# **Hướng dẫn chi tiết để tạo project insurance-claim từ đầu đến khi hoàn chỉnh**

Dưới đây là các bước chi tiết để tạo và cấu hình một project Spring Boot từ đầu, bao gồm việc tạo các file giống như trong project insurance-claim.zip đã tải lên.

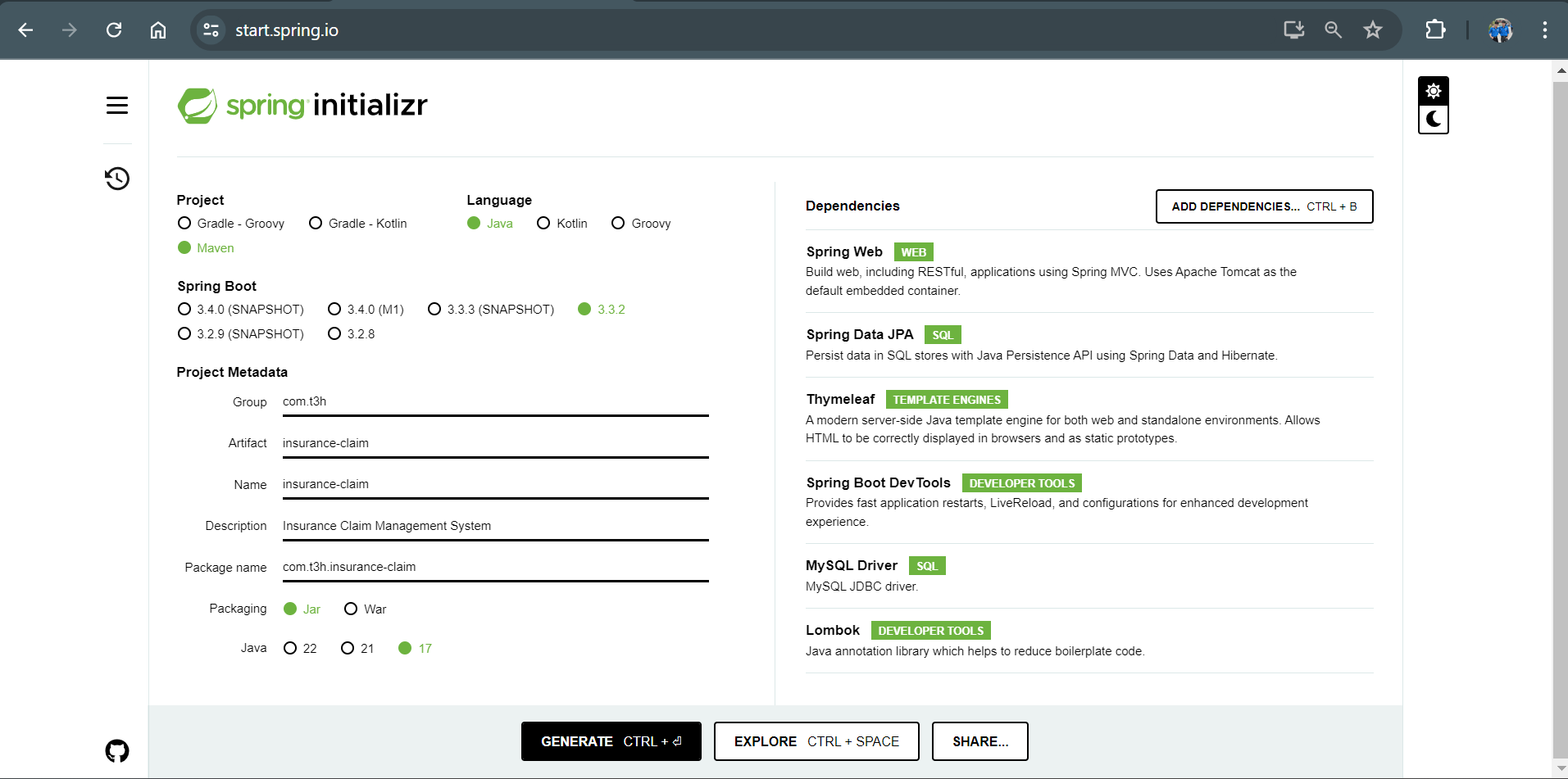
**Bước 1: Tạo project Spring Boot trên** [**https://start.spring.io/**](https://start.spring.io/)

