**Spring Data JPA**

**1. Giới Thiệu về Spring Data JPA**

**Định nghĩa**

Spring Data JPA là một phần của Spring Framework, giúp đơn giản hóa việc thao tác với cơ sở dữ liệu bằng cách cung cấp một lớp trừu tượng cao hơn so với JPA (Java Persistence API). Nó giúp bạn giảm thiểu việc viết mã xử lý CRUD (Create, Read, Update, Delete) và cung cấp các công cụ mạnh mẽ để thực hiện truy vấn dữ liệu.

**Mục đích**

Việc sử dụng Spring Data JPA mang lại nhiều lợi ích như:

* **Tự động hóa:** Giảm thiểu lượng mã phải viết cho các thao tác cơ bản như tìm kiếm, lưu trữ, và xóa dữ liệu.
* **Tăng hiệu suất:** Tận dụng khả năng caching, lazy loading, và eager fetching của JPA.
* **Tính linh hoạt:** Hỗ trợ các truy vấn tùy chỉnh thông qua @Query, JPQL, và native query.

**Khi nào sử dụng?**

Spring Data JPA phù hợp khi bạn làm việc với các hệ thống có cơ sở dữ liệu quan hệ và cần một công cụ mạnh mẽ để xử lý các thao tác CRUD cũng như truy vấn dữ liệu phức tạp.

**Cách triển khai**

Bạn sẽ sử dụng Spring Data JPA bằng cách định nghĩa các entity, repository, và áp dụng các kỹ thuật truy vấn thông qua @Query và các phương thức có sẵn.

**2. Repository trong Spring Data JPA**

**Định nghĩa**

Repository là một interface trong Spring Data JPA, cung cấp các phương thức mặc định để thao tác với cơ sở dữ liệu như save(), findById(), findAll(), delete(), v.v. Bạn có thể mở rộng từ JpaRepository hoặc CrudRepository.

**Mục đích**

Repository giúp bạn tập trung vào logic nghiệp vụ thay vì phải viết các thao tác cơ sở dữ liệu thủ công. Nó giảm thiểu lỗi và tăng tốc độ phát triển ứng dụng.

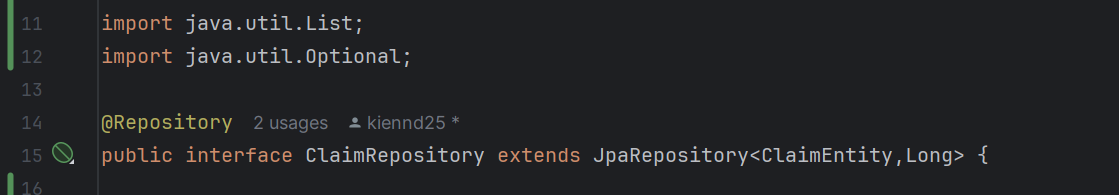
**Khi nào sử dụng?**

Sử dụng Repository khi bạn cần tương tác với cơ sở dữ liệu, đặc biệt là khi bạn cần thực hiện các thao tác CRUD hoặc các truy vấn đơn giản đến phức tạp.

**Cách triển khai**

Bạn định nghĩa một interface mở rộng từ JpaRepository hoặc CrudRepository và Spring Data JPA sẽ tự động triển khai các phương thức của nó.

**Ví dụ:**



**3. Các Hàm Có Sẵn trong Repository**

**Định nghĩa**

Spring Data JPA cung cấp một loạt các phương thức có sẵn trong repository để thao tác với dữ liệu.

**Mục đích**

Những phương thức có sẵn này giúp tiết kiệm thời gian và công sức khi thực hiện các thao tác thông thường như lưu trữ, tìm kiếm, cập nhật và xóa dữ liệu.

**Khi nào sử dụng?**

Bạn sử dụng các phương thức này khi cần thực hiện các thao tác CRUD cơ bản hoặc tìm kiếm dữ liệu mà không cần viết truy vấn tùy chỉnh.

**Cách triển khai**

Bạn chỉ cần gọi các phương thức này từ repository mà không cần viết bất kỳ mã truy vấn nào.

**Ví dụ:**

* save(S entity): Lưu một thực thể vào cơ sở dữ liệu.
* findById(ID id): Tìm một thực thể dựa trên ID.
* findAll(): Trả về tất cả các thực thể trong cơ sở dữ liệu.
* deleteById(ID id): Xóa một thực thể dựa trên ID.

**Chi tiết:**

* save(S entity): Lưu một thực thể, nếu thực thể đã tồn tại thì cập nhật, nếu không thì tạo mới.
* findById(ID id): Tìm một thực thể theo ID. Trả về một Optional, giúp tránh lỗi NullPointerException.
* findAll(): Trả về một danh sách tất cả các thực thể, thường sử dụng khi bạn cần liệt kê tất cả các bản ghi trong bảng.
* deleteById(ID id): Xóa một thực thể theo ID, thực hiện hành động DELETE trên cơ sở dữ liệu.

