INTRODUCTION TO SPRING BOOT PROJECT WITH MYSQL AND POSTMAN

TABLE OF CONTENTS:

1. OVERVIEW:

- This document provides an introduction to a Spring Boot project integrated with a MySQL database, along with using Postman for API testing.
- It offers a step-by-step guide for setting up the project and demonstrates how to connect to a database, create RESTful API endpoints, and test them with Postman.

2. INTRODUCTION:

This project serves as an introductory exploration into the world of Spring Boot, MySQL, and Postman, three essential components in modern web application development.

Purpose:

The purpose of this project is to provide an accessible and hands-on guide for those looking to embark on their journey in Spring Boot development. By the end of this project, you will have a foundational understanding of how to create a Spring Boot application, connect it to a MySQL database, and effectively test its API endpoints using Postman.

Key Components:

Spring Boot: Spring Boot simplifies the development of Java-based applications by providing a streamlined framework. You will learn how to set up a Spring Boot project.

MySQL Database: The integration of MySQL as a database management system will be explored, including creating tables, performing CRUD (Create, Read, Update, Delete) operations, and more.

Postman: Postman, a powerful API testing tool, will be used to interact with the API endpoints of your Spring Boot application. This ensures that your project is functioning as intended.

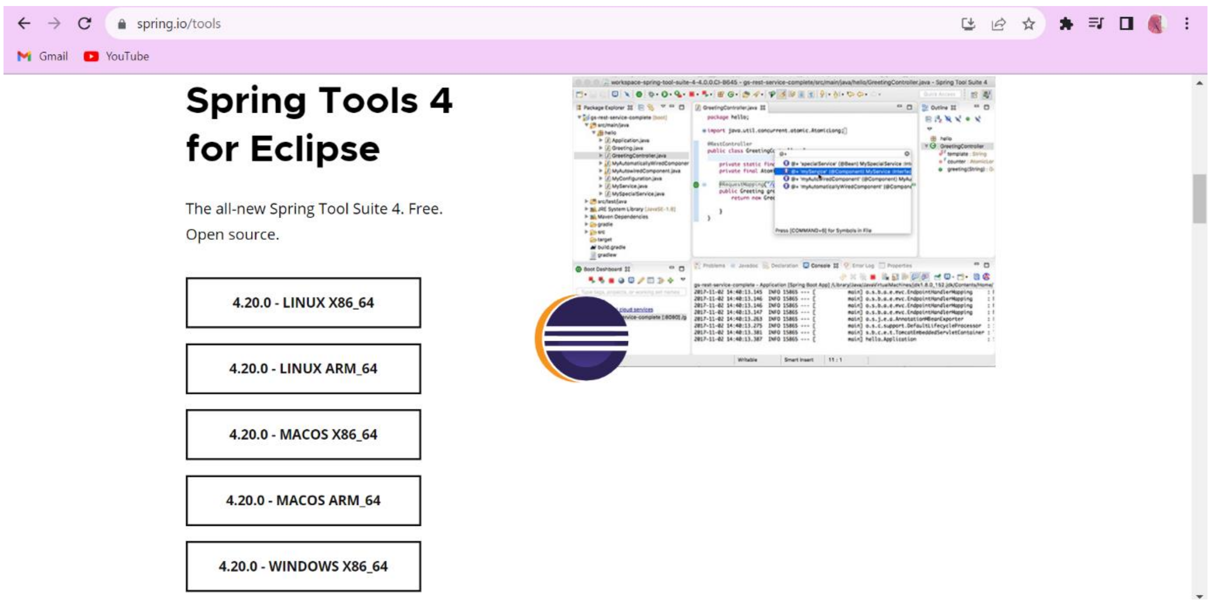
This project is for learners and developers seeking a practical understanding of these fundamental technologies. It will guide you through each step, from initial setup to hands-on application and testing. By the end of this project, you'll be equipped with the knowledge and skills needed to build your own Spring Boot projects, integrate databases, and ensure their functionality through comprehensive testing with Postman.

3. PREREQUISITES:

To get started with this project, you will need the following tools installed on your system:

a. Spring Tools Suite (STS): Spring Tools Suite is a specialized integrated development environment (IDE) designed for Spring-based applications. You can download and install it from the official website.

Website: https://spring.io/tools



spring.io/tools

Gmail YouTube

Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4. Free. Open source.

