new Dish( id: 6, name: "Caprese Salad", category: "Veg", price: 12.0));
31         dishService.addDish(new Dish( id: 7, name: "Polenta", category: "Vegan", price: 10.0));
32         dishService.addDish(new Dish( id: 8, name: "Risotto", category: "Veg", price: 19.0));
33         dishService.addDish(new Dish( id: 9, name: "Antipasto", category: "Non-veg", price: 12.0));
34         dishService.addDish(new Dish( id: 10, name: "Tiramisu", category: "Veg", price: 9.0));
35         dishService.addDish(new Dish( id: 11, name: "Pepperoni Pizza", category: "Non-veg", price: 18.0));
36         dishService.addDish(new Dish( id: 12, name: "Insalata Mista", category: "Veg", price: 10.0));
37         dishService.addDish(new Dish( id: 13, name: "Osso Buco", category: "Non-veg", price: 25.0));
38         dishService.addDish(new Dish( id: 14, name: "Cannoli", category: "Veg", price: 8.0));
39         dishService.addDish(new Dish( id: 15, name: "Fettuccine Alfredo", category: "Veg", price: 18.0));
40         dishService.addDish(new Dish( id: 16, name: "Vitello Tonnato", category: "Non-veg", price: 20.0));
41         dishService.addDish(new Dish( id: 17, name: "Arancini", category: "Veg", price: 12.0));
42         dishService.addDish(new Dish( id: 18, name: "Saltimbocca", category: "Non-veg", price: 22.0));
```

---

## *STEP 9 – Front-end Pages*

---

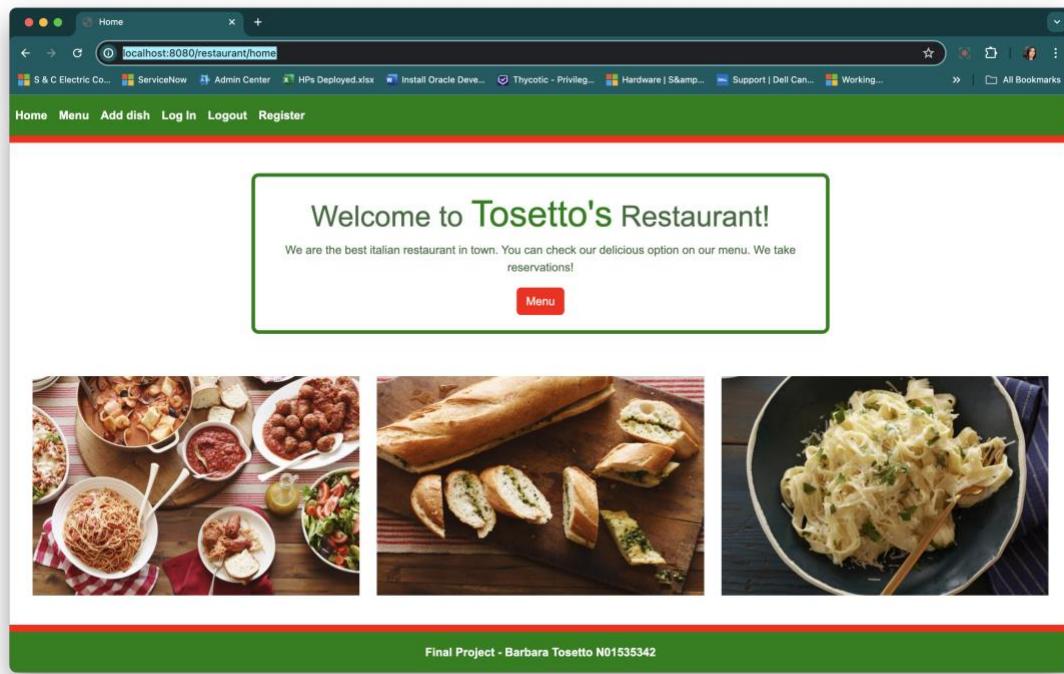
The front-end of this Spring Boot application is structured to provide a dynamic and user-friendly interface, primarily managed through Thymeleaf templates located in the `templates` directory. These templates are HTML files enriched with Thymeleaf syntax, allowing for server-side rendering of dynamic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

```
<> home.html <
1  <!DOCTYPE html>
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <meta charset="UTF-8">
5      <title>Home</title>
6      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoI6uLRA9TneNEoa7RxnatjcdSCnG" type="text/css">
7      <link rel="stylesheet" href="/home.css" type="text/css">
8  </head>
9  <body>
10     <div class="main">
11         <header th:include="@@/fragments/header.html">
12     </header>
13     <div class="content">
14         <div class="card">
15             <div class="card-body">
16                 <h1 class="card-title">Welcome to 
```



```

<> menu.html <
1  <!DOCTYPE html>
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <meta charset="UTF-8">
5      <title>Menus</title>
6      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha584-T3c6CoI6uLrA9TneNEo7Rxnatzc0SCm0IMXs" type="text/css"/>
7      <link rel="stylesheet" href="/home.css" type="text/css">
8  </head>
9  <body>
10
11     <div class="main">
12         <header th:include="@{/fragments/header.html}"></header>
13         <br>
14         <div class="content" style="padding: 40px">
15             <h1 class="title">Menu</h1>
16             <div class="alert alert-success" th:if="${message}" th:text="${message}"></div>
17
18             <form th:action="@{/restaurant/menu/1}" method="get">
19                 <input type="text" name="searchedCategory" placeholder="Category" />
20                 <input type="number" name="searchedPrice" placeholder="Price" />
21                 <button type="submit" class="btn btn-primary">filter</button>
22                 <a class="btn btn-primary" th:href="@{/restaurant/menu/' + ${pageNo} }">Reset</a>
23             </form>
24
25             <table class="table table-striped">
26                 <thead>
27                     <tr>
28                         <th scope="col">
29                             <a th:href="@{/restaurant/menu/' + ${currentPage} + '?sortField=id&sortDirection=' + ${reverseSortDirection} }">Id</a>
30                         </th>
31                         <th scope="col">
32                             <a th:href="@{/restaurant/menu/' + ${currentPage} + '?sortField=name&sortDirection=' + ${reverseSortDirection} }">Name</a>
33                         </th>
34                         <th scope="col">
35                             <a th:href="@{/restaurant/menu/' + ${currentPage} + '?sortField=category&sortDirection=' + ${reverseSortDirection} }">Category</a>
36                         </th>
37                         <th scope="col">
38                             <a th:href="@{/restaurant/menu/' + ${currentPage} + '?sortField=price&sortDirection=' + ${reverseSortDirection} }">Price</a>
39                         </th>
40                         <th scope="col" colspan="2">Operations</th>
41                     </tr>
42                 </thead>
43             </table>

```

```

<> menu.html <
1  <html lang="en" xmlns:th="http://www.thymeleaf.org">
2  <body>
3      <div class="main">
4          <div class="content" style="padding: 40px">
5              <table class="table table-striped">
6                  <tbody>
7                      <tr>
8                          <td th:text="${dish.name}"></td>
9                          <td th:text="${dish.category}"></td>
10                         <td><span th:text="${dish.price}"></td>
