

Lập trình

JAVA-Spring-Boot

**Biên soạn:**

Trần Nhật Nam

TÀI LIỆU LƯU HÀNH NỘI BỘ

CHƯƠNG 1: SPRING BOOT – GIỚI THIỆU

Spring Boot là một framework mã nguồn mở dựa trên Java được sử dụng để tạo các Micro Service. Nó được phát triển bởi nhóm Pivotal và được sử dụng để xây dựng các ứng dụng Spring độc lập và sẵn sàng cho việc triển khai. Chương này sẽ giới thiệu về Spring Boot và làm quen với các khái niệm cơ bản của nó.

1.1. Micro Service là gì?

Micro Service là một kiến trúc cho phép các nhà phát triển phát triển và triển khai các dịch vụ một cách độc lập. Mỗi dịch vụ khi chạy đều có quy trình riêng, từ đó đạt được mô hình nhẹ để hỗ trợ các ứng dụng kinh doanh.

Các Ưu điểm:

Micro Service mang đến những lợi ích sau cho các nhà phát triển:

Triển khai dễ dàng

Dễ dàng mở rộng

Tương thích với Containers

Yêu cầu cấu hình tối thiểu

Giảm thời gian sản xuất

1.2. Spring Boot là gì?

Spring Boot cung cấp một nền tảng tốt cho các nhà phát triển Java để phát triển các ứng dụng Spring độc lập và sẵn sàng cho việc triển khai. Bạn có thể bắt đầu với các cấu hình tối thiểu mà không cần thiết phải thiết lập toàn bộ cấu hình Spring.

Các Ưu điểm:

Spring Boot mang đến những lợi ích sau cho các nhà phát triển:

Dễ hiểu và phát triển ứng dụng Spring

Tăng năng suất làm việc

Giảm thời gian phát triển

Mục tiêu:

Spring Boot được thiết kế với những mục tiêu sau:

Tránh sử dụng cấu hình XML phức tạp trong Spring

Phát triển các ứng dụng Spring sẵn sàng cho việc triển khai một cách dễ dàng

Giảm thời gian phát triển và chạy ứng dụng một cách độc lập

Cung cấp một cách dễ dàng để bắt đầu với ứng dụng.

1.3. Tại sao chọn Spring Boot?

Bạn có thể chọn Spring Boot vì các tính năng và lợi ích mà nó cung cấp như sau:

Nó cung cấp một cách linh hoạt để cấu hình Java Beans, cấu hình XML và giao dịch cơ sở dữ liệu.

Nó cung cấp các thư viện có sẳn xử lý hàng loạt mạnh mẽ.

Trong Spring Boot, mọi thứ được tự động cấu hình; không cần cấu hình thủ công.

Nó cung cấp ứng dụng Spring dựa trên các chú thích (annotations).

Giảm thiểu việc quản lý phụ thuộc (dependency) trong dự án.

Bao gồm một máy chủ Servlet nhúng (Embedded Servlet Container).

1.4. Spring Boot hoạt động như thế nào?

Spring Boot tự động cấu hình ứng dụng của bạn dựa trên các phụ thuộc (dependencies) bạn đã thêm vào dự án bằng cách sử dụng chú thích @EnableAutoConfiguration. Ví dụ, nếu cơ sở dữ liệu MySQL có trong classpath của bạn, nhưng bạn chưa cấu hình bất kỳ kết nối cơ sở dữ liệu nào, thì Spring Boot sẽ tự động cấu hình một cơ sở dữ liệu in-memory (trong bộ nhớ).

Điểm vào của ứng dụng Spring Boot là class chứa chú thích @SpringBootApplication và phương thức main.

Spring Boot tự động quét tất cả các thành phần (components) có trong dự án bằng cách sử dụng chú thích @ComponentScan.

1.5. Spring Boot Starters

Xử lý việc quản lý phụ thuộc là một nhiệm vụ khó khăn đối với các dự án lớn. Spring Boot giải quyết vấn đề này bằng cách cung cấp một tập hợp các phụ thuộc cho sự tiện lợi của nhà phát triển.

Ví dụ, nếu bạn muốn sử dụng Spring và JPA để truy cập cơ sở dữ liệu, bạn chỉ cần bao gồm phụ thuộc (dependency) spring-boot-starter-data-jpa trong dự án của bạn.

Lưu ý: Tất cả các Spring Boot starters đều tuân theo cùng một mẫu đặt tên spring-boot-starter-\*, trong đó \* chỉ ra loại ứng dụng mà nó hỗ trợ.

Ví dụ:

Hãy xem các Spring Boot starters sau đây để hiểu rõ hơn:

Spring Boot Starter Actuator được sử dụng để giám sát và quản lý ứng dụng của bạn. Code được hiển thị dưới đây:

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

Phụ thuộc (dependency) Spring Boot Starter Web được sử dụng để viết các Rest Endpoints. Mã của nó được hiển thị dưới đây:

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

Phụ thuộc (dependency) Spring Boot Starter Thymeleaf được sử dụng để tạo một ứng dụng web. Mã của nó được hiển thị dưới đây:

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId>  </dependency> |

Phụ thuộc (dependency) Spring Boot Starter Test được sử dụng để viết các bài kiểm tra (test cases). Mã của nó được hiển thị dưới đây:

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  </dependency> |

Lưu ý: Mã trên là dạng cấu hình Maven để chỉ định các phụ thuộc cần thiết cho từng loại chức năng trong dự án Spring Boot.

