**1️⃣ Create the Spring Boot Project**

1. **Go to** [**Spring Initializr**](https://start.spring.io/)
2. **Set the Project Configuration:**
   * **Project: Maven**
   * **Language: Java**
   * **Spring Boot Version: Latest stable version (e.g., 3.x)**
   * **Group: com.example**
   * **Artifact: riskrating**
   * **Name: risk-rating-api**
   * **Package Name: com.example.riskrating**
   * **Packaging: Jar**
   * **Java Version: 17 or later**
3. **Add Dependencies:**
   * **✅ Spring Web (For REST API)**
   * **✅ Spring Security (For Authentication)**
   * **✅ Spring Data JPA (For Database Connection)**
   * **✅ MySQL Driver (For Database)**
   * **✅ Lombok (To Reduce Boilerplate Code)**
   * **✅ Spring Boot Validation (For Request Validation)**
   * **✅ JWT (Manually added for authentication)**
4. **Click "Generate" and download the project.**
5. **Extract the ZIP file and open it in IntelliJ IDEA, Eclipse, or VS Code.**

**Step 3: Create Each Package & Class**

**📍 Navigate to src/main/java/com/example/riskrating/ and create the following packages.**

**1️⃣ model/ (Entity Classes for Database Tables)**

**📌 Location: src/main/java/com/example/riskrating/model/  
📌 Stores database table structure using JPA entities.**

**Create the following entity classes inside the model package:**

* **User.java (Broker Login)**
* **Location.java (FEMA Risk Factors)**
* **Property.java (Building Characteristics)**
* **InsurancePolicy.java (Policy Coverage)**
* **PremiumCalculation.java (Final Premium Calculation)**

**2️⃣ repository/ (Database Access)**

**📌 Location: src/main/java/com/example/riskrating/repository/  
📌 Handles interaction with the database.**

**Create the following repository interfaces inside repository package:**

* **UserRepository.java**
* **LocationRepository.java**
* **PropertyRepository.java**
* **InsurancePolicyRepository.java**
* **PremiumCalculationRepository.java**

**3️⃣ service/ (Business Logic Layer)**

**📌 Location: src/main/java/com/example/riskrating/service/  
📌 Contains business logic and interacts with the repository layer.**

**Create the following service classes inside service package:**

* **UserService.java**
* **LocationService.java**
* **PropertyService.java**
* **InsurancePolicyService.java**
* **PremiumCalculationService.java**

**4️⃣ controller/ (REST API Controllers)**

**📌 Location: src/main/java/com/example/riskrating/controller/  
📌 Exposes REST API endpoints for frontend interaction.**

**Create the following controller classes inside controller package:**

* **AuthController.java (User Registration & Login)**
* **LocationController.java (FEMA Risk Data)**
* **PropertyController.java (Building Information)**
* **InsurancePolicyController.java (Coverage Information)**
* **PremiumCalculationController.java (Premium Calculation API)**

**5️⃣ dto/ (Data Transfer Objects)**

**📌 Location: src/main/java/com/example/riskrating/dto/  
📌 Transfers structured data between frontend & backend.**

**Create the following DTO classes inside dto package:**

* **UserDTO.java**
* **LocationDTO.java**
* **PropertyDTO.java**
* **InsurancePolicyDTO.java**
* **PremiumCalculationDTO.java**

**6️⃣ security/ (Spring Security & JWT Authentication)**

**📌 Location: src/main/java/com/example/riskrating/security/  
📌 Handles authentication, authorization, and JWT token management.**

**Create the following security classes inside security package:**

* **JwtUtil.java (Generate & Validate JWT Tokens)**
* **JwtFilter.java (Extracts Token from Requests)**
* **SecurityConfig.java (Configures Spring Security)**
* **CustomUserDetailsService.java (Loads User Data from DB)**

**7️⃣ RiskRatingApiApplication.java (Main Application Class)**

**📌 Location: src/main/java/com/example/riskrating/  
📌 This is the entry point of the Spring Boot application.**

**java**

**CopyEdit**

**package com.example.riskrating;**

**import org.springframework.boot.SpringApplication;**

**import org.springframework.boot.autoconfigure.SpringBootApplication;**

**@SpringBootApplication**

**public class RiskRatingApiApplication {**

**public static void main(String[] args) {**

**SpringApplication.run(RiskRatingApiApplication.class, args);**

**}**

**}**

**📌 Starts the backend server.**

**ALL MODEL ENTITY CLASSES**

package com.example.riskrating.model;

