**Insurance Management System (IMS) Application**

1. **Policy management:** -

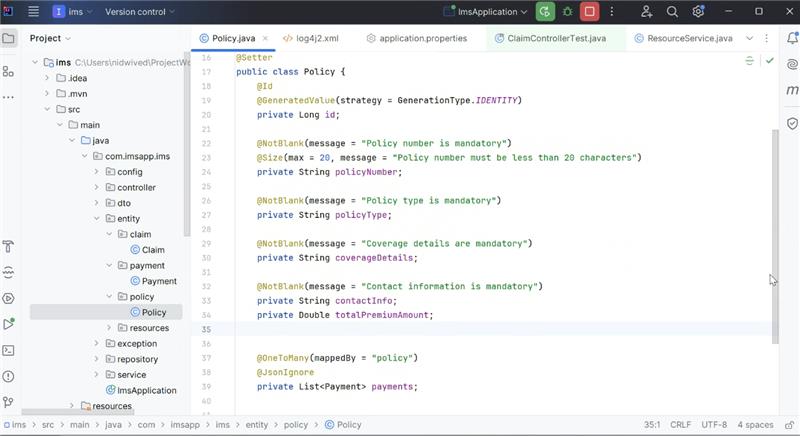
The IMS allows users to store their policy information, including their policy number, policy type, coverage details, and contact information for their insurance company. To achieve policy module provides CRUD operations (Create, Read, Update, Delete) for Policy entities using the PolicyRepository. It includes methods to get all policies, get a policy by ID, save a policy, and delete a policy.

1. **getAllPolicies()**: Retrieves all policies from the repository and logs the action.
2. **getPolicyById(Long id)**: Fetches a policy by its ID from the repository.
3. **savePolicy(Policy policy)**: Saves a policy to the repository.
4. **deletePolicy(Long id)**: Deletes a policy by its ID from the repository.

A screenshot of a computer

AI-generated content may be incorrect.

**Policy Entity**



**Exception Handling**

A screenshot of a computer

AI-generated content may be incorrect.

1. **Claim management: -**

The IMS allows users to track their claims, including the status of their claim, the amount of their claim, and the date their claim was filed

Claim module provides CRUD operations for Claim entities using the ClaimRepository. It includes methods to get all claims, get claims by status, create a new claim, update an existing claim, and delete a claim. The convertToResponse method is used to transform Claim entities into ClaimResponse DTOs

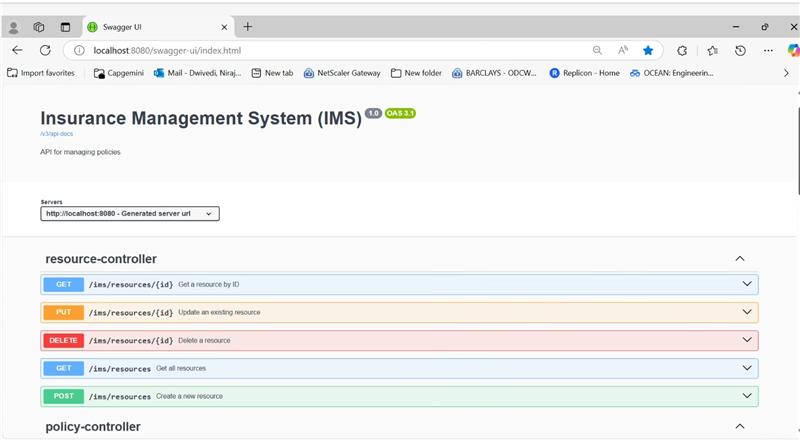
1. **getAllClaims(Pageable pageable)**: Retrieves all claims in a paginated format and converts them to ClaimResponse objects.
2. **getClaimsByStatus(String status, Pageable pageable)**: Fetches claims by their status in a paginated format and converts them to ClaimResponse objects.
3. **createClaim(ClaimRequest claimRequest)**: Creates a new claim from a ClaimRequest object, saves it to the repository, and converts it to a ClaimResponse object.
4. **updateClaim(Long id, ClaimRequest claimRequest)**: Updates an existing claim based on its ID and a ClaimRequest object. If the claim is found, it updates the details and saves it; otherwise, it throws a ClaimNotFoundException.
5. **deleteClaim(Long id)**: Deletes a claim by its ID. If the claim is found, it deletes it; otherwise, it throws a ClaimNotFoundException.
6. **convertToResponse(Claim claim)**: Converts a Claim entity to a ClaimResponse DTO.