11                         <td>
12                             <a
13                                 th:href="@{/restaurant/admin/delete/{id}{id=${dish.id}}}"
14                                 class="btn btn-danger btn-sm">Delete
15                             </a>
16                         </td>
17                         <td>
18                             <a
19                                 th:href="@{/restaurant/admin/update/{id}{id=${dish.id}}}"
20                                 class="btn btn-update btn-sm">Update
21                             </a>
22                         </td>
23                     </tr>
24                 </tbody>
25             </table>
26
27             <div class="pagination" th:if="${totalPages != null}">
28                 <div class="pagination-info">
29                     <div>Total dishes: <span th:text="${totalItems}"></span></div>
30                     <div>Current Page: <span th:text="${currentPage}"></span></div>
31                     <div>Total pages: <span th:text="${totalPages ?: 0}"></span></div>
32                 </div>
33                 <div class="pagination-no" >
34                     <div th:each="i : ${#numbers.sequence(1, totalPages)}">
35                         <a th:href="@{/restaurant/menu/' + ${i} + '?sortField=' + ${sortField} + '&sortDirection=' + ${sortDirection} }" th:text="${i}"></a>
36                     </div>
37                 </div>
38             </div>
39             <footer th:include="@{/fragments/footer.html}">
40         </footer>
41     </div>
42 </body>
43 </html>

```

The screenshot shows a web application window titled "Menu" with the URL "localhost:8080/restaurant/menu". The page includes a navigation bar with links for Home, Menu, Add dish, Log in, Logout, and Register. Below the navigation is a search/filter section with fields for Category and Price, and buttons for Filter and Reset. A table displays a list of dishes with columns for Id, Name, Category, Price, and Operations (Delete and Update). The table contains five entries:

Id	Name	Category	Price	Operations
1	Margherita Pizza	Veg	\$17.9	<button>Delete</button> <button>Update</button>
2	Pasta Carbonara	Non-veg	\$16.5	<button>Delete</button> <button>Update</button>
3	Lasagna	Veg	\$119.0	<button>Delete</button> <button>Update</button>
4	Chicken Parmigiana	Non-veg	\$20.0	<button>Delete</button> <button>Update</button>
5	Caprese Salad	Veg	\$12.0	<button>Delete</button> <button>Update</button>

Below the table, there is a footer with the text "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.

```
<> add-dish.html <
1  <!DOCTYPE html>
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <meta charset="UTF-8">
5      <title>Add a dish</title>
6      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-TfVGFbGcaB6+PCuLuJaBVC7LuzqQ5BoOCnx8MbGbH8Spmp4AcqPS1zKgjPMEo98" crossorigin="anonymous">
7      <link rel="stylesheet" href="/home.css" type="text/css">
8  <style>
9      input{
10          display: block;
11          width: 100%;
12      }
13      .content{
14          display: flex;
15          flex-direction: column;
16          align-items: center;
17      }
18  </style>
19 </head>
20 <body>
21 <div class="main" >
22     <header th:include="@{/fragments/header.html}">
23
24     </header>
25     <div class="content" style="padding: 40px">
26         <h1 th:text="${dish.name} == null ? 'Add an item to our menu' : 'Update a menu item'"></h1>
27         <form method="POST"
28             th:action="${(dish.name) == null ? '@{/restaurant/admin/add-dish}' : '@{/restaurant/admin/update-dish}'}"
29             th:object="${dish}"
30             class="form"
31             style="...">
32             <div class="form-item">
33                 <label>Id</label>
34                 <input type="text" th:field="*{id}">
35             </div>
36             <div class="form-item">
37                 <label>Name</label>
38                 <input type="text" th:field="*{name}">
39             </div>
40             <div class="form-item">
41                 <label>Category</label>
42                 <input type="text" th:field="*{category}">
43             </div>
44             <div class="form-item">
45                 <label>Price</label>
46                 <input type="number" th:field="*{price}">
47             </div>
48             <button type="submit" class="btn btn-primary mt-2"
49                 th:text="${(dish.name) == null ? 'Add' : 'Update'}">
50             </button>
51         </form>
52         <div th:if="${error}" class="alert alert-danger mt-3">
53             <span th:text="${error}"></span>
54         </div>
55         <div th:include="@{/fragments/footer.html}">
56
57         </div>
58     </div>
59     <footer th:include="@{/fragments/footer.html}">
60
61         </footer>
62     </div>
63     </body>
64 </html>
```

Add a dish

localhost:8080/restaurant/admin/add-dish

S & C Electric Co... ServiceNow Admin Center HPs Deployed.xlsx Install Oracle Dev... Thycotic - Privileg... Hardware | S&C Support | Del Can... Working... IT - My S&C IT Standard Work Institution Page CA-L22-0182 - M...

