**Spring Boot và Cấu trúc dự án**

1. **Tổng quan về Spring Boot**
2. **Java Spring Boot là gì?**

- Là một module của Spring framework.

- Cung cấp các tính năng RAD (Rapid Application Development - Phát triển ứng dụng nhanh chóng) cho phép tạo ứng dụng độc lập dựa trên Spring nhanh chóng.

- Mục đích: loại bỏ cấu hình phức tạp của Spring (không yêu cầu cấu hình XML, nâng cao hiệu suất làm việc).

1. **Ưu điểm**

- Có đầy đủ các tính năng của Spring framework.

- Đơn giản hóa cấu hình và xây dựng được các ứng dụng độc lập có khả năng chạy bằng java-jar nhờ các dependency starter.

- Dễ dàng deploy: các ứng dụng server được nhúng trực tiếp vào ứng dụng, tránh những khó khăn khi triển khai lên môi trường production và không cần thiết phải tải file WAR.

1. **Đặc tính cơ bản của Spring Boot**

- Cung cấp sẵn một lớp (Spring Application) chứa hàm main, là lớp mở đầu cho toàn bộ chương trình.

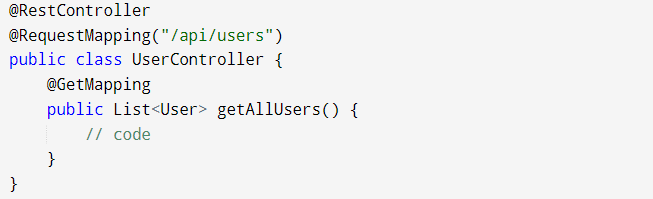
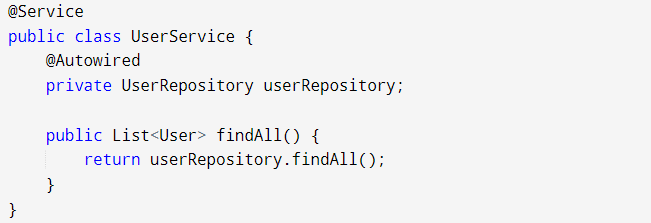
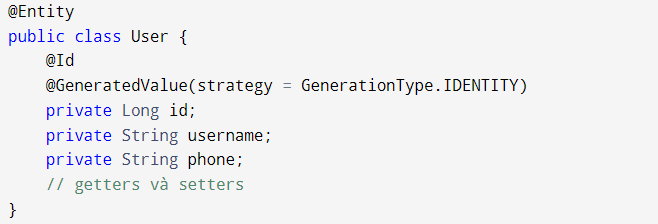
- Giảm bớt độ phức tạp trong việc cấu hình ứng dụng trong trường hợp sử dụng nhiều môi trường.

- Externalized Configuration cho phép có khả năng cấu hình được từ bên ngoài

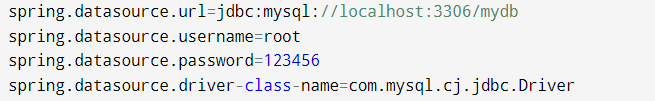
- Tất cả tính năng log nội bộ của Spring Boot đều sử dụng common log và được quản lý mặc định.

1. **Cấu trúc dự án Spring Boot**
2. **Cấu trúc dự án Spring Boot**
3. **Theo mô hình MVC**

- **src/main/java**: thư mục chứa mã nguồn Java của dự án, các thư mục con như:

