**RESTful api**

Để phục vụ cho kiến trúc 3 lớp, spring tạo ra 3 anotation đại diện cho 3 layer: **@Controller**, **@Service**, **@Repository**. Về bản chất 3 anotation này củng chính là **@Component** tuy nhiên từ gốc độ thiết kế ta cần phân rõ các annotation này cho các class đảm nhiệm đúng nhiệm vụ của nó

**@Controller**

* **@Controller** được sử dụng để đánh dấu một class là một controller trong mô hình MVC. Bằng cách đánh dấu một class bằng **@Controller**, Spring sẽ nhận biết rằng class này là một controller và nó có thể xử lý các yêu cầu từ client
* Controller đóng vai trò trong việc định tuyến các yêu cầu từ client đến các phương thức xử lý tương ứng. Bằng cách sử dụng các annotation như **@RequestMapping**, **@GetMapping**, **@PostMapping**,...chúng ta có thể xác định các URL và HTTP methods mà các phương thức trong controller sẽ xử lý
* Các phương thức trong controller có thể trả về bất kỳ loại dữ liệu nào, chẳng hạn như trang HTML, hoặc định dạng khác như JSON, XML. **@Controller** thường được sử dụng để trả về view, và chúng ta thường sử dụng các template engine như Thymeleaf, JSP, hoặc FreeMarker để tạo và hiển thị các trang web động. Tuy nhiên, với cấu hình và xử lý phù hợp, **@Controller** cũng có thể trả về các định dạng khác như JSON thông qua sử dụng **@ResponseBody** hoặc các cơ chế chuyển đổi dữ liệu khác như Jackson hoặc Gson
* Tóm lại, **@Controller** cho phép chúng ta xử lý các yêu cầu từ client, định tuyến chúng đến các phương thức xử lý tương ứng, và trả về các đối tượng phản hồi, bao gồm cả view và các định dạng khác như JSON
* Phân biệt 2 controller giống tên class bằng cách thêm định nghĩa thuộc tính *value*:

A screenshot of a computer code

Description automatically generated with low confidence

**@RestController**

* **@RestController** = **@Controller** + **@ResponseBody**
* **@RestController** là một biến thể của **@Controller** và thường được sử dụng trong việc xây dựng các API RESTful trong ứng dụng Spring. Khi ta đánh dấu một class controller với **@RestController**, nó sẽ tự động áp dụng **@ResponseBody** cho tất cả các phương thức trong class đó
* **@ResponseBody** được sử dụng để chỉ định rằng kết quả trả về từ phương thức controller sẽ được chuyển đổi tự động thành định dạng phù hợp với yêu cầu từ client. Mặc định, **@ResponseBody** chuyển đổi đối tượng trả về từ controller thành định dạng JSON. Tuy nhiên, chúng ta có thể tuỳ chỉnh định dạng trả về bằng cách sử dụng thuộc tính *produces* trong annotation **@RequestMapping**, **@GetMapping**, **@PostMapping** và các annotation tương tự

A picture containing text, screenshot, font, line

Description automatically generated

**@RequestHeader**

**@RequestHeader** là một annotation để truy cập và sử dụng thông tin từ các header trong HTTP request trong các ứng dụng web

**@RequestBody**

* Khi một yêu cầu HTTP được gửi đến server, dữ liệu thường được gửi kèm trong phần body của yêu cầu, bao gồm các định dạng như JSON, XML hoặc form data. Khi sử dụng **@RequestBody** trên một tham số trong một controller của Spring, dữ liệu từ phần body của yêu cầu (ví dụ: JSON) sẽ tự động ánh xạ vào tham số đó
* Spring sử dụng các *HTTP Message Converters* để chuyển đổi dữ liệu từ định dạng yêu cầu thành đối tượng Java tương ứng. Mặc định, Spring Boot hỗ trợ chuyển đổi dữ liệu JSON. Để hỗ trợ chuyển đổi dữ liệu các định dạng khác, bạn cần thêm thư viện tương ứng và cấu hình các *HttpMessageConverter* phù hợp

**@RequestMapping**

* Đây là annotation tổng quát để ánh xạ một phương thức của controller với một URL cụ thể. Nó có thể được sử dụng cho các phương thức HTTP như GET, POST, PUT, DELETE và các phương thức khác
* Khi **@RequestMapping** được đặt trên tên của một class controller, nó ánh xạ một tiền tố URL chung cho tất cả các phương thức trong class đó