1. **Truy cập trang web**: Mở trình duyệt và truy cập trang [start.spring.io](https://start.spring.io/).
2. **Cấu hình project**:
   * Project: Maven Project
   * Language: Java
   * Spring Boot: chọn phiên bản **3.3.2**
   * Project Metadata:
     + Group: com.t3h
     + Artifact: insurance-claim
     + Name: insurance-claim
     + Description: Insurance Claim Management System
     + Package name: com.t3h.insuranceclaim
     + Packaging: Jar
     + Java: 17
3. **Dependencies**: Chọn các dependencies cần thiết cho project, bao gồm:
   * Spring Web
   * Spring Data JPA
   * Thymeleaf
   * Spring Boot DevTools
   * MySQL Driver
   * Lombok
   * Spring Boot Starter Test

Các dependences dưới đây sẽ add sau khi tạo project

* + MapStruct
  + SpringFox
  + Servlet API

1. **Generate project**: Nhấn nút Generate để tải về file zip của project.

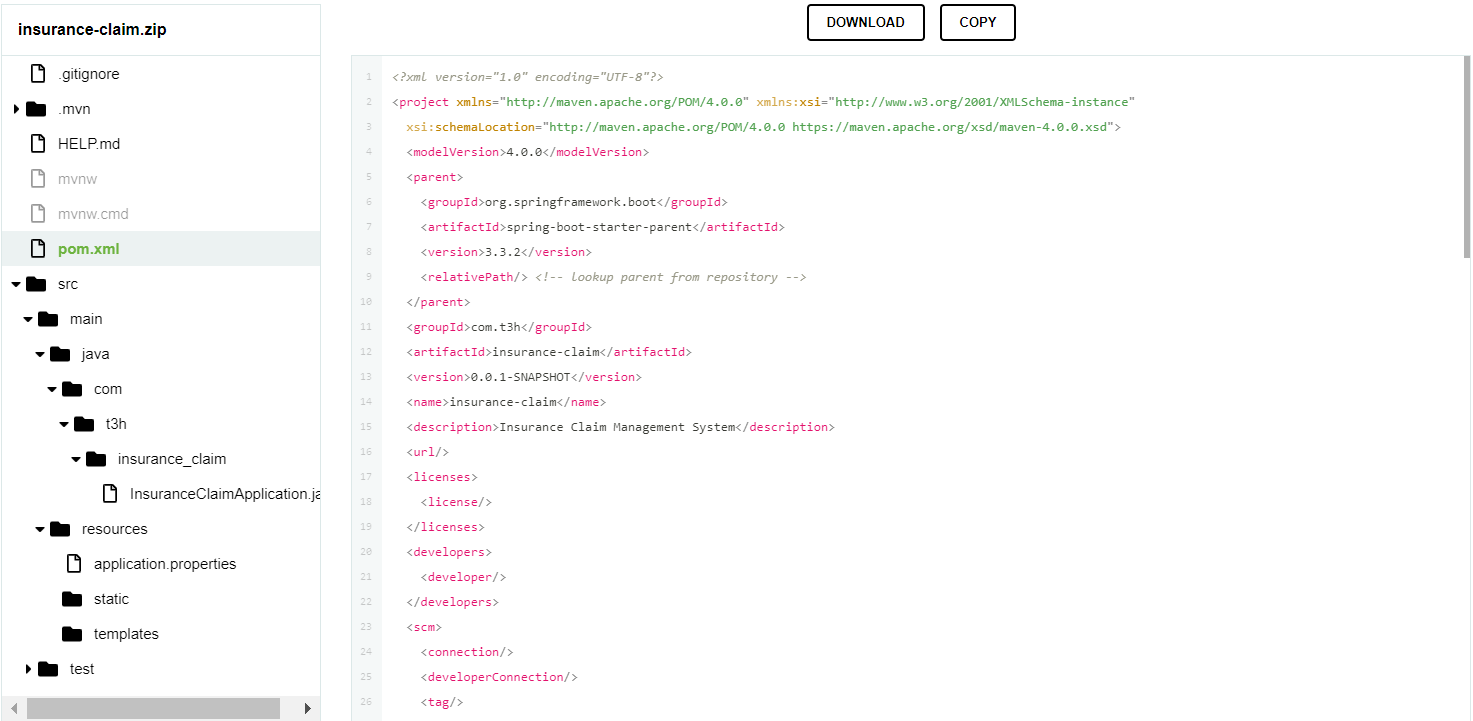


**Bước 2: Giải nén và mở project**

1. **Giải nén**: Giải nén file zip vừa tải về.
2. **Mở project**: Mở project bằng IDE Intellij

**Bước 3: Cấu trúc thư mục của project**

Khi mở project, bạn sẽ thấy cấu trúc thư mục như sau:



**Bước 4: Cấu hình file application.properties**

**Trước đó hãy tạo sẵn database insurance-claim trong database mysql**

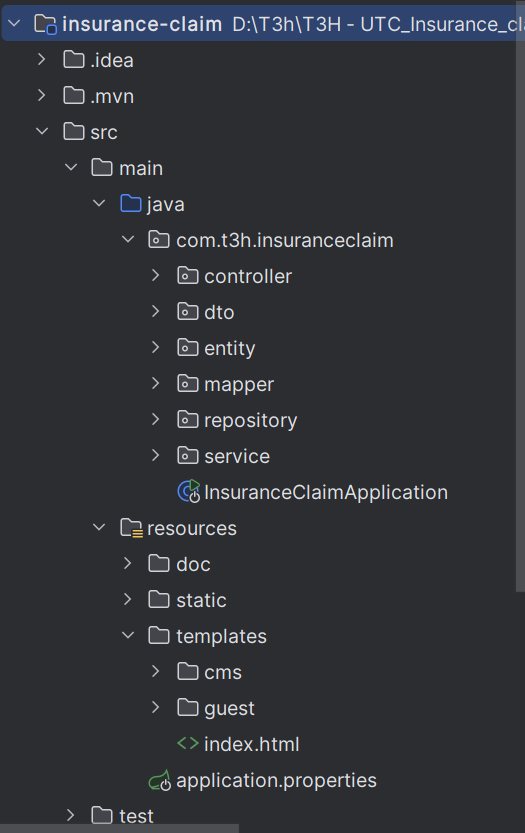
Mở file src/main/resources/application.properties và cấu hình cơ sở dữ liệu như sau:

spring.application.name=insurance-claim  
# MySQL configurations  
spring.datasource.url=jdbc:mysql://localhost:3306/insuranceclaim?allowPublicKeyRetrieval=true&useSSL=false  
spring.datasource.username=root  
#password sẽ tùy thuộc vào password bạn đặt cho mysql server, nêsu ai dùng mysql phpmyadmin có thể bỏ trống nếu ko có   
spring.datasource.password=root  
# JPA configurations  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.hikari.maximum-pool-size=10

**Bước 5: Tạo các package và classes cần thiết**

Tạo các package và class như sau:

1. **Entities**: Tạo package entity
2. **Repositories**: Tạo package repository
3. **Services**: Tạo package service
4. **Controllers**: Tạo package controller
5. **Mapper:** Tạo package mapper
6. **Dto:** Tạo package dto



# **Cấu trúc project**

**Tổng quan về cấu trúc của Project**

Project "insurance-claims" bao gồm các tầng sau:

1. **Entity Layer**
2. **Repository Layer**
3. **Service Layer**
4. **Controller Layer**
5. **DTO Layer**
6. **Mapper Layer**

Định nghĩa về các Annotation sử dụng trong project

* Annotation là một dạng metadata trong Java, được sử dụng để cung cấp các thông tin bổ sung cho chương trình. Các annotation không trực tiếp ảnh hưởng đến logic chương trình nhưng có thể được sử dụng bởi trình biên dịch hoặc các công cụ khác để xử lý logic nhất định.