* **Tên package của dự án**: thư mục gốc cho tất cả các class và packages khác của dự án, bao gồm controller, service,.... Đặt tên theo tiêu chuẩn domain ngược, ví dụ: com.mypackage, org.demo,...
* **controller**: chứa các lớp Controller, có nhiệm vụ xử lý các yêu cầu HTTP, ánh xạ chúng đến phương thức xử lý tương ứng, và điều hướng dữ liệu đến View hoặc trả về phản hồi. Ví dụ: 
* **service**: chứa các lớp Service, nơi xử lý các logic nghiệp vụ của ứng dụng. Các lớp trong tầng này thường gọi đến các lớp trong repository và cung cấp dữ liệu cho controller. Ví dụ: 
* **repository**: chứa các lớp Repository dùng để truy xuất dữ liệu từ CSDL.
* **model** hoặc **entity**: chứa các lớp Model hoặc Entity, biểu diễn các đối tượng dữ liệu (bản sao của các bảng trong CSDL). Ví dụ: 

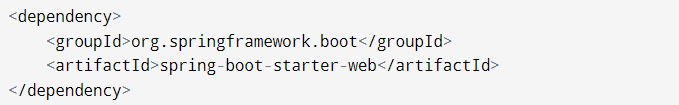
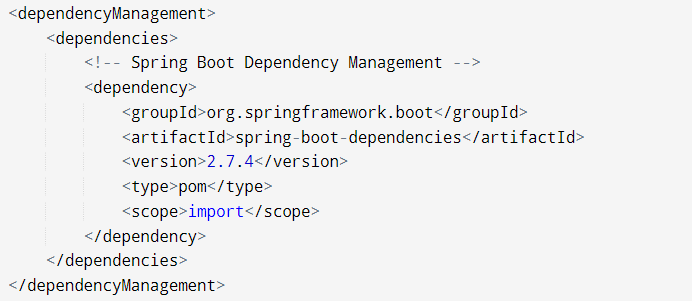
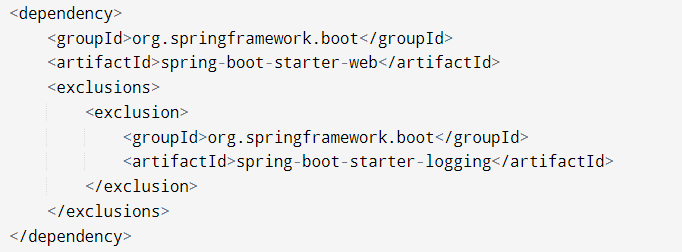
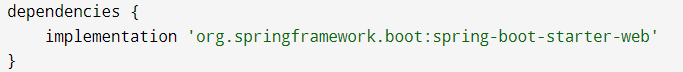
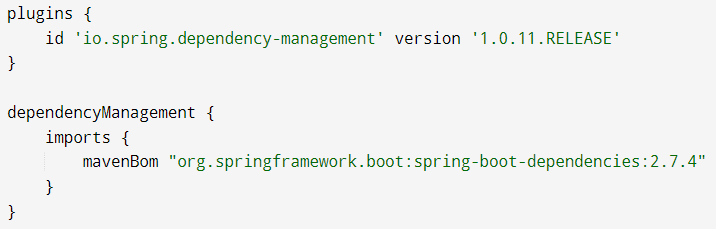
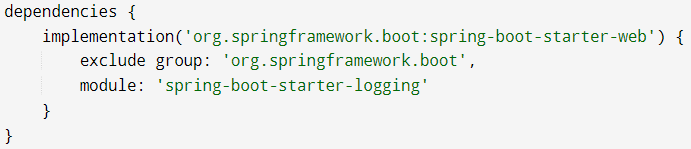
- **src/main/resources**: chứa các file cấu hình và các tài nguyên tĩnh khác:

* **application.properties** hoặc **application.yml**: File cấu hình chính của Spring Boot, nơi khai báo các thông số cấu hình như kết nối cơ sở dữ liệu, cổng server, logging... Ví dụ (application.properties**)**:
* Cấu hình server: server.port=8080
* Cấu hình kết nối CSDL: 
* Cấu hình JPA/Hibernate: 
* Cấu hình logging: 
* **static**: chứa các tài nguyên tĩnh như file CSS, JavaScript, hình ảnh,...
* **messages**: chứa các file cấu hình **messages.properties** cho việc xử lý đa ngôn ngữ.
* **data.sql** hoặc **schema.sql**: file SQL để khởi tạo hoặc nhập dữ liệu mẫu vào cơ sở dữ liệu khi ứng dụng khởi động.