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "users")

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

@Column(unique = true, nullable = false)

private String email;

@Column(nullable = false)

private String password;

}

**2️⃣ Location.java (FEMA Risk Factors)**

java

CopyEdit

package com.example.riskrating.model;

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "locations")

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class Location {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String state;

private String zipCode;

private double distanceToRiver;

private double elevation;

private String floodZone;

private double crsDiscount;

}

📌 **Stores FEMA risk factors for a location.**  
📌 **Data can be fetched from the FEMA API or a preloaded database.**

**📌 3️⃣ Property.java (Building Characteristics)**

java

CopyEdit

package com.example.riskrating.model;

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "properties")

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class Property {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne

@JoinColumn(name = "user\_id", nullable = false)

private User user;

@ManyToOne

@JoinColumn(name = "location\_id", nullable = false)

private Location location;

private String foundationType;

private double firstFloorHeight;

private int numFloors;

private boolean floodVents;

}

📌 **Stores property details provided by the broker.**  
📌 **Linked to a Location (risk factors) and a User (broker).**

**📌 4️⃣ InsurancePolicy.java (Coverage Details)**

java

CopyEdit

package com.example.riskrating.model;

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "insurance\_policies")

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class InsurancePolicy {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne

@JoinColumn(name = "property\_id", nullable = false)

private Property property;

private double coverageBuilding;

private double coverageContents;

private double deductible;

private int priorClaims;

}

📌 **Stores coverage details for the building and its contents.**  
📌 **Linked to a Property.**

**📌 5️⃣ PremiumCalculation.java (Premium Calculation Results)**

java

CopyEdit

package com.example.riskrating.model;

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "premium\_calculations")

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class PremiumCalculation {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@OneToOne

@JoinColumn(name = "policy\_id", nullable = false)

private InsurancePolicy policy;

private double baseRate;

private double geographicFactor;

private double propertyFactor;

private double deductibleFactor;

private double finalPremium;

}

📌 **Stores the final premium calculated for the policy.**  
📌 **Includes adjustments based on risk factors, property details, and coverage.**