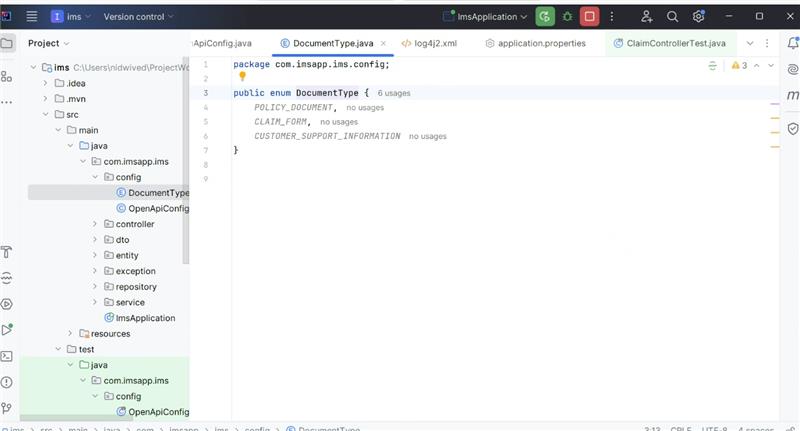


1. **Resource centre: -**

The IMS provides users with access to a variety of resources, such as policy documents, claim forms, and customer support information

1. **getAllResources()**: Retrieves all resources from the repository.
2. **getResourceById(Long id)**: Fetches a resource by its ID, checks if the corresponding file exists in the static folder, and returns it as a downloadable file. If the resource or file is not found, it throws a ResourceNotFoundException.
3. **createResource(Resources resource)**: Saves a new resource to the repository.
4. **updateResource(Long id, Resources resourceDetails)**: Updates an existing resource based on its ID with the provided details. If the resource is not found, it throws a ResourceNotFoundException.
5. **deleteResource(Long id)**: Deletes a resource by its ID. If the resource is not found, it throws a ResourceNotFoundException.

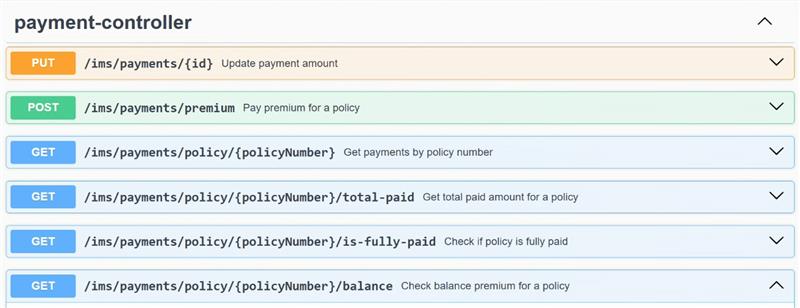




1. **Payment management: -**

The IMS allows users to make payments for their insurance premium

1. **payPremium(String policyNumber, Double amount)**: Processes the payment for a given policy number and amount. It checks if the policy exists, creates a new payment, and saves it to the repository. If the policy is not found, it throws a PolicyNotFoundException.
2. **checkBalancePremium(String policyNumber)**: Calculates the remaining balance of the premium for a given policy number. It retrieves the policy and its payments, sums the paid amounts, and subtracts this from the total premium amount. If the policy is not found, it throws a PolicyNotFoundException.
3. **validateAmount(Policy policy, Double amount)**: Validates the payment amount to ensure it is positive and does not exceed the total premium amount of the policy. If the amount is invalid, it throws an IllegalArgumentException.
4. **getPaymentsByPolicy(String policyNumber)**: Retrieves all payments associated with a given policy number. If the policy is not found, it throws a PolicyNotFoundException.
5. **updatePayment(Long paymentId, Double newAmount)**: Updates the amount of an existing payment based on its ID. It validates the new amount and saves the updated payment. If the payment is not found, it throws a PaymentProcessingException.
6. **getTotalPaidAmount(String policyNumber)**: Calculates the total amount paid for a given policy number by summing all associated payments. If the policy is not found, it throws a PolicyNotFoundException.
7. **isPolicyFullyPaid(String policyNumber)**: Checks if the total paid amount for a given policy number meets or exceeds the total premium amount, indicating the policy is fully paid. If the policy is not found, it throws a PolicyNotFoundException.



