



TẬP ĐOÀN CÔNG NGHIỆP – VIỄN THÔNG QUÂN ĐỘI

BÁO CÁO MINI-PROJECT

Tìm Hiểu Spring Framework

Nguyễn Tuấn Thành

tuanthanhnguyen1007@gmail.com

Chương trình Viettel Digital Talent 2023

Lĩnh vực: Software & Data Engineering

Mentor: Hoàng Khắc Hiếu

Đơn vị: Viettel Solution

HÀ NỘI, 06/2023

Lời mở đầu

Spring Framework là một trong những framework phổ biến cho việc phát triển các ứng dụng Java. Nó được phát triển bởi Rod Johnson vào năm 2002 và trở thành một trong những framework phổ biến nhất cho phát triển các ứng dụng Java. Spring framework có một cộng đồng phát triển lớn, với nhiều tài liệu, hướng dẫn và ví dụ đầy đủ trên Internet. Nó cũng được sử dụng rộng rãi trong các ứng dụng doanh nghiệp cũng như các dự án phần mềm khác.

Trong quá trình thực hiện tìm hiểu về Spring framework em có gặp một số khó khăn do kiến thức còn hạn chế. Em rất cảm ơn anh Hoàng Khắc Hiếu đã giúp đỡ, giải đáp các thắc mắc của mình để em có thể hoàn thiện được mini project.

Tóm tắt nội dung và đóng góp

- Báo cáo tìm hiểu về Spring Framework, lịch sử ra đời, tổng quan các mô đun và tìm hiểu chi tiết về Spring Boot, Spring Security, Spring Jpa. Cách xây dựng API với Spring.
- Phương pháp thực hiện: Thực hiện bằng phương pháp phân tích – tổng hợp các kiến thức, tài liệu.
- Một số kết quả đạt được:
 - + Trình bày tổng quan về Spring Framework, chi tiết các Spring Boot, Spring JPA, Spring Security.
 - + Xây dựng được API với Spring.
 - + Xây dựng được hệ thống Backend sử dụng Spring Boot, xác thực Spring Security, truy vấn CSDL Mysql.
- Đóng góp cá nhân:
 - + Tìm hiểu về các thành phần của Spring Framework.
 - + Tìm hiểu và lấy ví dụ phân spring jpa, vẽ sơ đồ và phân tích luồng xác thực của spring security.
 - + Thực hiện xây dựng API để hiểu rõ hơn các thành phần cần thiết để xây dựng một api cơ bản với spring boot.
- Các kỹ năng, kiến thức đạt được:
 - + Nắm được các kiến thức về Spring, Spring Boot, Spring JPA, Spring Security.
 - + Áp dụng các kiến thức tìm hiểu về Spring Framework để thực hiện xây dựng api cơ bản, hệ thống backend xác thực.

Mục Lục

1.	Lịch sử ra đời của Spring Framework.....	6
2.	Một số khái niệm và khái quát từng chức năng của từng module trong spring framework .	7
2.1.	Một số khái niệm	7
2.2.	Các module trong spring.....	8
2.2.1.	Core Container	8
2.2.2.	Data Access/Integration	9
2.2.3.	Web	10
2.2.4.	AOP and Instrumentation.....	11
2.2.5.	Test.....	11
2.2.6.	Các Module bổ sung khác	11
3.	Tìm hiểu về Spring boot, Spring JPA, Spring Security.....	12
3.1.	Spring boot.....	12
3.1.1.	Kiến trúc của Spring Boot.....	12
3.1.2.	Các tính năng của spring boot.....	13
3.1.2.1.	Tự động cấu hình	13
3.1.2.2.	Nhúng máy chủ	14
3.1.2.3.	Cấu hình dựa trên các file properties.....	14
3.1.2.4.	Hỗ trợ tích hợp các module khác	15
3.1.2.5.	Hỗ trợ các công cụ đi kèm	15
3.2.	Spring Data JPA.....	16
3.2.1.	Giới thiệu JPA	16
3.2.1.	Đặc điểm của Spring Data JPA	17
3.2.1.1.	Không lặp lại việc tạo quá nhiều Repository	17
3.2.1.2.	Tùy chỉnh các phương thức và câu truy vấn	17
3.2.1.3.	Cấu hình Transaction.....	18
3.3.	Spring Security	19
3.3.1.	Giới thiệu chung về Spring Security	19
3.3.2.	Quy trình xác thực của Spring Security	20
3.3.2.1.	Security Filter	20
3.3.2.2.	JWT Authentication Filter	21
3.3.2.3.	UsernamePasswordAuthenticationToken	22
3.3.2.4.	AuthenticationManager	22
3.3.2.5.	AuthenticationProvider	22
3.3.2.6.	SecurityContextHolder.....	23
4.	Cách xây dựng API với Spring Framework.....	23

Danh mục hình ảnh

Hình 1. Logo của spring framework	6
Hình 2. Các module của spring framework	8
Hình 3. Mô tả ORM Mapping.....	10
Hình 4. Các layer của spring boot.....	12
Hình 5. Kiến trúc spring boot	13
Hình 6. Thêm dependency spring web vào pom.xml.....	14
Hình 7. Thêm cấu hình vào file application.properties	14
Hình 8. Tích hợp các module khác vào spring boot project	15
Hình 9. Spring initializr	16
Hình 10. Tạo mới interface UserRepository	18
Hình 11. Custom query với @Query	18
Hình 12. Ví dụ về @Transaction.....	19
Hình 13. Tích hợp Spring Security vào ứng dụng Spring Boot.....	20
Hình 14. Sơ đồ quy trình xác thực spring security.....	20

1. Lịch sử ra đời của Spring Framework

- Spring Framework là một trong những framework phổ biến nhất cho phát triển ứng dụng Java. Nó được phát triển bởi Rod Johnson vào năm 2003 như một giải pháp thay thế cho các công nghệ phát triển truyền thống như EJB (Enterprise JavaBeans).
- Spring Framework được xây dựng trên nền tảng của các công nghệ phát triển Java hiện đại như Inversion of Control (IoC) và Dependency Injection (DI). Nó cũng cung cấp một số tính năng khác như AOP (Aspect-Oriented Programming), JDBC (Java Database Connectivity), và hỗ trợ MVC (Model-View-Controller) cho phát triển web.
- Ngày nay, Spring Framework đã trở thành một trong những framework phổ biến nhất cho phát triển ứng dụng Java, được sử dụng rộng rãi trong các dự án phát triển lớn và nhỏ trên khắp thế giới. Spring Framework cũng đã phát triển và mở rộng để hỗ trợ các nền tảng và ngôn ngữ khác nhau như Kotlin, Groovy và Scala. Ngoài ra, Spring Framework cũng cung cấp các dự án phụ như Spring Boot, Spring Data và Spring Security để hỗ trợ phát triển ứng dụng web và thông tin liên lạc với cơ sở dữ liệu.
- Trong quá trình phát triển, Spring Framework đã trải qua nhiều phiên bản khác nhau, từ phiên bản đầu tiên Spring 1.0 ra mắt vào năm 2004 đến phiên bản mới nhất Spring Framework 6.0 được phát hành vào năm 2022.
- Từ khi ra đời, Spring Framework đã có một vai trò quan trọng trong việc thúc đẩy sự phát triển và tiêu chuẩn hóa phát triển ứng dụng Java, cũng như cung cấp cho các nhà phát triển một cách tiếp cận dễ dàng và linh hoạt để xây dựng các ứng dụng Java hiệu quả và bảo trì dễ dàng.



Hình 1. Logo của spring framework

2. Một số khái niệm và khái quát từng chức năng của từng module trong spring framework

2.1. Một số khái niệm

- **Dependency Injection(DI):** là một design pattern cho phép giảm thiểu các sự liên kết giữa các thành phần trong một ứng dụng. Bằng cách sử dụng DI, spring framework cho phép giảm bớt sự phụ thuộc giữa các thành phần với nhau, giúp cho ứng dụng trở nên module hóa và dễ dàng cho việc bảo trì. Hay có thể hiểu DI là việc các Object nên phụ thuộc vào các abstract class và thể hiện chi tiết(instance) của nó sẽ được inject vào đối tượng lúc runtime.