Home Menu Add dish Log In Logout Register

### Add an item to our menu

ID

Name

Category

Price

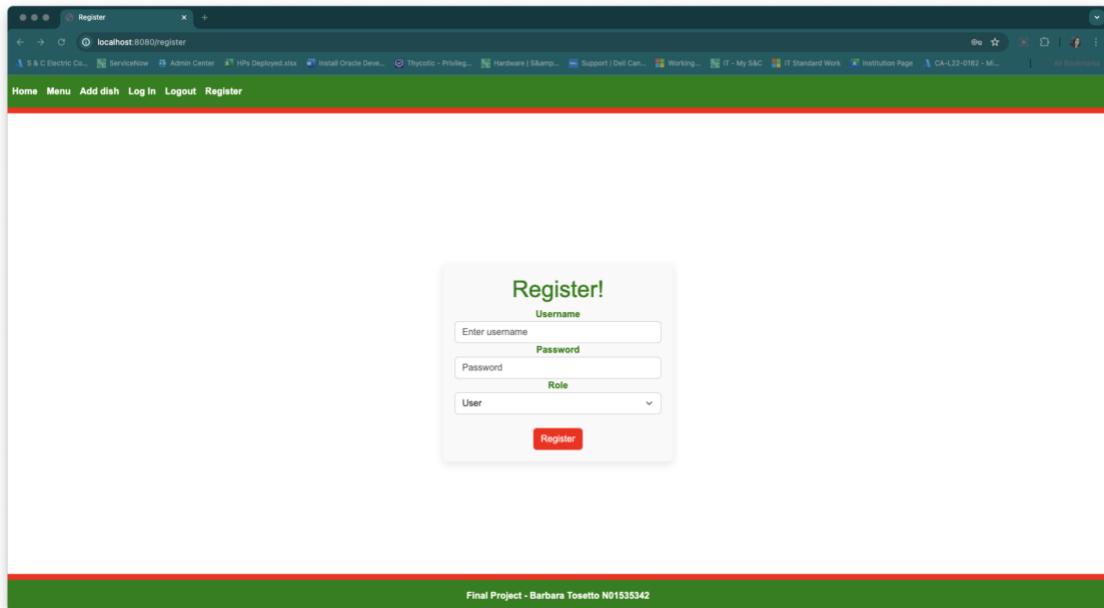
**Add**

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.

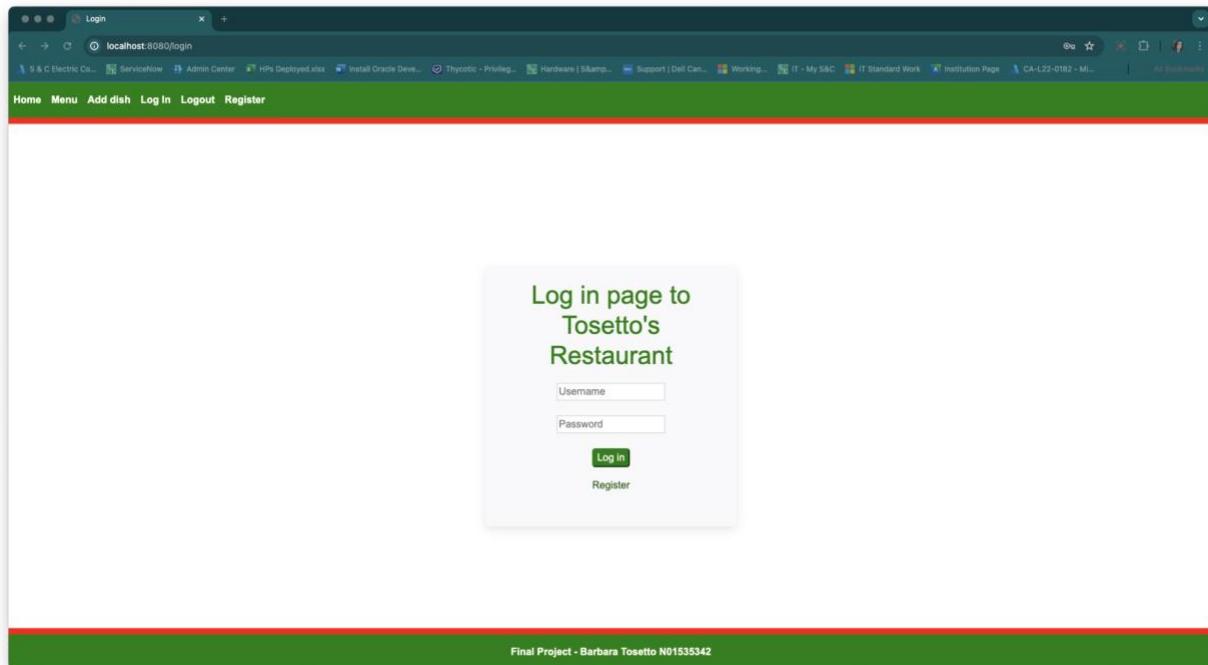
```
<> register.html <
1  <!DOCTYPE html>
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <meta charset="UTF-8">
5      <title>Register</title>
6      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
7          integrity="sha384-T3c6Colu1rAVineNca/RknatzjUScm0lMxxSR1gAsxEV/UwYKZMPK8H2HN" crossorigin="anonymous">
8      <link rel="stylesheet" href="/home.css" type="text/css">
9      <style></style>
10 </head>
11 <body>
12     <header th:include="@{/fragments/header.html}">
13     </header>
14     <div class="content">
15         <div class="main">
16             <h1>Register!</h1>
17             <div class="alert alert-info" roles="alert" th:if="${message}">
18                 <span th:text="${message}"></span>
19             </div>
20             <form th:object="${user}" th:action="@{/register}" method="post">
21                 <div class="form-group">
22                     <label for="username">Username</label>
23                     <input type="text" class="form-control" id="username" placeholder="Enter username" th:field="*{username}" required>
24                 </div>
25                 <div class="form-group">
26                     <label for="password">Password</label>
27                     <input type="password" class="form-control" id="password" placeholder="Password" th:field="*{password}" required>
28                 </div>
29                 <div class="form-group">
30                     <label for="role">Role</label>
31                     <select class="form-select" aria-label="Role" id="role" th:field="*{role}" required>
32                         <option value="USER">User</option>
33                         <option value="ADMIN">Admin</option>
34                     </select>
35                 </div>
36                 <br>
37                 <button type="submit" class="btn btn-primary">Register</button>
38             </form>
39         </div>
40         <div th:include="@{/fragments/footer.html}"></div>
41     </div>
42 </body>
```



```

<> 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-T3c6CoIi6uLrA9TneNeoa7RxatzjcDSCmG1MxxSR16AsXEV/Dwykc2MPK8M2HN" crossorigin="anonymous">
8       <link rel="stylesheet" href="/home.css" type="text/css">
9       <style></style>
10      </head>
11      <body>
12          <header th:include="@{/fragments/header.html}"></header>
13          <div class="content">
14              <div class="main" >
15                  <h1>Log in page to <span th:text="${restaurantName}"> </span> Restaurant</h1>
16                  <div class="alert alert-info" role="alert" th:if="${message}">