**4. Sử Dụng @Query và Các Kết Quả Nhận Được**

**Định nghĩa**

@Query là một annotation trong Spring Data JPA, cho phép bạn viết các truy vấn tùy chỉnh sử dụng JPQL hoặc Native SQL.

**Mục đích**

Sử dụng @Query cho phép bạn thực hiện các truy vấn phức tạp mà không thể thực hiện được chỉ với các phương thức có sẵn.

**Khi nào sử dụng?**

Bạn sử dụng @Query khi cần thực hiện các truy vấn phức tạp, như truy vấn với nhiều điều kiện, kết hợp nhiều bảng, hoặc khi cần sử dụng các chức năng SQL nâng cao.

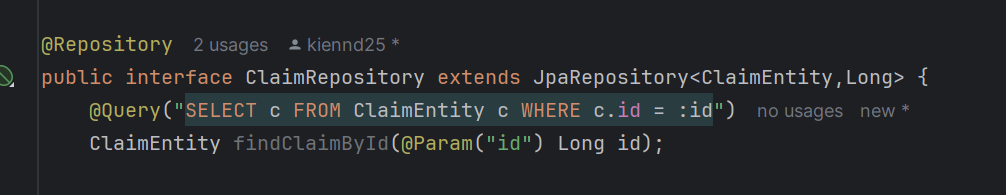
**Cách triển khai**

Bạn có thể sử dụng @Query với các kiểu trả về khác nhau như Object, Option, List, Page, v.v.

**3.1) Trả Về Object**

**Định nghĩa**  
Trả về một thực thể(Entity) hoặc một đối tượng tùy chỉnh từ truy vấn.

**Cách triển khai**

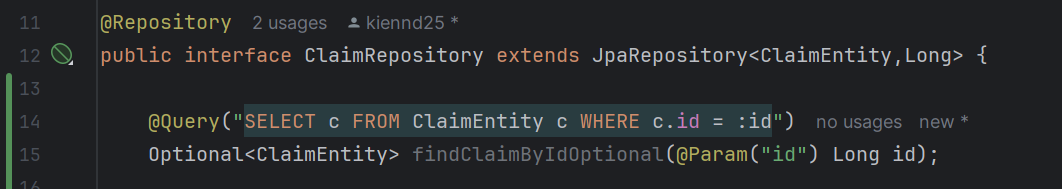


**Giải thích:**  
Truy vấn này trả về một thực thể ClaimEntity có ID tương ứng.

**3.2) Trả Về Optional**

**Định nghĩa**  
Optional là một container object có thể chứa giá trị null. Điều này giúp tránh lỗi NullPointerException.

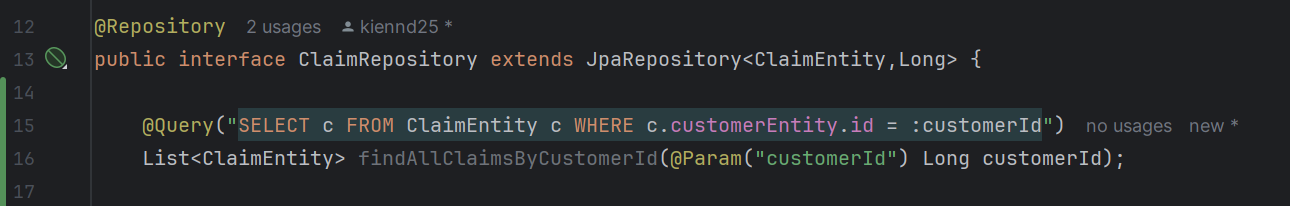
**Cách triển khai**

**Giải thích:**  
Truy vấn này trả về một Optional chứa ClaimEntity nếu tìm thấy, nếu không sẽ trả về Optional.empty().

**3.3) Trả Về List**

**Định nghĩa**  
Trả về danh sách các thực thể hoặc đối tượng tùy chỉnh từ truy vấn.

**Cách triển khai**



**Giải thích:**  
Truy vấn này trả về một danh sách các yêu cầu bồi thường liên quan đến customerId cung cấp.

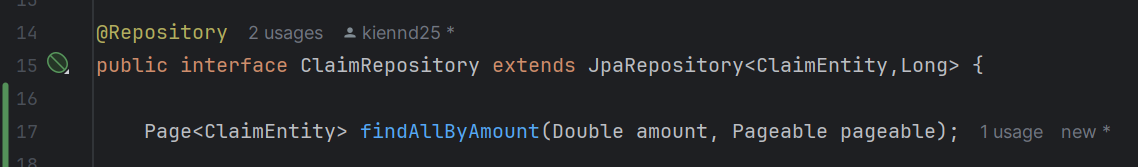
**3.4) Trả Về Page, Pageable và Các Thuộc Tính của Page và Pageable**

**Pageable là gì?** Pageable là một interface định nghĩa thông tin về phân trang và sắp xếp.

**Page là gì?** Page là một interface đại diện cho một trang kết quả của truy vấn, chứa danh sách kết quả và thông tin về phân trang.