4.20.0 - LINUX X86_64

4.20.0 - LINUX ARM_64

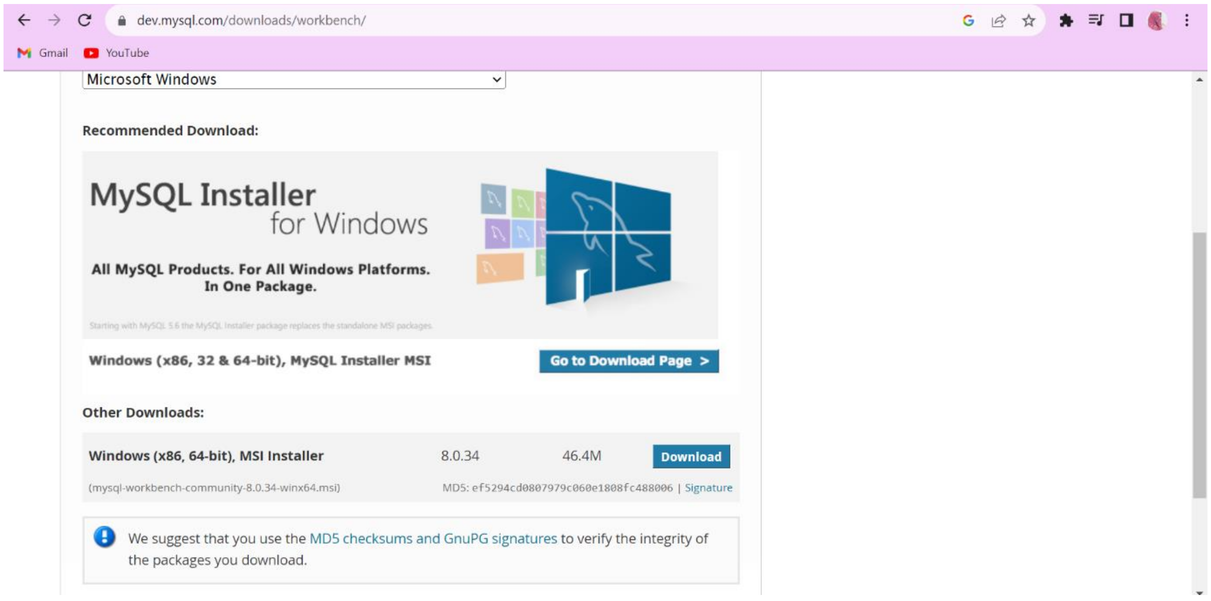
4.20.0 - MACOS X86_64

4.20.0 - MACOS ARM_64

4.20.0 - WINDOWS X86_64

b. MySQL Workbench: MySQL Workbench is a visual database design and management tool for MySQL. Install it to facilitate database management and interactions.

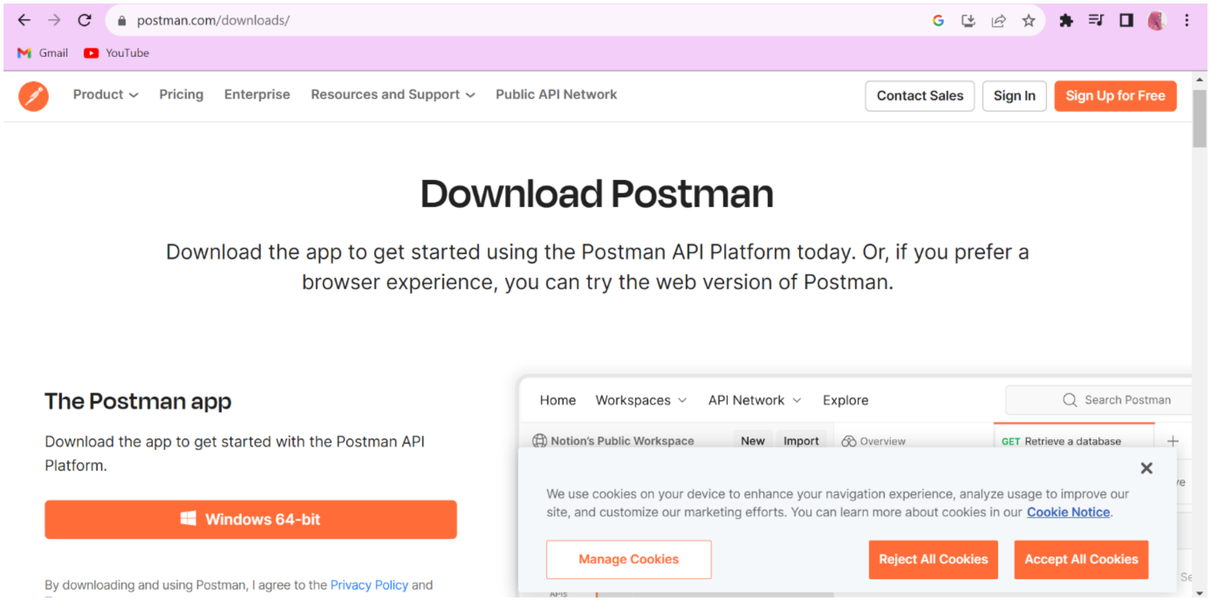
Website: https://dev.mysql.com/downloads/workbench/


dev.mysql.com/downloads/workbench/
Gmail
YouTube
Microsoft Windows
Recommended Download:
MySQL Installer for Windows
All MySQL Products. For All Windows Platforms. In One Package.
Windows (x86, 32 & 64-bit), MySQL Installer MSI
Go to Download Page >
Other Downloads:
Windows (x86, 64-bit), MSI Installer
8.0.34
46.4M
Download
(mysql-workbench-community-8.0.34-winx64.msi)
MD5: ef5294cd0807979c060e1808fc488006 | Signature

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

c. Postman: Postman is a powerful API testing tool used to send and receive HTTP requests. It will be essential for testing your Spring Boot application's API endpoints. Download and install Postman from the official website.