**REPOSITORY CLASSES**

**1️⃣ UserRepository.java (Broker Authentication)**

**java**

**CopyEdit**

**package com.example.riskrating.repository;**

**import com.example.riskrating.model.User;**

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

**import org.springframework.stereotype.Repository;**

**import java.util.Optional;**

**@Repository**

**public interface UserRepository extends JpaRepository<User, Long> {**

**Optional<User> findByEmail(String email);**

**}**

**📌 Used for user authentication.  
📌 Finds users by email for login.**

**📌 2️⃣ LocationRepository.java (FEMA Risk Data)**

**java**

**CopyEdit**

**package com.example.riskrating.repository;**

**import com.example.riskrating.model.Location;**

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

**import org.springframework.stereotype.Repository;**

**@Repository**

**public interface LocationRepository extends JpaRepository<Location, Long> {**

**Location findByZipCode(String zipCode);**

**}**

**📌 Fetches FEMA risk factors for a ZIP code.  
📌 Can be used with real-time FEMA API or a preloaded database.**

**📌 3️⃣ PropertyRepository.java (Building Details)**

**java**

**CopyEdit**

**package com.example.riskrating.repository;**

**import com.example.riskrating.model.Property;**

**import com.example.riskrating.model.User;**

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

**import org.springframework.stereotype.Repository;**

**import java.util.List;**

**@Repository**

**public interface PropertyRepository extends JpaRepository<Property, Long> {**

**List<Property> findByUser(User user);**

**}**

**📌 Fetches properties added by a specific broker.  
📌 Each property is linked to a user and a location.**

**📌 4️⃣ InsurancePolicyRepository.java (Coverage Details)**

**java**

**CopyEdit**

**package com.example.riskrating.repository;**

**import com.example.riskrating.model.InsurancePolicy;**

**import com.example.riskrating.model.Property;**

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

**import org.springframework.stereotype.Repository;**

**import java.util.Optional;**

**@Repository**

**public interface InsurancePolicyRepository extends JpaRepository<InsurancePolicy, Long> {**

**Optional<InsurancePolicy> findByProperty(Property property);**

**}**

**📌 Finds insurance policies linked to a property.  
📌 Used to retrieve coverage details for premium calculation.**

**📌 5️⃣ PremiumCalculationRepository.java (Premium Storage)**

**java**

**CopyEdit**

**package com.example.riskrating.repository;**

**import com.example.riskrating.model.PremiumCalculation;**

**import com.example.riskrating.model.InsurancePolicy;**

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

**import org.springframework.stereotype.Repository;**

**import java.util.Optional;**

**@Repository**

**public interface PremiumCalculationRepository extends JpaRepository<PremiumCalculation, Long> {**

**Optional<PremiumCalculation> findByPolicy(InsurancePolicy policy);**

**}**

**📌 Finds premium calculations for a given policy.  
📌 Stores the final calculated premium in the database.**

**CONTROLLER CODE**

**📌 1️⃣ AuthController.java (User Authentication)**

**java**

**CopyEdit**

**package com.example.riskrating.controller;**

**import com.example.riskrating.model.User;**

**import com.example.riskrating.repository.UserRepository;**

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

**import org.springframework.http.ResponseEntity;**

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

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

**import java.util.Optional;**

**@RestController**

**@RequestMapping("/auth")**

**public class AuthController {**

**@Autowired**

**private UserRepository userRepository;**

**@PostMapping("/register")**

**public ResponseEntity<String> register(@RequestBody User user) {**

**user.setPassword(new BCryptPasswordEncoder().encode(user.getPassword()));**

**userRepository.save(user);**

**return ResponseEntity.ok("User Registered Successfully");**

**}**

**@PostMapping("/login")**

**public ResponseEntity<String> login(@RequestBody User loginRequest) {**

**Optional<User> user = userRepository.findByEmail(loginRequest.getEmail());**

**if (user.isPresent() && new BCryptPasswordEncoder().matches(loginRequest.getPassword(), user.get().getPassword())) {**

**return ResponseEntity.ok("Login Successful");**

**}**

**return ResponseEntity.status(401).body("Invalid Credentials");**

**}**

**}**

**📌 Allows brokers to register and login.  
📌 Encrypts passwords before storing.**

**📌 2️⃣ LocationController.java (FEMA Risk Data)**

**java**

**CopyEdit**

**package com.example.riskrating.controller;**

**import com.example.riskrating.model.Location;**

**import com.example.riskrating.repository.LocationRepository;**

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

**import org.springframework.http.ResponseEntity;**

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