- **src/test/java**: chứa các lớp phục vụ cho unit test, thường có cấu trúc tương tự src/main/java:

* **controller**: các lớp kiểm thử controller.
* **service**: các lớp kiểm thử logic nghiệp vụ.
* **repository**: các lớp kiểm thử truy xuất dữ liệu từ cơ sở dữ liệu.

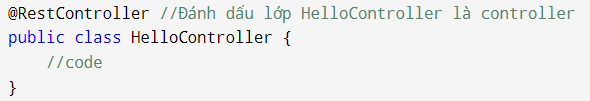
- **pom.xml** (với Maven) hoặc **build.gradle** (với Gradle): file cấu hình dependencies và build dự án.

* **pom.xml**:
* **Spring Boot Starter POMs**: tập hợp phụ thuộc được cấu hình sẵn cho các chức năng phổ biến. Ví dụ với ứng dụng web: 
* **Dependency Management Section**: Spring Boot sử dụng một phần dependencyManagement trong POM để quản lý phiên bản của các phụ thuộc, tránh việc phải xác định phiên bản cho từng phụ thuộc 
* **Exclusions**: loại bỏ các phụ thuộc không cần thiết từ các starter POMs bằng cách sử dụng phần tử <exclusions> 
* **build.gradle:**
* **Spring Boot Starter Dependencies**: Sử dụng các starter dependencies trong phần dependencies. Ví dụ: 
* **Dependency Management Plugin**: Gradle sử dụng plugin io.spring.dependency-management để hỗ trợ quản lý phụ thuộc tương tự như Maven 
* **Exclusions**: loại bỏ các phụ thuộc không cần thiết, ví dụ: 

1. **Theo tính năng**

- Mỗi tính năng tương ứng với package phù hợp. Ví dụ, **user package** chứa UserController, UserService,...

1. **Controller đơn giản xử lý request, response**
2. **Một số annotation**
3. **@RestController**

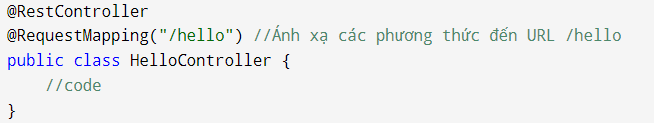
**-** Đánh dấu class là controller để xử lý các yêu cầu REST, ví dụ: 

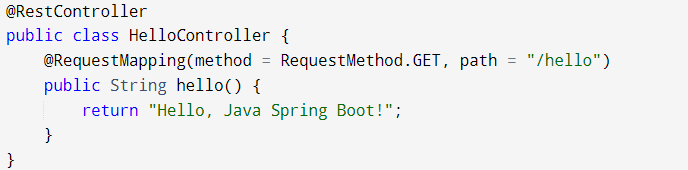
- Kết hợp **@Controller** và **@ResponseBody**: tự động chuyển đổi đối tượng được trả về thành JSON hoặc XML.

1. **@RequestMapping**

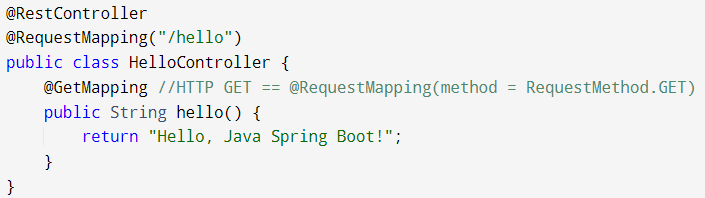
- Ánh xạ các yêu cầu HTTP tới các phương thức xử lý của controller