A screen shot of a computer

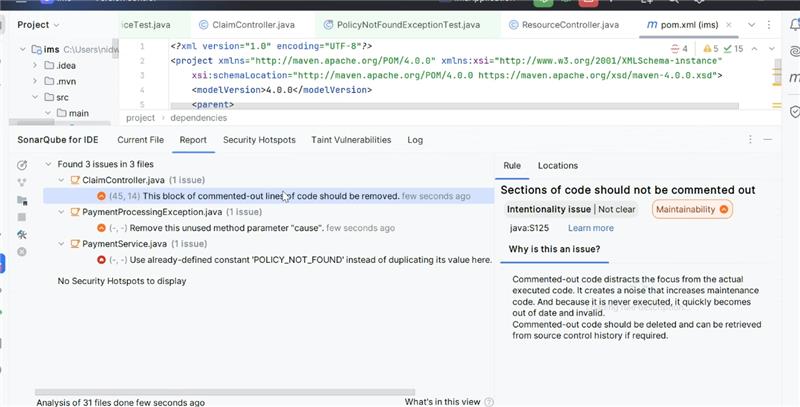
AI-generated content may be incorrect.

1. **Code analyser to maintain code quality (Sonar Cube/SonarLint ):** Steps to install in intellij

* Open IntelliJ IDEA.
* Go to File > Settings > Plugins.
* Click on the Marketplace tab.
* Search for "SonarLint/SonarCube".
* Click Install and restart IntelliJ IDEA

**Analyze Code**:

* + Right-click on a file or project in the Project Explorer.
  + Select SonarQube/Lint> Analyze with SonarQube/Lint.

s

1. **Log4j2 Integration:**

Update your `pom.xml` to exclude `spring-boot-starter-logging` and include `spring-boot-starter-log4j2`

<dependencies>

<dependency>

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

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

<exclusions>

<exclusion>

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

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

</exclusion>

</exclusions>

</dependency>

<dependency>

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

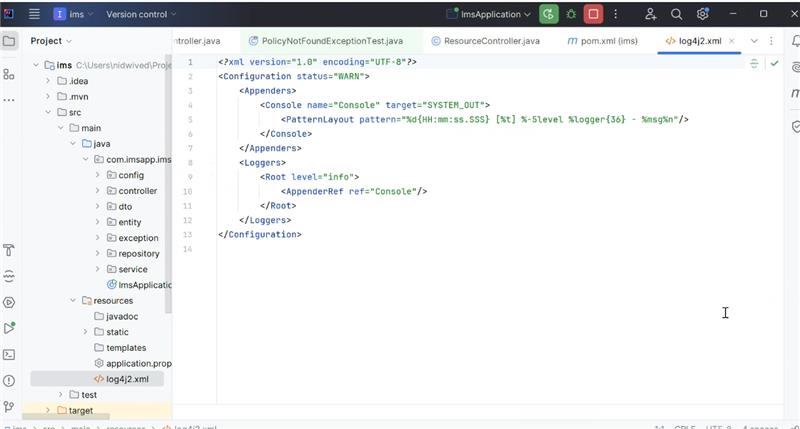
<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

</dependencies>

**Add Log4j2 Configuration File**

Create a `log4j2.xml` or `log4j2-spring.xml` file in the `src/main/resources` directory. The `-spring` variant is recommended for Spring Boot applications



**Add in Application Properties**

You can also set log levels in the `application.properties` file:

logging.level.root=INFO

logging.level.com.imsapp.ims=DEBUG

logging.file.name=logs/app.log

logging.pattern.console=%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n

logging.pattern.file=%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} - %msg%n