Có các cách để inject dependency vào một đối tượng:

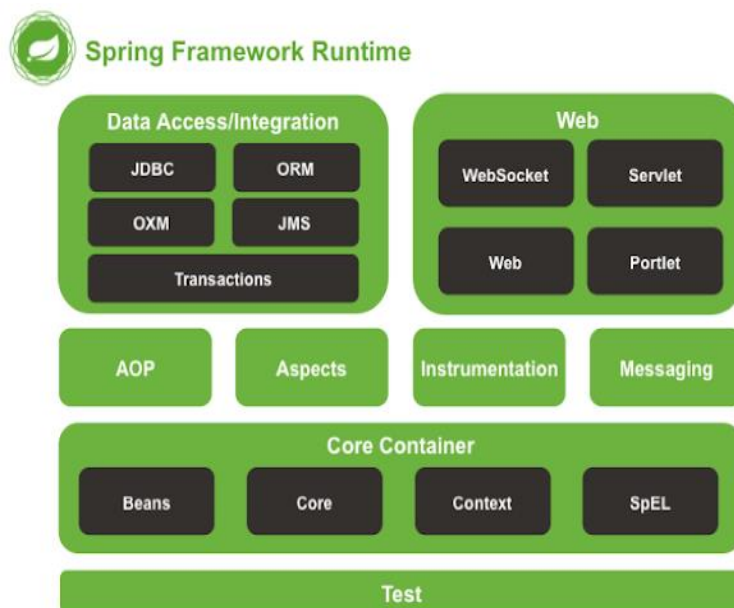
- + **Constructor injection:** Thêm dependency vào hàm khởi tạo (constructor). Cách này được recommend bởi các nhà phát triển spring framework.
- + **Setter injection:** Các dependency sẽ được truyền vào thông qua hàm Setter.
- + **Interface Injection:** Class cần inject sẽ implement 1 interface. Interface này chứa 1 hàm tên Inject. Container sẽ injection dependency vào 1 class thông qua việc gọi hàm Inject của interface đó. Đây là cách rườm rà và cũng ít được sử dụng.
- **Inversion of Control(IoC):** là một nguyên tắc chính của spring framework, IoC chịu trách nhiệm quản lý vòng đời của các bean và đưa vào các phụ thuộc của chúng.
- **Bean:** là một đối tượng được quản lý bởi Spring Framework. Nó là một đối tượng Java thông thường, nhưng được quản lý bởi ApplicationContext của spring và được cấu hình thông qua các metadata như XML, Java annotation. Scope của bean xác định vòng đời của bean nằm trong container và với đối tượng khác trong ứng dụng. Các Scope của bean bao gồm:
 - **Singleton:** đây là scope mặc định của bean. Một singleton bean chỉ được khởi tạo một lần bởi container và được chia sẻ bởi tất cả các đối tượng yêu cầu nó.
 - **Prototype:** scope này có nghĩa là container sẽ tạo ra một instance mới cho bean mỗi khi có yêu cầu đến nó.
 - **Request:** scope này chỉ áp dụng cho các ứng dụng web và container sẽ tạo một instance mới cho bean với mỗi yêu cầu HTTP.

- **Session:** tương tự request, nhưng bean sẽ được tạo cho mỗi yêu cầu HTTP.
 - **Application:** scope của bean dành cho dữ liệu cấp ứng dụng toàn cục và chỉ hợp lệ trong một Spring ApplicationContext hỗ trợ web.
 - **Websocket:** scope này dành cho dữ liệu Websocket.
- **Container:** Spring container là một phần chính của spring framework, nó tạo và quản lý các bean. Quản lý các dependencies của chúng.
 - **ApplicationContext:** ApplicationContext là một triển khai của Spring container. Nó chịu trách nhiệm cho việc tải và quản lý các cấu hình metadata và tạo các bean được xác định trong metadata đó.
 - **Aspect-Oriented Programming(AOP):** Spring hỗ trợ lập trình hướng khía cạnh(AOP), cho phép ta phân tách các mối quan tâm chung(cross-cutting concern) như ghi nhật ký(logging) hoặc bảo mật(security) ra khỏi logic của ứng dụng.

2.2. Các module trong spring

Các module trong spring framework có thể chia thành các phần chính: Core Container, Data Access/Integration, Web, AOP and Instrumentation và Test.

2.2.1. Core Container



Hình 2. Các module của spring framework

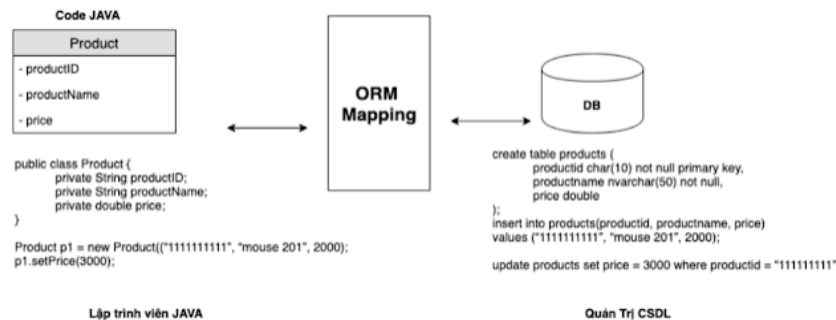
Core Container cung cấp các chức năng cơ bản của spring framework, bao gồm Inversion of Control(IoC) và Application Context. Nó bao gồm các module sau:

- **Spring core:** Module này cũng cấp các chức năng cơ bản của Spring framework, bao gồm Ioc và DI. Ioc Container được coi như trái tim của Spring framework, nó chịu trách nhiệm khởi tạo và quản lý các instance của JavaBean. Nó sử dụng dependency injection nó liên kết các bean lại với nhau.
- **Spring beans:** Module này cung cấp một BeanFactory, chịu trách nhiệm quản lý vòng đời của một bean. BeanFactory là một lớp chính để truy cập vào IoC container. Nó cung cấp phương thức để lấy bean.
- **Spring context:** Module này cung cấp ApplicationContext, nó là một phiên bản nâng cấp của BeanFactory và nó cũng cấp thêm các tính năng ví dụ như hỗ trợ tải các nguồn tài nguyên như file cấu hình, hỗ trợ các tính năng AOP,...
- **Spring Expression Language (SpEL):** Module này cung cấp một ngôn ngữ biểu thức mạnh mẽ để truy vấn và thao tác với các đối tượng trong thời gian chạy. SpEL hỗ trợ nhiều tính năng, bao gồm truy cập các thuộc tính, gọi các phương thức, điều kiện, vòng lặp và chuyển đổi kiểu.

2.2.2. Data Access/Integration

Data Access/Integration hỗ trợ cho việc tích hợp với cơ sở dữ liệu và các nguồn dữ liệu khác. Nó bao gồm các module sau:

- **Spring JDBC:** Module này cũng cấp một lớp JDBC trừu tượng cơ bản để giảm thiểu số lượng boilerplate code yêu cầu khi làm việc với JDBC. Spring JDBC hỗ trợ cho việc quản lý các transaction, cho phép lập trình viên có thể quản lý các transaction với database sử dụng trình quản lý transaction của Spring.
- **Spring ORM:** Module này cung cấp khả năng tích hợp với các framework ánh xạ đối tượng quan hệ, có thể ví dụ như Hibernate và JPA. Spring ORM cung cấp một lớp trừu tượng ở mức cao nằm trên các framework ORM, cho phép lập trình viên viết ít hơn các boilerplate code và dễ dàng tích hợp các công nghệ ORM với các tính năng khác của Spring, chẳng hạn như transaction hay caching.



Hình 3. Mô tả ORM Mapping

- **Spring data:** Module này cung cấp một mô hình lập trình một cách nhất quán và dễ sử dụng để làm việc với các công nghệ truy cập dữ liệu, bao gồm database, NoSQL và dữ liệu dựa trên dịch vụ đám mây. Spring data cung cấp nhiều tính năng, bao gồm các hành động tự động CRUD, câu truy vấn từ tên các phương thức, hỗ trợ phân trang và sắp xếp, tích hợp quản lý các transaction của Spring. Thêm vào đó, Spring data còn cung cấp các pattern truy xuất dữ liệu phổ biến như.

2.2.3. Web

Module này hỗ trợ cho việc xây dựng các ứng dụng web. Nó bao gồm:

- **Spring MVC:** Module này cung cấp Model-View-Controller(MVC) để xây dựng ứng dụng web. Spring MVC cung cấp một loạt các tính năng, bao gồm cả việc hỗ trợ xử lý các HTTP request và response, xử lý form, ràng buộc dữ liệu, kiểm tra tính hợp lệ của dữ liệu và hơn thế nữa. Nó cũng hỗ trợ các công nghệ cho việc hiển thị, ví dụ như JSP, Thymeleaf,... cho phép lập trình viên có thể chọn các công nghệ hiển thị này phù hợp với nhu cầu của mình.
- **Spring WebFlux:** Là module để xây dựng các ứng dụng web reactive với hiệu suất cao và khả năng mở rộng tốt. Các ứng dụng WebFlux được xây dựng bằng cách sử dụng các API reactive như Flux và Mono, cung cấp một cách tiếp cận dựa trên sự kiện cho việc xử lý các yêu cầu web và giải phóng tài nguyên của máy chủ. Spring WebFlux cũng hỗ trợ các tính năng như định tuyến, xử lý lỗi, xử lý các yêu cầu HTTP và WebSocket, và hỗ trợ các thư viện như Jackson hoặc Gson để truyền thông dữ liệu giữa máy khách và máy chủ.
- **Spring Web Service:** Mô-đun này cung cấp hỗ trợ để xây dựng các dịch vụ web dựa trên SOAP và RESTful. Spring web service cung cấp hỗ trợ để

tạo WSDL (Ngôn ngữ mô tả dịch vụ web) từ các lớp Java và để tạo các lớp Java từ WSDL.