**Mục đích** Phân trang giúp bạn xử lý một lượng lớn dữ liệu hiệu quả hơn bằng cách chỉ lấy một phần nhỏ dữ liệu trong mỗi lần truy vấn.

**Cách triển khai tại tầng repository và service**



****

**Giải thích chi tiết các thuộc tính:**

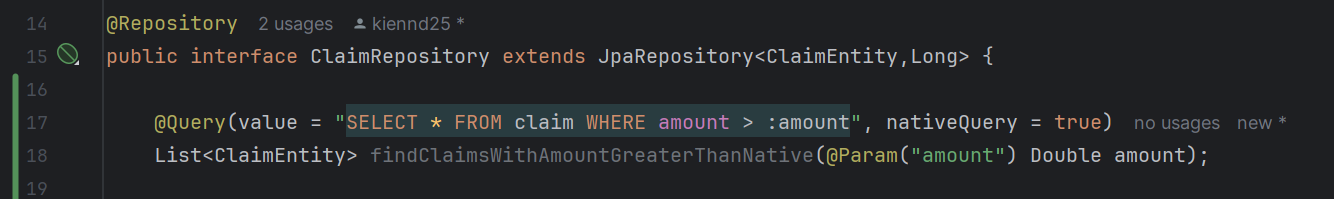
* pageNumber: Số thứ tự của trang (bắt đầu từ 0).
* pageSize: Số lượng phần tử trong một trang.
* sort: Sắp xếp kết quả theo thuộc tính cụ thể.

**3.5) Sử Dụng @Query với Native Query**

**Định nghĩa**  
Native query là các truy vấn SQL gốc mà không thông qua JPQL.

**Mục đích**  
Native query cho phép bạn tận dụng toàn bộ sức mạnh của SQL và truy cập các tính năng cụ thể của hệ quản trị cơ sở dữ liệu.

**Cách triển khai**



**Giải thích:**  
Truy vấn này trả về một danh sách các yêu cầu bồi thường có số tiền lớn hơn giá trị được chỉ định.

**5. Sắp Xếp (Sorting), Các Hàm và Toán Tử**

**Định nghĩa**

Sắp xếp là quá trình sắp xếp các kết quả truy vấn theo một hoặc nhiều thuộc tính theo thứ tự tăng dần hoặc giảm dần.

**Mục đích**

Sắp xếp giúp bạn dễ dàng quản lý và hiển thị dữ liệu theo thứ tự mong muốn, chẳng hạn như theo ngày tạo, tên, hoặc số tiền.

**Khi nào sử dụng?**

Bạn sử dụng sắp xếp khi cần hiển thị dữ liệu theo một thứ tự nhất định hoặc khi cần xử lý dữ liệu có thứ tự.

**Cách triển khai**

****

Bạn có thể sử dụng Sort trong Spring Data JPA để sắp xếp kết quả truy vấn.

**Giải thích:**  
Truy vấn này trả về tất cả các yêu cầu bồi thường và sắp xếp chúng theo số tiền giảm dần.

**6. Cách Sử Dụng @Param**

**Định nghĩa**

@Param là một annotation trong Spring Data JPA, được sử dụng để gán các giá trị tham số cho các biến trong truy vấn.

**Mục đích**

@Param giúp bạn truyền các giá trị từ phương thức vào truy vấn một cách rõ ràng và dễ quản lý.

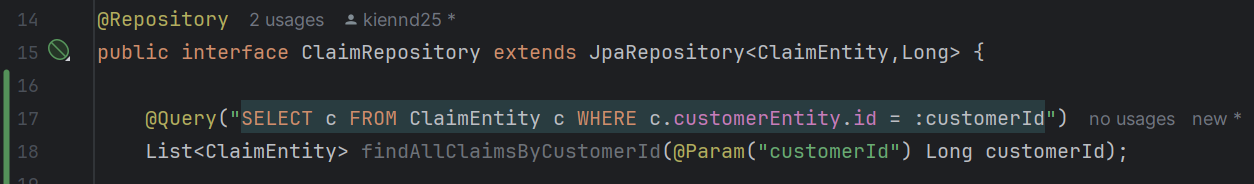
**Khi nào sử dụng?**

Bạn sử dụng @Param khi cần truyền các tham số từ phương thức vào truy vấn JPQL hoặc native query.