**import java.util.Optional;**

**@RestController**

**@RequestMapping("/api/location")**

**public class LocationController {**

**@Autowired**

**private LocationRepository locationRepository;**

**@GetMapping("/{zipCode}")**

**public ResponseEntity<Location> getLocationData(@PathVariable String zipCode) {**

**Optional<Location> location = Optional.ofNullable(locationRepository.findByZipCode(zipCode));**

**return location.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());**

**}**

**@PostMapping("/add")**

**public ResponseEntity<Location> addLocation(@RequestBody Location location) {**

**locationRepository.save(location);**

**return ResponseEntity.ok(location);**

**}**

**}**

**📌 Fetches location risk factors by ZIP code.  
📌 Can be used with FEMA API or local database.**

**📌 3️⃣ PropertyController.java (Building Details)**

**java**

**CopyEdit**

**package com.example.riskrating.controller;**

**import com.example.riskrating.model.Property;**

**import com.example.riskrating.repository.PropertyRepository;**

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

**import org.springframework.http.ResponseEntity;**

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

**import java.util.List;**

**@RestController**

**@RequestMapping("/api/property")**

**public class PropertyController {**

**@Autowired**

**private PropertyRepository propertyRepository;**

**@GetMapping("/user/{userId}")**

**public ResponseEntity<List<Property>> getUserProperties(@PathVariable Long userId) {**

**List<Property> properties = propertyRepository.findByUser(new User(userId, "", "", ""));**

**return ResponseEntity.ok(properties);**

**}**

**@PostMapping("/add")**

**public ResponseEntity<Property> addProperty(@RequestBody Property property) {**

**propertyRepository.save(property);**

**return ResponseEntity.ok(property);**

**}**

**}**

**📌 Allows brokers to add & view properties they manage.  
📌 Each property is linked to a user (broker).**

**📌 4️⃣ InsurancePolicyController.java (Coverage Details)**

**java**

**CopyEdit**

**package com.example.riskrating.controller;**

**import com.example.riskrating.model.InsurancePolicy;**

**import com.example.riskrating.repository.InsurancePolicyRepository;**

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

**import org.springframework.http.ResponseEntity;**

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

**import java.util.Optional;**

**@RestController**

**@RequestMapping("/api/policy")**

**public class InsurancePolicyController {**

**@Autowired**

**private InsurancePolicyRepository policyRepository;**

**@GetMapping("/{propertyId}")**

**public ResponseEntity<InsurancePolicy> getPolicyByProperty(@PathVariable Long propertyId) {**

**Optional<InsurancePolicy> policy = policyRepository.findByProperty(new Property(propertyId, null, null, "", 0, 0, false));**

**return policy.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());**

**}**

**@PostMapping("/add")**

**public ResponseEntity<InsurancePolicy> addPolicy(@RequestBody InsurancePolicy policy) {**

**policyRepository.save(policy);**

**return ResponseEntity.ok(policy);**

**}**

**}**

**📌 Allows brokers to add coverage details for properties.  
📌 Each policy is linked to a property.**

**📌 5️⃣ PremiumCalculationController.java (Premium Calculation)**

**java**

**CopyEdit**

**package com.example.riskrating.controller;**

**import com.example.riskrating.model.\*;**

**import com.example.riskrating.repository.PremiumCalculationRepository;**

**import com.example.riskrating.service.PremiumCalculationService;**

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

**import org.springframework.http.ResponseEntity;**

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

**@RestController**

**@RequestMapping("/api/premium")**

**public class PremiumCalculationController {**

**@Autowired**

**private PremiumCalculationService premiumCalculationService;**

**@Autowired**

**private PremiumCalculationRepository premiumCalculationRepository;**

**@PostMapping("/calculate")**

**public ResponseEntity<PremiumCalculation> calculatePremium(@RequestBody InsurancePolicy policy) {**

**PremiumCalculation premiumCalculation = premiumCalculationService.calculatePremium(policy);**

**premiumCalculationRepository.save(premiumCalculation);**

**return ResponseEntity.ok(premiumCalculation);**

**}**

**}**

**SERVICE CLASS**

**1️⃣ UserService.java (Broker Authentication Service)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.model.User;**

**import com.example.riskrating.repository.UserRepository;**

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

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

**import org.springframework.stereotype.Service;**

**import java.util.Optional;**

**@Service**

**public class UserService {**

**@Autowired**

**private UserRepository userRepository;**

**private final BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();**

**public String registerUser(User user) {**

**user.setPassword(passwordEncoder.encode(user.getPassword()));**

**userRepository.save(user);**

**return "User Registered Successfully";**

**}**

**public String loginUser(String email, String password) {**

**Optional<User> user = userRepository.findByEmail(email);**

**if (user.isPresent() && passwordEncoder.matches(password, user.get().getPassword())) {**

**return "Login Successful";**

**}**

**return "Invalid Credentials";**

**}**

**}**

**📌 Handles broker registration and login.  
📌 Encrypts passwords before saving.**

**📌 2️⃣ LocationService.java (FEMA Risk Data Service)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.model.Location;**

