Class Link :- <https://meet.google.com/avn-hmcc-zqp> { Mon - Fri } – { 7.00 pm - 8.30 pm }

Mentor Linkedin Profile URL <https://www.linkedin.com/in/anil-boppuri-7665132a?utm_source=share&utm_campaign=share_via&utm_content=profile&utm_medium=android_app>

<https://start.spring.io/>

<https://docs.spring.io/spring-framework/reference/index.html>

When Java is so powerful, why do we need spring and spring boot ?

10 Differences between Java and SpringBoot?

Here are the main SDLC model headings:

1. **Waterfall Model**
2. **V-Model (Verification and Validation)**
3. **Incremental Model**
4. **Spiral Model**
5. **Agile Model**
6. **Scrum Model**
7. **Kanban Model**
8. **RAD (Rapid Application Development) Model**
9. **DevOps Model**
10. **Prototyping Model**
11. **Big Bang Model**
12. **Iterative Model**

<https://www.agileforvalue.com/blog/start-your-first-scrum-project-in-12-1-steps>

<https://github.com/topics/projects>

Team members :-

[prashantjadhav4april@gmail.com](mailto:prashantjadhav4april@gmail.com)

[rohit.22210261@viit.ac.in](mailto:rohit.22210261@viit.ac.in)

[santhoshkaruppusamy5@gmail.com](mailto:santhoshkaruppusamy5@gmail.com)

[ashokbethamcharla29@gmail.com](mailto:ashokbethamcharla29@gmail.com)

Google Chat Space Link :- <https://chat.google.com/room/AAAAt_ssfME?cls=7>

Project has objectives, which have to be achieved at given cost, within given time and at an accepted quality.

Project :- Food Express App

LucidChart – Tool for UML Diagrams

<https://blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams/>

Spring is all about dependency management

Google Meeting Link for Team : <https://meet.google.com/jqw-woon-amz> { Mon, Wed, Fri }-{ 6.30 pm - 7.00 pm}

Maven File Download link :- <https://maven.apache.org/download.cgi>

Also download the file :- apache-maven-3.9.9-bin.zip

<https://mvnrepository.com/repos/central>

<beans xmlns="<http://www.springframework.org/schema/beans>"

xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

xsi:schemaLocation="<http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans.xsd>"></beans>

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

Analysis of requirements (10)

Development of functionalities as required(25) - split it module wise

Testing of functionality (12)

Deployment and functioning/working of the project as per needs (20)

Documentation(8)

total (100)

2 weeks

<http://localhost:8080/app/greet?emailParam=springboard4245@gmail.com>

Creating a **Tasty Express Food Web App** using Java with **Spring Boot**, **Thymeleaf**, and **SQL** in **VS Code** involves multiple steps, including setting up your environment, creating a Spring Boot application, integrating Thymeleaf templates, and using a relational database like MySQL or H2 for data persistence. Below is a step-by-step guide on how to create this application:

### **Prerequisites:**

1. **VS Code** installed with Java extensions (Java, Spring Boot, etc.)
2. **JDK 11 or higher** installed (ensure JAVA\_HOME is set)
3. **Maven** installed for dependency management
4. **MySQL** (or another SQL database like H2) installed for persistence

### **Step 1: Set Up the Spring Boot Project**

1. **Create a new Spring Boot project**: You can use Spring Initializr to set up your project. Go to [Spring Initializr](https://start.spring.io/), and configure the project as follows:  
   * Project: Maven Project
   * Language: Java
   * Spring Boot Version: 2.x or the latest
   * Group: com.tastyexpress
   * Artifact: tastyexpress
   * Dependencies:
     + Spring Web
     + Thymeleaf
     + Spring Data JPA
     + MySQL Driver (or H2 for simplicity)
     + Spring Boot DevTools (optional)
2. **Download the project** and unzip it. Open the project in **VS Code**.

### **Step 2: Set Up Application Properties**

In the src/main/resources/application.properties file, configure the connection to the MySQL database.

spring.datasource.url=jdbc:mysql://localhost:3306/tastyexpress\_db

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.thymeleaf.cache=false

Replace the username and password with your own MySQL credentials.

### **Step 3: Create the Database and Table in MySQL**

Log in to your MySQL console and create the database:

CREATE DATABASE tastyexpress\_db;

Create a table for storing food items. Here's a simple food table:

CREATE TABLE food (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

description TEXT,

price DECIMAL(10, 2) NOT NULL,

image\_url VARCHAR(255)

);

### **Step 4: Add Entity Class**

Create an entity class for the food table in the src/main/java/com/tastyexpress directory.

package com.tastyexpress.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Food {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String description;

private Double price;

private String imageUrl;

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public Double getPrice() {

return price;

}

public void setPrice(Double price) {

this.price = price;

}

public String getImageUrl() {

return imageUrl;

}

public void setImageUrl(String imageUrl) {

this.imageUrl = imageUrl;

