EIGHT WEEKS SUMMER TRAINING REPORT

On

Java Backend Development in Spring Boot

Submitted by

Nitin Padal

Registration No.12217618

Program Name: B.Tech CSE

Under the Guidance of

Mr. Anoop Garg

School of Computer Science & Engineering Lovely professional University,

Phagwara

(May-July, 2024)

DECLARATION

I hereby declare that I have completed my eight weeks summer training on **Java Backend Development in Spring Boot** at **Program by Programming Pathshala** from 15th May 2024 to 20th July 2024 under the guidance of Mr.Anoop Garg. I have declare that I have worked with full dedication during these eight weeks of training and my learning outcomes fulfil the requirements of training for the award of degree of Bachelor of Technology, Lovely Professional university, Phagwara.

Name of Student: Nitin Padal

Registration no: 12217618

ACKNOLEDGEMENT

It is with sense of gratitude; I acknowledge the efforts of entire hosts of well-wishers who have in some way or other contributed in their own special ways to the success and completion of the Summer Training.

Successfully completion of any type technology requires helps from a number of people. I have also taken help from different people for the preparation of the report. Now, there is little efforts to show my deep gratitude to those helpful people.

I would like to also thank **Program by Programming Pathshala** and my own college Lovely Professional University for offering such a course which not only improve my programming skill but also taught me other new technology.

Then I would like to thank my parents and friends who have helped me with their valuable suggestions and guidance for choosing this course.

Last but not least I would like to thank my all classmates who have helped me a lot.

Training certificate from organization

Programming Pathshala
669ba48675386515a8255966

Kindly verify Programming Pathshala's document on Blockchain by scanning the QR code.
https://certificates.programmingpathshala.com/docs/d3667e1ec1bf557999ee195cb135b3544d06a705fdc0cca120e4488a409334e8

TABLE OF CONTENTS

INTRODUCTION

We followed a structured methodology for our 8 Weeks Summer training Program by Programming Pathshala for Java backend development in SpringBoot

Which starts from Core Java and OOPS Fundamentals to Advanced Java Concepts, Maven: build automation tool, Core Spring Boot Concepts

And at the end prepared a Well planned mini Project, The goal of this project is to build a powerful REST API that handles student data management. The API will enable creating, reading, updating, and deleting student records, with an interactive HTML front-end for testing and verification. Any changes made through the API will be instantly reflected on the web page.

With the idea of imparting programming knowledge**, Mr. Anoop Garg** who taught 10,000+ students through on campus workshops and online courses on Interview Preparation. He has worked at Amazon with its High Scale Systems for three years after graduating from IIT. He is loved by his students for his lucid in-depth explanations and ability to make people think through problems.

Programming Pathshala is an organisation focused on making high quality Tech Education accessible, and helping anyone who wants to make it big in tech - get where they want to be. With a 4.9/5Google rating, our IIT trained mentors have taught 20,000+ students, and enabled them to be high in demand tech professionals.

1. Core Java and OOPS Fundamentals

- **Classes and Objects:**
    - Class: Blueprint for creating objects.
    - Object: Instance of a class with attributes and behavior.
- **Inheritance:**
    - One class inherits properties and behaviors from another class (parent-child relationship).
- **Polymorphism:**
    - Ability to take many forms:
        - **Compile-time polymorphism:** Method overloading.
        - **Runtime polymorphism:** Method overriding.
- **Abstraction:**
    - Hides implementation details and exposes only functionality through abstract classes or interfaces.
- **Encapsulation:**
    - Bundling of data and methods in a class with restricted access using access modifiers (**private**, **public**, **protected**).
- **Abstract Classes and Interfaces:**
    - **Abstract Class:** Contains abstract methods; cannot be instantiated directly.
    - **Interface:** Defines a contract with abstract methods that implementing classes must follow.

2. Advanced Java Concepts

- **Exception Handling with Try, Catch, and Finally:**
    - **Try Block:** Contains code that might throw an exception.
    - **Catch Block:** Catches and handles exceptions.
    - **Finally Block:** Always executed (for cleanup), regardless of exception occurrence.

3.Maven: Simplifying Java Development

Maven is a powerful build automation tool primarily used for Java projects. It simplifies project setup, managing dependencies, and automating the build process.

What is Maven?

Maven is a build automation tool used to manage the entire lifecycle of a Java project. It automates tasks such as compiling code, running tests, packaging, and deployment.

Maven Project Object Model (POM)

The POM (Project Object Model) is an XML file that contains information about the project and its configuration, such as dependencies, plugins, and build configurations. It helps manage project builds, reporting, and documentation.

Managing Dependencies with Maven

Maven uses a repository-based dependency management system, where you can declare libraries (dependencies) in the POM file. Maven automatically downloads and manages these dependencies, ensuring compatibility.

Running Maven Builds from the IDE

Maven can be integrated into most popular IDEs (e.g., IntelliJ IDEA, Eclipse), allowing developers to run builds, tests, and other Maven goals directly from the IDE interface.

Maven for Spring Boot MVC, Security, and REST API

Maven simplifies setting up Spring Boot applications. By declaring Spring Boot starters (like **spring-boot-starter-web** or **spring-boot-starter-security**) in the POM, Maven resolves and downloads the required dependencies to build MVC, security, and REST API features effortlessly.

Maven and Hibernate/JPA Integration

Maven facilitates the integration of Hibernate and JPA by managing dependencies like **hibernate-core** or **spring-boot-starter-data-jpa** through the POM file. This helps in building applications that use ORM for database interactions.
Maven's main purpose is to streamline project management, dependency handling, and automated build processes, making Java development more efficient.