2.2.4. AOP and Instrumentation

- Module AOP (Aspect-Oriented Programming) trong Spring Framework là một công nghệ lập trình giúp tách rời các khía cạnh của một ứng dụng, như thể hiện, logic và quản lý lỗi, để giảm thiểu sự phức tạp của mã và tăng tính tái sử dụng của các thành phần.
- AOP cho phép các nhà phát triển chia nhỏ các chức năng của ứng dụng thành các mô-đun nhỏ hơn, gọi là các "Aspect" (khía cạnh), mà có thể được áp dụng cho nhiều đối tượng khác nhau trong ứng dụng. Những khía cạnh này có thể được sử dụng để thêm hoặc loại bỏ các chức năng của một ứng dụng mà không cần thay đổi mã nguồn của ứng dụng.
- Module AOP cũng hỗ trợ các chức năng như logging, caching, transaction management và security, giúp các lập trình viên triển khai các tính năng này một cách dễ dàng và hiệu quả. Bằng cách sử dụng AOP, các nhà phát triển có thể tập trung vào các chức năng chính của ứng dụng và giảm thiểu sự trùng lặp của code, tăng tính tái sử dụng và hiệu quả của ứng dụng.

2.2.5. Test

- Module này cung cấp hỗ trợ về Testing với JUnit và TestNg.

2.2.6. Các Module bổ sung khác

- **Spring security:** Module này cung cấp các tính năng xác thực và ủy quyền cho các ứng dụng Spring. Spring Security cung cấp một loạt các cơ chế ủy quyền, chẳng hạn như kiểm soát truy cập dựa trên vai trò. Nó cũng cung cấp hỗ trợ để bảo mật các phần khác nhau của ứng dụng bằng cách sử dụng các cấu hình bảo mật khác nhau, cho phép các lập trình viên áp dụng các chính sách bảo mật chi tiết.
- **Spring integration:** Module này hỗ trợ việc xây dựng ứng dụng hướng sự kiện và hướng thông điệp. Spring integration hỗ trợ các hệ thống messaging systems như JMS, AMQP và Apache Kafka.
- **Spring batch:** Module này cung cấp hỗ trợ xử lý hàng loạt và tích hợp với các hệ thống doanh nghiệp. Spring Batch cung cấp một loạt các công cụ và tiện ích để xây dựng và quản lý các ứng dụng xử lý hàng loạt, chẳng hạn như hỗ trợ kiểm tra và gỡ lỗi các công việc hàng loạt, ghi nhật ký và giám

sát cũng như tích hợp với các module Spring khác, chẳng hạn như Spring Data và Spring Integration.

- **Spring cloud:** Module này cung cấp hỗ trợ để xây dựng các ứng dụng trên đám mây bằng các công nghệ Spring. Spring Cloud cung cấp một loạt các tính năng để xây dựng các ứng dụng trên đám mây, chẳng hạn như service discovery, quản lý cấu hình và cân bằng tải. Nó cung cấp hỗ trợ để tích hợp với các nền tảng đám mây khác nhau, chẳng hạn như AWS và GCP, cũng như để sử dụng các công nghệ trên đám mây khác nhau, chẳng hạn như container và điện toán không có máy chủ.

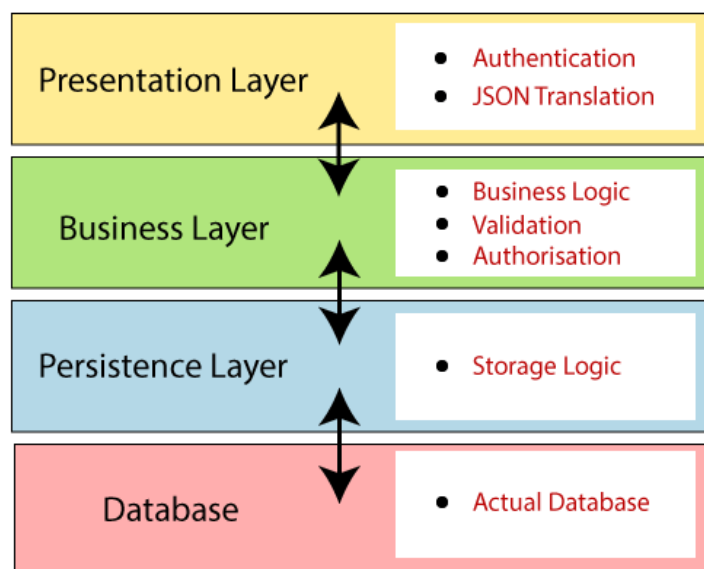
3. Tìm hiểu về Spring boot, Spring JPA, Spring Security

3.1. Spring boot

Spring Boot là một framework của Spring giúp đơn giản hóa việc phát triển ứng dụng Java. Nó giúp đơn giản hóa việc cấu hình và triển khai ứng dụng Spring bằng cách cung cấp các cấu hình mặc định và tự động cấu hình các thành phần cần thiết cho ứng dụng của bạn.

3.1.1. Kiến trúc của Spring Boot

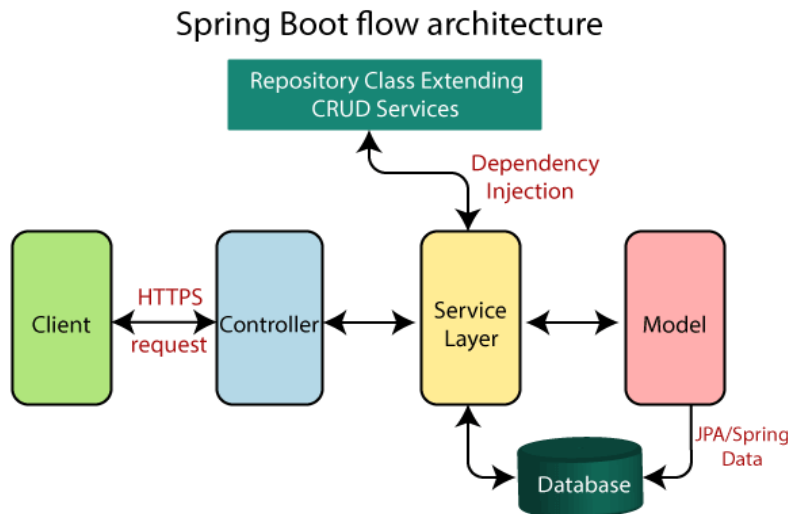
- Spring boot có 4 lớp chính là: Presentation Layer, Business Layer, Persistence Layer, Database Layer.



Hình 4. Các layer của spring boot

- **Presentation Layer:** Lớp này xử lý các yêu cầu HTTP, chuyển JSON thành object, và xác thực các request và chuyển nó đến business layer.

- **Business Layer:** Lớp này xử lý tất cả các business logic. Nó bao gồm các service class. Nó chịu trách nhiệm validation và authorization.
 - **Persistence Layer:** Chịu trách nhiệm chuyển các object thành các hàng trong cơ sở dữ liệu và ngược lại.
 - **Database Layer:** Thực hiện các hành động CRUD.
- Kiến trúc của Spring boot



Hình 5. Kiến trúc spring boot

- Client gửi đến một HTTP request(GET, PUT, POST,...)
- HTTP request được chuyển đến controller. Controller sẽ xử lý và gọi đến lớp logic.
- Logic nghiệp vụ được nằm trong lớp Service. Spring Boot sẽ thực hiện tất cả các logic trên dữ liệu của cơ sở dữ liệu và ánh xạ tới lớp Model thông qua JPA.
- Sau đó controller trả về response cho client.

3.1.2. Các tính năng của spring boot

3.1.2.1. Tự động cấu hình

- Spring Boot có thể tự động cấu hình cho ứng dụng của bạn dựa trên các thư viện và dependencies được sử dụng trong ứng dụng. Đây là tính năng quan trọng, khi một dependency được thêm vào project thì Spring Boot sẽ tự động tìm kiếm các bean và cấu hình chúng để sử dụng trong ứng dụng mà không cần phải cấu hình thủ công.