1.6. Tự động cấu hình (Auto Configuration)

Spring Boot Auto Configuration tự động cấu hình ứng dụng Spring của bạn dựa trên các phụ thuộc JAR bạn đã thêm vào dự án. Ví dụ, nếu cơ sở dữ liệu MySQL có trong classpath của bạn, nhưng bạn chưa cấu hình bất kỳ kết nối cơ sở dữ liệu nào, thì Spring Boot sẽ tự động cấu hình một cơ sở dữ liệu in-memory (trong bộ nhớ).

Để thực hiện điều này, bạn cần thêm chú thích @EnableAutoConfiguration hoặc @SpringBootApplication vào file class chính của bạn. Sau đó, ứng dụng Spring Boot của bạn sẽ được tự động cấu hình.

Hãy xem mã sau để hiểu rõ hơn:

|  |
| --- |
| // Spring Boot Application  @SpringBootApplication  public class MyApp {  public static void main(String[] args) {  SpringApplication.run(MyApp.class, args);  }  } |

1.7. Spring Boot Application

Điểm vào của ứng dụng Spring Boot là class chứa chú thích @SpringBootApplication. Class này nên có phương thức main để chạy ứng dụng Spring Boot. Chú thích @SpringBootApplication bao gồm cả Auto Configuration, Component Scan và Spring Boot Configuration.

Nếu bạn đã thêm chú thích @SpringBootApplication vào class, bạn không cần thêm @EnableAutoConfiguration, @ComponentScan và @SpringBootConfiguration. Chú thích @SpringBootApplication đã bao gồm tất cả các chú thích khác.

Hãy xem mã sau để hiểu rõ hơn:

|  |
| --- |
| // Spring Boot Application  @SpringBootApplication  public class MyApp {  public static void main(String[] args) {  SpringApplication.run(MyApp.class, args);  }  } |