17                      <span th:text="${message}"></span>
18                  </div>
19                  <div th:if="${param.error}" class="alert alert-danger">
20                      Invalid username and password.
21                  </div>
22                  <div th:if="${param.logout}" class="alert alert-warning">
23                      You have been logged out.
24                  </div>
25                  <form th:action="@{/login}" method="post">
26                      <div>
27                          <input type="text" name="username" placeholder="Username"/>
28                      </div>
29                      <br>
30                      <div>
31                          <input type="password" name="password" placeholder="Password"/>
32                      </div>
33                      <br>
34                      <input type="submit" value="Log in" />
35                  </form>
36                  <div class="alert">
37                      <a class="nav-link" th:href="@{/register}">Register</a>
38                  </div>
39                  <footer th:include="@{/fragments/footer.html}"></footer>
40              </div>
41          </div>
42      </body>
43  html : body > div.content

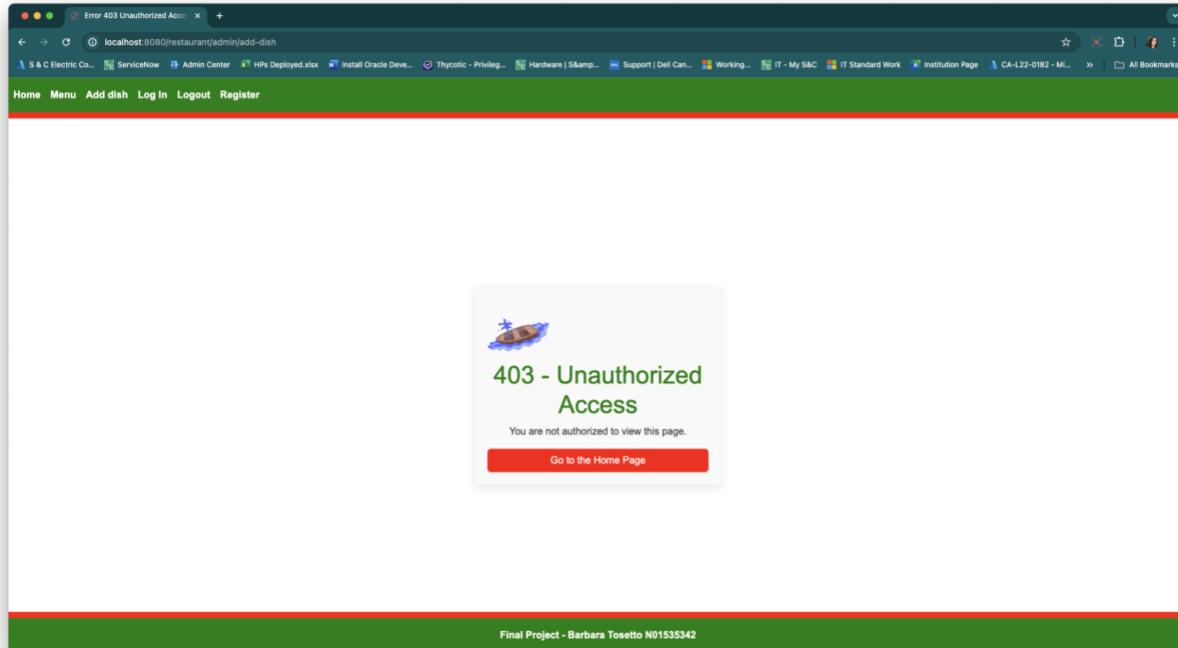
```



```

<> error.html <
1   <!DOCTYPE html>
2   <html lang="en" xmlns:th="http://www.thymeleaf.org">
3   <head>
4       <meta charset="UTF-8">
5       <title>Error 403 Unauthorized Access</title>
6       <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
7             integrity="sha384-T3c6CoI6uLr9TneNEoa7RxatzcDCmG1MXSR16AsXEV/Dwykc2MPK8M2HN" crossorigin="anonymous">
8       <link rel="stylesheet" href="/home.css" type="text/css">
9   >     <style>~~~</style>
10  </head>
11  <body>
12  <header th:include="@{/fragments/header.html}">
13  </header>
14  <div class="content">
15      <div class="main">
16          
17          <h1>403 - Unauthorized Access</h1>
18          <p>You are not authorized to view this page.</p>
19          <a href="/restaurant/home" class="btn btn-primary">Go to the Home Page</a>
20      </div>
21  </div>
22  <footer th:include="@{/fragments/footer.html}"></footer>
23  </body>
24 </html>
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61

```



## Reusable Fragments

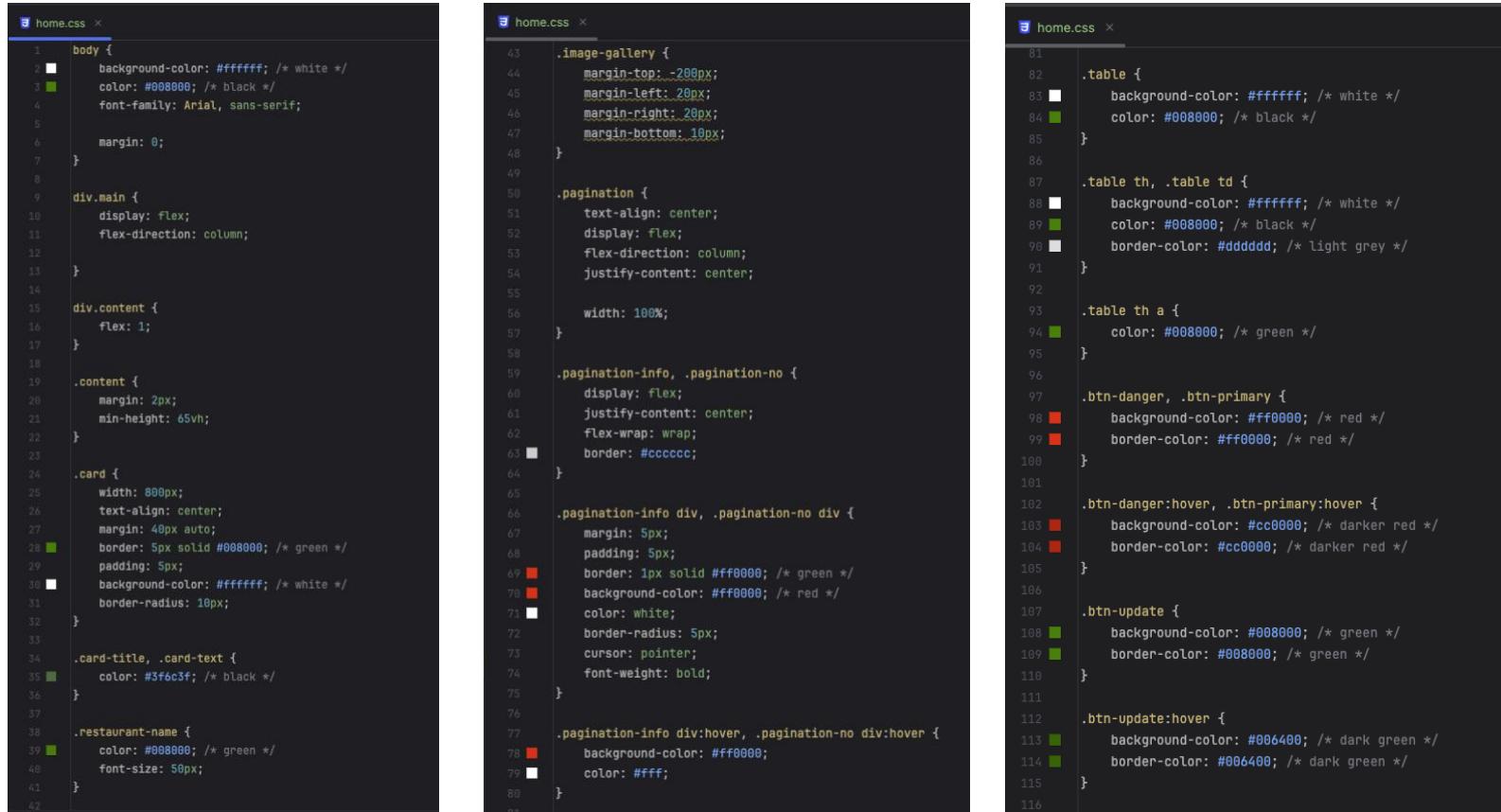
Such as `header.html` and `footer.html` in the `fragments` folder, which are incorporated across various templates to ensure a consistent layout and reduce redundancy.