- Tính năng Auto-configuration giúp giảm thời gian và công sức trong việc cấu hình và triển khai ứng dụng Spring Boot, đồng thời giúp đảm bảo tính nhất quán và độ tin cậy của ứng dụng.
- Để sử dụng tính năng này, ta cần thêm các dependency cần thiết vào file pom.xml hoặc build.gradle. Sau đó, Spring Boot sẽ tự động phát hiện các dependency này và cấu hình các bean tương ứng.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Hình 6. Thêm dependency spring web vào pom.xml

- Ta có thể cho phép tính năng auto-configuration bằng cách sử dụng annotation `@EnableAutoConfiguration`. Nhưng annotation này không sử dụng vì nó nằm trong `@SpringBootApplication` (là annotation được kết hợp bởi 3 annotation khác là `@ComponentScan`, `@SpringBootConfiguration` và `@EnableAutoConfiguration`). Vì vậy, ta sử dụng annotation `@SpringBootApplication` thay cho `@EnableAutoConfiguration`.

3.1.2.2. Nhúng máy chủ

- Nhúng máy chủ (Embedded Server) là một tính năng tiện ích và mạnh mẽ của Spring Boot. Nó cho phép ta có thể chạy trực tiếp web server bên trong ứng dụng mà không cần phải deploy ra một web server riêng biệt khác.
- Spring Boot có thể nhúng các máy chủ web như Tomcat, Jetty hoặc Undertow vào ứng dụng, giúp cho việc triển khai ứng dụng trở nên dễ dàng hơn.

Nhiều Spring Boot starter mặc định chứa embedded server

- spring-boot-starter-web chứa Tomcat server bởi nó bao gồm spring-boot-starter-tomcat.
- spring-boot-starter-webflux chứa Reactor Netty vì nó bao gồm spring-boot-starter-reactor-netty.

3.1.2.3. Cấu hình dựa trên các file properties

Nếu cần tùy chỉnh cấu hình mặc định, ta có thể thêm các cấu hình vào file application.properties hoặc application.yml để thay đổi các giá trị mặc định.

```
server.port=8082
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/mini-project
```

Hình 7. Thêm cấu hình vào file application.properties

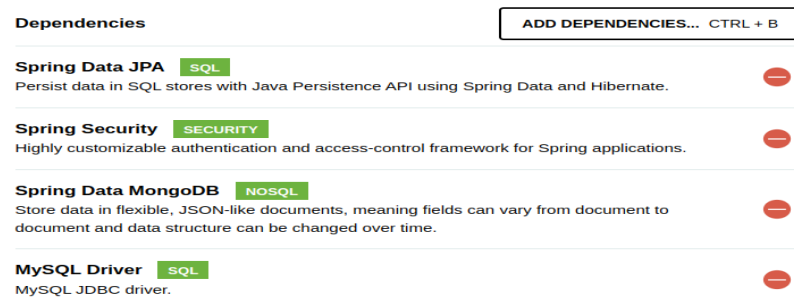
Trong Spring, spring profile là một cách để cấu hình ứng dụng cho các môi trường khác nhau(dev, product, test) và thường được cấu hình bằng các file cấu hình riêng biệt cho từng môi trường. Các file có thể được đặt tên theo cú pháp application-{tên môi trường}.properties hoặc application-{tên môi trường}.yml. Để có thể chạy môi trường nào mong muốn, ta có thể sử dụng tham số trong file cấu hình application.properties hoặc application.yml với **spring.profiles.active=dev**. Ngoài ra ta có thể thêm tham số vào câu lệnh khi chạy ứng dụng ví dụ như **java -jar myapp.jar --spring.profiles.active=dev**.

3.1.2.4. Hỗ trợ tích hợp các module khác

Spring Boot tích hợp nhiều tính năng của Spring Data như JPA, MongoDB, Redis, Cassandra, và nhiều hơn nữa.

Spring Boot tích hợp với Spring Security để cung cấp tính năng bảo mật cho ứng dụng.

Ta có thể dễ dàng tích hợp các module khác bằng cách thêm các dependency vào pom.xml.



Hình 8. Tích hợp các module khác vào spring boot project

3.1.2.5. Hỗ trợ các công cụ đi kèm

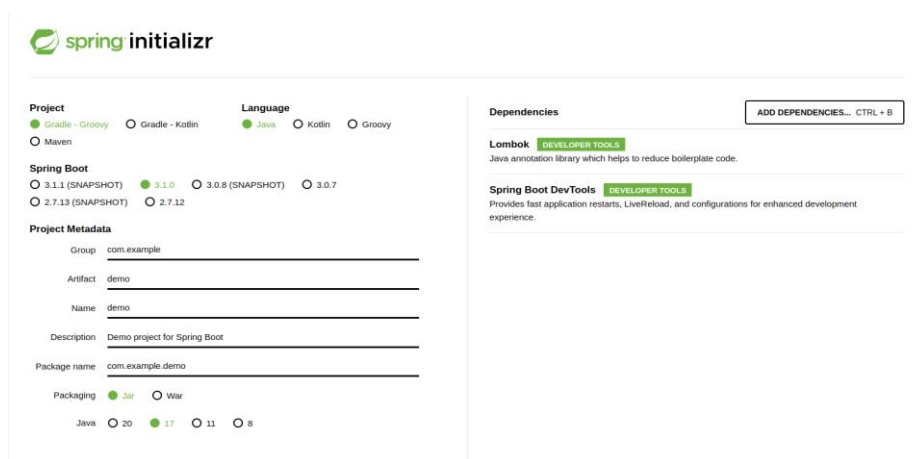
- Spring Boot cung cấp các công cụ đi kèm như Spring Boot CLI, Spring Initializr, Spring Boot Actuator, giúp cho việc phát triển và quản lý ứng dụng trở nên thuận tiện hơn.

- Spring Boot CLI(Command Line Interface) là công cụ dòng lệnh dùng để phát triển ứng dụng Spring Boot một cách nhanh chóng và thuận tiện. CLI cung cấp một số lệnh để tạo, chạy và kiểm thử ứng dụng một cách dễ dàng như spring init, spring run, spring test,...

- Spring Boot Actuator cung cấp các tính năng quản lý và giám sát ứng dụng Spring Boot. Actuator cung cấp một tập các endpoint HTTP và các công cụ quản lý để theo dõi và kiểm soát ứng dụng. Các endpoint HTTP

được cung cấp bởi Actuator cho phép xem thông tin về ứng dụng như thông tin các request được xử lý, thông tin về các thông số cấu hình ứng dụng,...

- Spring Initializr là một công cụ giúp ta tạo ra một project Spring Boot mới một cách nhanh chóng và dễ dàng. Với Spring Initializr, ta không cần phải cài đặt và cấu hình các công cụ phát triển như Maven hay Gradle, mà chỉ cần chọn các dependency và cấu hình cho project của mình thông qua giao diện đơn giản. Spring Initializr cung cấp một trang web cho phép tạo một project Spring Boot mới chỉ trong vài phút. Trên trang web này, ta có thể chọn các thông số cấu hình cho project như ngôn ngữ lập trình, version của Spring Boot, các dependency cần thiết cho project, tên artifact, package name,...

The image shows the Spring Initializr web interface. It has a header with the Spring logo and 'spring initializr'. Below the header, there are two main sections. The left section is for project configuration, including 'Project' (with radio buttons for Gradle - Groovy, Gradle - Kotlin, and Maven), 'Language' (with radio buttons for Java, Kotlin, and Groovy), 'Spring Boot' (with radio buttons for 3.1.1 (SNAPSHOT), 3.1.0, 3.0.8 (SNAPSHOT), 3.0.7, 2.7.13 (SNAPSHOT), and 2.7.12), 'Project Metadata' (with input fields for Group, Artifact, Name, Description, and Package name), and 'Packaging' (with radio buttons for Jar and War). The right section is for 'Dependencies', with a button 'ADD DEPENDENCIES... CTRL + B'. It lists two dependencies: 'Lombok' and 'Spring Boot DevTools', both marked as 'DEVELOPER TOOLS'.

Hình 9. Spring initializr

3.2. Spring Data JPA

Spring Data JPA là một phần của Spring Framework, nó tích hợp JPA với Spring Framework giúp cho việc phát triển ứng dụng Java khi tương tác với cơ sở dữ liệu trở nên dễ dàng hơn. Spring Data JPA cung cấp cho lập trình viên một cách tiếp cận hiệu quả để thao tác với cơ sở dữ liệu mà không phải viết nhiều code lặp lại. Spring Data JPA được xây dựng phía trên tầng JPA. Có nghĩa là nó sẽ sử dụng tất cả các chức năng của JPA như entity mapping, JPA Query,...

3.2.1. Giới thiệu JPA

- **ORM** viết tắt của Object Relational Mapping, là quá trình ánh xạ dữ liệu từ ngôn ngữ hướng đối tượng sang cơ sở dữ liệu quan hệ và ngược lại. ORM giúp ánh xạ các bảng, cột, kiểu dữ liệu và các quan hệ(1-1, 1-n, n-n) trong database thành các Class.