**import com.example.riskrating.repository.LocationRepository;**

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

**import org.springframework.stereotype.Service;**

**import org.springframework.web.client.RestTemplate;**

**@Service**

**public class LocationService {**

**@Autowired**

**private LocationRepository locationRepository;**

**private static final String FEMA\_API\_URL = "https://api.fema.gov/flood-risk?zip=";**

**public Location fetchLocationData(String zipCode) {**

**// Check if location data already exists**

**Location existingLocation = locationRepository.findByZipCode(zipCode);**

**if (existingLocation != null) {**

**return existingLocation;**

**}**

**// Fetch data from FEMA API**

**RestTemplate restTemplate = new RestTemplate();**

**Location location = restTemplate.getForObject(FEMA\_API\_URL + zipCode, Location.class);**

**// Save to database**

**locationRepository.save(location);**

**return location;**

**}**

**public Location addLocation(Location location) {**

**return locationRepository.save(location);**

**}**

**}**

**📌 Fetches FEMA risk factors by ZIP code.  
📌 Saves data to the database to avoid redundant API calls.**

**📌 3️⃣ PropertyService.java (Building Details Service)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.model.Property;**

**import com.example.riskrating.model.User;**

**import com.example.riskrating.repository.PropertyRepository;**

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

**import org.springframework.stereotype.Service;**

**import java.util.List;**

**@Service**

**public class PropertyService {**

**@Autowired**

**private PropertyRepository propertyRepository;**

**public List<Property> getUserProperties(User user) {**

**return propertyRepository.findByUser(user);**

**}**

**public Property addProperty(Property property) {**

**return propertyRepository.save(property);**

**}**

**}**

**📌 Allows brokers to add & view properties they manage.**

**📌 4️⃣ InsurancePolicyService.java (Coverage Details Service)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.model.InsurancePolicy;**

**import com.example.riskrating.model.Property;**

**import com.example.riskrating.repository.InsurancePolicyRepository;**

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

**import org.springframework.stereotype.Service;**

**import java.util.Optional;**

**@Service**

**public class InsurancePolicyService {**

**@Autowired**

**private InsurancePolicyRepository policyRepository;**

**public Optional<InsurancePolicy> getPolicyByProperty(Property property) {**

**return policyRepository.findByProperty(property);**

**}**

**public InsurancePolicy addPolicy(InsurancePolicy policy) {**

**return policyRepository.save(policy);**

**}**

**}**

**📌 Handles coverage details for each property.**

**📌 5️⃣ PremiumCalculationService.java (Premium Calculation Logic)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.model.\*;**

**import org.springframework.stereotype.Service;**

**@Service**

**public class PremiumCalculationService {**

**public PremiumCalculation calculatePremium(InsurancePolicy policy) {**

**Property property = policy.getProperty();**

**Location location = property.getLocation();**

**// Base Premium Calculation**

**double baseRate = 1.5 \* (policy.getCoverageBuilding() / 1000);**

**double geographicFactor = 1.0 + (location.getElevation() / 10) + (location.getDistanceToRiver() / 500);**

**double propertyFactor = property.isFloodVents() ? 0.9 : 1.1;**

**double deductibleFactor = 1.0 - (policy.getDeductible() / 10000);**

**double finalPremium = baseRate \* geographicFactor \* propertyFactor \* deductibleFactor;**

**// Store Premium Calculation**

**PremiumCalculation premiumCalc = new PremiumCalculation();**

**premiumCalc.setPolicy(policy);**

**premiumCalc.setBaseRate(baseRate);**

**premiumCalc.setGeographicFactor(geographicFactor);**

**premiumCalc.setPropertyFactor(propertyFactor);**

**premiumCalc.setDeductibleFactor(deductibleFactor);**

**premiumCalc.setFinalPremium(finalPremium);**

**return premiumCalc;**

**}**

**}**

**📌 Applies FEMA methodology for premium calculation.  
📌 Uses location risk factors, property details, and coverage options.**

**📌 Summary of Service Classes**

| **Service Class** | **Purpose** |
| --- | --- |
| **UserService** | **Handles broker registration and login.** |
| **LocationService** | **Fetches FEMA risk factors from API/database.** |
| **PropertyService** | **Manages property details.** |
| **InsurancePolicyService** | **Stores and retrieves insurance policies.** |
| **PremiumCalculationService** | **Computes flood insurance premiums** |