- Có thể áp dụng trên class hoặc phương thức

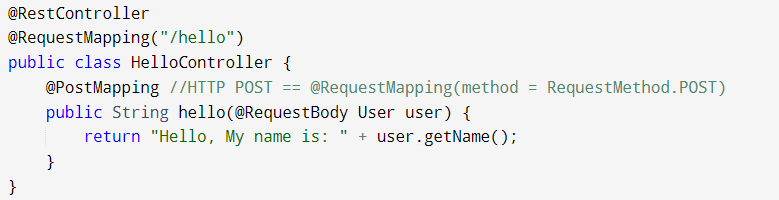
- Trên class: 

- Trên phương thức: 

1. **@GetMapping**

- Dành cho các HTTP GET, ví dụ: 

1. **@PostMapping**

- Dành cho HTTP POST, ví dụ: 

1. **@RequestBody**

- Chuyển đổi JSON từ body của request thành kiểu đối tượng (data binding)

1. **Cấu hình dự án với application.properties**
2. **application.properties là gì?**

- File application.properties được dùng để cấu hình ngữ cảnh ứng dụng (application context). Ngữ cảnh được tự động tạo ra khi ứng dụng Spring Boot được khởi động.

- Là nơi chứa cấu hình mặc định của ứng dụng.

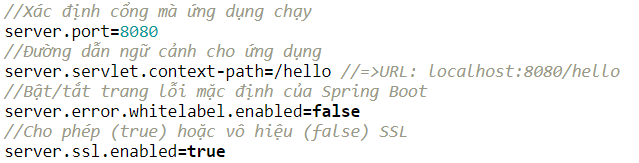
- Các cấu hình trong application.properties có độ ưu tiên thấp hơn khi ngữ cảnh ứng dụng được ghi đè ở nhiều nơi (tham số dòng lệnh, biến môi trường,...). Ví dụ:

* Tham số dòng lệnh: --server.port=8081
* Trong application.properties: server.port=8080

=> Ứng dụng chạy trên cổng 8081

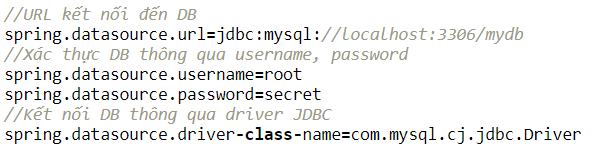
1. **Cấu hình ứng dụng**
2. **Cấu hình Server (Server configuration)**

- Cấu hình liên quan đến máy chủ nhúng mà Spring Boot sử dụng.

- Ví dụ: 

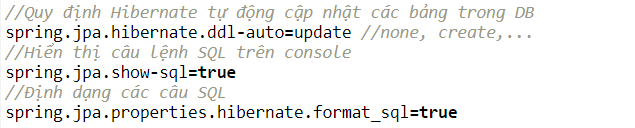
1. **Cấu hình cơ sở dữ liệu (Data source configuration)**

- Thiết lập thông tin kết nối cơ sở dữ liệu (CSDL) khi sử dụng JPA hoặc JDBC.

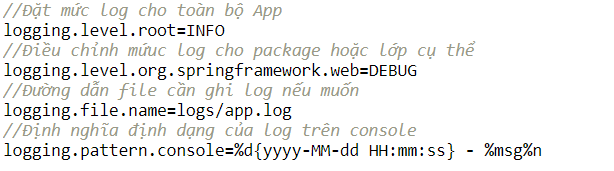
- Ví dụ: 

1. **Cấp hình JPA/Hibernate**

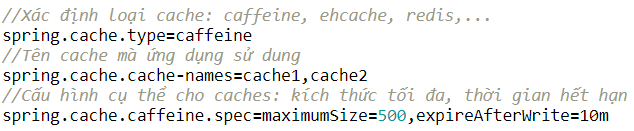
- Kiểm soát cách xử lý cơ sở dữ liệu và các đối tượng.