- **JPA** viết tắt của Java Persistence API, là một đặc tả các tiêu chuẩn của Java để làm việc với cơ sở dữ liệu quan hệ. Nó là một tập hợp các interface, cung cấp một API **EntityManager** để xử lý các câu query và các transaction trên các object dựa trên database. Nó sử dụng ngôn ngữ truy vấn đối tượng java là JPQL thay vì các câu truy vấn SQL truyền thống. JPA đơn giản, gọn gàng hơn và ít tốn công sức hơn là JDBC, SQL hay ánh xạ xử lý thuần.
 - + Ví dụ: ta lấy tên học sinh ở thành phố Hà Nội thì câu query sẽ như sau: **“SELECT s FROM Student s where s.city = ‘Hà Nội’”**. Trong ví dụ này thì Student ứng với tên của lớp đối tượng Java ứng với bảng “student” trong cơ sở dữ liệu.
- Kiến trúc của JPA gồm:
 - Entity: Các entity là đối tượng được lưu trữ lâu dài như một bản ghi trong database.
 - EntityManagerFactory: Có nhiệm vụ khởi tạo và quản lý các EntityManager instance.
 - EntityManager: Là một interface cung cấp các API tương tác giữa ứng dụng với database. Một số chức năng cơ bản như(Persist: Tạo mới thực thể, Merge: Cập nhật một thực thể, Remove: Xóa một thực thể).
 - EntityTransaction: Là một tập hợp các thao tác trong đó chúng phải thực hiện thành công hoặc thất bại, nếu thất bại các dữ liệu sẽ được roll-back.

3.2.1. Đặc điểm của Spring Data JPA

3.2.1.1. Không lặp lại việc tạo quá nhiều Repository

- Với mỗi Repository chúng ta phải triển khai các hàm CRUD, việc này khiến code bị lặp lại nhiều lần. Spring Data JPA cung cấp cho ta một tập hợp các repository interface (@CrudRepository, @PagingAndSortingRepository, @JpaRepository), khi dùng ta chỉ cần extends từ chúng thì mặc định nó sẽ cung cấp các hàm như CRUD, tìm kiếm,...

3.2.1.2. Tùy chỉnh các phương thức và câu truy vấn

- Khi Spring Data JPA tạo mới một Repository, nó sẽ phân tích các phương thức được định nghĩa trong interface và cố gắng tạo các truy vấn từ tên phương thức.

```

4 usages TuanThanh
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

    4 usages TuanThanh
    Optional<User> findByEmail(String email);

    no usages TuanThanh
    Boolean existsByEmail(String email);
}

```

Hình 10. Tạo mới interface UserRepository

Ở phương thức trên ta tạo hai hàm là `findByEmail` và `existsByEmail`. Với method `findByEmail` chuyển vào tham số email, hàm này có tác dụng trả về user có email trùng với tham số truyền vào, câu truy vấn (SQL) sẽ có dạng là: **Select * from user where user.email = ‘tên-email’**. Tương tự vậy thì hàm `existsByEmail` sẽ trả về giá trị true hoặc false để kiểm tra xem email này đã tồn tại trong database hay chưa.

- Khi ta phải truy vấn với câu truy vấn phức tạp mà các câu truy vấn trong Spring Data JPA không hỗ trợ sẵn, ta có thể dùng annotation `@Query` để custom lại câu truy vấn. Với việc sử dụng `@Query`, ta có thể sử dụng câu truy vấn JPQL hoặc raw SQL.

```

no usages new *
@Query("select u from User u where u.email =?1")
User customUserQuery(String email);

no usages new *
@Query(value = "SELECT * FROM users u where u.email = ?1", nativeQuery = true)
User customUserQueryNative(String email);

```

Hình 11. Custom query với @Query

Với method `customUserQuery` ta sử dụng JPQL để truy vấn. Phương thức `customUserQueryNative` ta sử dụng raw SQL, khi sử dụng thì ta thêm `nativeQuery=true` để Spring Data JPA biết phương thức này sử dụng raw SQL để truy vấn.

3.2.1.3. Cấu hình Transaction

- Transaction cung cấp các đặc điểm ACID để đảm bảo tính nhất quán và hợp lệ của dữ liệu. Trong đó:
 - + A - Atomicity: các hành động thực thi hoặc là thành công hết hoặc là không có bất cứ một hành động nào được thực thi khi có một hành động thực thi không thành công.

- + C - Consistency: tất cả các ràng buộc toàn vẹn dữ liệu(constraints, key, data types,...) phải được thực thi thành công cho mọi transaction, đảm bảo tính đúng đắn giữ liệu.
 - + I - Isolation: các transaction xảy ra xen kẽ sẽ không ảnh hưởng đến tính nhất quán của dữ liệu. Nếu cùng lúc có một hoặc nhiều hành động cùng diễn ra thì cần có cơ chế đảm bảo rằng các hành động này có thể thực hiện đồng thời mà không ảnh hưởng đến nhau.
 - + D - Durability: Đảm bảo một transaction thực thi thành công thì tất cả những thay đổi trong transaction phải được đồng bộ xuống database kể cả khi gặp sự cố.
- Với Spring và Spring Data JPA nó hỗ trợ việc quản lý transaction khiến cho việc này cực kỳ đơn giản, tất cả những gì chúng ta cần làm là thêm annotation `@Transactional` trên phương thức hay trên một class.

```
@Transactional
public void deleteAllById(Iterable<? extends ID> ids) {
    Assert.notNull(ids, message: "Ids must not be null!");
    Iterator var2 = ids.iterator();

    while(var2.hasNext()) {
        ID id = var2.next();
        this.deleteById(id);
    }
}
```

Hình 12. Ví dụ về @Transaction

3.3. Spring Security

3.3.1. Giới thiệu chung về Spring Security

Spring Security là một framework bảo mật mạnh mẽ và linh hoạt cho các ứng dụng Java. Nó cung cấp các tính năng bảo mật cho ứng dụng web và ứng dụng Spring Boot bao gồm xác thực người dùng, phân quyền truy cập, bảo vệ chống tấn công CSRF và XSS, xử lý các vấn đề bảo mật liên quan đến phiên làm việc, cookie và nhiều tính năng bảo mật khác.

Spring Security cung cấp một kiến trúc mở rộng và tùy chỉnh để triển khai các tính năng bảo mật phù hợp với nhu cầu của mỗi dự án. Nó

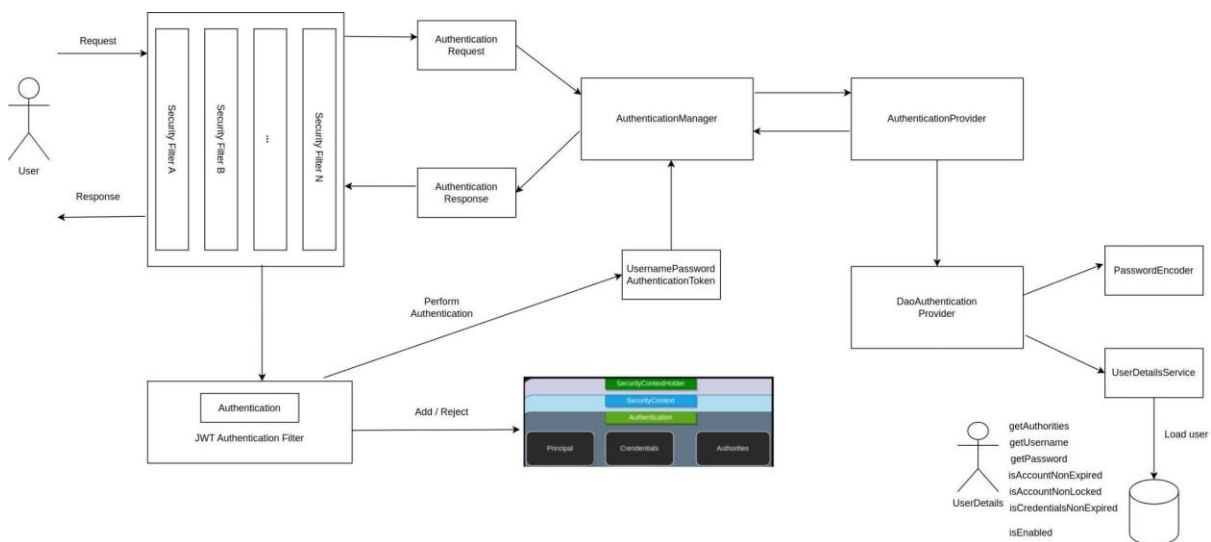
hỗ trợ cơ chế xác thực như xác thực dựa trên form, xác thực dựa trên token và xác thực dựa trên OAuth2.