**1️⃣ DTO Classes**

**1️⃣ UserDTO.java (User Registration & Login)**

**java**

**CopyEdit**

**package com.example.riskrating.dto;**

**import lombok.Data;**

**@Data**

**public class UserDTO {**

**private String name;**

**private String email;**

**private String password;**

**}**

**📌 Used for broker registration and login requests.**

**2️⃣ LocationDTO.java (Location Data Transfer)**

**java**

**CopyEdit**

**package com.example.riskrating.dto;**

**import lombok.Data;**

**@Data**

**public class LocationDTO {**

**private String state;**

**private String zipCode;**

**private double distanceToRiver;**

**private double elevation;**

**private String floodZone;**

**private double crsDiscount;**

**}**

**📌 Transfers FEMA risk factors between frontend and backend.**

**3️⃣ PropertyDTO.java (Property Details)**

**java**

**CopyEdit**

**package com.example.riskrating.dto;**

**import lombok.Data;**

**@Data**

**public class PropertyDTO {**

**private Long userId;**

**private Long locationId;**

**private String foundationType;**

**private double firstFloorHeight;**

**private int numFloors;**

**private boolean floodVents;**

**}**

**📌 Used to receive property details from the frontend.**

**4️⃣ InsurancePolicyDTO.java (Policy Details)**

**java**

**CopyEdit**

**package com.example.riskrating.dto;**

**import lombok.Data;**

**@Data**

**public class InsurancePolicyDTO {**

**private Long propertyId;**

**private double coverageBuilding;**

**private double coverageContents;**

**private double deductible;**

**private int priorClaims;**

**}**

**📌 Transfers insurance coverage details between frontend and backend.**

**5️⃣ PremiumCalculationDTO.java (Premium Response)**

**java**

**CopyEdit**

**package com.example.riskrating.dto;**

**import lombok.Data;**

**@Data**

**public class PremiumCalculationDTO {**

**private double baseRate;**

**private double geographicFactor;**

**private double propertyFactor;**

**private double deductibleFactor;**

**private double finalPremium;**

**}**

**📌 Used to return premium calculation results to the frontend.**

**📌 2️⃣ Service Classes**

**📌 Services interact with the repository layer to handle business logic.**

**1️⃣ UserService.java (Broker Authentication)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.dto.UserDTO;**

**import com.example.riskrating.model.User;**

**import com.example.riskrating.repository.UserRepository;**

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

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

**import org.springframework.stereotype.Service;**

**import java.util.Optional;**

**@Service**

**public class UserService {**

**@Autowired**

**private UserRepository userRepository;**

**private final BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();**

**public String registerUser(UserDTO userDTO) {**

**User user = new User();**

**user.setName(userDTO.getName());**

**user.setEmail(userDTO.getEmail());**

**user.setPassword(passwordEncoder.encode(userDTO.getPassword()));**

**userRepository.save(user);**

**return "User Registered Successfully";**

**}**

**public String loginUser(String email, String password) {**

**Optional<User> user = userRepository.findByEmail(email);**

**if (user.isPresent() && passwordEncoder.matches(password, user.get().getPassword())) {**

**return "Login Successful";**

**}**

**return "Invalid Credentials";**

**}**

**}**

**📌 Handles broker registration and login.**

**2️⃣ LocationService.java (FEMA Risk Data)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.dto.LocationDTO;**

**import com.example.riskrating.model.Location;**

**import com.example.riskrating.repository.LocationRepository;**

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

**import org.springframework.stereotype.Service;**

**import org.springframework.web.client.RestTemplate;**

**@Service**

**public class LocationService {**

**@Autowired**

**private LocationRepository locationRepository;**

**private static final String FEMA\_API\_URL = "https://api.fema.gov/flood-risk?zip=";**

**public Location fetchLocationData(String zipCode) {**

**Location existingLocation = locationRepository.findByZipCode(zipCode);**

**if (existingLocation != null) {**

**return existingLocation;**

**}**

**RestTemplate restTemplate = new RestTemplate();**

**Location location = restTemplate.getForObject(FEMA\_API\_URL + zipCode, Location.class);**

**locationRepository.save(location);**

**return location;**

**}**

**public Location addLocation(LocationDTO locationDTO) {**

**Location location = new Location();**

**location.setState(locationDTO.getState());**

**location.setZipCode(locationDTO.getZipCode());**

**location.setDistanceToRiver(locationDTO.getDistanceToRiver());**

**location.setElevation(locationDTO.getElevation());**

**location.setFloodZone(locationDTO.getFloodZone());**

**location.setCrsDiscount(locationDTO.getCrsDiscount());**

**return locationRepository.save(location);**

**}**

**}**

**📌 Fetches FEMA risk factors and stores them in the database.