Website: https://www.postman.com/downloads/



Download Postman

Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

The Postman app

Download the app to get started with the Postman API Platform.

By downloading and using Postman, I agree to the Privacy Policy and

We use cookies on your device to enhance your navigation experience, analyze usage to improve our site, and customize our marketing efforts. You can learn more about cookies in our Cookie Notice.

Ensure that you have these tools installed and configured on your system before proceeding with the project. Having these tools in place will enable you to work seamlessly with Spring Boot, MySQL, and Postman.

4. STEP-BY-STEP GUIDE:

a. Setting Up Spring Boot Project:

- ➢ Download and Install Spring Tools Suite (STS). (Refer 3.a)
- ➢ Create a New Spring Boot Project using Spring Initializer.

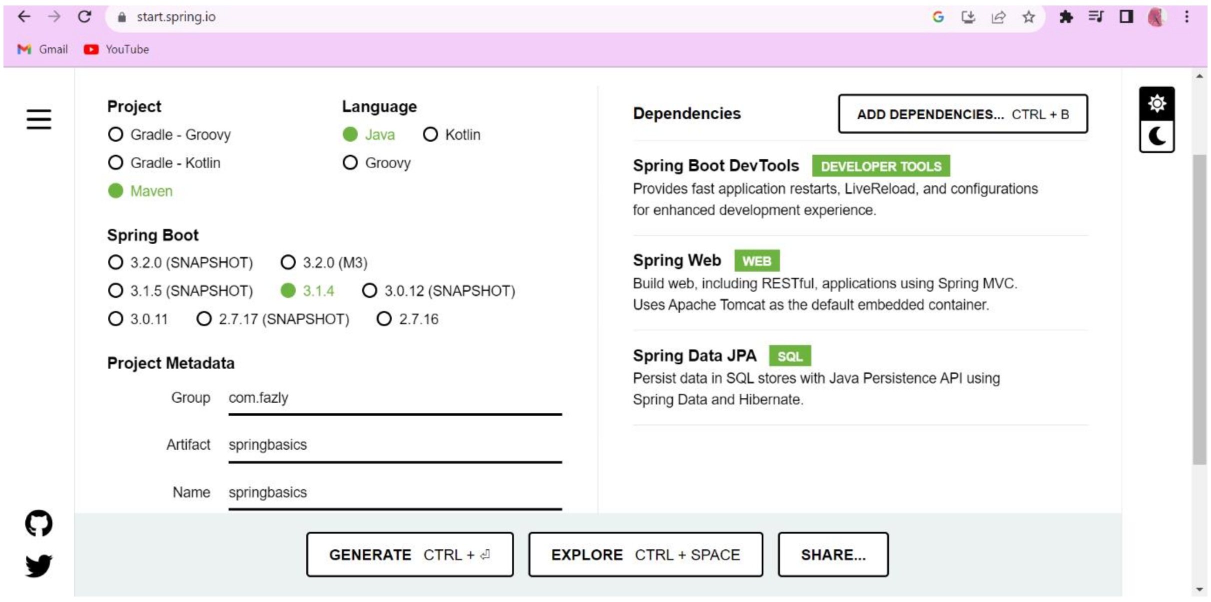
Website: https://start.spring.io/

Give Project-> Maven

- ➢ Choose Project Dependencies and Configuration.

For this project add dependencies:

- ✓ Spring Boot DevTools: For efficient development and automatic application restart.
- ✓ Spring Web: For building web applications.
- ✓ Spring Data JPA: For data access with JPA.Leave all the other as default.
- ➢ Explore the Project Structure.
- ➢ Give Generate.



Project
Gradle - Groovy
Gradle - Kotlin
[✓ Maven]

Language
[✓ Java] Kotlin
Groovy

Spring Boot
3.2.0 (SNAPSHOT) 3.2.0 (M3)
3.1.5 (SNAPSHOT) [✓ 3.1.4] 3.0.12 (SNAPSHOT)
3.0.11 2.7.17 (SNAPSHOT) 2.7.16

Project Metadata

Group	com.fazly
Artifact	springbasics
Name	springbasics

Dependencies

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

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.

- ➢ Import the generated project into Spring Tools Suite.
- ➢ Click "File" > "Import" > "Existing Maven Projects" to import the project into your IDE.
- ➢ STS will download the required dependencies and set up the project for you.