Spring Security được tích hợp sẵn với khung Spring Framework, cho phép lập trình viên dễ dàng tích hợp các tính năng bảo mật vào các ứng dụng của mình. Nó được tích hợp sẵn với Spring Boot, giúp triển khai các ứng dụng Spring Boot an toàn và dễ dàng hơn.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Hình 13. Tích hợp Spring Security vào ứng dụng Spring Boot

3.3.2. Quy trình xác thực của Spring Security



Hình 14. Sơ đồ quy trình xác thực spring security

3.3.2.1. Security Filter

- Mỗi một yêu cầu tới máy chủ sẽ đi phải thông qua một chuỗi quá trình lọc và phân quyền. SecurityFilterChain là một interface trong Spring Security được sử dụng để quản các filter trong ứng dụng web. Nó được sử dụng để xác định thứ tự và cấu hình các filter mà Spring Security sẽ áp dụng cho các request đến ứng dụng.

3.3.2.2. JWT Authentication Filter

- JWT(JSON Web Token) - một tiêu chuẩn mở của RFC 7519 được sử dụng như một phương tiện đại diện nhỏ gọn, để truyền đạt thông tin giữa server và client thông qua một chuỗi JSON. Nó thường được sử dụng để xác thực người dùng và cho phép truy cập vào các tài nguyên được bảo vệ trong các ứng dụng web. Khi người dùng đăng nhập, hệ thống sẽ tạo ra một chuỗi JWT và trả về cho người dùng. Người dùng sẽ gửi chuỗi JWT này trong mỗi yêu cầu để xác thực và ủy quyền truy cập.

- JWT bao gồm ba phần chính là header, payload và signature.

- + header: gồm hai phần là loại token(JWT) và thuật toán ký đang được sử dụng(ví dụ như HMAC SHA256,...)
- + payload: là phần chứa các nội dung thông tin(claim) mà người dùng muốn truyền đi. Claims gồm 3 loại là registered, public và private claims.
- + signature: là thành phần quan trọng nhất của JWT, để tạo chữ ký ta cần mã hóa header, payload, khóa bí mật(secret key) và thuật toán sử dụng để mã hóa.

- JWT Authentication Filter là một lớp lọc được sử dụng để xác thực JWT trên các HTTP Request. Lớp này sẽ kiểm tra token bằng cách kiểm tra chữ ký và thời gian hết hạn của token, kiểm tra thông tin người dùng và quyền truy cập của họ.

- Để sử dụng JWTAuthenticationFilter ta cần thêm dependency io.jsonwebtoken vào file pom.xml. Tạo một lớp **JWTAuthenticationFilter** kế thừa từ lớp **OncePerRequestFilter**, một lớp JWT Service để kiểm tra hợp lệ của jwt token. Sau đó ở **SecurityFilterChain** ta thêm **.addFilterBefore(jwtAuthenticationFilter,**

UsernamePasswordAuthenticationFilter.class) trong đó:

- + **.addFilterBefore()**: được sử dụng để đăng ký một filter mới vào bộ lọc.
- + **UsernamePasswordAuthenticationFilter** là một filter mặc định của spring security, được sử dụng nhằm mục đích xác thực thông tin đăng nhập bằng username và password.
- + Khi ta sử dụng phương thức ở trên, khi có một yêu cầu đến, **jwtAuthenticationFilter** sẽ thực hiện trước(kiểm tra token có hợp lệ hay không) sau đó mới đến **UsernamePasswordAuthenticationFilter**. Nếu quá trình xác thực **jwtAuthenticationFilter** xác thực thành công, xác thực kết thúc tại đó và yêu cầu được cho phép truy cập vào các

tài nguyên. Nếu không thành công, thì lớp `UsernamePasswordAuthenticationFilter` sẽ được thực hiện để xác thực thông tin đăng nhập bằng tên người dùng và mật khẩu.

3.3.2.3. UsernamePasswordAuthenticationToken

- Sau khi xác thực thành công ở lớp ***JWTAuthenticationFilter***, một đối tượng `UsernamePasswordAuthenticationToken` được tạo ra để lưu trữ thông tin xác thực người dùng bao gồm tên đăng nhập, mật khẩu và quyền truy cập.
- Khi người dùng đăng nhập vào hệ thống, thông tin đăng nhập của họ được thu thập và chuyển đến `UsernamePasswordAuthenticationToken`. Sau đó `UsernamePasswordAuthenticationToken` được truyền đến `AuthenticationManager` để xác thực thông tin đăng nhập của user.

3.3.2.4. AuthenticationManager

- `AuthenticationManager` là một interface với phương thức `authenticate()` làm nhiệm vụ xác thực thông tin đăng nhập người dùng. Khi người dùng đăng nhập vào hệ thống, thông tin đăng nhập của họ được thu thập và được chuyển đến `AuthenticationManager` để xác thực thông tin đăng nhập. `AuthenticationManager` sau đó sẽ sử dụng một hoặc nhiều `AuthenticationProvider` để xác thực thông tin đăng nhập của người dùng.

3.3.2.5. AuthenticationProvider

- `AuthenticationProvider` là những class implement interface `AuthenticationProvider` với phương thức `authenticate()` làm nhiệm vụ xử lý các logic liên quan đến authentication.
- `DaoAuthenticationProvider` là một lớp cung cấp tính năng xác thực người dùng bằng cách tìm kiếm thông tin người dùng trong cơ sở dữ liệu. Nó sử dụng `UserDetailsService` (là interface có duy nhất phương thức `loadUserByUsername()` để tìm kiếm thông tin người dùng và sử dụng `PasswordEncoder` để mã hóa mật khẩu và so sánh với mật khẩu đã được mã hóa trong cơ sở dữ liệu (thuật toán mã hóa hay thường sử dụng là `BCrypt`).
- `UserDetails` là một interface bao gồm các phương thức:
 - + `getAuthorities()`: trả về danh sách các quyền của người dùng.
 - + `getPassword()`: trả về password đã dùng trong quá trình xác thực.
 - + `getUsername()`: trả về username đã dùng trong quá trình xác thực.

- + `isAccountNonExpired()`: trả về true nếu tài khoản của người dùng chưa hết hạn.
- + `isAccountNonLocked()`: trả về true nếu người dùng chưa bị khóa.
- + `isCredentialsNonExpired()`: trả về true nếu chứng thực (mật khẩu) của người dùng chưa hết hạn.
- + `isEnabled()`: trả về true nếu người dùng đã được kích hoạt.

3.3.2.6. SecurityContextHolder

- `SecurityContextHolder` là một lớp trong Spring Security được sử dụng để lưu trữ và truy xuất thông tin xác thực của người dùng. Sau khi Spring Security đã validate đủ, user details sẽ được lưu vào Security context. Ứng với sơ đồ bên trên, nếu user xác thực và đăng nhập thành công sẽ cấp cho người dùng một jwt token. Sau khi đi qua lớp lọc JWT Authentication Filter, nếu token hợp lệ nó sẽ lưu thông tin user vào security context holder. Nếu xác thực bị lỗi, nó sẽ từ chối lưu thông tin user này.

4. Cách xây dựng API với Spring Framework

- Tạo Rest API với Spring Boot, JPA và MySQL để thực hiện 4 tính năng cơ bản(CRUD) là thêm(C), xem(R), cập nhật(U) và xóa(D).
- Đầu tiên, truy cập: <https://start.spring.io/> để tạo project

The screenshot shows the Spring Initializr web application interface. It is divided into two main sections: Project Configuration and Dependencies.

Project Configuration:

- Project:** Radio buttons for Gradle - Groovy, Gradle - Kotlin, **Java** (selected), Kotlin, and Groovy.
- Language:** Radio buttons for **Java** (selected), Kotlin, and Groovy.
- Spring Boot:** Radio buttons for 3.1.1 (SNAPSHOT), 3.1.0, 3.0.8 (SNAPSHOT), 3.0.7, 2.7.13 (SNAPSHOT), and **2.7.12** (selected).
- Project Metadata:**
 - Group: `com.viettel`
 - Artifact: `demo-api`
 - Name: `demo-api`
 - Description: `Demo project for Spring Boot`
 - Package name: `com.viettel.demo-api`
- Packaging:** Radio buttons for **Jar** (selected) and War.
- Java:** Radio buttons for 20, 17, **11** (selected), and 8.