**

**3️⃣ PropertyService.java (Building Details)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.dto.PropertyDTO;**

**import com.example.riskrating.model.Property;**

**import com.example.riskrating.model.User;**

**import com.example.riskrating.repository.PropertyRepository;**

**import com.example.riskrating.repository.UserRepository;**

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

**import org.springframework.stereotype.Service;**

**import java.util.List;**

**@Service**

**public class PropertyService {**

**@Autowired**

**private PropertyRepository propertyRepository;**

**@Autowired**

**private UserRepository userRepository;**

**public List<Property> getUserProperties(Long userId) {**

**User user = userRepository.findById(userId).orElseThrow(() -> new RuntimeException("User not found"));**

**return propertyRepository.findByUser(user);**

**}**

**public Property addProperty(PropertyDTO propertyDTO) {**

**Property property = new Property();**

**property.setFoundationType(propertyDTO.getFoundationType());**

**property.setFirstFloorHeight(propertyDTO.getFirstFloorHeight());**

**property.setNumFloors(propertyDTO.getNumFloors());**

**property.setFloodVents(propertyDTO.isFloodVents());**

**return propertyRepository.save(property);**

**}**

**}**

**📌 Manages building details for each broker.**

**4️⃣ InsurancePolicyService.java (Coverage Details)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.dto.InsurancePolicyDTO;**

**import com.example.riskrating.model.InsurancePolicy;**

**import com.example.riskrating.repository.InsurancePolicyRepository;**

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

**import org.springframework.stereotype.Service;**

**@Service**

**public class InsurancePolicyService {**

**@Autowired**

**private InsurancePolicyRepository policyRepository;**

**public InsurancePolicy addPolicy(InsurancePolicyDTO policyDTO) {**

**InsurancePolicy policy = new InsurancePolicy();**

**policy.setCoverageBuilding(policyDTO.getCoverageBuilding());**

**policy.setCoverageContents(policyDTO.getCoverageContents());**

**policy.setDeductible(policyDTO.getDeductible());**

**policy.setPriorClaims(policyDTO.getPriorClaims());**

**return policyRepository.save(policy);**

**}**

**}**

**📌 Handles insurance policy storage and retrieval.**

**5️⃣ PremiumCalculationService.java (Premium Calculation)**

**java**

**CopyEdit**

**package com.example.riskrating.service;**

**import com.example.riskrating.dto.PremiumCalculationDTO;**

**import com.example.riskrating.model.InsurancePolicy;**

**import com.example.riskrating.model.Location;**

**import com.example.riskrating.model.PremiumCalculation;**

**import org.springframework.stereotype.Service;**

**@Service**

**public class PremiumCalculationService {**

**public PremiumCalculation calculatePremium(InsurancePolicy policy) {**

**Location location = policy.getProperty().getLocation();**

**double baseRate = 1.5 \* (policy.getCoverageBuilding() / 1000);**

**double geographicFactor = 1.0 + (location.getElevation() / 10) + (location.getDistanceToRiver() / 500);**

**double finalPremium = baseRate \* geographicFactor;**

**PremiumCalculation premiumCalc = new PremiumCalculation();**

**premiumCalc.setFinalPremium(finalPremium);**

**return premiumCalc;**

**}**

**}**

**📌 Calculates the final flood insurance premium.**

**1️⃣ application.properties (Backend Configuration)**

Modify this file to configure **database, JWT secret, and security settings**.

properties

CopyEdit

# Database Configuration

spring.datasource.url=jdbc:mysql://localhost:3306/risk\_rating

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate Configuration

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

# JWT Secret Key (Used for Token Signing)

jwt.secret=mysupersecretkey

# Server Port

server.port=8080

📌 **Configures database connection.**  
📌 **Sets JWT secret key for authentication.**  
📌 **Runs backend on port 8080.**