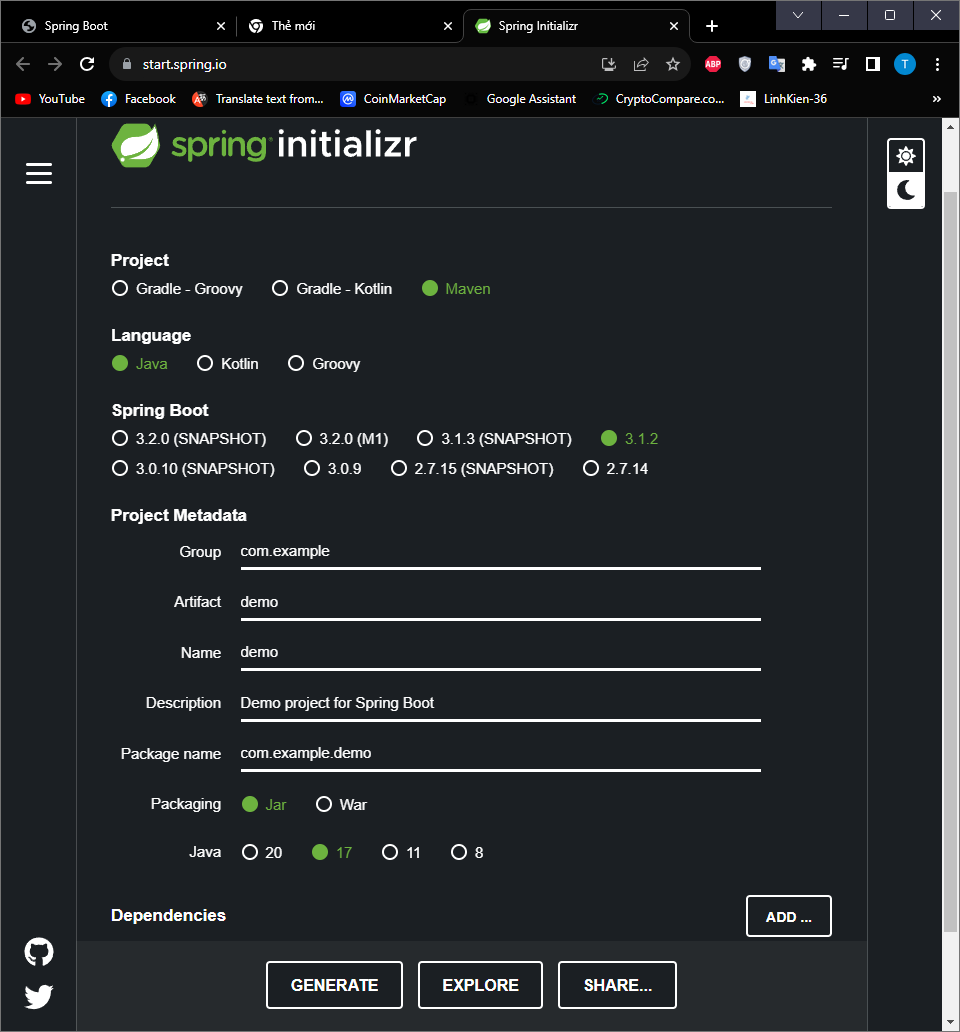
CHƯƠNG 2: SPRING BOOT – BOOTSTRAPPING

Bootstrapping là quá trình khởi tạo và cấu hình ban đầu cho một ứng dụng hoặc hệ thống, đặc biệt là trong ngữ cảnh của phát triển phần mềm. Với Spring Boot, quá trình bootstrapping trở nên đơn giản hơn nhờ vào tính năng tự động cấu hình (Auto Configuration). Khi bạn khởi đầu một ứng dụng Spring Boot, Spring sẽ tự động cấu hình và tùy chỉnh ứng dụng dựa trên các phụ thuộc (dependencies) và cấu hình có sẵn. Điều này giúp bạn tiết kiệm thời gian và công sức so với việc cấu hình thủ công và giúp ứng dụng nhanh chóng được triển khai và chạy. Chương này sẽ giải thích cách thực hiện bootstrapping trên một ứng dụng Spring Boot.

2.1. Spring Initializer

Một trong những cách để bootstrapping một ứng dụng Spring Boot là sử dụng Spring Initializer. Để làm điều này, bạn sẽ phải truy cập trang web Spring Initializer http://start.spring.io/ và chọn Công cụ Build, Phiên bản Spring Boot và nền tảng của bạn. Bạn cũng cần cung cấp Nhóm (Group), Tên dự án (Artifact) và các phụ thuộc cần thiết để chạy ứng dụng.

Hãy xem hình ảnh minh họa dưới đây, cho thấy một ví dụ khi chúng ta đã thêm phụ thuộc (dependency) springboot-starter-web để viết REST Endpoints.



Hình : Khởi tạo Project Spring Boot

Sau khi cung cấp Nhóm, Tên dự án, Phụ thuộc, Công cụ Build, Nền tảng và Phiên bản, hãy nhấp vào nút Generate Project. Một tập tin zip sẽ được tải xuống và các tập tin sẽ được giải nén.

Phần này sẽ giải thích các ví dụ bằng cách sử dụng cả Maven và Gradle.

Maven

Sau khi bạn tải xuống dự án, giải nén tập tin. Bây giờ, tệp pom.xml của bạn sẽ trông như dưới đây:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2022/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.tutorialspoint</groupId>  <artifactId>demo</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>demo</name>  <description>Demo project for Spring Boot</description>    <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.8.RELEASE</version>  <relativePath/>  </parent>    <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>    <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>    <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

Gradle:

Sau khi bạn tải xuống dự án, giải nén tệp. Bây giờ tệp build.gradle của bạn trông như dưới đây:

|  |
| --- |
| buildscript {  ext {  springBootVersion = '1.5.8.RELEASE'  }  repositories {  mavenCentral()  }  dependencies {  classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  }  }  apply plugin: 'java'  apply plugin: 'eclipse'  apply plugin: 'org.springframework.boot'  group = 'com.tutorialspoint'  version = '0.0.1-SNAPSHOT'  sourceCompatibility = 1.8  repositories {  mavenCentral()  }  dependencies {  compile('org.springframework.boot:spring-boot-starter-web')  testCompile('org.springframework.boot:spring-boot-starter-test')  } |

2.2. Phụ thuộc Class Path (Class Path Dependencies)

Spring Boot cung cấp một số Starters để thêm các tệp jar vào class path của chúng ta. Ví dụ, để viết một Rest Endpoint, chúng ta cần thêm phụ thuộc (dependency) spring-boot-starter-web vào class path của chúng ta. Hãy xem mã sau đây để hiểu rõ hơn:

Maven dependency

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

Gradle dependency

|  |
| --- |
| dependencies {  compile('org.springframework.boot:spring-boot-starter-web')  } |

2.3. Phần main (Main Method)

Phương thức main nên được viết trong class của ứng dụng Spring Boot. Class này nên được chú thích bằng @SpringBootApplication. Đây là điểm vào của ứng dụng Spring Boot để bắt đầu chạy. Bạn có thể tìm thấy tệp main class trong thư mục src/java/main với package mặc định.

Trong ví dụ này, tệp main class nằm trong thư mục src/java/main với package mặc định là com.tutorialspoint.demo. Hãy xem mã sau để hiểu rõ hơn:

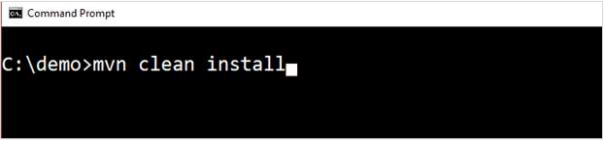
|  |
| --- |
| package com.tutorialspoint.demo;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  @SpringBootApplication  @RestController  public class DemoApplication {  public static void main(String[] args) {  SpringApplication.run(DemoApplication.class, args);  }  @RequestMapping(value="/")  public String hello() {  return "Hello World";  }  } |

2.4. Tạo một JAR thực thi được (Executable JAR)

Hãy tạo một tệp JAR thực thi được để chạy ứng dụng Spring Boot bằng các lệnh Maven và Gradle trong dòng lệnh như sau:

Sử dụng lệnh Maven:

mvn clean install



Hình : Lệnh MVN

Sau khi thực thi lệnh, bạn sẽ thấy thông báo BUILD SUCCESS xuất hiện ở dòng lệnh.