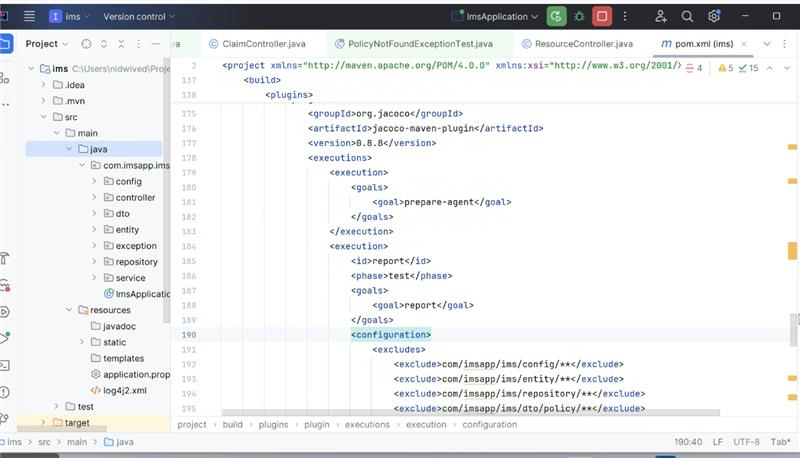
1. **Jacoco Integration:**

This configuration ensures that JaCoCo collects coverage data during tests, generates reports, and optionally checks coverage thresholds.

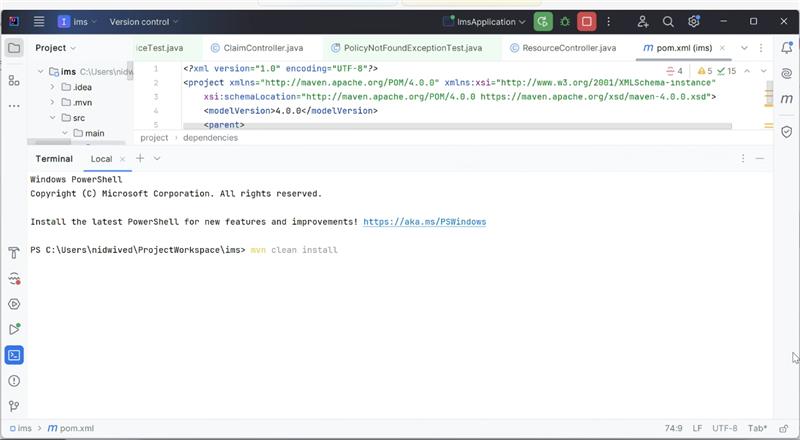
* Prepare Agent: Sets up the JaCoCo agent to collect coverage data This execution prepares the JaCoCo agent to collect coverage data during the test phase.
* **Report**: Generates a coverage report, excluding specified packages and classes.
* **Check**: Validates the coverage against defined rules (currently commented out).

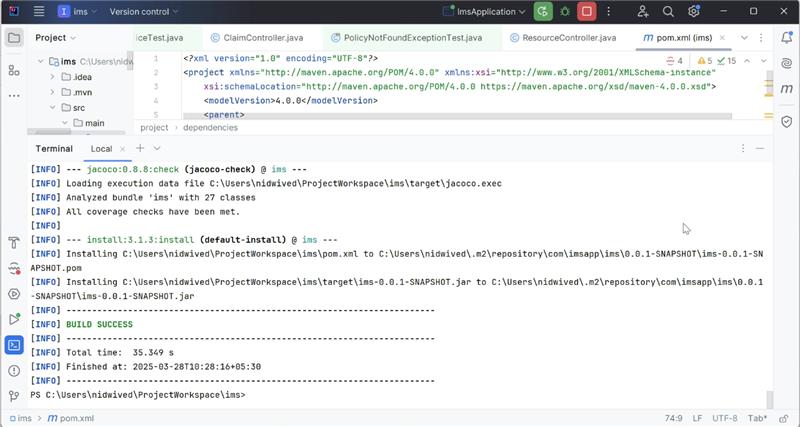
**Integrate plugin in pom.xml**

The `jacoco-maven-plugin` is declared with its group ID, artifact ID, and version.