**Cách triển khai**

**5.1. Truyền Tham Số Đơn Giản với @Param**

**Cách triển khai**

**Giải thích:**  
Truy vấn này nhận tham số customerId từ phương thức và sử dụng nó trong truy vấn.

**5.2. Truyền Tham Số với Object Filter**

**Định nghĩa**  
Bạn có thể truyền một đối tượng filter chứa nhiều thuộc tính để xây dựng các truy vấn linh hoạt.

**Cách triển khai**

**Giải thích:**  
Hàm findByCondition được sử dụng để truy vấn danh sách các yêu cầu bồi thường (ClaimEntity) dựa trên các điều kiện tìm kiếm được truyền vào thông qua đối tượng ClaimRequest. Kết quả truy vấn sẽ được trả về dưới dạng phân trang (Page<ClaimEntity>), cho phép người dùng nhận được dữ liệu trong từng trang cụ thể.

**Điều Kiện Truy Vấn**

* (:#{#request.claimCode} = '' OR LOWER(c.code) LIKE LOWER(CONCAT('%', :#{#request.claimCode}, '%')))

**Giải thích:**

* + Điều kiện này kiểm tra xem giá trị claimCode trong ClaimRequest có rỗng hay không.
  + Nếu claimCode rỗng (''), điều kiện này sẽ luôn đúng (TRUE), tức là không lọc dựa trên mã yêu cầu bồi thường.
  + Nếu claimCode không rỗng, hệ thống sẽ so sánh mã yêu cầu bồi thường (c.code) với giá trị claimCode trong yêu cầu, không phân biệt chữ hoa chữ thường và sử dụng toán tử LIKE với wildcard % để tìm kiếm các mã có chứa claimCode.
* AND (:#{#request.statusCode} = '' OR c.claimStatusEntity.code = :#{#request.statusCode})

**Giải thích:**

* + Điều kiện này kiểm tra statusCode trong ClaimRequest.
  + Nếu statusCode rỗng, điều kiện này sẽ luôn đúng (TRUE).
  + Nếu statusCode không rỗng, hệ thống sẽ lọc các yêu cầu bồi thường có mã trạng thái (claimStatusEntity.code) trùng khớp với statusCode.
* AND (:#{#request.fromDateSearch} IS NULL OR c.claimDate >= :#{#request.fromDateSearch})

**Giải thích:**

* + Điều kiện này kiểm tra giá trị fromDateSearch.
  + Nếu fromDateSearch là NULL, điều kiện này sẽ luôn đúng (TRUE).
  + Nếu không, hệ thống sẽ lọc các yêu cầu bồi thường có ngày yêu cầu bồi thường (claimDate) lớn hơn hoặc bằng fromDateSearch.
* AND (:#{#request.toDateSearch} IS NULL OR c.claimDate <= :#{#request.toDateSearch})

**Giải thích:**

* + Điều kiện này kiểm tra giá trị toDateSearch.
  + Nếu toDateSearch là NULL, điều kiện này sẽ luôn đúng (TRUE).
  + Nếu không, hệ thống sẽ lọc các yêu cầu bồi thường có ngày yêu cầu bồi thường (claimDate) nhỏ hơn hoặc bằng toDateSearch.
* AND (c.deleted = false OR c.deleted IS NULL)

**Giải thích:**

* + Điều kiện này lọc các yêu cầu bồi thường chưa bị xóa (deleted = false) hoặc có giá trị deleted là NULL. Điều này đảm bảo rằng chỉ những yêu cầu bồi thường hợp lệ, chưa bị xóa mới được trả về.

Hàm findByCondition là một hàm mạnh mẽ cho phép bạn truy vấn các yêu cầu bồi thường dựa trên nhiều tiêu chí khác nhau như mã yêu cầu, mã trạng thái, ngày yêu cầu, và trạng thái xóa. Các tiêu chí này có thể được kết hợp với nhau để tạo ra các truy vấn tùy chỉnh. Hơn nữa, kết quả của truy vấn được phân trang, giúp quản lý và hiển thị dữ liệu hiệu quả hơn trong các ứng dụng có dữ liệu lớn.

**7. Kết Luận**

Spring Data JPA cung cấp một loạt các công cụ mạnh mẽ để tương tác với cơ sở dữ liệu. Từ việc sử dụng các phương thức CRUD mặc định, đến việc tạo các truy vấn tùy chỉnh với @Query, bạn có thể dễ dàng xây dựng các ứng dụng mạnh mẽ và linh hoạt. Việc kết hợp Pageable, @Param, và các đối tượng filter giúp bạn quản lý dữ liệu phức tạp một cách hiệu quả. Hy vọng rằng bài giảng này đã giúp bạn nắm vững các khái niệm và cách sử dụng Spring Data JPA trong dự án của mình.

**RESTful API trong Spring Boot: Hướng dẫn chi tiết**

RESTful API là một trong những chuẩn phổ biến nhất hiện nay để xây dựng các dịch vụ web. Trong bài viết này, chúng ta sẽ tìm hiểu về RESTful API là gì, cách hoạt động, và cách xây dựng một RESTful API cơ bản bằng Spring Boot. Đồng thời, chúng ta cũng sẽ đi sâu vào các phương thức truyền tham số phổ biến như @PathVariable, @RequestBody, @RequestParam, cách sử dụng các mã trạng thái HTTP chuẩn, và những quy tắc quan trọng trong RESTful API.