**1. Entity Layer**

**Mô tả:**

Tầng này đại diện cho các thực thể (entity) trong cơ sở dữ liệu. Các lớp trong tầng này thường ánh xạ đến các bảng trong cơ sở dữ liệu và được sử dụng để thao tác dữ liệu.

**Chức năng:**

* Định nghĩa các thuộc tính tương ứng với các cột trong bảng cơ sở dữ liệu.
* Quản lý các mối quan hệ giữa các thực thể.
* Chứa các chú thích (annotations) từ JPA hoặc Hibernate để ánh xạ các trường đến các cột cơ sở dữ liệu.

**Ví dụ:**

@Entity  
@Table(name = "claim\_document")  
@Getter  
@Setter  
public class ClaimDocumentEntity extends BaseEntity {  
  
 @ManyToOne  
 @JoinColumn(name = "claim\_id")  
 private ClaimEntity claimEntity;  
  
 private String documentType;  
 private String documentName;  
 private String filePath;  
 private LocalDate uploadDate;  
  
 public ClaimDocumentEntity() {  
  
 }  
}

**Giải thích các Annotation đã dùng:**

* @Entity: Đánh dấu lớp này là một thực thể JPA, ánh xạ đến một bảng trong cơ sở dữ liệu.
* @Id: Chỉ định trường này là khóa chính của thực thể.
* @GeneratedValue(strategy = GenerationType.IDENTITY): Chỉ định chiến lược tạo giá trị tự động cho khóa chính.
* @ManyToOne: Chỉ định mối quan hệ nhiều-đến-một giữa ClaimEntity và CustomerEntity.
* @JoinColumn(name = "customer\_id"): Chỉ định cột trong bảng cơ sở dữ liệu liên kết với thực thể khác.

**2. Repository Layer**

**Mô tả:**

Tầng này chứa các lớp repository để truy cập và thao tác dữ liệu từ cơ sở dữ liệu. Repository sử dụng các phương thức tiêu chuẩn để thực hiện các thao tác CRUD (Create, Read, Update, Delete).

**Chức năng:**

* Truy xuất dữ liệu từ cơ sở dữ liệu thông qua các phương thức tiêu chuẩn hoặc các truy vấn tùy chỉnh.
* Lưu trữ, cập nhật và xóa dữ liệu.
* Tối ưu hóa truy vấn thông qua việc sử dụng các chú thích (annotations) như @Query.

**Ví dụ:**

@Repository  
public interface ClaimRepository extends JpaRepository<ClaimEntity, Long> {  
}

**Giải thích các Annotation đã dùng:**

* @Repository: Đánh dấu lớp này là một repository trong Spring, quản lý truy cập dữ liệu.
* JpaRepository<ClaimEntity, Long>: Interface của Spring Data JPA, cung cấp các phương thức CRUD tiêu chuẩn.

**3. Service Layer**

**Mô tả:**

Tầng này chứa logic nghiệp vụ của ứng dụng. Các lớp trong tầng này thường gọi các phương thức của repository để thao tác dữ liệu và thực hiện các logic nghiệp vụ cần thiết.

**Chức năng:**

* Thực hiện các thao tác nghiệp vụ như tính toán, xác thực dữ liệu, và xử lý logic phức tạp.
* Gọi các phương thức từ repository để truy xuất hoặc thao tác dữ liệu.
* Đảm bảo tính toàn vẹn của dữ liệu bằng cách kiểm tra và xác thực dữ liệu trước khi lưu vào cơ sở dữ liệu.