}

}

### **Step 5: Create Repository**

Create a repository interface that extends JpaRepository for database operations:

package com.tastyexpress.repository;

import com.tastyexpress.model.Food;

import org.springframework.data.jpa.repository.JpaRepository;

public interface FoodRepository extends JpaRepository<Food, Long> {

}

### **Step 6: Create Service Layer**

Create a service class to handle business logic. This will interact with the repository.

package com.tastyexpress.service;

import com.tastyexpress.model.Food;

import com.tastyexpress.repository.FoodRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class FoodService {

private final FoodRepository foodRepository;

@Autowired

public FoodService(FoodRepository foodRepository) {

this.foodRepository = foodRepository;

}

public List<Food> getAllFood() {

return foodRepository.findAll();

}

public Food getFoodById(Long id) {

return foodRepository.findById(id).orElse(null);

}

public void saveFood(Food food) {

foodRepository.save(food);

}

}

### **Step 7: Create Controller**

Create a controller class to map HTTP requests to services and return views using Thymeleaf.

package com.tastyexpress.controller;

import com.tastyexpress.model.Food;

import com.tastyexpress.service.FoodService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import java.util.List;

@Controller

public class FoodController {

private final FoodService foodService;

@Autowired

public FoodController(FoodService foodService) {

this.foodService = foodService;

}

@GetMapping("/")

public String homePage(Model model) {

List<Food> foodList = foodService.getAllFood();

model.addAttribute("foodList", foodList);

return "index";

}

}

### **Step 8: Create Thymeleaf Templates**

In the src/main/resources/templates/ directory, create the index.html file.

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Tasty Express</title>

</head>

<body>

<h1>Welcome to Tasty Express!</h1>

<div th:each="food : ${foodList}">

<h2 th:text="${food.name}">Food Name</h2>

<p th:text="${food.description}">Food Description</p>

<p th:text="${food.price}">Food Price</p>

<img th:src="${food.imageUrl}" alt="Food Image" />

</div>

</body>

</html>

This template displays a list of foods and their details using Thymeleaf.

### **Step 9: Run the Application**

1. Open the **Terminal** in VS Code.
2. Run the application using Maven:

mvn spring-boot:run

1. Once the app is running, visit http://localhost:8080 in your browser, and you should see a list of food items fetched from your MySQL database.

### **Step 10: Additional Features (Optional)**

* **Create a food item**: Add a form to create new food items and save them to the database.
* **Update/Delete functionality**: Add features to update or delete food items.
* **Authentication**: Implement user authentication for restaurant admins.

### **Conclusion**

You've now set up a **Tasty Express** web application using **Spring Boot**, **Thymeleaf**, and **SQL**. This app lists food items stored in a MySQL database and renders them using Thymeleaf templates.

If you'd like to enhance the project, you could add more features such as user authentication, food categorization, shopping cart integration, and more.

Certainly! Here are several additional features you can implement to enhance your **Tasty Express Food Web App**. These will allow you to expand the functionality, making your app more interactive, dynamic, and user-friendly.

### **1. User Authentication (Login and Registration)**

Allow users to register and log in to place orders or manage food items.

#### **Steps:**

1. **Add Spring Security Dependency**: Update your pom.xml to include Spring Security for authentication.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

1. **Create User Entity**: Create a User entity for storing user credentials.

package com.tastyexpress.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import java.util.List;

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String password;

private String role; // Can be "USER", "ADMIN", etc.

// Getters and Setters

}

1. **Create UserRepository**: Create a repository for the User entity.

package com.tastyexpress.repository;

import com.tastyexpress.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

User findByUsername(String username);

}

1. **Configure Spring Security**: Create a SecurityConfig class to set up authentication and authorization.

package com.tastyexpress.config;

import com.tastyexpress.service.UserDetailsServiceImpl;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

private final UserDetailsServiceImpl userDetailsService;

public SecurityConfig(UserDetailsServiceImpl userDetailsService) {

this.userDetailsService = userDetailsService;

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/login", "/register").permitAll()

.antMatchers("/admin/\*\*").hasRole("ADMIN")

.anyRequest().authenticated()

.and()

.formLogin().loginPage("/login").permitAll()

.and()

.logout().permitAll();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

1. **Create Login and Registration Pages**: Use Thymeleaf to create registration and login pages for users. Make use of Spring Security's built-in login handling.
2. **UserDetailsService Implementation**: Implement UserDetailsService for loading user data from the database.

### **2. Food Management for Admins (CRUD Operations)**

Allow admins to add, update, and delete food items.