- Ví dụ: 

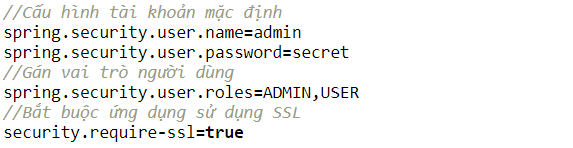
1. **Cấu hình logging (Logging configuration)**

- Điều chỉnh cách ghi log, ví dụ: 

1. **Cấu hình cache**

- Điều chỉnh cách ứng dụng xử lý bộ nhớ đệm, ví dụ: 

1. **Cấu hình bảo mật (Security configuration)**

- Thiết lập bảo mật ứng dụng, ví dụ: 

1. **Một số cấu hình khác**

- Cấu hình Mail: thiết lập việc gửi email thông qua server SMTP

- Cấu hình Thymeleaf (nếu sử dụng làm công cụ template engine)

- Cấu hình Web: điều chỉnh các thành phần như MVC, tài nguyên tĩnh, session

- Cấu hình File Upload

- Cấu hình CORS (Cross-Origin Resource Sharing): cấu hình cho phép chia sẻ tài nguyên giữa các nguồn khác nhau

1. **Cơ bản về MySQL**
2. **Select**

- Truy xuất dữ liệu từ cơ sở dữ liệu, kết quả trả về được lưu trong bảng kết quả, được gọi là tập kết quả.

- Cú pháp:

* Lấy dữ liệu theo các cột cụ thể: 
* Hoặc lấy tất cả các cột: 

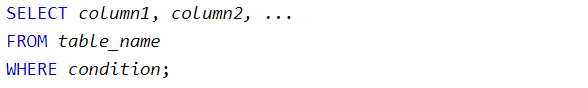
- Ví dụ: 

- Từ khóa **DISTINCT** để trả về các bản ghi là duy nhất, ví dụ: 

- Các từ khóa cho phép thực hiện tính toán trên dữ liệu trả về như: **MAX, MIN, COUNT, ...**

1. **Where**

-Từ khóa **WHERE** được sử dụng để lọc các bản ghi dữ liệu thõa mãn theo điều kiện xác định.

- Cú pháp: 

- Các toán tử: **=**, **<**, **<**, **>=**, **<=**, <>, BETWEEN, LIKE, IN,...

- Các toán tử logic: AND, OR, NOT

1. **Insert**

- Câu lệnh **INSERT INTO** dùng đề chèn thêm bản ghi mới vào bảng trong CSDL.

- Cú pháp: (chỉ rõ từng cột cần thêm dữ liệu) 

hoặc: (thêm vào tất cả các cột ứng với các giá trị truyền vào, số lượng giá trị truyền vào phải bằng số cột) 

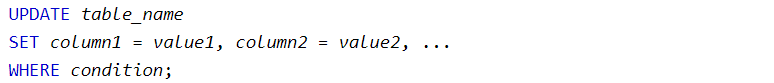
- Ví dụ: 

1. **Update**

- Cho phép sửa đổi bản ghi tồn tại trong bảng trong CSDL.

- Nên kết hợp với từ khóa **WHERE** để đảm bảo chỉ các bản ghi thõa mãn điều kiện mới được cập nhật.

- Nếu không có mệnh đề **WHERE**, tất cả các bản ghi sẽ được cập nhật.

- Cú pháp: 

1. **Delete**

- Xóa bản ghi tồn tại trong bảng trong CSDL.

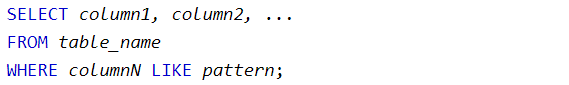
- Nên kết hợp mệnh đề **WHERE**, nếu không tất cả các bản ghi sẽ bị xóa

- Cú pháp: 

1. **Like**

- Kết hợp với WHERE để tìm kiếm một mẫu cụ thể.

- Các ký tự đại diện thường dùng như: **dấu phần trăm (%)** - đại diện cho bất kỳ chuỗi ký tự nào - và **dấu gạch dưới ( \_ )** - đại diện cho một ký tự duy nhất.