4.Spring Boot

Spring Boot is built upon the Spring Framework, a comprehensive platform for Java development. Spring Boot offers a streamlined approach to creating standalone, production-ready Spring applications with minimal configuration. Key features include:

Auto-configuration: Spring Boot automatically configures common application settings, reducing the amount of boilerplate code needed. This speeds up development and simplifies the setup process.

Embedded Servers: Spring Boot integrates popular web servers like Tomcat, Jetty, and Undertow directly into your application, eliminating the need for separate server installations.

Starter Dependencies: Spring Boot provides curated starter dependencies that include essential libraries for various functionalities like web development, data access, security, and more.

Opinionated Defaults: Spring Boot sets up sensible defaults for various configurations, promoting best practices and consistency across projects.

Command Line Interface (CLI): Spring Boot offers a command-line interface for creating, running, and managing applications with ease.

Health Checks: Spring Boot provides built-in health checks to monitor the health of your application and identify potential issues.

Actuator Endpoints: Spring Boot Actuator exposes endpoints that provide valuable insights into the runtime behavior of your application, such as metrics, logs, and environment details.

Spring Boot Architecture

The **Spring Boot** is built on top of the core Spring framework. It is a simplified and automated version of the spring framework. The spring boot follows a **layered architecture** in which each layer communicates to other layers (Above or below in hierarchical order). The spring boot documentation provides the following definition to the Spring Boot Framework.

Spring Boot makes it easy to create stand-alone, production-grade Spring based application that you can "Just Run"

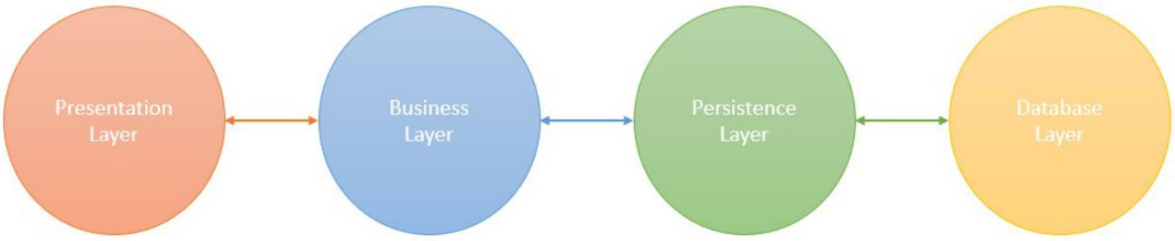
The main aim of spring boot is to remove the XML and annotations-based configuration settings from the application. Along with this spring boot provides the following benefits such as opinionated (options to later change the configuration), convention over configuration, stand-alone, and production-ready.

Spring Boot Layers

The spring boot consists of the following four layers:

1. **Presentation Layer** – Authentication & Json Translation
2. **Business Layer** – Business Logic, Validation & Authorization
3. **Persistence Layer** – Storage Logic

4. **Database Layer** – Actual Database



1. Presentation Layer
The presentation layer is the top layer of the spring boot architecture. It consists of Views. i.e., the front-end part of the application. It handles the HTTP requests and performs authentication. It is responsible for converting the JSON field's parameter to Java Objects and vice-versa. Once it performs the authentication of the request it passes it to the next layer. i.e., the business layer.

2. Business Layer
The business layer contains all the business logic. It consists of services classes. It is responsible for validation and authorization.

3. Persistence Layer
The persistence layer contains all the database storage logic. It is responsible for converting business objects to the database row and vice-versa.

4. Database Layer
The database layer contains all the databases such as MySql, H2 DataBase etc. This layer can contain multiple databases. It is responsible for performing the CURD operations.

Understanding the Spring Boot Architecture

Spring Boot is built upon the foundation of the Spring Framework, a powerful and comprehensive Java framework. Spring Boot leverages several key components to achieve its simplicity and efficiency. These include:

- Dependency Injection (DI): Spring Boot uses DI to manage dependencies between different components of your application. This promotes loose coupling and makes your code more modular and maintainable.

- Auto-configuration: Spring Boot provides intelligent auto-configuration, automatically configuring your application based on the dependencies you include. This reduces boilerplate code and streamlines development.
- Spring Data JPA: This component simplifies database interactions, allowing you to easily work with persistence layers and interact with databases like MySQL or PostgreSQL.
- Spring Security: Spring Boot integrates seamlessly with Spring Security, providing robust security features for your web applications, such as user authentication and authorization.

By understanding these core components, you'll grasp the principles behind Spring Boot's architecture and be better equipped to build scalable and reliable applications.

Setting up the Development Environment

Before you begin your journey with Spring Boot, it's crucial to have a solid development environment in place. This involves installing the necessary tools and configuring your system to support Spring Boot development. Here's what you'll need:

- Java Development Kit (JDK): Download and install the latest JDK from Oracle or AdoptOpenJDK.
- Integrated Development Environment (IDE): Popular choices include IntelliJ IDEA, Eclipse, or VS Code. Choose an IDE that suits your preferences and provides good support for Java and Spring Boot.
- Maven or Gradle: These are build tools essential for managing dependencies and compiling your Spring Boot applications.

Once you've installed the necessary tools, you can set up a new Spring Boot project using Spring Initializr, a web-based tool that generates a basic project structure with essential dependencies. This simplifies the setup process and allows you to start coding quickly.

Spring Boot Starters and Dependency Management

Spring Boot Starters are pre-configured dependency sets designed for specific functionalities. These starters provide a convenient way to include common

dependencies without manually configuring each one. They reduce the complexity of dependency management and ensure compatibility between dependencies.

Some popular starters include: spring-boot-starter-web for web applications, spring-boot-starter-data-jpa for database integration, spring-boot-starter-security for security features, and spring-boot-starter-test for testing. Starters simplify dependency management and reduce boilerplate code, allowing developers to focus on core application logic.