**1) Diễn giải các thành phần**

**API (Application Programming Interface)** là một tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML.

**REST (REpresentational State Transfer)** là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. Nó sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy. Vì vậy, thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE, vv đến một URL để xử lý dữ liệu.

**RESTful API là** một tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các resource. RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile…) khác nhau giao tiếp với nhau.

**Chức năng quan trọng nhất của REST** là quy định cách sử dụng các HTTP method (như GET, POST, PUT, DELETE…) và cách định dạng các URL cho ứng dụng web để quản các resource. RESTful không quy định logic code ứng dụng và không giới hạn bởi ngôn ngữ lập trình ứng dụng, bất kỳ ngôn ngữ hoặc framework nào cũng có thể sử dụng để thiết kế một RESTful API.

**RESTful hoạt động như thế nào?**

REST hoạt động chủ yếu dựa vào giao thức HTTP. Các hoạt động cơ bản nêu trên sẽ sử dụng những phương thức HTTP riêng.

* GET (SELECT): Trả về một Resource hoặc một danh sách Resource.
* POST (CREATE): Tạo mới một Resource.
* PUT (UPDATE): Cập nhật thông tin cho Resource.
* DELETE (DELETE): Xoá một Resource.

Những phương thức hay hoạt động này thường được gọi là **CRUD** tương ứng với Create, Read, Update, Delete – Tạo, Đọc, Sửa, Xóa.

**2) Các Nguyên Tắc Cơ Bản của REST**

Một API RESTful phải tuân thủ một số nguyên tắc cơ bản, bao gồm:

* **Stateless**: Mỗi yêu cầu từ client đến server phải tự chứa đầy đủ thông tin cần thiết. Server không lưu trữ bất kỳ trạng thái nào giữa các yêu cầu.
* **Client-Server**: Giao diện giữa client và server phải rõ ràng và tách biệt.
* **Cacheable**: Các phản hồi có thể được lưu trữ tạm thời để cải thiện hiệu suất.
* **Uniform Interface**: RESTful API phải tuân theo một giao diện nhất quán.

**Quy Tắc Sử Dụng Các Phương Thức HTTP trong RESTful API**

* **GET**: Nên được sử dụng cho các thao tác đọc dữ liệu. Không bao giờ thay đổi trạng thái của tài nguyên.
* **POST**: Sử dụng khi cần tạo mới một tài nguyên. Dữ liệu được gửi qua body của yêu cầu.
* **PUT**: Được sử dụng để thay thế toàn bộ tài nguyên. Nếu tài nguyên không tồn tại, nó có thể được tạo mới.
* **DELETE**: Sử dụng để xóa tài nguyên.

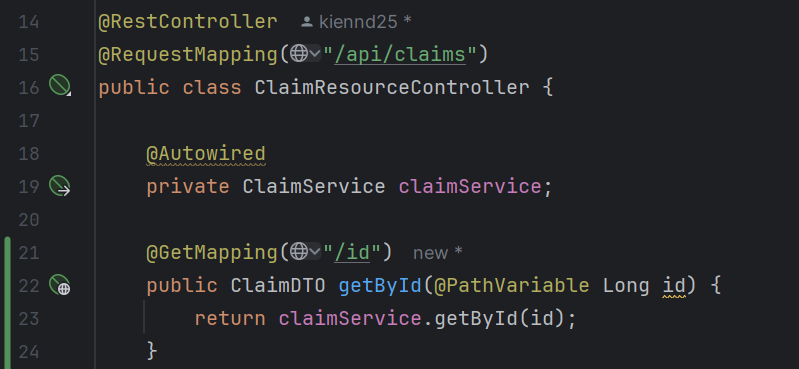
**Các Annotation mô phỏng phương thức HTTP trong Spring Boot**

Trong Spring Boot, các phương thức HTTP của RESTful API được mô phỏng bằng các annotation cụ thể. Dưới đây là một số annotation chính và cách sử dụng chúng.