**Ví dụ:**

public interface ClaimService {  
 ClaimDTO getById(Long id);  
 ClaimDTO create(ClaimDTO claimDTO);  
 ClaimDTO update(Long id, ClaimDTO claimDTO);  
 void delete(Long id);  
}

@Service  
public class ClaimServiceImpl implements ClaimService {  
  
 @Autowired  
 private ClaimRepository claimRepository;  
  
 @Autowired  
 private ClaimMapper claimMapper;  
  
 @Override  
 public ClaimDTO getById(Long id) {  
 ClaimEntity claimEntity = claimRepository.findById(id)  
 .orElseThrow(() -> new RuntimeException("Claim not found"));  
 return claimMapper.toDTO(claimEntity);  
 }  
  
 @Override  
 public ClaimDTO create(ClaimDTO claimDTO) {  
 ClaimEntity claimEntity = claimMapper.toEntity(claimDTO);  
 claimEntity = claimRepository.save(claimEntity);  
 return claimMapper.toDTO(claimEntity);  
 }

**Giải thích các Annotation đã dùng:**

* @Service: Đánh dấu lớp này là một service trong Spring, chứa logic nghiệp vụ.
* @Autowired: Cho phép Spring tự động tiêm dependency, trong trường hợp này là ClaimRepository và ClaimMapper.

**4. Controller Layer**

**Mô tả:**

Tầng này là nơi xử lý các yêu cầu HTTP từ phía người dùng. Controller chịu trách nhiệm nhận các yêu cầu, gọi các phương thức từ service, và trả về phản hồi tương ứng.

**Chức năng:**

* Nhận yêu cầu từ người dùng và phân tích các thông tin đầu vào.
* Gọi các phương thức từ service để xử lý yêu cầu.
* Trả về phản hồi dưới dạng JSON hoặc các định dạng khác để gửi lại cho client.

**Ví dụ:**

@Controller  
@RequestMapping("/cms/")  
public class ClaimManagerController {  
  
  
 @GetMapping("claim-manager")  
 public String claimManager() {  
 return "cms/claim/claim-manager";  
 }  
  
 @GetMapping("claim/{id}")  
 public String claim(@PathVariable int id) {  
 return "cms/claim/claim-detail";  
 }  
}

@RestController  
@RequestMapping("/api/claims")  
public class ClaimResourceController {  
  
 @Autowired  
 private ClaimService claimService;  
  
  
 @GetMapping("/id")  
 public ClaimDTO getById(@PathVariable Long id) {  
 return claimService.getById(id);  
 }  
  
 @PutMapping("/id")  
 public ClaimDTO update(@PathVariable Long id, @RequestBody ClaimDTO claimDTO) {  
 return claimService.update(id, claimDTO);  
 }  
  
 @DeleteMapping("/id")  
 public void delete(@PathVariable Long id) {  
 claimService.delete(id);  
 }  
}

**Giải thích các Annotation đã dùng:**

* @RestController: Kết hợp giữa @Controller và @ResponseBody, chỉ định rằng lớp này là một controller RESTful, trả về dữ liệu trực tiếp thay vì trả về một trang web.
* @RequestMapping("/api/claims"): Chỉ định đường dẫn cơ bản cho tất cả các yêu cầu HTTP của lớp controller này.
* @PostMapping: Xử lý các yêu cầu HTTP POST.
* @GetMapping: Xử lý các yêu cầu HTTP GET.
* @RequestBody: Gán dữ liệu từ yêu cầu HTTP vào đối tượng DTO.
* @PathVariable: Trích xuất biến từ URL để sử dụng làm tham số phương thức.
* ResponseEntity: Đại diện cho toàn bộ phản hồi HTTP, bao gồm trạng thái và dữ liệu trả về.