Spring Boot Configuration and Properties

Spring Boot provides a flexible and extensible configuration system using application properties. These properties define application settings and configurations, including database connections, server ports, and logging levels. Spring Boot supports multiple configuration sources, including application.properties, application.yml, and environment variables.

Spring Boot automatically scans for configuration files and loads properties based on their priority. You can customize the default behavior by specifying specific profiles or using custom configuration files. This allows you to tailor your application configuration to different environments, such as development, testing, and production.

Spring Boot Annotations and Conventions

Spring Boot leverages annotations to simplify configuration and reduce boilerplate code. Annotations are used to define components, controllers, services, and other Spring-managed beans. They provide a declarative way to configure application behavior, making code more concise and readable. Spring Boot follows convention-over-configuration, where sensible defaults are applied to reduce the need for explicit configuration. It provides auto-configuration for common dependencies and components, eliminating the need for manual configuration in many cases. This convention-based approach streamlines development and reduces the effort required to set up and run a Spring Boot application.

Setting up a Spring Boot Project

Setting up a Spring Boot project is straightforward, thanks to the Spring Initializr tool. This web-based tool allows you to generate a basic Spring Boot project with your desired dependencies. Follow these steps:

1. Go to the Spring Initializr website (https://start.spring.io/).
2. Choose your project dependencies, including Spring Web for web development, Spring Data JPA for database integration, and Spring Security for security.
3. Select your preferred language (Java) and build tool (Maven or Gradle).
4. Click "Generate" to download the project as a ZIP file.
5. Unzip the file, open the project directory in your IDE, and build the project using the build tool you selected.

You'll have a basic Spring Boot project ready to start building your application.

Creating a Basic Spring Boot Application

Once you have a Spring Boot project, you can create a simple "Hello World" application to test your setup. Here's how:

1. Create a new Java class, for example, `HelloWorldController`.
2. Annotate the class with `@RestController` to indicate it's a REST controller. This annotation automatically configures the controller to handle HTTP requests.
3. Create a method annotated with `@GetMapping` to handle HTTP GET requests. The method should return a String, for example, "Hello World!".
4. Run your Spring Boot application, and you should be able to access the "Hello World!" message by accessing the root endpoint in your browser (e.g., `http://localhost:8080`).

This demonstrates the ease of creating a basic web application with Spring Boot.

Handling HTTP Requests with Controllers

Spring Boot controllers are the heart of your application's web interface. They handle incoming HTTP requests, process them, and return responses. Here's a breakdown of common concepts:

- Annotations: `@RestController`, `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping` - These annotations define which HTTP methods (GET, POST, PUT, DELETE) a controller method handles. You can use them to map specific URLs to methods.
- Request Parameters: Controller methods can accept request parameters using `@RequestParam`. This allows you to retrieve values from the request URL or body.
- Response Bodies: Controller methods return data that will be sent back to the client. You can return plain text, JSON objects, or other data formats.
- Path Variables: `@PathVariable` allows you to extract values from the request URL path. For example, you could use it to retrieve a user ID from a URL like `/users/{userId}`.

By utilizing these concepts, you can create controllers that efficiently manage HTTP requests and responses in your Spring Boot application.

Handling HTTP Requests and Responses

Spring Boot provides a powerful mechanism for handling HTTP requests and responses. It uses annotations to define how different HTTP methods, like GET, POST, PUT, and DELETE, are mapped to specific controller methods.
For example, you can create a controller method with the @PostMapping annotation to handle requests sent using the POST method. You can access the data sent in the request body using the @RequestBody annotation.
To send a response, you can use the ResponseEntity class, which allows you to control the status code, headers, and body of the response. Spring Boot automatically handles serialization and deserialization of data to and from JSON or XML formats, making it easy to work with RESTful APIs.

Method	Annotation	Description
GET	@GetMapping	Retrieves data from the server
POST	@PostMapping	Sends data to the server
PUT	@PutMapping	Updates existing data on the server
DELETE	@DeleteMapping	Deletes data from the server

Integrating a Database with Spring Boot

Spring Boot simplifies database integration through Spring Data JPA, an abstraction layer over JPA (Java Persistence API). This allows you to work with databases in a more object-oriented way, simplifying data access and manipulation.

- Entity Classes: Create Java classes that represent your database tables. Annotate these classes with `@Entity` and map fields to database columns using annotations like `@Id`, `@Column`, and `@GeneratedValue`.
- Repositories: Create interfaces that extend `JpaRepository` to access data from your entities. The interface provides methods for common operations like `findAll()`, `findById()`, `save()`, and `delete()`.
- Database Configuration: Spring Boot automatically configures the database connection using properties in your application's `application.properties` file. You can specify the database type, username, password, and other details.
- Data Access: Inject your repositories into other classes to access database data. Spring Boot will handle the underlying connection and transaction management for you.

Spring Data JPA allows you to interact with databases seamlessly, making it easy to persist, retrieve, and manage data in your Spring Boot application.

What is APIs and REST APIs

An API (Application Programming Interface) is a set of rules and specifications that define how software components interact with each other. REST (Representational State Transfer) is an architectural style for building APIs that focuses on using standard HTTP methods (GET, POST, PUT, DELETE) and common data formats like JSON to interact with web services.

REST APIs are widely used in modern web applications for communication between client applications and backend services. They allow developers to expose data and functionalities from their systems to other applications, enabling seamless integration and data sharing.

Here are some real-life examples of REST APIs:

Social Media Platforms: APIs allow developers to integrate features from social media platforms like Facebook, Twitter, or Instagram into their applications. For example, you can use a Twitter API to display tweets on your website.