**1. @GetMapping**

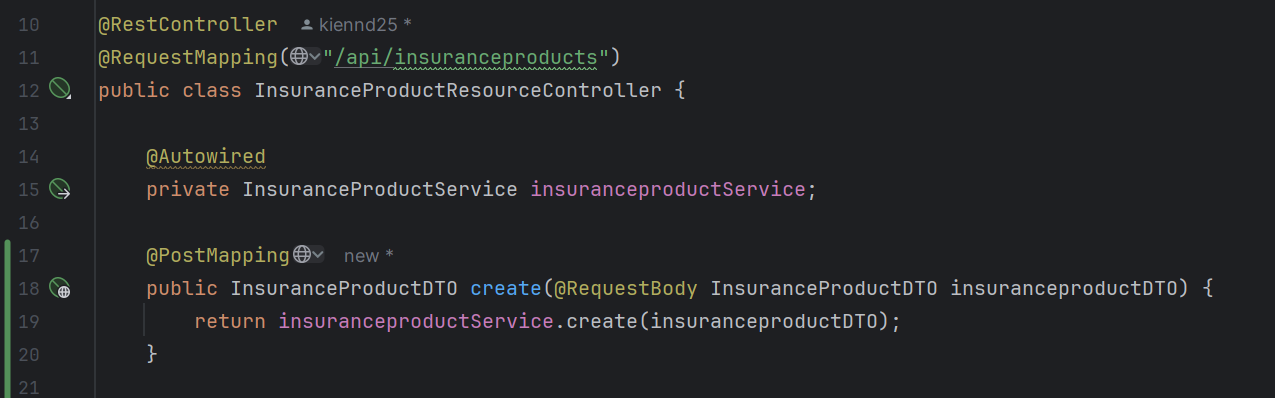
@GetMapping tương ứng với phương thức GET trong HTTP. Nó được sử dụng để truy vấn dữ liệu từ server mà không làm thay đổi trạng thái của tài nguyên.

**Ví dụ:**

**2. @PostMapping**

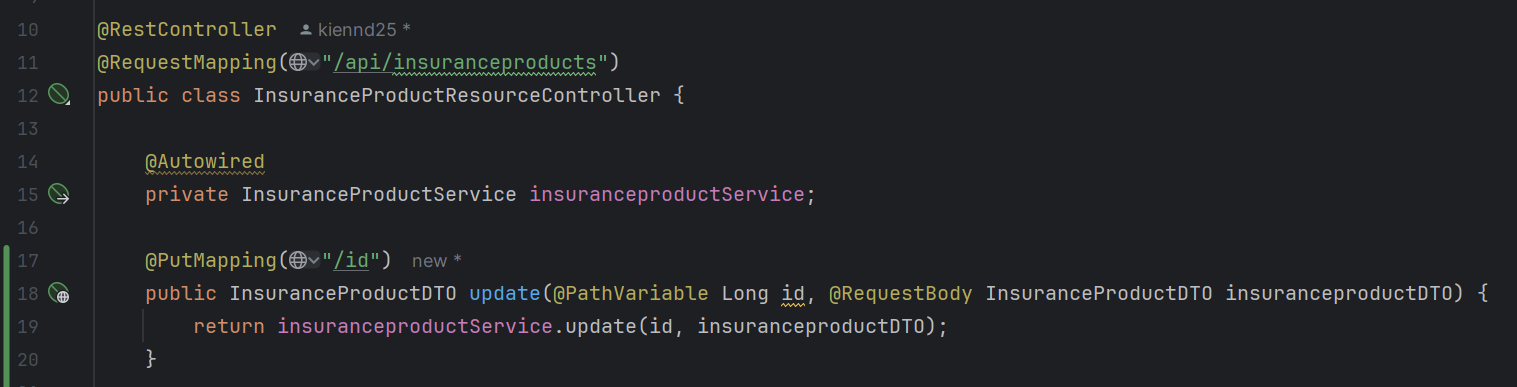
@PostMapping tương ứng với phương thức POST trong HTTP. Được sử dụng để tạo mới một tài nguyên trên server.

**Ví dụ:**

**3. @PutMapping**

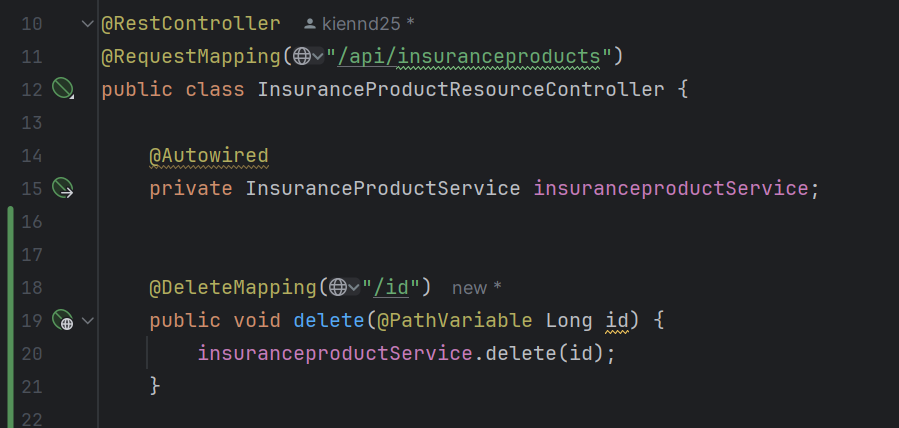
@PutMapping tương ứng với phương thức PUT trong HTTP. Sử dụng để cập nhật toàn bộ một tài nguyên hiện có trên server.