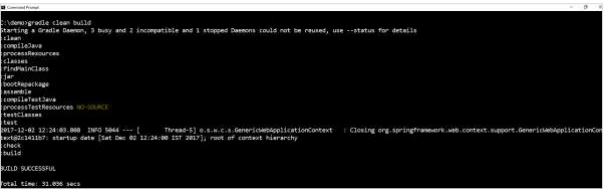
Hình : Xây dựng thành công

Sử dụng lệnh Gradle:

gradle clean build



Hình : Lệnh gradle

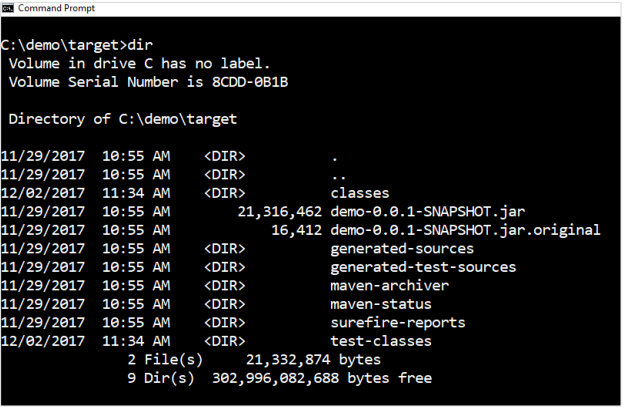
Sau khi thực thi lệnh, bạn sẽ thấy thông báo BUILD SUCCESSFUL xuất hiện ở dòng lệnh.

Hình : Xây dựng thành công

2.5. Hello World với Java Spring Boot

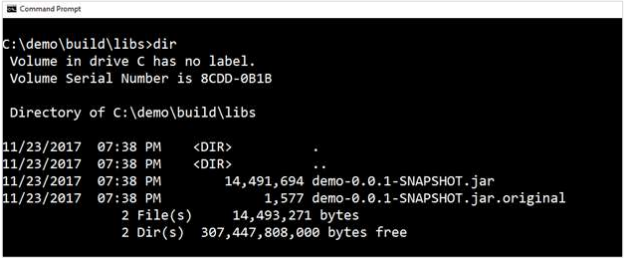
Sau khi tạo một tệp JAR thực thi được, bạn có thể tìm thấy nó ở các thư mục sau.

Với Maven, bạn có thể tìm thấy tệp JAR trong thư mục target như sau:



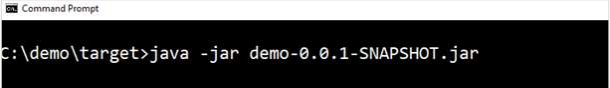
Hình : Thư mục cho Maven

Với Gradle, bạn có thể tìm thấy tệp JAR trong thư mục build/libs như sau:



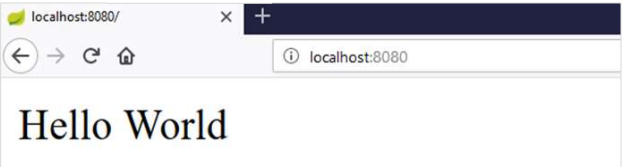
Hình : Thư mục cho Gradle

Bây giờ, hãy chạy tệp JAR bằng lệnh java -jar <JARFILE>. Trong ví dụ trên, tệp JAR được đặt tên là demo-0.0.1-SNAPSHOT.jar.



Hình : Chạy java Hello World

Bây giờ, nhìn vào cửa sổ console, Tomcat đã khởi động trên cổng 8080 (http). Bây giờ, hãy mở trình duyệt web và truy cập vào URL http://localhost:8080/, bạn sẽ thấy kết quả như sau: (Dữ liệu hình ảnh không thể hiện trong phiên dịch).



Hình : Hello World trên trình duyệt web

CHƯƠNG 3: SPRING BOOT – BUILDING RESTFUL WEB SERVICES

3.1. Web services RESTful và các chú thích:

Spring Boot cung cấp một hỗ trợ rất tốt để xây dựng các dịch vụ web RESTful cho các ứng dụng doanh nghiệp. Chương này sẽ giải thích chi tiết về việc xây dựng các dịch vụ web RESTful bằng Spring Boot.

Lưu ý: Để xây dựng một dịch vụ web RESTful, chúng ta cần thêm phụ thuộc Spring Boot Starter Web vào tệp cấu hình build.

Nếu bạn đang sử dụng Maven, sử dụng mã sau đây để thêm phụ thuộc sau vào tệp pom.xml của bạn:

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

Nếu bạn đang sử dụng Gradle, sử dụng mã sau đây để thêm phụ thuộc sau vào tệp build.gradle của bạn:

|  |
| --- |
| dependencies {  compile('org.springframework.boot:spring-boot-starter-web')  } |

Dưới đây là mã cho tệp cấu hình build Maven hoàn chỉnh - pom.xml:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <!-- Các thông tin khác của dự án -->  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <!-- Các plugin khác của dự án -->  </project> |

Dưới đây là mã cho tệp cấu hình build Gradle hoàn chỉnh - build.gradle:

|  |
| --- |
| buildscript {  ext {  springBootVersion = '1.5.8.RELEASE'  }  repositories {  mavenCentral()  }  dependencies {  classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  }  }  apply plugin: 'java'  apply plugin: 'eclipse'  apply plugin: 'org.springframework.boot'  group = 'com.tutorialspoint'  version = '0.0.1-SNAPSHOT'  sourceCompatibility = 1.8  repositories {  mavenCentral()  }  dependencies {  compile('org.springframework.boot:spring-boot-starter-web')  testCompile('org.springframework.boot:spring-boot-starter-test')  } |