```
<> footer.html ×  
1 ■ <div class="bg-grey d-flex justify-content-around pt-3 pb-3" style="color: #ffffff;  
2 ■   background-color: #008000;  
3 ■   border-top: 10px solid red;  
4 ■   font-weight: bold;  
5 ■   text-align: center;  
6 ■   position: fixed;  
7 ■   bottom: 0;  
8 ■   left: 0;  
9 ■   padding: 10px 20px;  
10 ■  width: 100%;">  
11 ■  <span>Final Project - Barbara Tosetto N01535342</span>  
12 ■ </div>  
13 ■
```

```
<> header.html ×  
1 ■ ■ <nav class="navbar navbar-expand-lg" style="background-color: #008000; /* white */ border-bottom: 10px solid red; /* red */"  
2 ■ <div class="collapse navbar-collapse" id="navbarSupportedContent">  
3 ■   <ul class="navbar-nav mr-auto">  
4 ■     <li class="nav-item">  
5 ■       <a class="nav-link" th:href="@{/restaurant/home}" style="color: #ffffff; font-weight: bold;">Home</a>  
6 ■     </li>  
7 ■     <li class="nav-item">  
8 ■       <a class="nav-link" th:href="@{/restaurant/menu/1}" style="color: #ffffff; font-weight: bold;">Menu</a>  
9 ■     </li>  
10 ■    <li class="nav-item">  
11 ■      <a class="nav-link" th:href="@{/restaurant/admin/add-dish}" style="color: #ffffff; font-weight: bold;">Add dish</a>  
12 ■    </li>  
13 ■    <li class="nav-item">  
14 ■      <a class="nav-link" th:href="@{/login}" style="color: #ffffff; font-weight: bold;">Log In</a>  
15 ■    </li>  
16 ■    <li class="nav-item">  
17 ■      <a class="nav-link" th:href="@{/logout}" style="color: #ffffff; font-weight: bold;">Logout</a>  
18 ■    </li>  
19 ■    <li class="nav-item">  
20 ■      <a class="nav-link" th:href="@{/register}" style="color: #ffffff; font-weight: bold;">Register</a>  
21 ■    </li>  
22 ■  </ul>  
23 ■ </div>  
24 ■ </nav>  
25 ■
```

## 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.

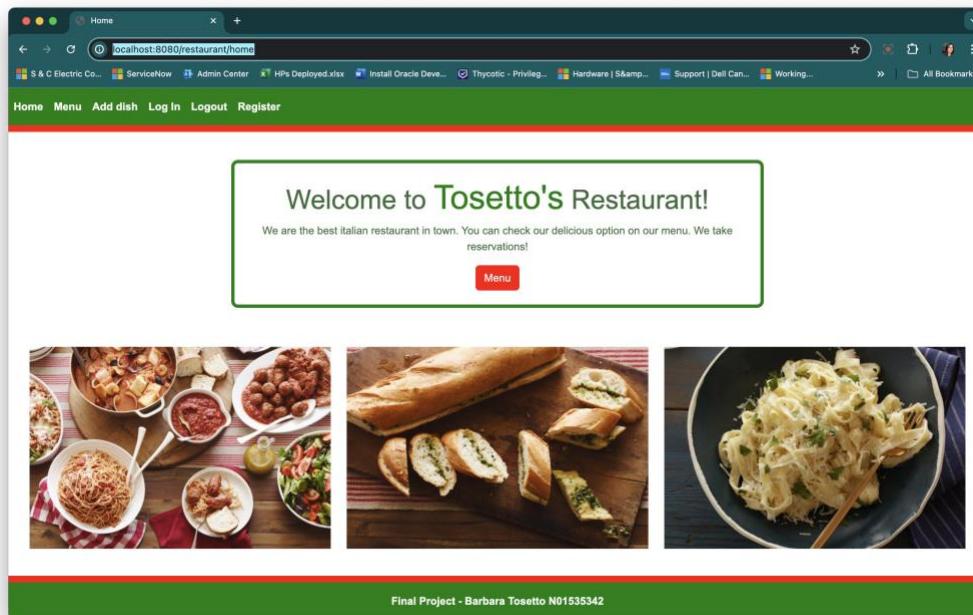


The image displays three separate code editor windows, each showing a snippet of CSS code from a file named `home.css`. The code is color-coded with syntax highlighting.

- Left Window:** Contains the beginning of the `home.css` file, including the `body`, `div.main`, `div.content`, `.content`, `.card`, `.card-title`, and `.card-text` rules.
- Middle Window:** Contains the `.image-gallery`, `.pagination`, `.pagination-info`, and `.pagination-no` rules.