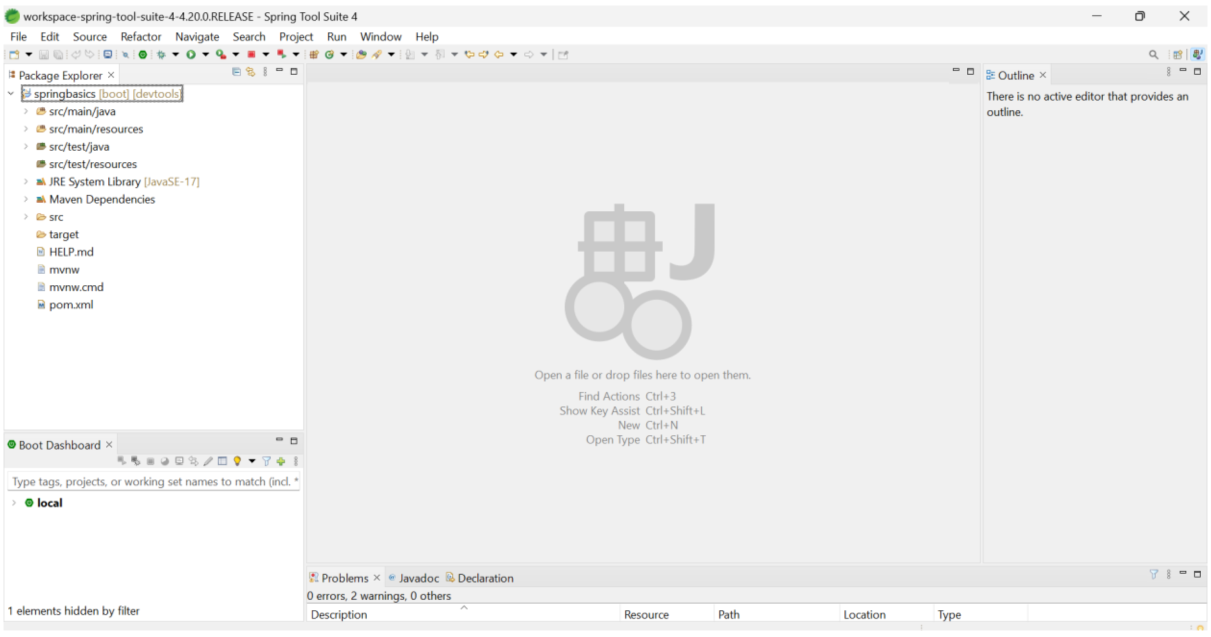
For me:

Group: com.fazly

Artifact: springbasics

Package: com.fazly.springbasics:

b. Creating Model Classes:


workspace-spring-tool-suite-4-4.20.0.RELEASE - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
springbasics [boot] [devtools]
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-17]
Maven Dependencies
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml
Outline
There is no active editor that provides an outline.
Open a file or drop files here to open them.
Find Actions Ctrl+3
Show Key Assist Ctrl+Shift+L
New Ctrl+N
Open Type Ctrl+Shift+T
Boot Dashboard
Type tags, projects, or working set names to match (incl. *
local
1 elements hidden by filter
Problems
Javadoc
Declaration
0 errors, 2 warnings, 0 others
Description
Resource
Path
Location
Type

- ➢ Open the project springbasics in Spring Tool Suite
- ➢ Locate default package com.fazly.springbasics inside src/main/java
- ➢ Inside src/main/java create a package com.fazly.springbasics.model to create model classes:
  - ✓ Users.java
  - ✓ Department.java
- ➢ Open each model class and add JPA annotations. For example, you might use @Entity to specify that the class is an entity, and @Table to define the table name.
- ➢ Inside each model class, define the fields and methods that represent the attributes and behavior of the data.
- ➢ User class may have fields for id, name, email, and password use @Id to specify primary key.
- ➢ Department class may have fields for id, name and section use @Id to specify primary key.
- ➢ Give constructors, getters and setters methods for both the Model Classes.

- ❖ The @GeneratedValue annotation with the strategy of GenerationType.UUID is used to specify how the primary key of an entity should be generated. In the context of JPA (Java Persistence API) and Hibernate, this annotation is often used to generate unique identifiers for database records.
- ❖ **@GeneratedValue:** This annotation marks a field in your entity class as an auto-generated value for the primary key. It indicates that the value of this field should be generated by the database or the JPA provider (e.g., Hibernate) and not explicitly set by the application.
- ❖ **strategy = GenerationType.UUID**: This part specifies the strategy for generating the primary key. In this case, UUID stands for universally unique identifier. A UUID is a 128-bit identifier that is guaranteed to be unique across time and space.

User.java:

```
package com.fazly.springbasics.model;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
@Entity
@Table(name="_users")
public class User {
	@Id
	@GeneratedValue(strategy=GenerationType.UUID)
	private String id;
	private String name;
	private String email;
	private String password;
	public User(String id, String name, String email, String password) {
		super();
		this.id = id;
		this.name = name;
		this.email = email;
		this.password = password;
	}
	public User() {
		super();
		// TODO Auto-generated constructor stub
	}
	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;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
}
```

Department.java:

```
package com.fazly.springbasics.model;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
@Entity
@Table(name="_departments")
public class Department {
	@Id
	@GeneratedValue(strategy=GenerationType.UUID)
	private String id;
	private String name;
	private String section;
	public Department(String id, String name, String section) {
		this.id = id;
		this.name = name;
		this.section = section;
	}
	public Department() {
		super();
		// TODO Auto-generated constructor stub
	}
	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;
	}
	public String getSection() {
		return section;
	}
	public void setSection(String section) {
		this.section = section;
	}
}
```

c. Configuring MySQL Database:

- **Create the MySQL Database Schema:** Use MySQL Workbench or any MySQL client to create the ‘springbasics’ schema in your MySQL database. You'll need to do this manually.
- **Open the application.properties File:** Locate the application.properties file in your Spring Boot project. This file is typically located in the src/main/resources directory.

application.properties:

```
# Server Configuration

server.port=8181

# MySQL Database Configuration

spring.datasource.url=jdbc:mysql://localhost:3306/springbasics

spring.datasource.username=root

spring.datasource.password=password

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate/JPA Configuration

spring.jpa.generate-ddl=true

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

- **Run the Application:** When you run your Spring Boot application, it will check if the tables corresponding to your entity classes exist in the springbasics schema. If not, it will create them automatically based on the JPA entity definitions and Hibernate settings.

d. Creating Repository Interfaces:

- Inside src/main/java create package com.fazly.springbasics.repository and in this package create interfaces called UserRepository and DepartmentRepository.
- Extend JpaRepository<EntityClass, ID> where EntityClass is the entity class (User or Department) and ID is the type of the entity's primary key (String).

UserRepository.java:

```
package com.fazly.springbasics.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.fazly.springbasics.model.User;
```

```
@Repository
public interface UserRepository extends JpaRepository<User,String>{
	Optional<User> findByEmail(String email);
}
```

DepartmentRepository.java:

```
package com.fazly.springbasics.repository;
import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.fazly.springbasics.model.Department;
@Repository
public interface DepartmentRepository extends JpaRepository<Department,String>{
	Optional<Department> findBySection(String section);
}
```

e. Creating Service Classes:

- Inside src/main/java create package com.fazly.springbasics.service and in this package create classes called UserService and DepartmentService.
- In your service class, add the @Service annotation to indicate that it's a Spring service component.
- Use @Autowired to inject the corresponding repository interface. This allows your service class to access data through the repository.
- Implement the business logic of your service class. This logic can include tasks such as data validation, data transformation, or any custom operations related to the entity you're working with.

UserService.java:

```
package com.fazly.springbasics.service;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.fazly.springbasics.model.User;
import com.fazly.springbasics.repository.UserRepository;
@Service
public class UserService {
	@Autowired
	UserRepository userRepository;
public String sayHello() {
	return "Hello";
}
public String saveUser(User user) {
        Optional<User> isUser=userRepository.findByEmail(user.getEmail());
        if(!isUser.isPresent()) {
                userRepository.save(user);
                return "User saved Successfully";/*In the Database rows are inserted*/
        }
        else{
                return "User already exists by email: "+user.getEmail();
        }
}
public List<User> getAllUsers(){
        return userRepository.findAll();
}
}
```

DepartmentService.java:

```
package com.fazly.springbasics.service;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.fazly.springbasics.model.Department;

import com.fazly.springbasics.repository.DepartmentRepository;

@Service

public class DepartmentService {

        @Autowired

        DepartmentRepository departmentRepository;

public String sayHello() {

        return "Hello";

}

public String saveDepartment(Department department) {

        Optional<Department>
isDepartment=departmentRepository.findBySection(department.getSection());

        if(!isDepartment.isPresent()) {

                departmentRepository.save(department);

                return "Department saved Successfully";

        }

        else{

                return "Department already exists by section: "+department.getSection();
        }
}
public List<Department> getAllDepartments(){
        return departmentRepository.findAll();
}
}
```

f. Creating Controller Classes:

- Inside src/main/java create package com.fazly.springbasics.controller and in this package create classes called UserController and DepartmentController.
- In your controller class, add the @RestController annotation to indicate that it's a Spring controller.
- Use @RequestMapping or other mapping annotations to define the request mappings (endpoints) that this controller will handle. These annotations specify the URL paths for the controller's methods.
- Use @Autowired to inject the corresponding service classes. This allows your controller to interact with the service layer.
- Create methods in your controller to handle specific HTTP requests (GET, POST, PUT, DELETE, etc.). These methods can use @GetMapping, @PostMapping, @PutMapping, or @DeleteMapping annotations to specify the HTTP method and URL path.

UserController.java:

```
package com.fazly.springbasics.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.fazly.springbasics.model.User;
import com.fazly.springbasics.service.UserService;
@RestController
@RequestMapping("/api/v1/user")
public class UserController {
        @Autowired
```

```
	UserService userService;
	@GetMapping("/hello")
	public String sayHello() {
		return userService.sayHello();
	}
	@GetMapping("/get")
	public List<User> getAllUsers(){
		return userService.getAllUsers();
	}
	@PostMapping("/save")
	public String saveUser(@RequestBody User user) {
		return userService.saveUser(user);
	}
}
```

DepartmentController.java:

```
package com.fazly.springbasics.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.fazly.springbasics.model.Department;
import com.fazly.springbasics.service.DepartmentService;
@RestController
@RequestMapping("/api/v1/department")
public class DepartmentController {
	@Autowired
	DepartmentService departmentService;
	@GetMapping("/hello")
	public String sayHello() {
```

```
                return departmentService.sayHello();
        }
        @GetMapping("/get")
        public List<Department> getAllDepartments(){
                return departmentService.getAllDepartments();
        }
        @PostMapping("/save")
        public String saveUser(@RequestBody Department department) {
                return departmentService.saveDepartment(department);
        }
}
```

g. Testing with Postman:

- Launch Postman from your computer. Create your own account.
- In Postman, click the "New" button to create a new request.
- Give your request a name to help identify it.
- Choose the appropriate HTTP method (GET, POST, PUT, DELETE, etc.) based on the endpoint you want to test.
- In the request URL field, enter the URL of the endpoint you want to test. For example, if you want to test a GET request to retrieve all users, enter the URL like http://localhost:8181/api/v1/user/get.

For Example,

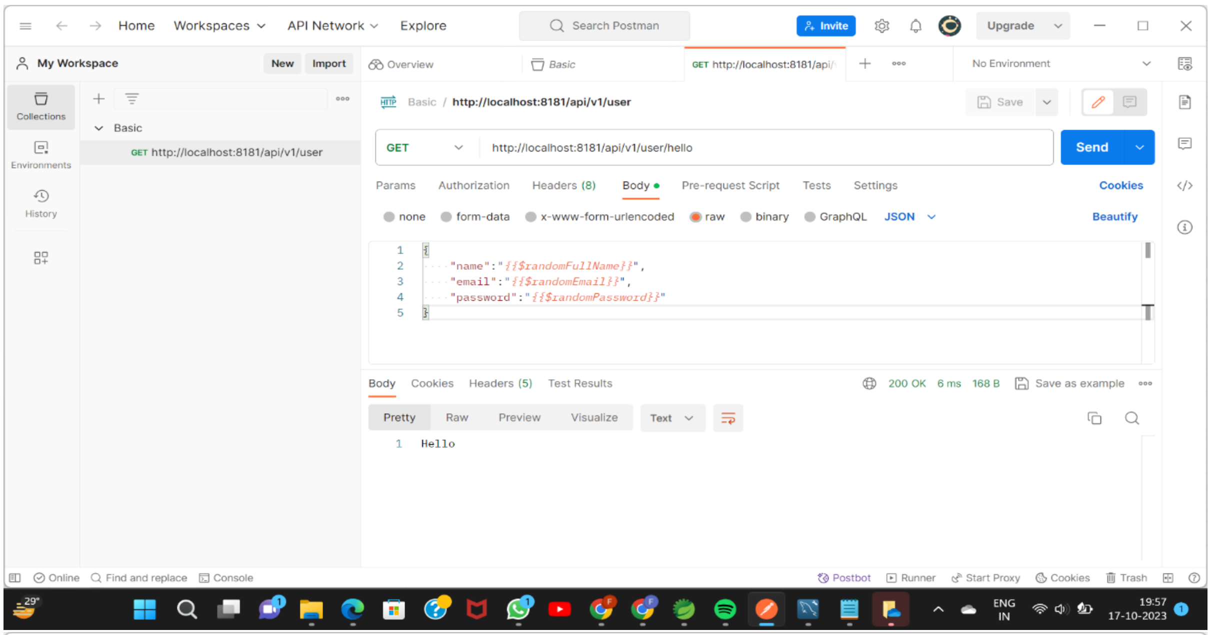
1. POST http://localhost:8181/api/v1/user/hello
2. GET http://localhost:8181/api/v1/user/get