E-commerce: APIs enable developers to integrate shopping cart functionalities, payment gateways, and shipping services into their applications. For example, you can use an API from a payment gateway to process online payments in your e-commerce store.

Mapping Services: APIs from Google Maps or Mapbox allow developers to embed interactive maps, calculate routes, and display location information within their applications.

Weather Services: APIs from weather providers like OpenWeatherMap or Weather Underground allow developers to retrieve real-time weather data for specific locations and display it on their applications.

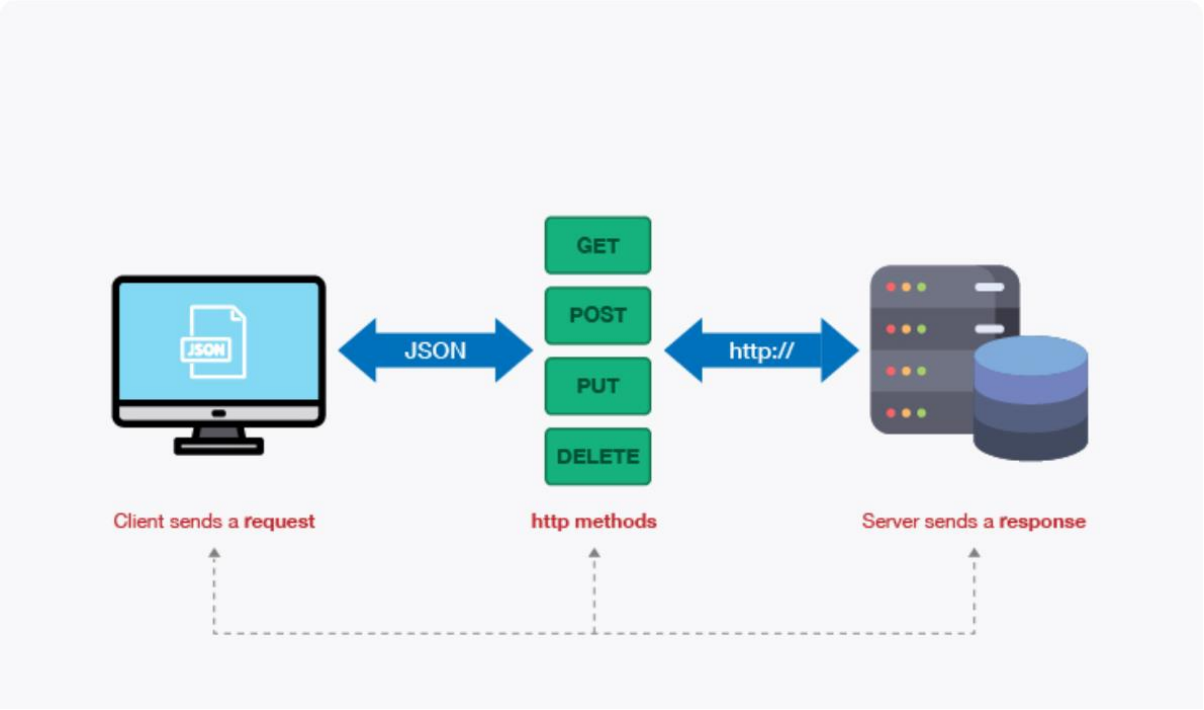
Introduction to RESTful Web Services

RESTful Web Services REST stands for **REpresentational State Transfer**. It was developed by **Roy Thomas Fielding**, one of the principal authors of the web protocol HTTP. Consequently, REST was an **architectural approach** designed to make the optimum use of **HTTP protocol**. It uses the concepts and verbs already present in HTTP to develop web services. This made REST incredibly easy to use and consume, so much so that it is the go-to standard for building web services today.

A resource can be anything, it can be accessed through a **URI (Uniform Resource Identifier)**. Unlike SOAP, REST does not have a standard messaging format. We can build REST web services using many representations, including both XML and JSON, although JSON is the more popular option. An important thing to consider is that REST is not a standard but a style whose purpose is to constrain our architecture to a client-server architecture and is designed to use stateless communication protocols like HTTP.

Important Methods of HTTP

The main methods of HTTP we build web services for are:



1. **GET**: Reads an existing data.
2. **PUT**: Updates existing data.

3. **POST**: Creates new data.
4. **DELETE**: Deletes the data.