**5. DTO Layer**

**Mô tả:**

DTO (Data Transfer Object) là các đối tượng đơn giản được sử dụng để chuyển dữ liệu giữa các lớp và các tầng khác nhau của ứng dụng. DTO thường được sử dụng để gom nhóm các dữ liệu lại với nhau mà không bao gồm các logic nghiệp vụ.

**Chức năng:**

* Chuyển dữ liệu giữa client và server mà không liên quan đến logic nghiệp vụ.
* Giảm thiểu số lượng các trường dữ liệu cần thiết trong một request/response.
* Tăng cường bảo mật bằng cách không để lộ các thực thể trực tiếp cho client.

**Ví dụ:**

@Data  
public class ClaimDTO {  
  
 private Long id;  
 private String claimCode;  
 private String customerName;  
 private String insuranceProductName;  
 private LocalDate claimDate;  
 private String description;  
 private String status;  
 private String statusDescription;  
 private Double amount;  
  
  
}

**6. Mapper Layer**

**Mô tả:**

Mapper là các lớp chịu trách nhiệm chuyển đổi giữa các đối tượng DTO và các đối tượng Entity. Điều này giúp tách biệt logic chuyển đổi khỏi các tầng khác và giữ cho mã nguồn sạch sẽ và dễ bảo trì.

**Chức năng:**

* Chuyển đổi từ Entity sang DTO và ngược lại.
* Đảm bảo tính nhất quán trong việc chuyển đổi dữ liệu giữa các lớp.
* Đơn giản hóa mã nguồn bằng cách giữ logic chuyển đổi ở một nơi duy nhất.

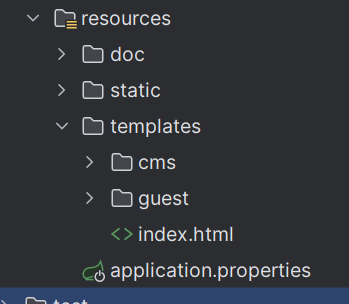
**Ví dụ:**

@Mapper(componentModel = "spring")  
public interface ClaimMapper {  
 @Mapping(source = "code", target = "claimCode")  
 @Mapping(source = "customerEntity.name", target = "customerName")  
 @Mapping(source = "insuranceProductEntity.name", target = "insuranceProductName")  
 @Mapping(source = "claimStatusEntity.code", target = "status")  
 @Mapping(source = "claimStatusEntity.description", target = "statusDescription")  
 ClaimDTO toDTO(ClaimEntity entity);

**Giải thích các Annotation đã dùng:**

* @Mapper(componentModel = "spring"): Đánh dấu lớp này là một mapper, cho phép Spring quản lý mapper này như một bean.

### 7. Giải thích về các thư mục và file trong resources



#### 1. Thư mục static

##### Mô tả:

Thư mục static chứa các tài nguyên tĩnh như CSS, JavaScript, và các hình ảnh. Các tài nguyên này sẽ được phục vụ trực tiếp từ thư mục static mà không cần bất kỳ xử lý nào từ phía server.

##### Ý nghĩa:

* **css**: Chứa các file CSS để định dạng giao diện người dùng (thư mục này không có trong cấu trúc hiện tại của bạn, nhưng thường có thể xuất hiện).
* **js**: Chứa các file JavaScript để xử lý logic phía client (thư mục này không có trong cấu trúc hiện tại của bạn, nhưng thường có thể xuất hiện).
* **images**: Chứa các hình ảnh được sử dụng trong ứng dụng (thư mục này không có trong cấu trúc hiện tại của bạn, nhưng thường có thể xuất hiện).

#### 2. Thư mục templates

##### Mô tả:

Thư mục templates chứa các template (thường là các file HTML) được sử dụng bởi Spring MVC hoặc Spring WebFlux để tạo các trang web động.

##### Ý nghĩa:

* Các file HTML trong thư mục này có thể chứa các phần tử động được sinh ra từ dữ liệu trên server.