**Ví dụ:**

**4. @DeleteMapping**

@DeleteMapping tương ứng với phương thức DELETE trong HTTP. Được sử dụng để xóa một tài nguyên khỏi server.

**Ví dụ:**



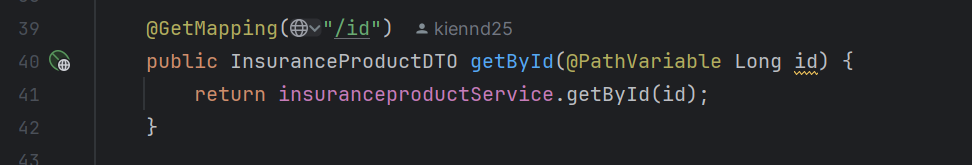
**Truyền Tham Số trong RESTful API với Spring Boot**

Trong Spring Boot, có ba cách chính để truyền tham số vào các phương thức xử lý yêu cầu trong RESTful API: @PathVariable, @RequestBody, và @RequestParam.

**1. @PathVariable**

@PathVariable được sử dụng để lấy giá trị từ đường dẫn URL. Đây là cách tốt nhất khi dữ liệu cần truyền là một phần của URL.

**Ví dụ:**



**Ưu điểm:**

* Rất rõ ràng và dễ hiểu khi dữ liệu cần truyền là một phần của tài nguyên URL.
* Tốt cho việc thiết kế API RESTful thân thiện với SEO.

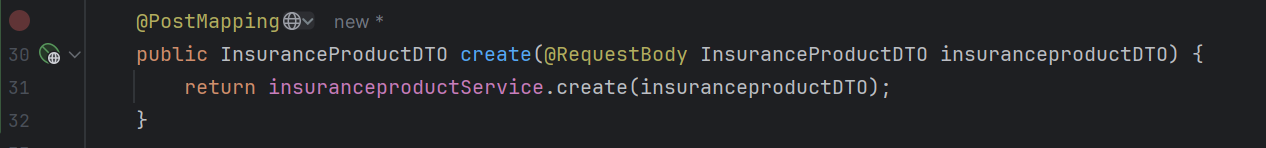
**Nhược điểm:**

* Không thích hợp cho các tham số phức tạp hoặc nhạy cảm.

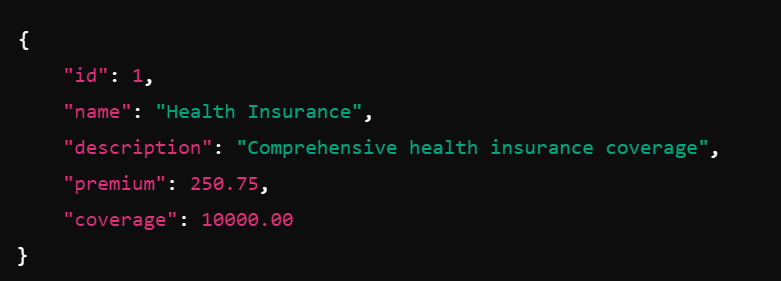
**2. @RequestBody**

@RequestBody được sử dụng để trích xuất dữ liệu từ phần body của HTTP request và ánh xạ nó vào một đối tượng Java. Thường dùng khi cần gửi dữ liệu phức tạp như JSON hoặc XML.

**Ví dụ:**

**Cách Truyền JSON:** Khi sử dụng @RequestBody, bạn có thể gửi dữ liệu JSON từ client đến server. Spring Boot sẽ tự động chuyển đổi JSON thành đối tượng Java.

**Ví dụ JSON:**



Dữ liệu này sẽ được chuyển thành đối tượng Product trong Spring Boot.

Trong ví dụ này:

* "id" là ID của sản phẩm bảo hiểm, giá trị là một số nguyên.
* "name" là tên của sản phẩm bảo hiểm.
* "description" là mô tả của sản phẩm.
* "premium" là mức phí bảo hiểm, ở đây được biểu diễn bằng số thập phân.
* "coverage" là mức bảo hiểm chi trả tối đa, cũng biểu diễn bằng số thập phân.

**Ưu điểm:**

* Dễ dàng xử lý dữ liệu phức tạp, đặc biệt là khi cần gửi nhiều thông tin.
* Dữ liệu JSON phổ biến và dễ đọc.