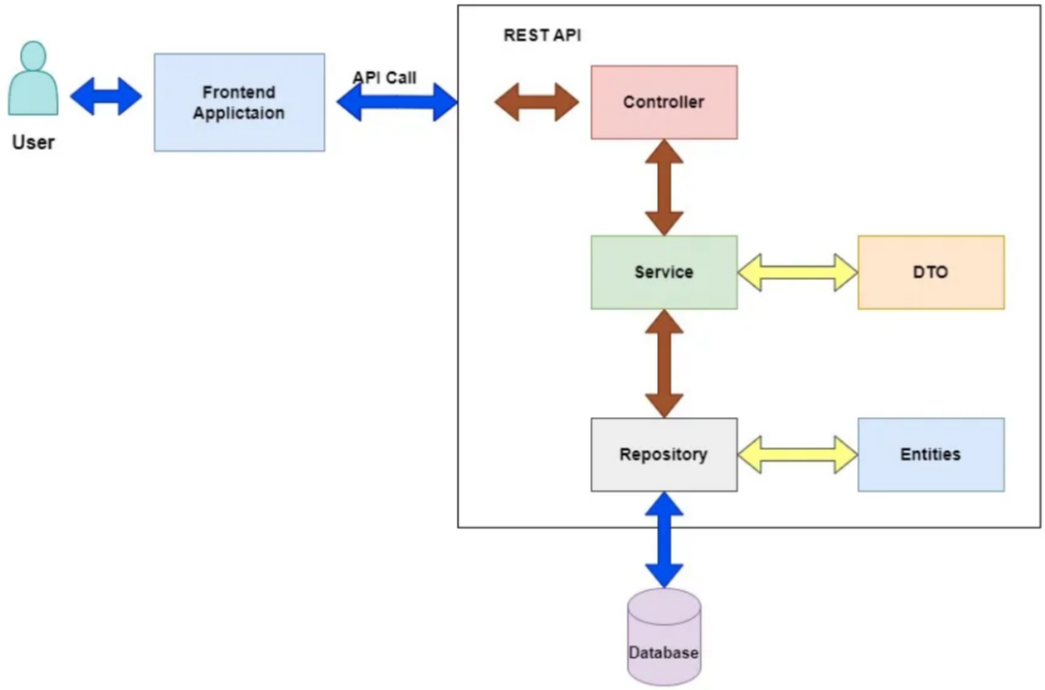
How APIs are Built using Spring Boot

Spring Boot provides an excellent framework for building REST APIs. By combining its features with Spring MVC, you can easily create well-structured and maintainable APIs.

Here are the key aspects of building REST APIs with Spring Boot:

- **Controllers:** Controllers act as entry points for handling HTTP requests. You can annotate methods with `@GetMapping`, `@PostMapping`, `@PutMapping`, and `@DeleteMapping` to map specific URLs to methods.
- **Request and Response Handling:** Spring Boot automatically handles request mapping, parameter binding, and response serialization, making it easy to work with HTTP requests and responses.
- **Data Serialization**: Spring Boot uses Jackson by default for JSON serialization and deserialization, enabling you to easily exchange data with clients. You can configure Jackson to customize serialization behaviors.
- **Validation:** You can use annotations like `@Valid` and validation constraints (e.g., `@NotNull`, `@Size`) to validate incoming data, ensuring data integrity and consistency.
- **Exception Handling:** Spring Boot provides mechanisms for handling exceptions gracefully. You can define custom exception handlers to provide informative error messages to clients.
- **Documentation:** Spring Boot integrates with tools like Swagger or SpringDoc to automatically generate API documentation, making it easier for developers to understand and use your API.

The architecture of the Spring Boot REST applications



Let's think the user has made an API call via the front-end application**(e.g. Click a button to view the student details)**. Then front-end application initiates an API request and it hits the controller layer of the REST API.
From the controller layer forward the request to the Service layer. Then the service layer forwards the request to the repository layer.
The repository layer interacts with the database layer. Methods like findAll, findById and save are available in the repository class. When we request all student details, the service class invoke the findAll method in the repository class.
When retrieving and inserting data into the database by the repository layer, it needs a set of variables to store the data. We use Entity class for that. Entity class is mapped to a table in the database. If we retrieve the Student details from the database then we need an entity class for the Student to keep the student details.

Project description:

The goal of this project is to build a powerful REST API that handles student data management. The API will enable creating, reading, updating, and deleting student records, with an interactive HTML front-end for testing and verification. Any changes made through the API will be instantly reflected on the web page, allowing for a seamless integration between the backend logic and the frontend presentation.

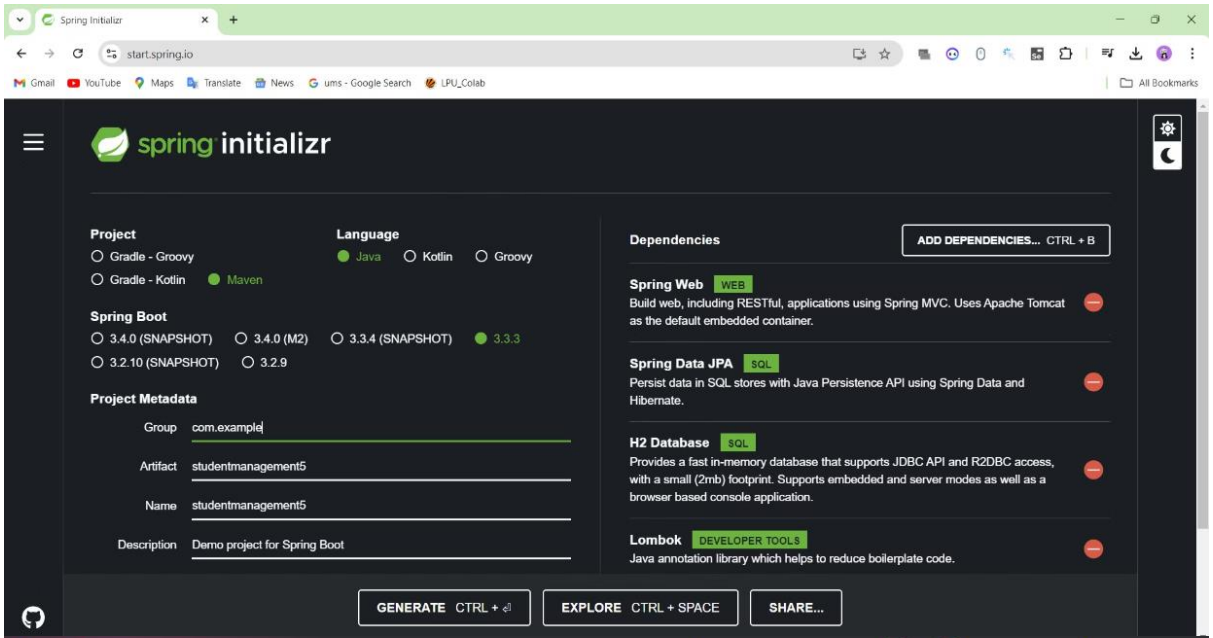
Project Setup
Initialize a Spring Boot project and configure the database connection, ensuring the right dependencies are included.