Trước khi tiếp tục xây dựng một dịch vụ web RESTful, nên hiểu rõ về các chú thích sau:

Rest Controller

Chú thích @RestController được sử dụng để định nghĩa các dịch vụ web RESTful. Nó hỗ trợ trả về các đối tượng dưới dạng JSON, XML và các định dạng tùy chỉnh khác. Cú pháp của nó được hiển thị dưới đây:

|  |
| --- |
| @RestController  public class ProductServiceController {  } |

Request Mapping

Chú thích @RequestMapping được sử dụng để xác định Request URI để truy cập vào các Endpoint REST. Chúng ta có thể xác định Request method để tiêu thụ và sản xuất đối tượng. Request method mặc định là GET.

|  |
| --- |
| @RequestMapping(value="/products")  public ResponseEntity<Object> getProducts() { } |

Request Body

Chú thích @RequestBody được sử dụng để xác định kiểu nội dung của request body.

|  |
| --- |
| public ResponseEntity<Object> createProduct(@RequestBody Product product) {  } |

Path Variable

Chú thích @PathVariable được sử dụng để xác định URI yêu cầu tùy chỉnh hoặc động. Biến đường dẫn trong URI yêu cầu được xác định bằng dấu ngoặc nhọn {} như được hiển thị dưới đây:

|  |
| --- |
| public ResponseEntity<Object> updateProduct(@PathVariable("id") String id) {  } |

Request Parameter

Chú thích @RequestParam được sử dụng để đọc các tham số yêu cầu từ URL Request. Theo mặc định, nó là một tham số bắt buộc. Chúng ta cũng có thể thiết lập giá trị mặc định cho các tham số yêu cầu như được hiển thị dưới đây:

|  |
| --- |
| public ResponseEntity<Object> getProduct(@RequestParam(value="name", required=false, defaultValue="honey") String name) {  } |

Các chú thích này giúp xác định và định dạng các Endpoint RESTful của chúng ta trong ứng dụng Spring Boot, cho phép chúng ta xử lý các yêu cầu từ client một cách linh hoạt và tiện lợi.

3.2. GET API

Phương thức yêu cầu HTTP mặc định là GET. Phương thức này không yêu cầu bất kỳ Request Body nào. Bạn có thể gửi các tham số yêu cầu và biến đường dẫn để xác định URL tùy chỉnh hoặc động. Đoạn mã ví dụ dưới đây mô tả cách định nghĩa phương thức yêu cầu HTTP GET. Trong ví dụ này, chúng ta sử dụng HashMap để lưu trữ các Sản phẩm. Chú ý rằng chúng ta sử dụng một lớp POJO làm Sản phẩm được lưu trữ.

Ở đây, URI yêu cầu là /products và nó sẽ trả về danh sách các sản phẩm từ bộ nhớ HashMap. Dưới đây là tệp lớp điều khiển chứa REST Endpoint của phương thức GET:

|  |
| --- |
| package com.tutorialspoint.demo.controller;  import java.util.HashMap;  import java.util.Map;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import com.tutorialspoint.demo.model.Product;  @RestController  public class ProductServiceController {  private static Map<String, Product> productRepo = new HashMap<>();  static {  Product honey = new Product();  honey.setId("1");  honey.setName("Honey");  productRepo.put(honey.getId(), honey);  Product almond = new Product();  almond.setId("2");  almond.setName("Almond");  productRepo.put(almond.getId(), almond);  }  @RequestMapping(value="/products")  public ResponseEntity<Object> getProduct() {  return new ResponseEntity<>(productRepo.values(), HttpStatus.OK);  }  } |

3.3. POST API

Yêu cầu HTTP POST được sử dụng để tạo một tài nguyên mới. Phương thức này chứa Request Body. Chúng ta có thể gửi các tham số yêu cầu và biến đường dẫn để xác định URL tùy chỉnh hoặc động.

Dưới đây là ví dụ mã để định nghĩa phương thức yêu cầu HTTP POST. Trong ví dụ này, chúng ta sử dụng HashMap để lưu trữ Sản phẩm, trong đó Sản phẩm là một lớp POJO.

Ở đây, URI yêu cầu là /products và nó sẽ trả về một String sau khi lưu trữ sản phẩm vào bộ nhớ HashMap.

|  |
| --- |
| package com.tutorialspoint.demo.controller;  import java.util.HashMap;  import java.util.Map;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.RequestBody;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;  import com.tutorialspoint.demo.model.Product;  @RestController  public class ProductServiceController {  private static Map<String, Product> productRepo = new HashMap<>();  @RequestMapping(value="/products", method=RequestMethod.POST)  public ResponseEntity<Object> createProduct(@RequestBody Product product) {  productRepo.put(product.getId(), product);  return new ResponseEntity<>("Product is created successfully", HttpStatus.CREATED);  }  } |

3.4. PUT API

Yêu cầu HTTP PUT được sử dụng để cập nhật tài nguyên hiện có. Phương thức này chứa Request Body. Chúng ta có thể gửi các tham số yêu cầu và biến đường dẫn để xác định URL tùy chỉnh hoặc động.