#### 3. File application.properties

##### Mô tả:

application.properties là file cấu hình chính cho ứng dụng Spring Boot. Nó chứa các thiết lập cấu hình như thông tin kết nối cơ sở dữ liệu, cấu hình các bean, và các thiết lập khác của ứng dụng.

##### Ý nghĩa:

* Quản lý các cấu hình của ứng dụng trong một nơi duy nhất.
* Dễ dàng thay đổi và kiểm soát các thiết lập của ứng dụng mà không cần thay đổi mã nguồn.

##### Ví dụ:

spring.application.name=insurance-claim  
# MySQL configurations  
spring.datasource.url=jdbc:mysql://localhost:3306/insuranceclaim?allowPublicKeyRetrieval=true&useSSL=false  
spring.datasource.username=root  
#password s? tùy thu?c vào password b?n ??t cho mysql server, nêsu ai dùng mysql phpmyadmin có th? b? tr?ng n?u ko có  
spring.datasource.password=root  
# JPA configurations  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.hikari.maximum-pool-size=10

### Tổng kết về các package và file trong resources

* **Thư mục static**: Chứa các tài nguyên tĩnh như CSS, JavaScript, và hình ảnh (hiện tại chưa có các file cụ thể trong cấu trúc của bạn).
* **Thư mục templates**: Chứa các template HTML để tạo các trang web động (hiện tại chưa có các file cụ thể trong cấu trúc của bạn).
* **File application.properties**: Chứa các thiết lập cấu hình cho ứng dụng Spring Boot.

### Tóm tắt chung về toàn bộ kiến trúc của Project

Project của bạn bao gồm các tầng sau:

1. **Entity Layer**
2. **Repository Layer**
3. **Service Layer**
4. **Controller Layer**
5. **DTO Layer**
6. **Mapper Layer**

#### 1. Entity Layer

* **@Entity**: Đánh dấu lớp này là một thực thể JPA.
* **@Id**: Chỉ định khóa chính của thực thể.
* **@GeneratedValue(strategy = GenerationType.IDENTITY)**: Chỉ định chiến lược tạo giá trị tự động cho khóa chính.
* **@ManyToOne**: Chỉ định mối quan hệ nhiều-đến-một.
* **@JoinColumn(name = "customer\_id")**: Chỉ định cột trong bảng cơ sở dữ liệu liên kết với thực thể khác.

#### 2. Repository Layer

* **@Repository**: Đánh dấu lớp này là một repository trong Spring.
* **JpaRepository<ClaimEntity, Long>**: Interface của Spring Data JPA, cung cấp các phương thức CRUD tiêu chuẩn.

#### 3. Service Layer

* **@Service**: Đánh dấu lớp này là một service trong Spring.
* **@Autowired**: Cho phép Spring tự động tiêm dependency.

#### 4. Controller Layer

* **@RestController**: Kết hợp giữa @Controller và @ResponseBody, chỉ định rằng lớp này là một controller RESTful.
* **@RequestMapping("/api/claims")**: Chỉ định đường dẫn cơ bản cho tất cả các yêu cầu HTTP của lớp controller này.
* **@PostMapping**: Xử lý các yêu cầu HTTP POST.
* **@GetMapping**: Xử lý các yêu cầu HTTP GET.
* **@RequestBody**: Gán dữ liệu từ yêu cầu HTTP vào đối tượng DTO.
* **@PathVariable**: Trích xuất biến từ URL để sử dụng làm tham số phương thức.

#### 5. DTO Layer

* Chuyển đổi dữ liệu giữa client và server.

#### 6. Mapper Layer

* **@Mapper(componentModel = "spring")**: Đánh dấu lớp này là một mapper, cho phép Spring quản lý mapper này như một bean.

### Thư mục resources

* **application.properties**: Chứa các thiết lập cấu hình cho ứng dụng Spring Boot.
* **static**: Chứa các tài nguyên tĩnh như CSS, JavaScript, và hình ảnh.
* **templates**: Chứa các template HTML để tạo các trang web động.