A screenshot of a computer code

Description automatically generated

* Khi xử lý các yêu cầu HTTP, chúng ta nên sử dụng annotation phù hợp với phương thức HTTP tương ứng. Ví dụ, nếu chúng muốn lấy dữ liệu, chúng nên sử dụng **@GetMapping**. Nếu chúng ta muốn gửi dữ liệu đến server, chúng ta nên sử dụng **@PostMapping**. Các annotation khác như **@PutMapping** và **@DeleteMapping** cũng nên được sử dụng tương ứng với phương thức HTTP tương ứng của yêu cầu

**@GetMapping**

Annotation này ánh xạ một phương thức của controller với một URL cụ thể và chỉ xử lý các yêu cầu HTTP GET. Nó thường được sử dụng khi chúng ta muốn lấy dữ liệu từ server

**@PostMapping**

Tương tự như **@GetMapping**, annotation này ánh xạ một phương thức của controller với một URL cụ thể, nhưng chỉ xử lý các yêu cầu HTTP POST. Nó thường được sử dụng khi chúng ta muốn gửi dữ liệu đến server

**@PutMapping**

Annotation này ánh xạ một phương thức của controller với một URL cụ thể và chỉ xử lý các yêu cầu HTTP PUT. Nó thường được sử dụng khi chúng ta muốn cập nhật dữ liệu trên server

**@DeleteMapping**

Annotation này ánh xạ một phương thức của controller với một URL cụ thể và chỉ xử lý các yêu cầu HTTP DELETE. Nó thường được sử dụng khi chúng ta muốn xóa dữ liệu từ server

**@PathVariable**

Annotation này được sử dụng để trích xuất giá trị từ một phần của URL và gán nó vào một tham số trong phương thức xử lý yêu cầu

A computer code with black text

Description automatically generated

**@RequestParam**

* Annotation này được sử dụng để trích xuất giá trị từ các tham số query trong URL và gán nó vào một tham số trong phương thức xử lý yêu cầu

A picture containing text, screenshot, font, line

Description automatically generated

* Ngoài ra **@RequestParam** còn thuộc tính *required* và *defaultlValue*
  + *required*: Mặc định là true. Nếu được đặt thành true và tham số không được cung cấp trong query string, một exception *MissingServletRequestParameterException* sẽ được ném ra. Nếu được đặt thành false, thì tham sốt có thể bị bỏ qua trong query string
  + *defaultValue*: Đây là giá trị mặc định được sử dụng cho tham số nếu nó không được cung cấp trong query string

A picture containing text, screenshot, font, line

Description automatically generated

**Triển khai API**

* Có 2 cách:
  + Trả về trực tiếp đối tượng:

A screenshot of a computer

Description automatically generated with low confidence

* + Sử dụng ResponseEntity:

A picture containing text, font, screenshot, line

Description automatically generated

* Trong cách sử dụng ResponseEntity, chúng ta có thể tùy chỉnh HTTP status code, headers và body của phản hồi. Điều này hữu ích khi chúng ta muốn tuỳ chỉnh phản hồi trả về dựa trên các trường hợp đặc biệt hoặc xử lý các lỗi tùy chỉnh. Bằng cách sử dụng ResponseEntity, chúng ta có toàn quyền kiểm soát thông tin phản hồi được trả về. Trong khi đó, khi trả về trực tiếp đối tượng, Spring Boot sẽ tự động chuyển đổi đối tượng thành định dạng phù hợp (ví dụ: JSON) và đặt HTTP status code mặc định là 200 OK
* Vì vậy, sự lựa chọn giữa hai cách phụ thuộc vào yêu cầu của ứng dụng của chúng ta. Nếu chúng ta chỉ cần trả về đối tượng mà không cần tùy chỉnh các thông tin phản hồi, thì việc trả về trực tiếp đối tượng là đủ. Tuy nhiên, nếu chúng ta muốn tùy chỉnh phản hồi hoặc xử lý các trường hợp đặc biệt, sử dụng ResponseEntity sẽ cung cấp linh hoạt hơn và cho phép chúng ta điều khiển thông tin phản hồi một cách tùy ý