Ví dụ dưới đây cho thấy cách định nghĩa phương thức yêu cầu HTTP PUT. Trong ví dụ này, chúng ta sử dụng HashMap để cập nhật Sản phẩm hiện có, trong đó Sản phẩm là một lớp POJO.

Ở đây, URI yêu cầu là /products/{id}, nó sẽ trả về một String sau khi cập nhật sản phẩm vào bộ nhớ HashMap. Chú ý rằng chúng ta sử dụng biến đường dẫn {id} để xác định ID của sản phẩm cần được cập nhật.

|  |
| --- |
| package com.tutorialspoint.demo.controller;  import java.util.HashMap;  import java.util.Map;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestBody;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;  import com.tutorialspoint.demo.model.Product;  @RestController  public class ProductServiceController {  private static Map<String, Product> productRepo = new HashMap<>();  @RequestMapping(value="/products/{id}", method=RequestMethod.PUT)  public ResponseEntity<Object> updateProduct(@PathVariable("id") String id, @RequestBody Product product) {  productRepo.remove(id);  product.setId(id);  productRepo.put(id, product);  return new ResponseEntity<>("Product is updated successfully", HttpStatus.OK);  }  } |

3.5. DELETE API

Yêu cầu HTTP DELETE được sử dụng để xóa tài nguyên hiện có. Phương thức này không chứa bất kỳ Request Body nào. Chúng ta có thể gửi các tham số yêu cầu và biến đường dẫn để xác định URL tùy chỉnh hoặc động.

Ví dụ dưới đây cho thấy cách định nghĩa phương thức yêu cầu HTTP DELETE. Trong ví dụ này, chúng ta sử dụng HashMap để xóa Sản phẩm hiện có, trong đó Sản phẩm là một lớp POJO.

URI yêu cầu là /products/{id} và nó sẽ trả về một String sau khi xóa sản phẩm khỏi bộ nhớ HashMap. Chúng ta sử dụng biến đường dẫn {id} để xác định ID của sản phẩm cần xóa.

|  |
| --- |
| package com.tutorialspoint.demo.controller;  import java.util.HashMap;  import java.util.Map;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;  import com.tutorialspoint.demo.model.Product;  @RestController  public class ProductServiceController {  private static Map<String, Product> productRepo = new HashMap<>();  static {  Product honey = new Product();  honey.setId("1");  honey.setName("Honey");  productRepo.put(honey.getId(), honey);  Product almond = new Product();  almond.setId("2");  almond.setName("Almond");  productRepo.put(almond.getId(), almond);  }  @RequestMapping(value="/products/{id}", method=RequestMethod.DELETE)  public ResponseEntity<Object> delete(@PathVariable("id") String id) {  productRepo.remove(id);  return new ResponseEntity<>("Product is deleted successfully", HttpStatus.OK);  }  } |

3.6. Lớp ứng dụng chính

Phần này cung cấp cho bạn toàn bộ mã nguồn hoàn chỉnh. Hãy xem các mã sau để hiểu chức năng của từng phần.

Spring Boot - DemoApplication.java

|  |
| --- |
| package com.tutorialspoint.demo;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class DemoApplication {  public static void main(String[] args) {  SpringApplication.run(DemoApplication.class, args);  }  } |

Lớp POJO - Product.java

|  |
| --- |
| package com.tutorialspoint.demo.model;  public class Product {  private String id;  private String name;  public String getId() {  return id;  }  public void setId(String id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  } |

Lớp Rest Controller - ProductServiceController.java

|  |
| --- |
| package com.tutorialspoint.demo.controller;  import java.util.HashMap;  import java.util.Map;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestBody;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;  import com.tutorialspoint.demo.model.Product;  @RestController  public class ProductServiceController {  private static Map<String, Product> productRepo = new HashMap<>();  static {  Product honey = new Product();  honey.setId("1");  honey.setName("Honey");  productRepo.put(honey.getId(), honey);  Product almond = new Product();  almond.setId("2");  almond.setName("Almond");  productRepo.put(almond.getId(), almond);  }  @RequestMapping(value="/products/{id}", method=RequestMethod.DELETE)  public ResponseEntity<Object> delete(@PathVariable("id") String id) {  productRepo.remove(id);  return new ResponseEntity<>("Product is deleted successfully", HttpStatus.OK);  }  @RequestMapping(value="/products/{id}", method=RequestMethod.PUT)  public ResponseEntity<Object> updateProduct(@PathVariable("id") String id, @RequestBody Product product) {  productRepo.remove(id);  product.setId(id);  productRepo.put(id, product);  return new ResponseEntity<>("Product is updated successfully", HttpStatus.OK);  }  @RequestMapping(value="/products", method=RequestMethod.POST)  public ResponseEntity<Object> createProduct(@RequestBody Product product) {  productRepo.put(product.getId(), product);  return new ResponseEntity<>("Product is created successfully", HttpStatus.CREATED);  }  @RequestMapping(value="/products")  public ResponseEntity<Object> getProduct() {  return new ResponseEntity<>(productRepo.values(), HttpStatus.OK);  }  } |

Sau đó, bạn có thể tạo một tệp JAR thực thi và chạy ứng dụng spring boot bằng cách sử dụng các lệnh Maven hoặc Gradle như sau:

Đối với Maven, sử dụng lệnh sau:

mvn clean install

Sau khi "BUILD SUCCESS", bạn có thể tìm thấy tệp JAR trong thư mục target.