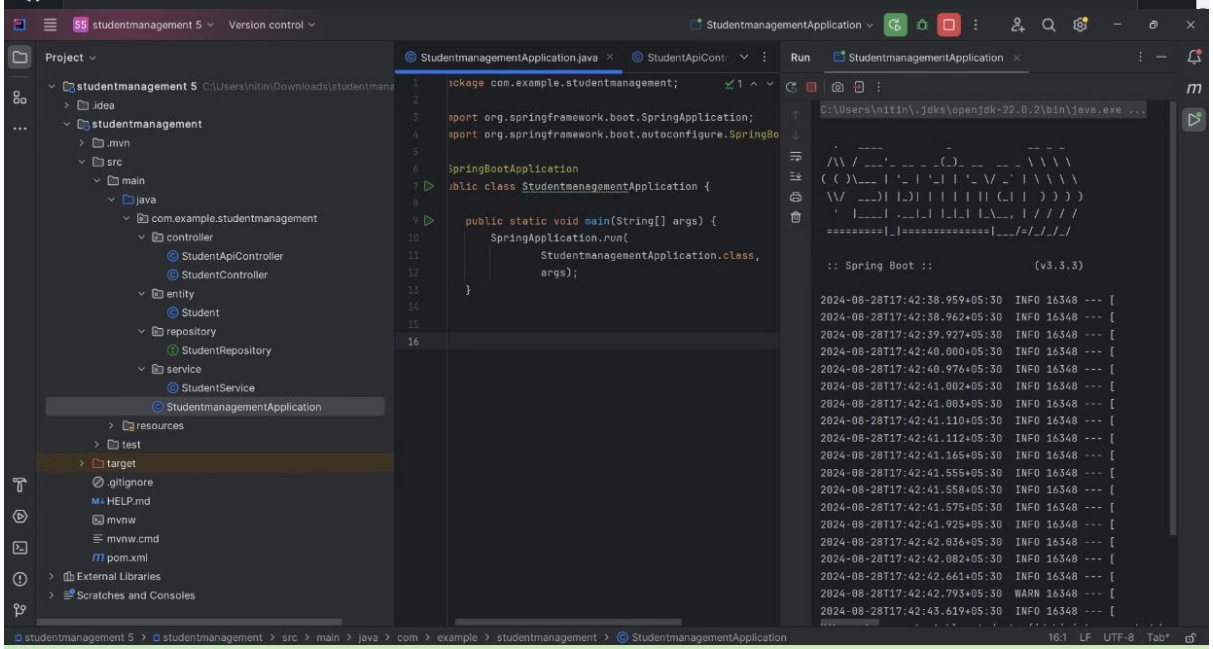
Model Design
Create a Student entity class that maps to the database, utilizing JPA annotations for defining the structure and relationships.

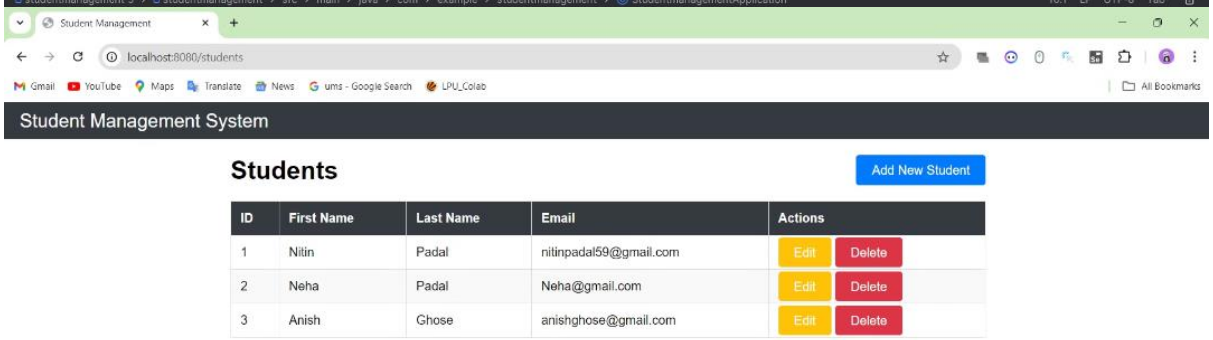
Repository Layer
Implement the CRUD operations for the Student entity using the JpaRepository interface, which provides convenient methods for interacting with the database.

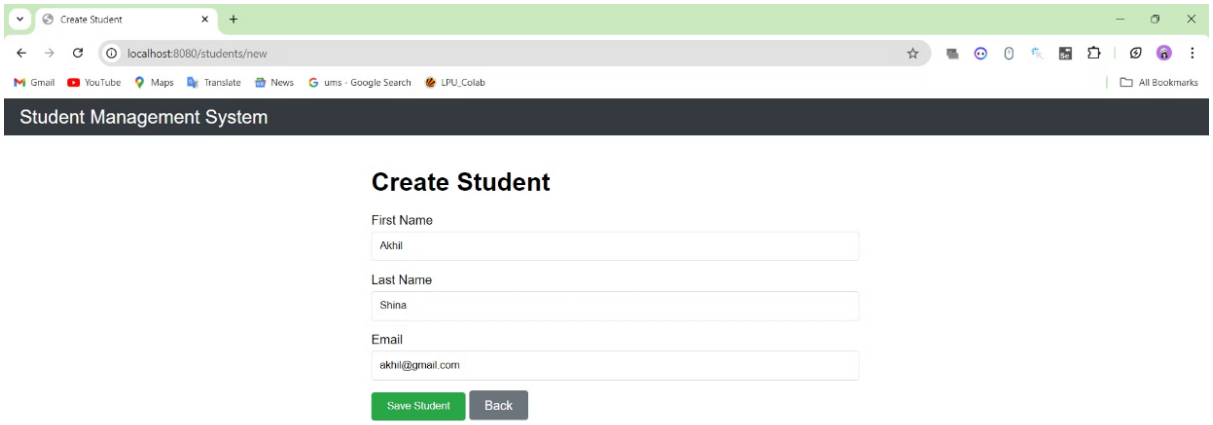
Service Layer
Develop the service layer, responsible for handling the business logic and performing complex operations related to student data management.