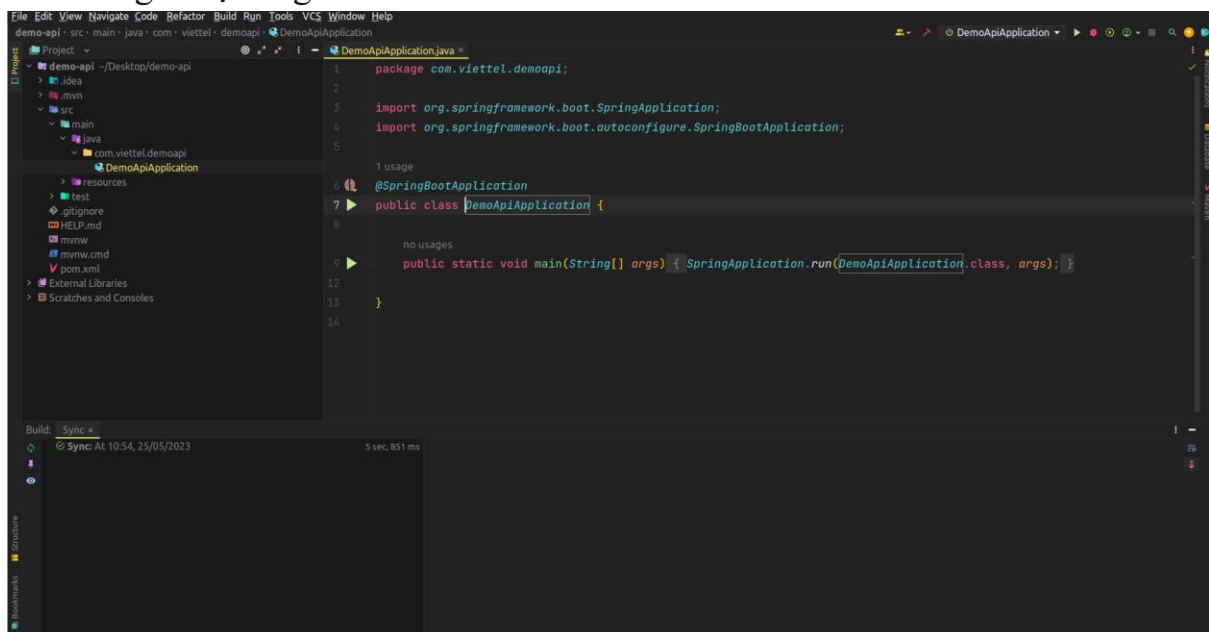
Dependencies:

- Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- MySQL Driver** (SQL): MySQL JDBC driver.
- Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.
- Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

At the top right of the Dependencies section, there is a button labeled "ADD DEPENDENCIES... CTRL + B".

- Chọn maven để quản lý dự án, sau đó chọn ngôn ngữ lập trình là Java và phiên bản spring boot là 2.7.12(không nên chọn phiên bản có SNAPSHOT vì đây là bản đang phát triển). Thêm các dependency là spring web, mysql driver, lombok và spring data jpa.

- Sau khi tải xuống, giải nén và mở idea để bắt đầu dự án. Như đã giải thích ở trên thì **@SpringBootApplication** là sự kết hợp của 3 annotation **@SpringBootConfiguration**, **@ComponentScan**, **@EnableAutoConfiguration** trong đó annotation **@EnableAutoConfiguration** là quan trọng nhất. Với **@EnableAutoConfiguration** annotation, Spring Boot sẽ tự động cấu hình cho ứng dụng của chúng ta dựa vào classpath, các annotations và các thông tin cấu hình mà chúng ta định nghĩa.



- Cấu hình kết nối MySQL

```
server.port=8082
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/demo-api
spring.datasource.username=<enter-username>
spring.datasource.password=<enter-password>
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```


- Tạo entity student và được gắn **@Entity** giúp hibernate có thể dựa vào các thông tin trong thuộc tính của class để mapping xuống bảng trong database. **@Id** xác định khóa chính, **@GeneratedValue** tự động tăng giá trị id, **@Data**, **@NoArgsConstructor**, **@AllArgsConstructor** là các annotation của lombok cho phép tạo các constructor có đầy đủ tham số hoặc không có tham số, các hàm getter, setter, toString.

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

24 usages
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Student {

    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    no usages
    private String name;
    no usages
    private String email;
    no usages
    private int age;
    no usages
    private double gpa;
}
```

- Tạo một interface kế thừa từ JpaRepository để truy xuất dữ liệu từ Database. Trong interface JpaRepository đã có các phương thức cho việc thực hiện CRUD với các entity. Ta có thể sử dụng các function như save(), findAll(), findById(),... Annotation **@Repository** được đặt trên đầu interface cho biết lớp này thực hiện việc thao tác cơ sở dữ liệu.

```
package com.viettel.demoapi.repository;

import com.viettel.demoapi.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

2 usages
@Repository
public interface StudentRepository extends JpaRepository<Student, Integer> {
}
```

- Tạo class StudentService để thực hiện các logic nghiệp vụ, **@Service** đánh dấu class này thực hiện các logic, **@RequiredArgsConstructor** để tự động tạo constructor với các thuộc tính final. Ở đây ta inject StudentRepository thông qua constructor injection để thực hiện các thao tác với database. Các phương thức getAll() tìm tất cả danh sách student, getSingle() tìm kiếm student theo id.

```
2 usages
@Service
@RequiredArgsConstructor
public class StudentService implements IStudentService {

    6 usages
    private final StudentRepository studentRepository;

    1 usage
    @Override
    public List<Student> getAll() {
        return studentRepository.findAll();
    }

    1 usage
    @Override
    public Student getSingle(int studentId) {
        return studentRepository.findById(studentId)
            .orElseThrow();
    }
}
```

- Hàm `create()` dùng để lưu thông tin student vào database, `update()` để cập nhật thông tin student. Logic ở hàm `update` sẽ thực lấy thông tin student dựa theo `id(studentRepo)`, sau đó kiểm tra các trường của tham số student truyền vào có hợp lệ không rồi cập nhật các thông tin này vào đối tượng `studentRepo` sau đó lưu lại vào database.

```
1 usage
@Override
public Student create(Student student) {
    return studentRepository.save(student);
}

1 usage
@Override
public Student update(Student student, int studentId) {
    Student studentRepo = studentRepository.findById(studentId)
        .orElseThrow();
    if(Objects.nonNull(student.getName()) && !"".equalsIgnoreCase(student.getName())) {
        studentRepo.setName(student.getName());
    }
    if(Objects.nonNull(student.getEmail()) && !"".equalsIgnoreCase(student.getEmail())) {
        studentRepo.setEmail(student.getEmail());
    }
    if(student.getAge() != 0) {
        studentRepo.setAge(student.getAge());
    }
    if(student.getGpa() != 0) {
        studentRepo.setGpa(student.getGpa());
    }
    return studentRepository.save(studentRepo);
}
```

- Hàm `delete()`: xóa student theo id

```
1 usage
@Override
public void delete(int studentId) {
    studentRepository.deleteById(studentId);
}
```

- Tạo lớp controller, **@RestController** là sự kết hợp của **@Controller** và **@ResponseBody** giúp cho việc xây dựng RESTful API được dễ dàng và đơn giản hơn. **@RequestMapping("/api/v1/student")** để khai báo các url của các api bắt đầu bằng **"/api/v1/student"**. **@GetMapping("/all")** để cho ta biết khi gọi đến phương thức GET với đường dẫn **/api/v1/student/all** sẽ trả về danh sách sinh viên. Tương tự như vậy thì **@GetMapping("/{id}")** sẽ lấy thông tin một sinh viên, **@PathVariable("id")** sẽ lấy giá trị id từ đường dẫn.

```
@RestController
@RequestMapping("/api/v1/student")
@RequiredArgsConstructor
public class StudentController {

    5 usages
    private final StudentService studentService;

    no usages
    @GetMapping("/all")
    public ResponseEntity<List<Student>> getAllStudent() {
        return ResponseEntity.status(HttpStatus.OK)
            .body(studentService.getAll());
    }

    no usages
    @GetMapping("/{id}")
    public ResponseEntity<Student> getSingleStudent(@PathVariable("id") int studentId) {
        return ResponseEntity.status(HttpStatus.OK)
            .body(studentService.getSingle(studentId));
    }
}
```

- **@PostMapping("/create")** để tạo mới một sinh viên, **@PutMapping("/update/{id}")** để cập nhật thông tin sinh viên, **@RequestBody** dùng để ánh xạ dữ liệu dạng JSON trong *HttpRequest body* sang Java Object (ở đây là lớp Student). **@DeleteMapping("/{id}")** để xóa sinh viên trong cơ sở dữ liệu bằng id.