Đối với Gradle, sử dụng lệnh sau:

gradle clean build

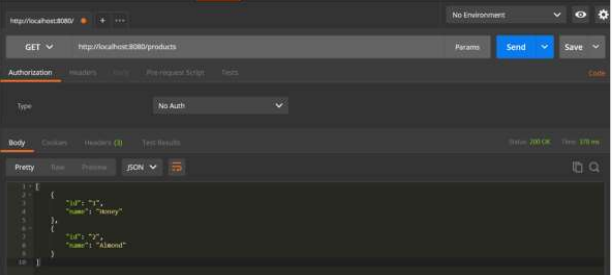
Sau khi "BUILD SUCCESSFUL", bạn có thể tìm thấy tệp JAR trong thư mục build/libs.

Bạn có thể chạy tệp JAR bằng cách sử dụng lệnh sau:

java –jar <JARFILE>

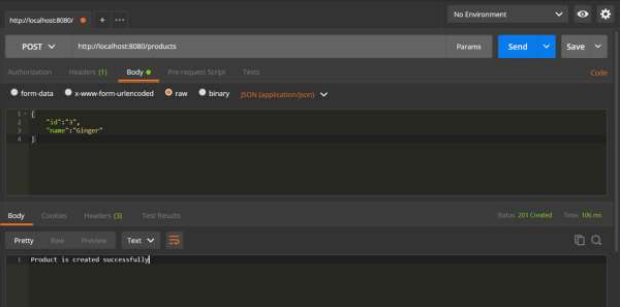
Điều này sẽ khởi động ứng dụng trên cổng Tomcat 8080. Bạn có thể gửi yêu cầu đến các URL của API theo cú pháp sau:

GET API URL: http://localhost:8080/products



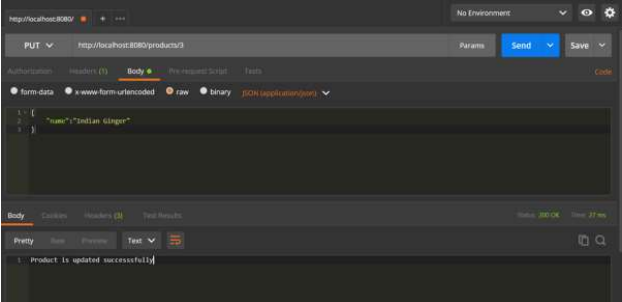
Hình : GET API

POST API URL: http://localhost:8080/products



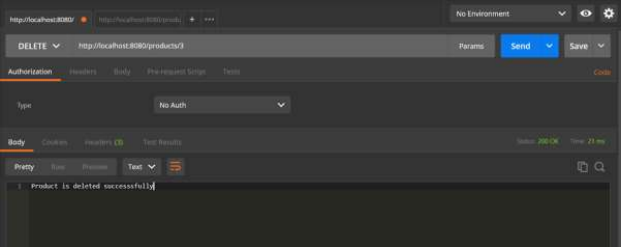
Hình : POST API

PUT API URL: http://localhost:8080/products/3



Hình : PUT API

DELETE API URL: http://localhost:8080/products/3



Hình : DELETE API

CHƯƠNG 4: SPRING BOOT – FLYWAY DATABASE – MYSQL

4.1. Giới thiệu Flyway

Flyway là một ứng dụng quản lý phiên bản giúp cải thiện schema của cơ sở dữ liệu của bạn dễ dàng và đáng tin cậy trên tất cả các phiên bản của ứng dụng. Để tìm hiểu thêm về Flyway, bạn có thể sử dụng liên kết: https://flywaydb.org/

Nhiều dự án phần mềm sử dụng cơ sở dữ liệu quan hệ. Điều này đòi hỏi xử lý các phiên bản cơ sở dữ liệu, thường được gọi là database migration hay schema migration.

Trong chương này, bạn sẽ tìm hiểu chi tiết về cách cấu hình cơ sở dữ liệu Flyway trong ứng dụng Spring Boot của bạn.

4.2. Cấu hình

Trước tiên, tải dự án Spring Boot từ trang Spring Initializer http://start.spring.io/ và chọn các dependencies sau:

Spring Boot Starter Web

Flyway

MySQL

JDBC

Người dùng Maven có thể thêm các dependencies sau vào tệp pom.xml:

|  |
| --- |
| <dependency>  <groupId>org.flywaydb</groupId>  <artifactId>flyway-core</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency> |

Người dùng Gradle có thể thêm các dependencies sau vào tệp build.gradle:

|  |
| --- |
| compile('org.flywaydb:flyway-core')  compile('org.springframework.boot:spring-boot-starter-jdbc')  compile('org.springframework.boot:spring-boot-starter-web')  compile('mysql:mysql-connector-java') |

Trong tệp application.properties, chúng ta cần cấu hình các thuộc tính cơ sở dữ liệu để tạo một DataSource và cũng cần cấu hình các thuộc tính flyway trong application.properties.