To prepare report , run command **mvn clean install** like below or **mvn clean test jacoco:report**



****

After successful build **Target** folder will get created and report we can find at path -**project-directory/target/site/jacoco/index.html**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **H2 Database Configuration:** we need to add below properties to application.properties /yml file and required plugin in pom.xml

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>2.1.214</version>

</dependency>

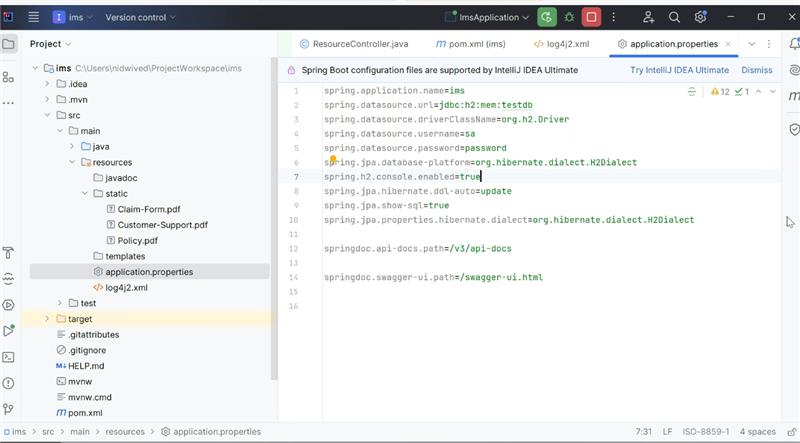
<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>



A screenshot of a computer

AI-generated content may be incorrect.

1. **Unit test cases using Junit for all functionalities with 100% coverage-**

Covered all the test case for this ims application for all layers using

<dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>4.0.0</version>  
 <scope>test</scope>  
</dependency>

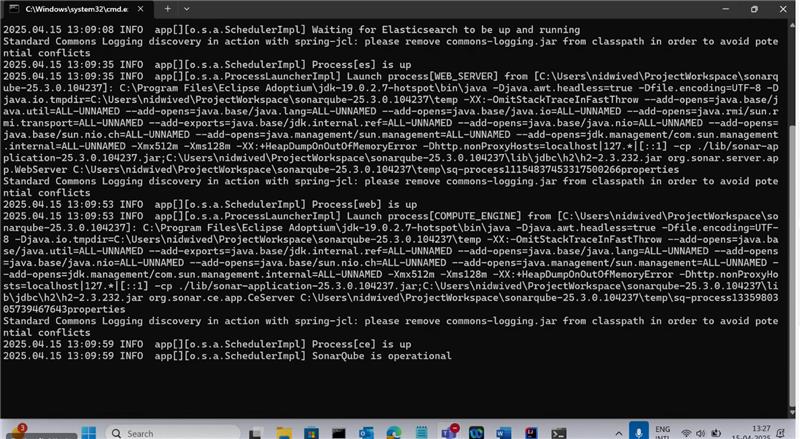
A screenshot of a computer

AI-generated content may be incorrect.

1. **Use static code analyser to maintain code (Integrated Sonar Cube Locally**): Created a separate document for how to setup sonar cube locally with document name **IMS-ConfigureSonarQube-Locally**

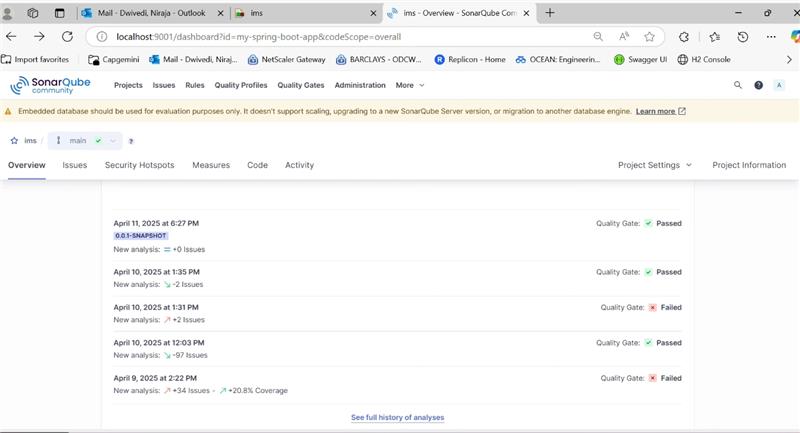
Started sonarcube.bat local machine and run this command from terminal of intellij **mvn clean verify sonar:sonar** to refresh and update the sonar cube anaysis report

1. **Sonar Cube Server Started locally**



1. **Report generated on dashbord when running on local:**

[**ims - Overview - SonarQube Community Build**](http://localhost:9001/dashboard?id=my-spring-boot-app&codeScope=overall)



1. **100 % Coverage with 0 bugs:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.