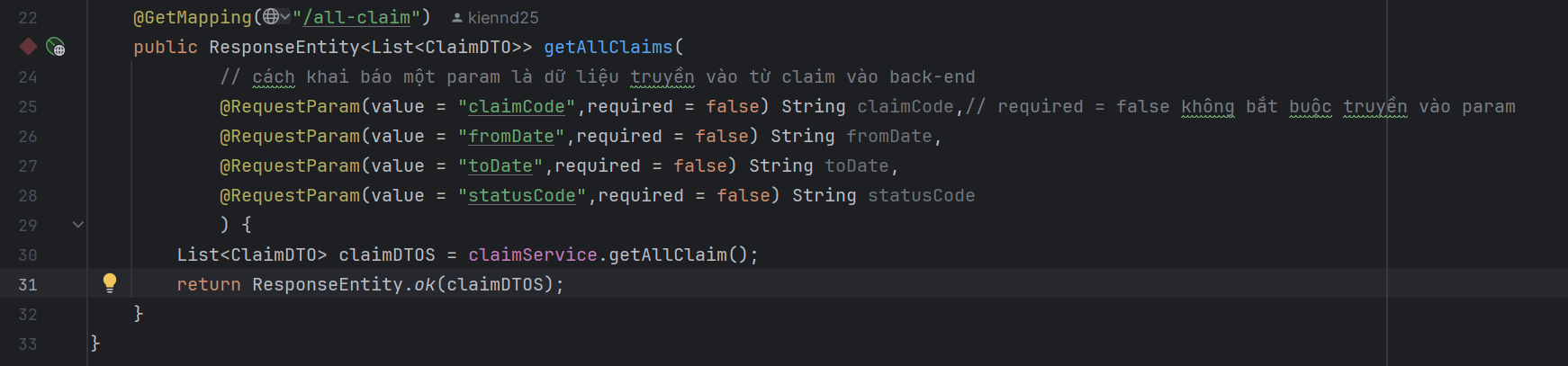
**Nhược điểm:**

* Khó bảo mật nếu không có các biện pháp kiểm tra và xác thực dữ liệu.
* Không thích hợp cho các tham số đơn giản.

**3. @RequestParam**

@RequestParam được sử dụng để lấy dữ liệu từ query string của URL, thường cho các tham số không thuộc phần đường dẫn của URL.

**Ví dụ:**

**Ưu điểm:**

* Rất tiện lợi cho các tham số đơn giản, như filter hoặc search.
* Dễ sử dụng và hiểu.

**Nhược điểm:**

* Không phù hợp cho dữ liệu phức tạp.
* URL có thể trở nên dài và khó đọc nếu có quá nhiều tham số.

**So Sánh và Gợi Ý Sử Dụng**

* **@PathVariable**: Sử dụng khi dữ liệu là một phần của tài nguyên và cần được chỉ định trong URL. Tốt cho các API thân thiện với người dùng và SEO. Tuy nhiên, tránh sử dụng cho dữ liệu nhạy cảm.
* **@RequestBody**: Sử dụng khi cần gửi dữ liệu phức tạp từ client đến server. Rất mạnh mẽ khi làm việc với JSON, nhưng cần phải thận trọng về bảo mật, đặc biệt khi làm việc với dữ liệu nhạy cảm.
* **@RequestParam**: Phù hợp cho các tham số đơn giản, chẳng hạn như các giá trị filter hoặc search. Dễ sử dụng, nhưng không lý tưởng cho dữ liệu phức tạp.

**Mã Trạng Thái HTTP (HTTP Status Codes)**

Mã trạng thái HTTP là các mã số đi kèm với phản hồi từ server để cho biết kết quả của yêu cầu.

* **200 OK** – Trả về thành công cho những phương thức GET, PUT, PATCH hoặc DELETE.
* **201 Created** – Trả về khi một Resouce vừa được tạo thành công.
* **204 No Content** – Trả về khi Resource xoá thành công.
* **304 Not Modified** – Client có thể sử dụng dữ liệu cache.
* **400 Bad Request** – Request không hợp lệ
* **401 Unauthorized** – Request cần có auth.
* **403 Forbidden** – bị từ chối không cho phép.
* **404 Not Found** – Không tìm thấy resource từ URI
* **405 Method Not Allowed** – Phương thức không cho phép với user hiện tại.
* **410 Gone** – Resource không còn tồn tại, Version cũ đã không còn hỗ trợ.
* **415 Unsupported Media Type** – Không hỗ trợ kiểu Resource này.
* **422 Unprocessable Entity** – Dữ liệu không được xác thực
* **429 Too Many Requests** – Request bị từ chối do bị giới hạn

**Kết Luận**

RESTful API là một phương pháp phát triển dịch vụ web phổ biến và hiệu quả. Sử dụng Spring Boot, bạn có thể dễ dàng xây dựng một RESTful API đáp ứng các nguyên tắc chuẩn của REST, đồng thời tận dụng sức mạnh của Spring để xử lý các yêu cầu HTTP phức tạp. Việc chọn lựa các phương thức truyền tham số phụ thuộc vào ngữ cảnh cụ thể của ứng dụng, đồng thời cần cân nhắc các yếu tố về bảo mật và tính tiện lợi.

Code demo:

<https://github.com/ndkien98/LJAVA2402_V2/tree/master/Buoi19_Spring/claim-manager>

<https://github.com/ndkien98/T3H---UTC_Insurance_claim/tree/master/insurance-claims/insurance-claim>