```
{
    "name":"{{$randomFullName}}",
    "email":"{{$randomEmail}}",
    "password":"{{$randomPassword}}"
 }
```

3. POST http://localhost:8181/api/v1/user/save

To execute these, give Send Request.

Then for Department also follow the same.



GET http://localhost:8181/api/v1/user

Basic / http://localhost:8181/api/v1/user

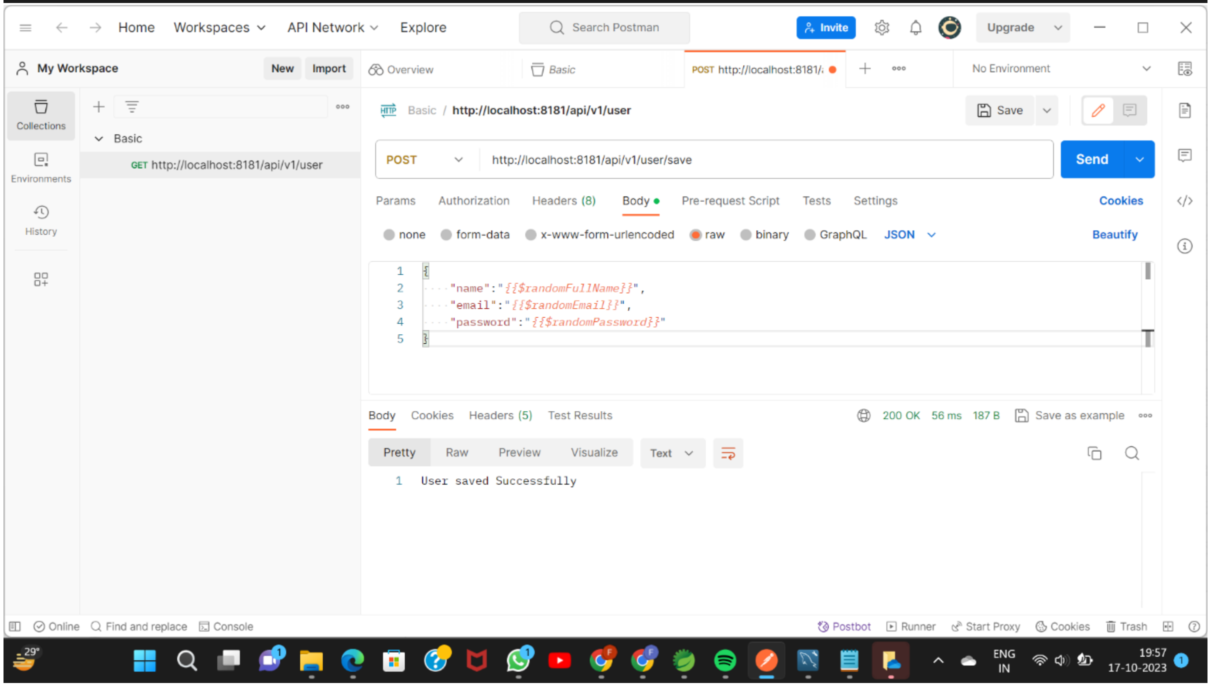
GET http://localhost:8181/api/v1/user/hello

Body: raw, JSON

```
{
  "name":"{{$randomFullName}}",
  "email":"{{$randomEmail}}",
  "password":"{{$randomPassword}}"
}
```

Body | Cookies | Headers (5) | Test Results — 200 OK 6 ms 168 B

```
Hello
```



POST http://localhost:8181/api/v1/user/save

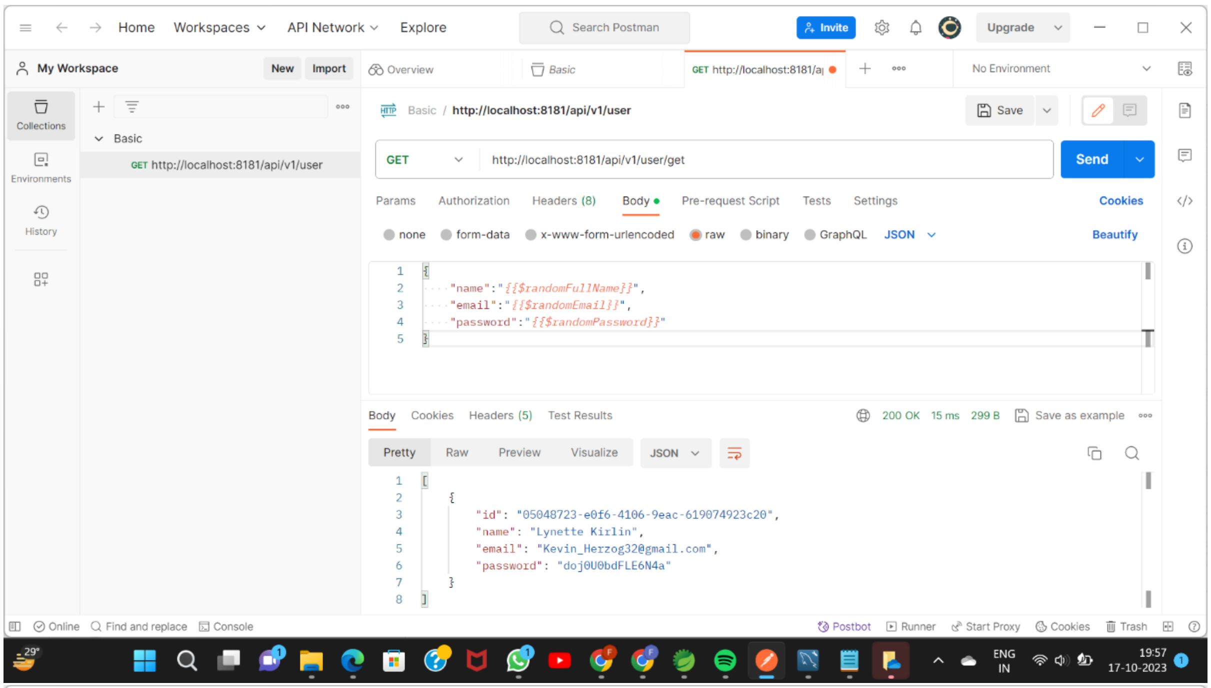
Basic / http://localhost:8181/api/v1/user

Body: raw, JSON