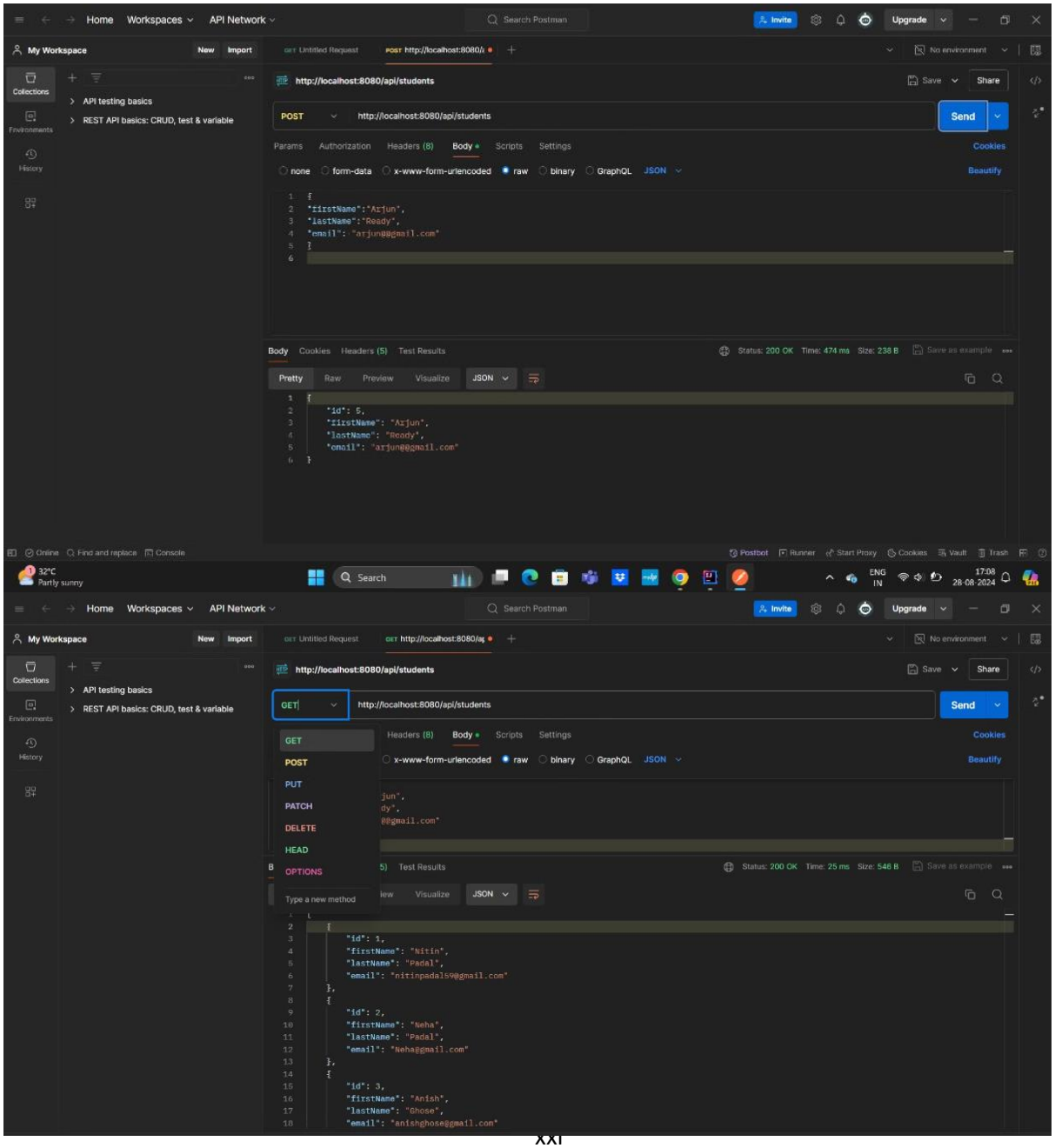
Controller Layer
Create RESTful endpoints using Spring MVC controllers, defining the routes and methods for accessing and managing student data through the API.