- Right Window:** Contains the `.table`, `.table th`, `.table td`, `.table th a`, `.btn-danger`, `.btn-primary`, `.btn-danger:hover`, `.btn-primary:hover`, `.btn-update`, and `.btn-update:hover` rules.

## Application Screenshots

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

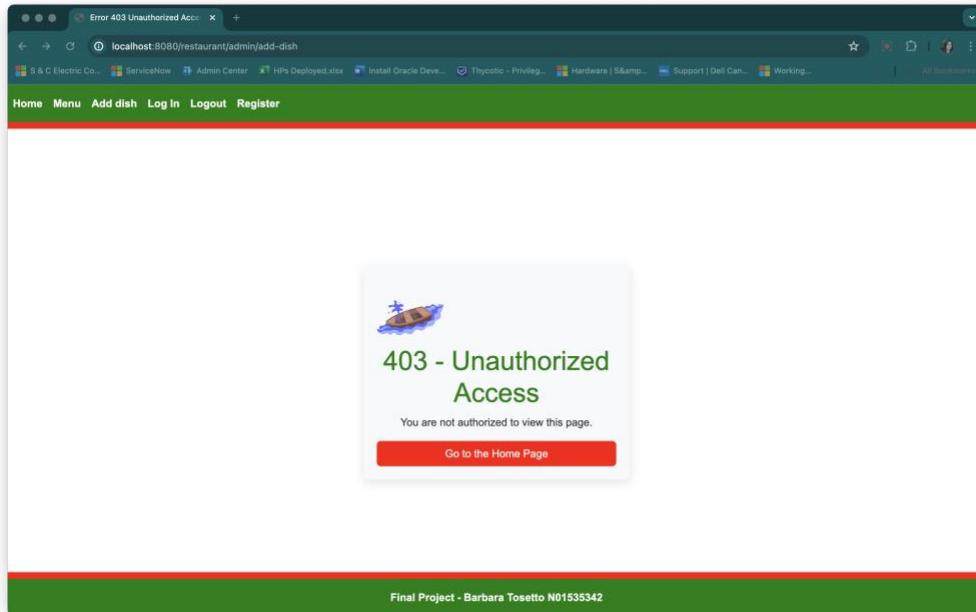


Menu: <http://localhost:8080/restaurant/menu/1>

The screenshot shows a web browser window for 'localhost:8080/restaurant/menu/1'. The title bar says 'Menu'. The page has a green header with 'Home' and other navigation links: 'Menu', 'Add dish', 'Log In', 'Logout', and 'Register'. Below the header is a section titled 'Menu' with a table. The table has columns: 'Id', 'Name', 'Category', 'Price', and 'Operations'. It lists five dishes: Margherita Pizza (Veg, \$17.9), Pasta Carbonara (Non-veg, \$18.5), Lasagna (Veg, \$119.0), Chicken Parmigiana (Non-veg, \$20.0), and Caprese Salad (Veg, \$12.0). Each row has 'Delete' and 'Update' buttons in the 'Operations' column. Below the table are buttons for 'Total dishes: 20', 'Current Page: 1', 'Total pages: 4', and a set of four small red navigation icons. At the bottom is a green footer with the text 'Final Project - Barbara Tosoetto N01535342'.

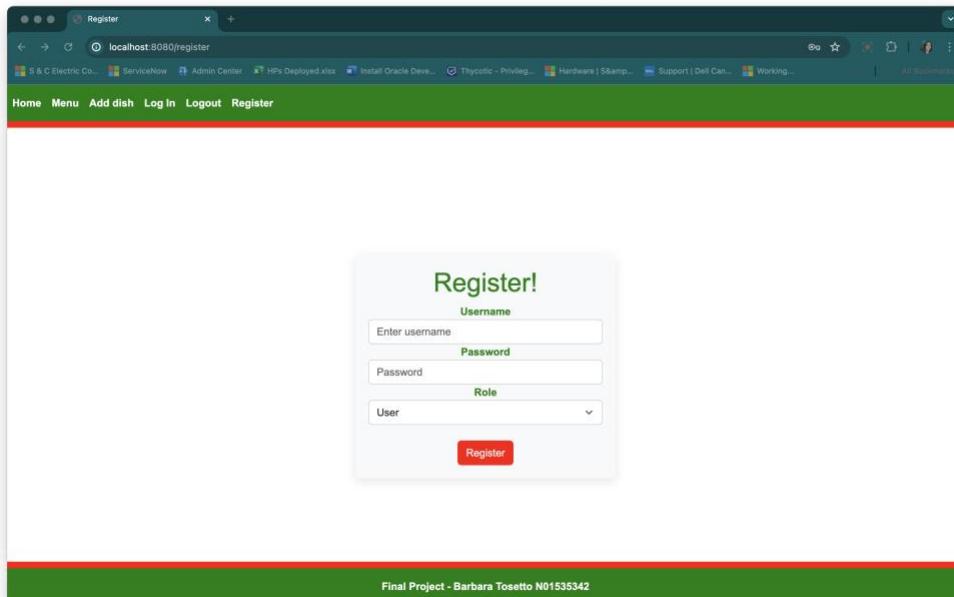
Id	Name	Category	Price	Operations
1	Margherita Pizza	Veg	\$17.9	<button>Delete</button> <button>Update</button>
2	Pasta Carbonara	Non-veg	\$18.5	<button>Delete</button> <button>Update</button>
3	Lasagna	Veg	\$119.0	<button>Delete</button> <button>Update</button>
4	Chicken Parmigiana	Non-veg	\$20.0	<button>Delete</button> <button>Update</button>
5	Caprese Salad	Veg	\$12.0	<button>Delete</button> <button>Update</button>

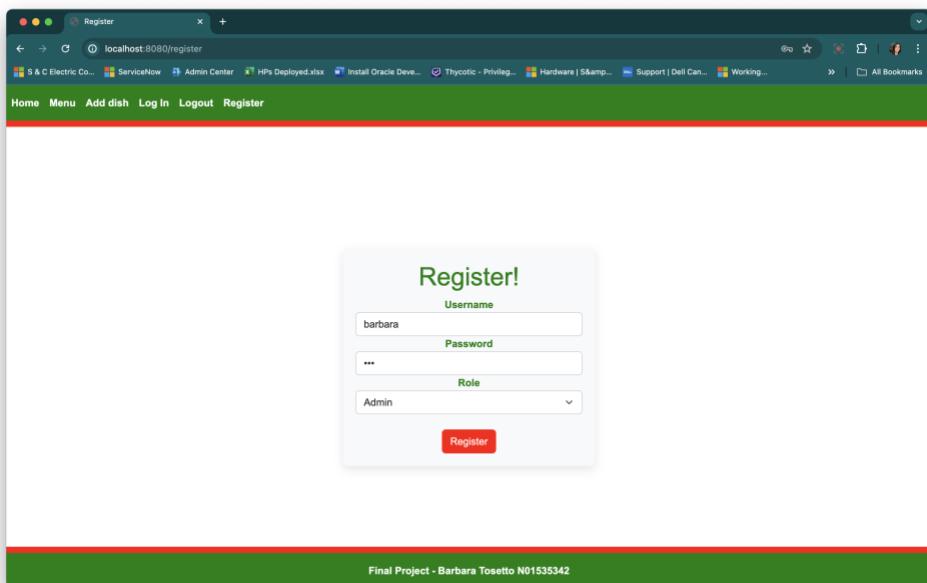
But if the user is not logged in and navigates to “Add Dish” page, or tries to perform “Delete” or “Update”, they will be redirected to an error page:



The user can Register as “Admin”, or “User” in order to have access to all features of the application.