```
{
  "name":"{{$randomFullName}}",
  "email":"{{$randomEmail}}",
  "password":"{{$randomPassword}}"
}
```

Body | Cookies | Headers (5) | Test Results — 200 OK 56 ms 187 B

```
User saved Successfully
```



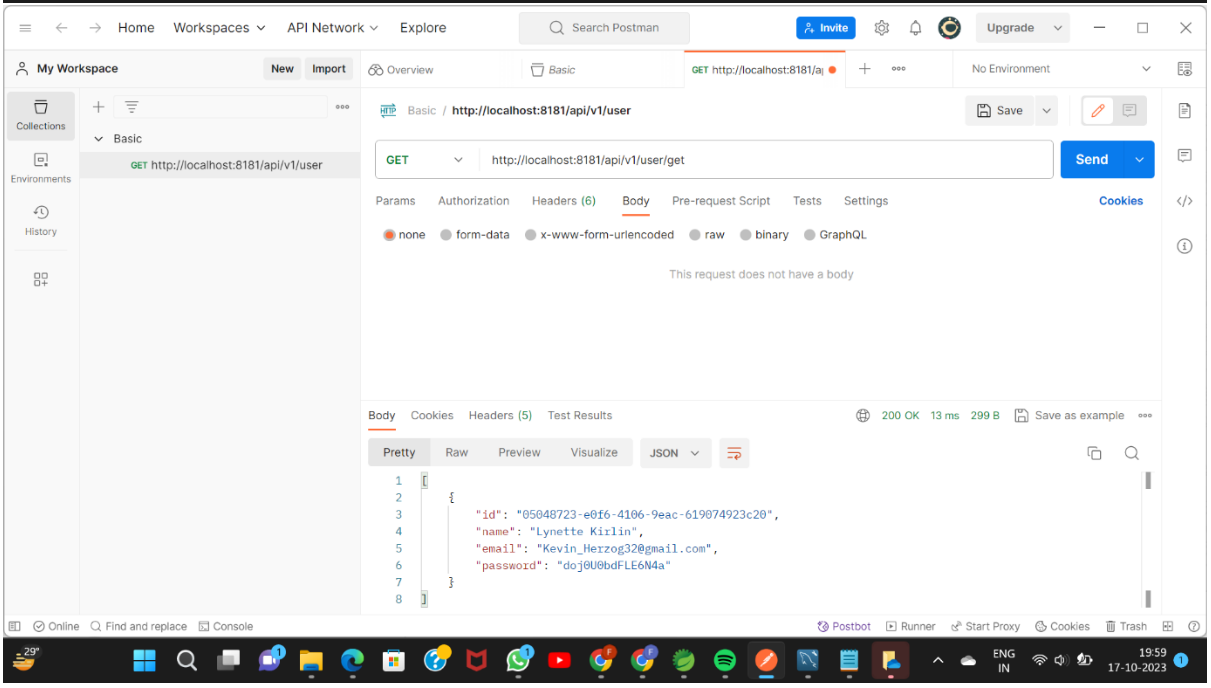
GET http://localhost:8181/api/v1/user/get

Body (raw, JSON):

```
{
    "name":"{{$randomFullName}}",
    "email":"{{$randomEmail}}",
    "password":"{{$randomPassword}}"
}
```

200 OK 15 ms 299 B

```
[
    {
        "id": "05048723-e0f6-4106-9eac-619074923c20",
        "name": "Lynette Kirlin",
        "email": "Kevin_Herzog32@gmail.com",
        "password": "doj0U0bdFLE6N4a"
    }
]
```



GET http://localhost:8181/api/v1/user/get

Body: none — This request does not have a body

200 OK 13 ms 299 B

```
[
    {
        "id": "05048723-e0f6-4106-9eac-619074923c20",
        "name": "Lynette Kirlin",
        "email": "Kevin_Herzog32@gmail.com",
        "password": "doj0U0bdFLE6N4a"
    }
]
```

5. CONCLUSION:

In this document, we have explored the process of building a Spring Boot project with MySQL and Postman integration. Here are the key takeaways:

Setting Up Spring Boot Project: We started by creating a Spring Boot project using Spring Initializer, configuring the project dependencies, and exploring the project structure.

Configuring MySQL Database: Before configuring our Spring Boot application, we created a MySQL database schema and configured the application.properties file to connect to the database.

Creating Repository Interfaces: We defined repository interfaces for our entity classes, enabling data access operations and providing methods for CRUD operations.

Creating Service Classes: Service classes were created to encapsulate business logic. These classes interact with the repositories and provide services to the controllers.

Creating Controller Classes: Controller classes were developed to handle incoming HTTP requests, define endpoints, and manage data flow between services and clients.

Testing with Postman: We employed Postman to test our API endpoints, ensuring that the application functions as expected, and data retrieval and manipulation work correctly.

This project demonstrates the fundamental steps in building a Spring Boot application with MySQL integration, emphasizing the importance of organized code structure and effective testing. It's crucial to remember that successful application development is an iterative process, and continuous testing and refinement are essential for achieving a reliable and efficient application.