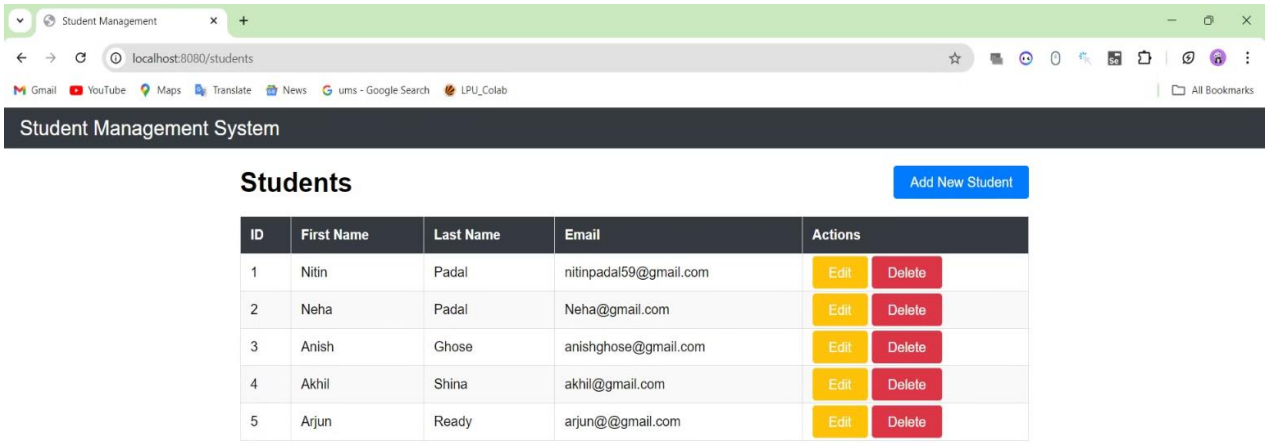

Spring Initializr
start.spring.io
spring initializr
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Language
Java
Kotlin
Groovy
Spring Boot
3.4.0 (SNAPSHOT)
3.4.0 (M2)
3.3.4 (SNAPSHOT)
3.3.3
3.2.10 (SNAPSHOT)
3.2.9
Project Metadata
Group com.example
Artifact studentmanagement5
Name studentmanagement5
Description Demo project for Spring Boot
Dependencies
ADD DEPENDENCIES... CTRL + B
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.
GENERATE CTRL + ⏎
EXPLORE CTRL + SPACE
SHARE...


studentmanagement 5
Version control
StudentmanagementApplication
Project
studentmanagement 5 C:\Users\nitin\Downloads\studentmana
.idea
studentmanagement
.mvn
src
main
java
com.example.studentmanagement
controller
StudentApiController
StudentController
entity
Student
repository
StudentRepository
service
StudentService
StudentmanagementApplication
resources
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
StudentmanagementApplication.java
StudentApiCont
ackage com.example.studentmanagement;
mport org.springframework.boot.SpringApplication;
mport org.springframework.boot.autoconfigure.SpringBo
SpringBootApplication
ublic class StudentmanagementApplication {
public static void main(String[] args) {
SpringApplication.run(
StudentmanagementApplication.class,
args);
}
Run
StudentmanagementApplication
C:\Users\nitin\.jdks\openjdk-22.0.2\bin\java.exe ...
:: Spring Boot :: (v3.3.3)
2024-08-28T17:42:38.959+05:30 INFO 16348 ---
2024-08-28T17:42:38.962+05:30 INFO 16348 ---
2024-08-28T17:42:39.927+05:30 INFO 16348 ---
2024-08-28T17:42:40.000+05:30 INFO 16348 ---
2024-08-28T17:42:40.976+05:30 INFO 16348 ---
2024-08-28T17:42:41.002+05:30 INFO 16348 ---
2024-08-28T17:42:41.003+05:30 INFO 16348 ---
2024-08-28T17:42:41.110+05:30 INFO 16348 ---
2024-08-28T17:42:41.112+05:30 INFO 16348 ---
2024-08-28T17:42:41.165+05:30 INFO 16348 ---
2024-08-28T17:42:41.555+05:30 INFO 16348 ---
2024-08-28T17:42:41.558+05:30 INFO 16348 ---
2024-08-28T17:42:41.575+05:30 INFO 16348 ---
2024-08-28T17:42:41.925+05:30 INFO 16348 ---
2024-08-28T17:42:42.036+05:30 INFO 16348 ---
2024-08-28T17:42:42.082+05:30 INFO 16348 ---
2024-08-28T17:42:42.661+05:30 INFO 16348 ---
2024-08-28T17:42:42.793+05:30 WARN 16348 ---
2024-08-28T17:42:43.619+05:30 INFO 16348 ---
16:1 LF UTF-8 Tab*


Student Management
localhost:8080/students
Student Management System
Students
Add New Student
ID
First Name
Last Name
Email
Actions
1 Nitin Padal nitinpadal59@gmail.com Edit Delete
2 Neha Padal Neha@gmail.com Edit Delete
3 Anish Ghose anishghose@gmail.com Edit Delete


Create Student
localhost:8080/students/new
Gmail
YouTube
Maps
Translate
News
ums - Google Search
LPU_Colab
All Bookmarks
Student Management System
Create Student
First Name
Akhil
Last Name
Shina
Email
akhil@gmail.com
Save Student
Back


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Environments
History
API testing basics
REST API basics: CRUD, test & variable
http://localhost:8080/api/students
POST
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"firstName":"Arjun",
"lastName":"Ready",
"email": "arjun@gmail.com"
}
Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Time: 474 ms
Size: 238 B
Pretty
Raw
Preview
Visualize
JSON
{
"id": 5,
"firstName": "Arjun",
"lastName": "Ready",
"email": "arjun@gmail.com"
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
32°C
Partly sunny
Search
ENG
IN
17:08
28-08-2024
GET
POST
PUT
PATCH
DELETE
HEAD
OPTIONS
Type a new method
Status: 200 OK
Time: 25 ms
Size: 546 B
[
{
"id": 1,
"firstName": "Nitin",
"lastName": "Padal",
"email": "nitinpadal59@gmail.com"
},
{
"id": 2,
"firstName": "Neha",
"lastName": "Padal",
"email": "Neha@gmail.com"
},
{
"id": 3,
"firstName": "Anish",
"lastName": "Ghose",
"email": "anishghose@gmail.com"



CONCLUSION

In summary, Spring Boot is a powerful framework that simplifies Java application development by providing default configurations and production-ready features. It helps developers focus on building the core functionality of their apps, rather than getting bogged down in low-level setup and infrastructure work.

By using Spring Boot, developers can create flexible and scalable applications that are easy to deploy and run in production environments. The framework's "just run" philosophy and layered architecture make it a great choice for building modern web applications, including powerful RESTful APIs.
Overall, Spring Boot is a valuable tool that can significantly streamline the Java development process and enable developers to build high-quality, enterprise-grade applications more efficiently.