Người dùng sử dụng tệp properties có thể thêm các thuộc tính sau vào tệp application.properties:

|  |
| --- |
| spring.application.name=flywayapp  spring.datasource.driverClassName=com.mysql.jdbc.Driver  spring.datasource.url=jdbc:mysql://localhost:3306/USERSERVICE?autoreconnect=true  spring.datasource.username=root  spring.datasource.password=root  spring.datasource.testOnBorrow=true  spring.datasource.testWhileIdle=true  spring.datasource.timeBetweenEvictionRunsMillis=60000  spring.datasource.minEvictableIdleTimeMillis=30000  spring.datasource.validationQuery=SELECT 1  spring.datasource.max-active=15  spring.datasource.max-idle=10  spring.datasource.max-wait=8000  flyway.url=jdbc:mysql://localhost:3306/mysql  flyway.schemas=USERSERVICE  flyway.user=root  flyway.password=root |

Người dùng sử dụng tệp YAML có thể thêm các thuộc tính sau vào tệp application.yml:

|  |
| --- |
| spring:  application:  name: flywayapp  datasource:  driverClassName: com.mysql.jdbc.Driver  url: "jdbc:mysql://localhost:3306/USERSERVICE?autoreconnect=true"  password: "root"  username: "root"  testOnBorrow: true  testWhileIdle: true  timeBetweenEvictionRunsMillis: 60000  minEvictableIdleTimeMillis: 30000  validationQuery: SELECT 1  max-active: 15  max-idle: 10  max-wait: 8000  flyway:  url: jdbc:mysql://localhost:3306/mysql  schemas: USERSERVICE  user: "root"  password: "root" |

Người dùng sử dụng tệp YAML có thể thêm các thuộc tính sau vào tệp application.yml:

|  |
| --- |
| spring:  application:  name: flywayapp  datasource:  driverClassName: com.mysql.jdbc.Driver  url: "jdbc:mysql://localhost:3306/USERSERVICE?autoreconnect=true"  password: "root"  username: "root"  testOnBorrow: true  testWhileIdle: true  timeBetweenEvictionRunsMillis: 60000  minEvictableIdleTimeMillis: 30000  validationQuery: SELECT 1  max-active: 15  max-idle: 10  max-wait: 8000  flyway:  url: jdbc:mysql://localhost:3306/mysql  schemas: USERSERVICE  user: "root"  password: "root" |

Bây giờ, tạo một tệp SQL trong thư mục src/main/resources/db/migration. Đặt tên cho tệp SQL là "V1\_\_Initial.sql":

|  |
| --- |
| CREATE TABLE USERS (ID INT AUTO\_INCREMENT PRIMARY KEY, USERID VARCHAR(45));  INSERT INTO USERS (ID, USERID) VALUES (1, 'tutorialspoint.com'); |

Mã lớp ứng dụng chính của Spring Boot được đưa ra dưới đây:

|  |
| --- |
| package com.tutorialspoint.flywayapp;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class FlywayappApplication {  public static void main(String[] args) {  SpringApplication.run(FlywayappApplication.class, args);  }  } |

Cấu hình hoàn chỉnh cho tệp cấu hình build như sau: Maven – pom.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.tutorialspoint</groupId>  <artifactId>flywayapp</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>flywayapp</name>  <description>Demo project for Spring Boot</description>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.9.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.flywaydb</groupId>  <artifactId>flyway-core</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

Gradle – build.gradle:

|  |
| --- |
| buildscript {  ext {  springBootVersion = '1.5.9.RELEASE'  }  repositories {  mavenCentral()  }  dependencies {  classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  }  }  apply plugin: 'java'  apply plugin: 'eclipse'  apply plugin: 'org.springframework.boot'  group = 'com.tutorialspoint'  version = '0.0.1-SNAPSHOT'  sourceCompatibility = 1.8  repositories {  mavenCentral()  }  dependencies {  compile('org.flywaydb:flyway-core')  compile('org.springframework.boot:spring-boot-starter-jdbc')  compile('org.springframework.boot:spring-boot-starter-web')  compile('mysql:mysql-connector-java')  testCompile('org.springframework.boot:spring-boot-starter-test')  } |

4.3. Kết nối dữ liệu và chạy ứng dụng Spring Boot

Bạn có thể tạo một tệp JAR có thể chạy được và chạy ứng dụng Spring Boot bằng cách sử dụng các lệnh Maven hoặc Gradle như sau:

Đối với Maven, bạn có thể sử dụng lệnh sau đây:

mvn clean install

Sau khi nhận được thông báo "BUILD SUCCESS", bạn có thể tìm thấy tệp JAR trong thư mục target.

Đối với Gradle, bạn có thể sử dụng lệnh sau đây:

gradle clean build

Sau khi nhận được thông báo "BUILD SUCCESSFUL", bạn có thể tìm thấy tệp JAR trong thư mục build/libs.

Bây giờ, chạy tệp JAR bằng lệnh sau:

java –jar <JARFILE>

Bây giờ, Tomcat đã bắt đầu trên cổng 8080 và bạn có thể thấy các bản ghi nhật ký của cơ sở dữ liệu flyway trong cửa sổ console.

TÀI LIỆU THAM KHẢO

[1]. Sourabh Sharma, Modern API Development with Spring and Spring Boot, 2020

[2]. https://www.javatpoint.com/spring-boot-tutorial