**📌 2️⃣ JwtUtil.java (Generate & Validate JWT Token)**

This class provides functions to **generate and validate JWT tokens**.

package com.example.riskrating.security;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

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

import org.springframework.stereotype.Component;

import java.util.Date;

import java.util.function.Function;

@Component

public class JwtUtil {

@Value("${jwt.secret}")

private String secretKey;

private static final long EXPIRATION\_TIME = 1000 \* 60 \* 60; // 1 Hour

public String generateToken(String email) {

return Jwts.builder()

.setSubject(email)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + EXPIRATION\_TIME))

.signWith(SignatureAlgorithm.HS256, secretKey)

.compact();

}

public String extractEmail(String token) {

return extractClaim(token, Claims::getSubject);

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

private <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

final Claims claims = Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token).getBody();

return claimsResolver.apply(claims);

}

public boolean validateToken(String token, String userEmail) {

return (userEmail.equals(extractEmail(token)) && !isTokenExpired(token));

}

private boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

}

📌 **Generates JWT token upon login.**  
📌 **Validates token when a user accesses protected resources.**

**3️⃣ JwtFilter.java (JWT Request Filter)**

This filter **checks incoming requests for a valid JWT token** before processing.

java

CopyEdit

package com.example.riskrating.security;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

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

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component

public class JwtFilter extends OncePerRequestFilter {

@Autowired

private JwtUtil jwtUtil;

@Autowired

private UserDetailsService userDetailsService;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

final String authorizationHeader = request.getHeader("Authorization");

String token = null;

String email = null;

if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {

token = authorizationHeader.substring(7);

email = jwtUtil.extractEmail(token);

}

if (email != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = userDetailsService.loadUserByUsername(email);

if (jwtUtil.validateToken(token, userDetails.getUsername())) {

UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

chain.doFilter(request, response);

}

}

📌 **Extracts JWT token from request header.**  
📌 **Validates token and sets authentication in the security context.**

**📌 4️⃣ SecurityConfig.java (Spring Security Configuration)**

Configures **which endpoints require authentication** and which are **publicly accessible**.

java

CopyEdit

package com.example.riskrating.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.ProviderManager;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

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

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.UserDetailsService;

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

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import java.util.Collections;

@Configuration

public class SecurityConfig {

@Bean

public AuthenticationManager authenticationManager(UserDetailsService userDetailsService) {

DaoAuthenticationProvider provider = new DaoAuthenticationProvider();

provider.setUserDetailsService(userDetailsService);

provider.setPasswordEncoder(passwordEncoder());

return new ProviderManager(Collections.singletonList(provider));

}

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public JwtFilter jwtFilter() {

return new JwtFilter();

}

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.requestMatchers("/auth/register", "/auth/login").permitAll()

.anyRequest().authenticated()

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(jwtFilter(), UsernamePasswordAuthenticationFilter.class);

}

}

📌 **Allows public access to /auth/register and /auth/login.**  
📌 **Requires authentication for all other endpoints.**

**📌 5️⃣ CustomUserDetailsService.java (User Authentication Service)**

Loads user details from the database for authentication.

java

CopyEdit

package com.example.riskrating.security;

import com.example.riskrating.model.User;

import com.example.riskrating.repository.UserRepository;

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

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.Collections;

@Service

public class CustomUserDetailsService implements UserDetailsService {

@Autowired

private UserRepository userRepository;

@Override

public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {

User user = userRepository.findByEmail(email)

.orElseThrow(() -> new UsernameNotFoundException("User not found"));

return new org.springframework.security.core.userdetails.User(

user.getEmail(),

user.getPassword(),

Collections.emptyList()

);

}

}

📌 **Fetches user details from the database during login.**

**📌 Summary of Security Classes**

| **Class Name** | **Purpose** |
| --- | --- |
| JwtUtil.java | Generates & validates JWT tokens. |
| JwtFilter.java | Extracts token from requests and authenticates users. |
| SecurityConfig.java | Configures security rules & endpoint access. |
| CustomUserDetailsService.java | Loads user details for authentication. |