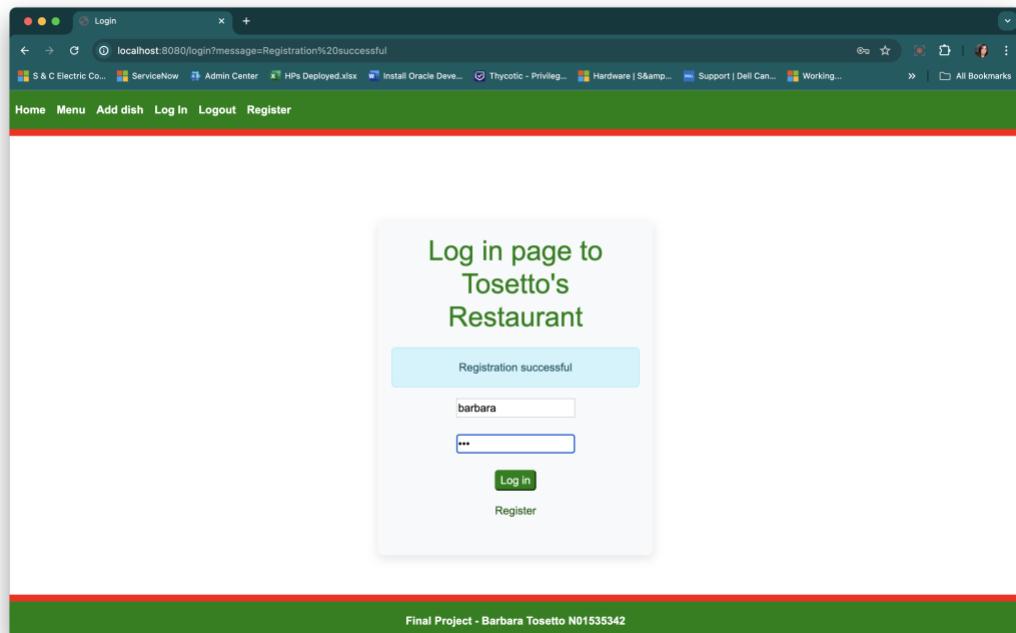
Register Page: <http://localhost:8080/register>





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>

The screenshot shows a web browser window with the title 'Add a dish'. The URL in the address bar is 'localhost:8080/restaurant/admin/add-dish'. The page has a green header bar with the following navigation links: Home, Menu, Add dish, Log In, Logout, and Register. Below the header, the main content area has a title 'Add an item to our menu'. It contains four input fields: 'ID' (with value '0'), 'Name' (empty), 'Category' (empty), and 'Price' (empty). At the bottom of the form is a red 'Add' button. The footer of the page is a dark green bar with the text 'Final Project - Barbara Tosetto N01535342'.

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

The screenshot shows a web browser window with the title 'Menu'. The URL in the address bar is 'localhost:8080/restaurant/menu'. The page has a green header bar with the following navigation links: Home, Menu, Add dish, Log In, Logout, and Register. Below the header, the main content area has a title 'Menu'. It features a search/filter section with 'Category' and 'Price' dropdowns, and 'Filter' and 'Reset' buttons. Below this is a table listing five dishes:

ID	Name	Category	Price	Operations
1	Margherita Pizza	Veg	\$17.9	<button>Delete</button> <button>Update</button>
2	Pasta Carbonara	Non-veg	\$18.5	<button>Delete</button> <button>Update</button>
3	Lasagna	Veg	\$119.0	<button>Delete</button> <button>Update</button>
4	Chicken Parmigiana	Non-veg	\$20.0	<button>Delete</button> <button>Update</button>
5	Caprese Salad	Veg	\$12.0	<button>Delete</button> <button>Update</button>

At the bottom of the table, there are buttons for 'Total dishes: 20', 'Current Page: 1', 'Total pages: 4', and a set of page navigation icons (1, 2, 3, 4).

The footer of the page is a dark green bar with the text 'Final Project - Barbara Tosetto N01535342'.