- Cú pháp: 

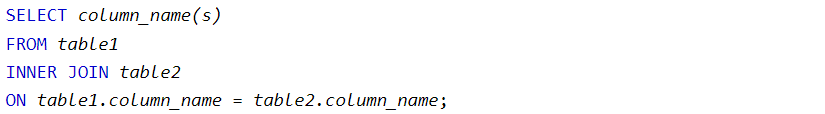
- Ví dụ, chọn tất cả khách hàng có tên bắt đầu bằng ‘a’: 

1. **Join**

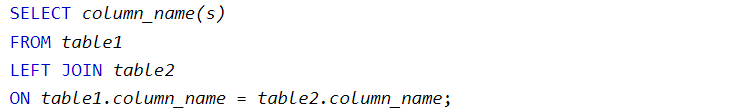
- Mệnh đề **JOIN** được sử dụng để kết hợp các hàng từ hai hoặc nhiều bảng, dựa trên một cột có liên quan giữa chúng.

- Có các loại như: **INNER JOIN**, **LEFT JOIN**, **RIGHT JOIN**, **CROSS JOIN**

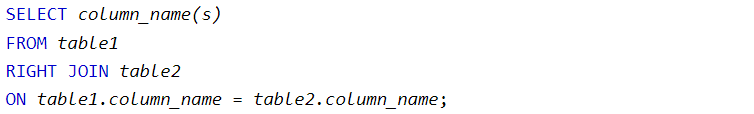
1. **Inner join**

- Chọn các bản ghi có giá trị trùng khớp trong cả hai bảng, cú pháp: 

1. **Left join**

- Trả về tất cả các bản ghi từ bảng bên trái (table1) và các bản ghi khớp (nếu có) từ bảng bên phải (table2), cú pháp: 

1. **Right join**

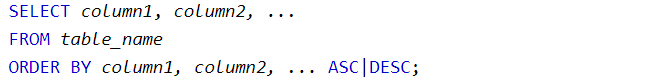
- Trả về tất cả các bản ghi từ bảng bên phải (table2) và các bản ghi khớp (nếu có) từ bảng bên trái (table1), cú pháp: 

1. **Cross join**

- Trả về tất cả bản ghi từ hai bảng (table1 và table2), cú pháp: 

1. **Order by**

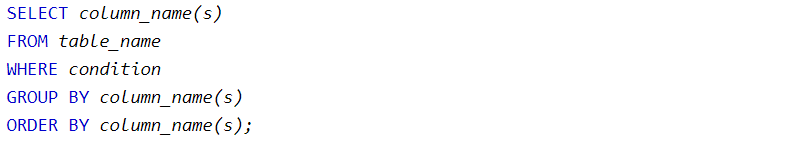
- Sử dụng để sắp xếp tập kết quả theo thứ tự tăng (**ASC**) hoặc giảm dần (**DESC**).

- Cú pháp: 

- Ví dụ, sắp xếp các bản ghi theo cột “Quốc gia”: 

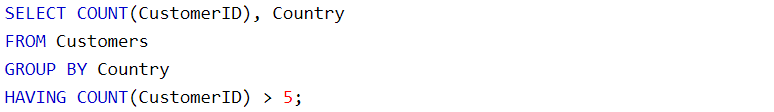
1. **Group by**

- Nhóm các hàng có cùng giá trị vào các hàng tóm tắt, thường được dùng với các hàm tổng hợp (**COUNT(), MAX(), MIN(), SUM(), AVG()**) để nhóm tập kết quả theo cột.

- Cú pháp: 

1. **Has**

- **HAS** được sử dụng với các hàm tổng hợp, cú pháp: 

- Ví dụ, liệt kê số lượng KH ở mỗi quốc gia thõa mãn điều kiện quốc gia đó có nhiều hơn 5 khách hàng: 

1. **Design pattern cơ bản trong Java**