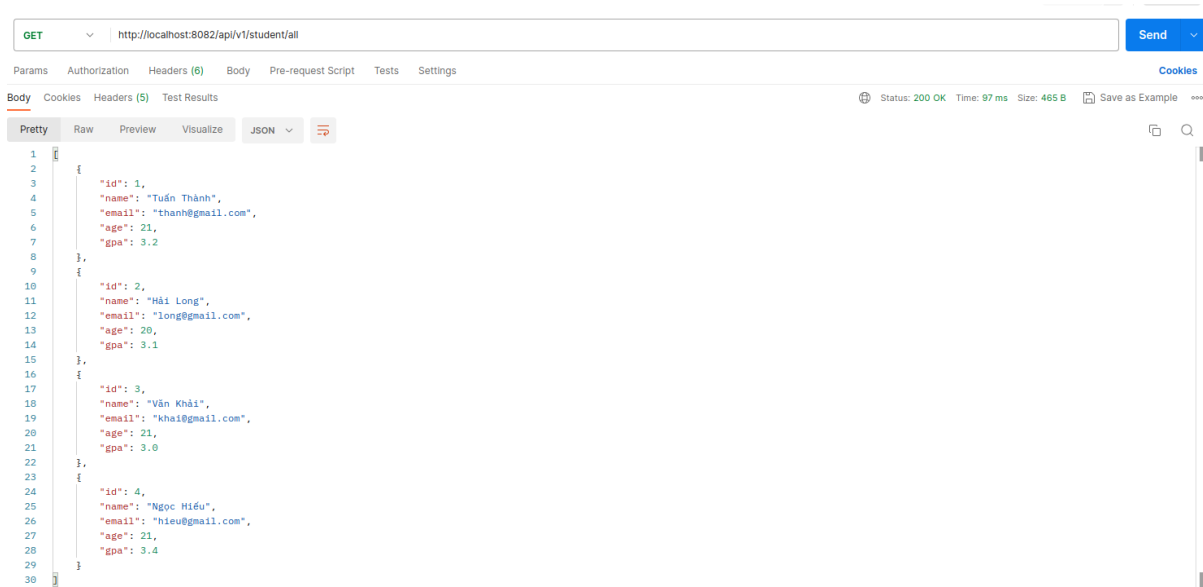
```
no usages
@PostMapping("/create")
public ResponseEntity<Student> createStudent(@RequestBody Student student) {
    return ResponseEntity.status(HttpStatus.CREATED)
        .body(studentService.create(student));
}

no usages
@PutMapping("/update/{id}")
public ResponseEntity<Student> updateStudent(@PathVariable("id") int studentId, @RequestBody Student student) {
    return ResponseEntity.status(HttpStatus.OK)
        .body(studentService.update(student, studentId));
}

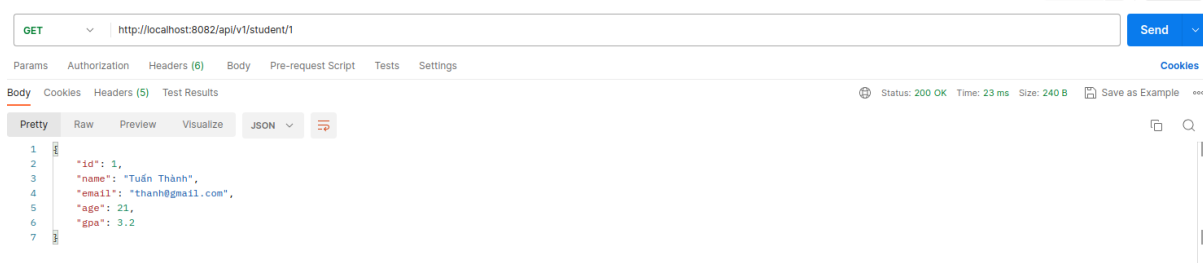
no usages
@DeleteMapping("/{id}")
public ResponseEntity<String> deleteStudent(@PathVariable("id") int studentId) {
    studentService.delete(studentId);
    return ResponseEntity.status(HttpStatus.OK)
        .body("Delete student successfully!!!");
}
```

- Sử dụng Postman để demo kết quả

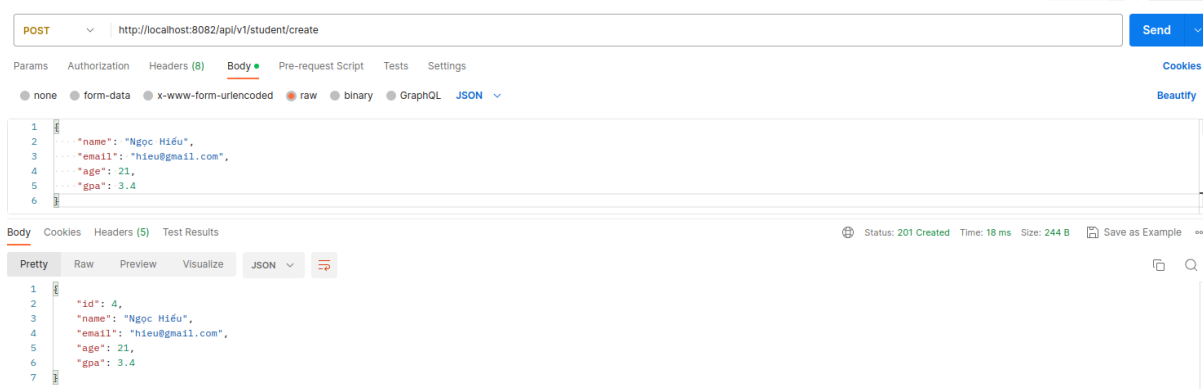
- Lấy danh sách sinh viên



- Lấy thông tin sinh viên có id là 1



- Tạo mới sinh viên



- Cập nhật thông tin sinh viên có id = 1

The screenshot shows a REST client interface with a PUT request to `http://localhost:8082/api/v1/student/update/1`. The request body is a JSON object: `{ "name": "Tuấn Thành", "email": "tuanthanh@gmail.com", "age": 21, "gpa": 3.4 }`. The response status is 200 OK, and the response body is a JSON object: `{ "id": 1, "name": "Tuấn Thành", "email": "tuanthanh@gmail.com", "age": 21, "gpa": 3.4 }`.

- Xóa sinh viên có id = 1

The screenshot shows a REST client interface with a DELETE request to `http://localhost:8082/api/v1/student/1`. The response status is 200 OK, and the response body is the text: `Delete student successfully!!!`.

- Kiểm tra lại kết quả sau khi xóa

The screenshot shows a REST client interface with a GET request to `http://localhost:8082/api/v1/student/all`. The response status is 200 OK, and the response body is a JSON array of three student objects: `[{ "id": 2, "name": "Hải Long", "email": "long@gmail.com", "age": 28, "gpa": 3.1 }, { "id": 3, "name": "Văn Khải", "email": "khai@gmail.com", "age": 21, "gpa": 3.0 }, { "id": 4, "name": "Ngọc Hiếu", "email": "hieue@gmail.com", "age": 21, "gpa": 3.4 }]`.

Kết Luận

- Trong báo cáo này, em đã tìm hiểu và demo những nội dung sau:
 - + Lịch sử ra đời của spring framework.
 - + Một số khái niệm quan trọng cần nắm được của spring framework.
 - + Các module của spring framework, khái quát từng module.
 - + Tìm hiểu về spring boot, khái niệm, kiến trúc và các tính năng của nó.
 - + Tìm hiểu về spring jpa, giới thiệu và các đặc điểm của nó.
 - + Tìm hiểu về spring security, giới thiệu và trình bày quy trình xác thực.
 - + Triển khai API với spring boot với các chức năng CRUD cơ bản.
- Hạn chế, điểm chưa đạt được:
 - + Phần tìm hiểu vẫn còn nhiều thiếu sót, chưa đầy đủ chi tiết.
 - + Một số phần tìm hiểu chưa áp dụng được vào trong demo.
 - + Chưa kiểm thử và đánh giá hiệu suất ứng dụng khi dùng spring.

Tài Liệu Tham Khảo

Giới thiệu về Spring JPA. (n.d.). From techmaster: <https://techmaster.vn/posts/36255/gioi-thieu-ve-spring-jpa>

Hướng dẫn lập trình Spring Security. (n.d.). From kipalog: <https://kipalog.com/posts/Huong-dan-lap-trinh-Spring-Security>

Spring Boot Architecture. (n.d.). From javatpoint: <https://www.javatpoint.com/spring-boot-architecture>

Spring Data JPA là gì? Tại sao chúng ta cần sử dụng nó? (n.d.). From shareprogramming: <https://shareprogramming.net/spring-data-jpa-la-gi-tai-sao-chung-ta-can-su-dung-no/>

Spring Framework Architecture. (n.d.). From geeksforgeek: <https://www.geeksforgeeks.org/spring-framework-architecture/>

Spring Security: Tìm hiểu về internal flow. (n.d.). From magz.techover.io: <https://magz.techover.io/2023/01/02/spring-security-tim-hieu-ve-internal-flow/>