#### **Steps:**

1. **Admin Food Management Controller**: Create a controller to handle CRUD operations for admins.

package com.tastyexpress.controller;

import com.tastyexpress.model.Food;

import com.tastyexpress.service.FoodService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.\*;

@Controller

@RequestMapping("/admin")

public class AdminController {

private final FoodService foodService;

@Autowired

public AdminController(FoodService foodService) {

this.foodService = foodService;

}

@GetMapping("/foods")

public String getAllFoods(Model model) {

model.addAttribute("foodList", foodService.getAllFood());

return "admin/foods";

}

@GetMapping("/food/new")

public String createFoodForm(Model model) {

model.addAttribute("food", new Food());

return "admin/food-form";

}

@PostMapping("/food/save")

public String saveFood(@ModelAttribute Food food) {

foodService.saveFood(food);

return "redirect:/admin/foods";

}

@GetMapping("/food/edit/{id}")

public String editFoodForm(@PathVariable Long id, Model model) {

Food food = foodService.getFoodById(id);

model.addAttribute("food", food);

return "admin/food-form";

}

@GetMapping("/food/delete/{id}")

public String deleteFood(@PathVariable Long id) {

foodService.deleteFood(id);

return "redirect:/admin/foods";

}

}

1. **Create Food Management Views**: Create templates for managing food (listing, adding, and editing).

### **3. Shopping Cart for Users**

Allow users to add food items to a cart and view the cart.

#### **Steps:**

1. **Create Cart Entity**: Define a simple cart to hold the items.

package com.tastyexpress.model;

import java.util.List;

public class Cart {

private List<Food> items;

public List<Food> getItems() {

return items;

}

public void addItem(Food food) {

items.add(food);

}

public void removeItem(Food food) {

items.remove(food);

}

public double getTotal() {

return items.stream().mapToDouble(Food::getPrice).sum();

}

}

1. **Add Cart Controller**: Create methods for adding, removing, and viewing cart items.

@Controller

@RequestMapping("/cart")

public class CartController {

private final FoodService foodService;

private final Cart cart;

@Autowired

public CartController(FoodService foodService) {

this.foodService = foodService;

this.cart = new Cart();

}

@GetMapping("/")

public String viewCart(Model model) {

model.addAttribute("cart", cart);

return "cart/view";

}

@PostMapping("/add/{id}")

public String addItemToCart(@PathVariable Long id) {

Food food = foodService.getFoodById(id);

cart.addItem(food);

return "redirect:/cart/";

}

@PostMapping("/remove/{id}")

public String removeItemFromCart(@PathVariable Long id) {

Food food = foodService.getFoodById(id);

cart.removeItem(food);

return "redirect:/cart/";

}

}

1. **Create Cart Views**: Create templates to display the cart items and the total price.

### **4. Order Management (Place Orders and View Orders)**

Allow users to place orders, and store order details in the database for tracking.

#### **Steps:**

1. **Create Order Entity**: Create an Order entity to store order details such as the user, food items, total price, and status.

package com.tastyexpress.model;

import javax.persistence.\*;

import java.util.List;

@Entity

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne

private User user;

private double totalPrice;

@ElementCollection

private List<String> foodItems;

private String status; // "PENDING", "COMPLETED", etc.

// Getters and Setters

}

1. **Order Service**: Implement a service to handle orders.

package com.tastyexpress.service;

import com.tastyexpress.model.Order;

import com.tastyexpress.repository.OrderRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class OrderService {

private final OrderRepository orderRepository;

@Autowired

public OrderService(OrderRepository orderRepository) {

this.orderRepository = orderRepository;

}

public void placeOrder(Order order) {

orderRepository.save(order);

}

public List<Order> getOrdersForUser(User user) {

return orderRepository.findByUser(user);

}

}

1. **Order Controller**: Handle order creation and order status management.

package com.tastyexpress.controller;

import com.tastyexpress.model.Order;

import com.tastyexpress.service.OrderService;

import com.tastyexpress.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.\*;

@Controller

@RequestMapping("/orders")

public class OrderController {

private final OrderService orderService;

private final UserService userService;

@Autowired

public OrderController(OrderService orderService, UserService userService) {

this.orderService = orderService;

this.userService = userService;

}

@PostMapping("/place")

public String placeOrder(@RequestParam Long userId, @RequestParam List<Long> foodIds) {

Order order = new Order();

order.setUser(userService.getUserById(userId));

order.setFoodItems(foodIds);

order.setTotalPrice(calculateTotalPrice(foodIds));

order.setStatus("PENDING");

orderService.placeOrder(order);

return "redirect:/orders/";

}

private double calculateTotalPrice(List<Long> foodIds) {

// Calculate the total price of the selected food items

return 0.0;

}

}

1. **Create Order Views**:
   * Order list for users
   * Order placement form

### **5. Payment Gateway Integration**

Implement payment functionality so users can pay for their orders using third-party payment gateways (like **Stripe** or **PayPal**).

#### **Steps:**

1. Integrate with payment APIs (Stripe, PayPal).
2. Add payment options on the checkout page.
3. Update the order status after payment confirmation.

### **Conclusion**

These features will make your **Tasty Express** web app more complete by allowing user authentication, food management by admins, shopping cart functionality, order management, and payment integration. These functionalities are important for turning your app into a full-fledged food ordering platform!