Update a menu item

ID	<input type="text" value="1"/>
Name	<input type="text" value="Margherita Pizza"/>
Category	<input type="text" value="Veg"/>
Price	<input type="text" value="19.9"/>

**Update**

Final Project - Barbara Tosetto N01535342

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

Dish updated

Category	Price	Filter	Reset	
Id	Name	Category	Price	Operations
1	Margherita Pizza	Veg	\$19.9	<b>Delete</b> <b>Update</b>
2	Pasta Carbonara	Non-veg	\$18.5	<b>Delete</b> <b>Update</b>
3	Lasagna	Veg	\$119.0	<b>Delete</b> <b>Update</b>
4	Chicken Parmigiana	Non-veg	\$20.0	<b>Delete</b> <b>Update</b>
5	Caprese Salad	Veg	\$12.0	<b>Delete</b> <b>Update</b>

Total dishes: 20 Current Page: 1 Total pages: 4  
1 2 3 4

Final Project - Barbara Tosetto N01535342

And then the first item was deleted:

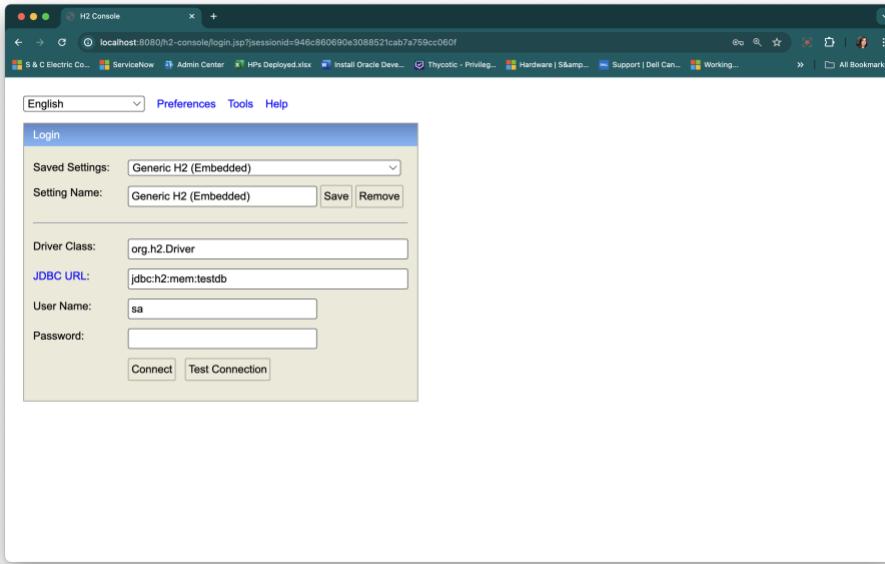
The screenshot shows a web browser window titled 'Menu' at the URL [localhost:8080/restaurant/menu/](http://localhost:8080/restaurant/menu/). A message bar at the top says 'Dish deleted successfully'. Below it is a table of dishes with columns: Id, Name, Category, Price, and Operations (Delete, Update). The first row (Id 2) has been deleted, leaving a gap. The table includes pagination controls: 'Total dishes: 10', 'Current Page: 1', 'Total pages: 4', and a set of four red navigation buttons.

ID	Name	Category	Price	Operations
2	Pasta Carbonara	Non-veg	\$18.5	<button>Delete</button> <button>Update</button>
3	Lasagna	Veg	\$119.0	<button>Delete</button> <button>Update</button>
4	Chicken Parmigiana	Non-veg	\$20.0	<button>Delete</button> <button>Update</button>
5	Caprese Salad	Veg	\$12.0	<button>Delete</button> <button>Update</button>
6	Polenta	Vegan	\$10.0	<button>Delete</button> <button>Update</button>

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

The screenshot shows a web browser window titled 'Login' at the URL [localhost:8080/login](http://localhost:8080/login). A message bar at the top says 'You have been logged out'. Below it is a login form with fields for 'Username' and 'Password', and buttons for 'Log in' and 'Register'. The footer of the page reads 'Final Project - Barbara Tosetto N01535342'.

## Access to H2-console:



The screenshot shows the H2 Console results page. On the left, there's a sidebar with database objects: 'pjbc:0:1', 'DISH', 'MY\_USER', 'INFORMATION\_SCHEMA', 'Sequences', and 'Users'. The main area has a toolbar with 'Run Selected', 'Auto complete', and 'SQL statement'. A query 'SELECT \* FROM DISH' is run, and the results are displayed in a table. The table has columns 'ID', 'PRICE', 'CATEGORY', and 'NAME'. The data is as follows:

ID	PRICE	CATEGORY	NAME
2	16.5	Non-veg	Pasta Carbonara
3	18.0	Veg	Lasagna
4	20.0	Non-veg	Chicken Parmigiana
5	12.0	Veg	Coprice Salad
6	10.0	Vegan	Poison
7	19.0	Veg	Risotto
8	12.0	Non-veg	Antipasto
9	9.0	Veg	Tramezzini
10	18.0	Non-veg	Pepperoni Pizza
11	10.0	Veg	Insalata Mista
12	25.0	Non-veg	Ossobuco
13	18.0	Veg	Fettuccine Alfredo
14	20.0	Non-veg	Vitello Tonnato
15	12.0	Veg	Aranzini
16	22.0	Non-veg	Salsimbocca
17	9.0	Vegan	Minestrone
18	16.0	Veg	Gnocchi
19	30.0	Non-veg	Bistecca Fiorentina
20	10.0	Veg	Panna Cotta

Below the table, it says '(19 rows, 6 ms)'. There's also an 'Edit' button at the bottom.

H2 Console

localhost:8080/h2-console/login.do?sessionid=946c860690e3088521cab7a759cc060f

Auto commit | Max rows: 1000 | Auto complete | SQL statement:

Run Selected Auto complete Clear SQL statement:

SELECT \* FROM MY\_USER;

ID	PASSWORD	ROLE	USERNAME
1	\$2a\$10\$ahM3hybw7LjVHC6Ln65cuN.dJB0XV1S9wOJS2dLYw.o7zctoGC	ADMIN	user
2	\$2a\$10\$4gPfr4c74A2nwf6llyfpuJDFBNuJ.wd.oh95TfB6CQE0sdm	USER	haha
3	\$2a\$10\$ozq2AMjNIA3t3mvPH22OrY1q1TT7sEMFWU6nQ4h0lqv8S2	ADMIN	barbara

(3 rows, 2 ms)

Edit

The screenshot shows the H2 Console interface running on a local host at port 8080. The URL in the address bar is localhost:8080/h2-console/login.do?sessionid=946c860690e3088521cab7a759cc060f. The main area displays a SQL query: 'SELECT \* FROM MY\_USER;'. Below the query, the results are shown in a table with four columns: ID, PASSWORD, ROLE, and USERNAME. There are three rows of data. The first row has ID 1, PASSWORD '\$2a\$10\$ahM3hybw7LjVHC6Ln65cuN.dJB0XV1S9wOJS2dLYw.o7zctoGC', ROLE 'ADMIN', and USERNAME 'user'. The second row has ID 2, PASSWORD '\$2a\$10\$4gPfr4c74A2nwf6llyfpuJDFBNuJ.wd.oh95TfB6CQE0sdm', ROLE 'USER', and USERNAME 'haha'. The third row has ID 3, PASSWORD '\$2a\$10\$ozq2AMjNIA3t3mvPH22OrY1q1TT7sEMFWU6nQ4h0lqv8S2', ROLE 'ADMIN', and USERNAME 'barbara'. The table has a header row with column names and three data rows. At the bottom left of the table, there is an 'Edit' button.