CHAPTER 1

ABOUT COMPANY

Name	HQV Technologies Private Limited
Address	No 17, Third Floor C Street, Bharathi Nagar, Bangalore – 560 001
Contact No.	+91 8892614271
Email	www.viqar@hqvtechnologies.com
Website	www.hqvtechnologies.com
Type of the Company	Private
Nature of the company	Information Technology
Company logo	hqvtechnologies

1.1 Company Services

HQV Technologies is a research and development center and educational institution based in Bangalore. They focus on providing quality education on the latest technologies and developing much-needed products for society. They operate project consultants who undertake a variety of projects from a variety of clients, assist in designing and manufacturing products, and provide services. They are continuously involved in researching future technologies and finding ways to simplify them for our customers.

1.2 Mission and Vision

To be a world-class IT Application Development Training and internship organization committed to helping students and providing quality services.

To Harness and train best brainpower to give solutions for real challenges of the world.

1.3 Values

- Respect for dignity and potential of individuals.
- Enthusiasm for top performance and desire for change.
- Honesty and fairness in everything**.**
- Ensure speed of response.
- Faster learning, creativity and team-work.
- Loyalty and pride in the company.

CHAPTER 2

ABOUT DEPARTMENT

2.1 Vision

The vision of a Java full stack web application using REST API is to provide a scalable, efficient, and flexible solution for building web applications that communicate with external services through a RESTful API.

The use of RESTful APIs allows for the exchange of data between a web application and external services in a standardized, lightweight, and platform-independent manner. This allows for the creation of highly modular and extensible web applications that can easily integrate with other services and systems.

The vision of a Java full stack web application using REST API is to provide a comprehensive solution for building dynamic, robust, and scalable web applications that leverage the power of Java and its associated frameworks, such as Spring and Hibernate, to provide a complete end-to-end solution for building complex and feature-rich web applications.

2.2 Main Areas

There are several main areas of a web application using REST API, including:

Resource Mapping: Mapping resources to URLs is an important aspect of RESTful architecture. The URLs should be descriptive and represent the resources they represent. The resource mappings should be flexible to allow for future expansion and changes.

HTTP Methods: RESTful APIs use HTTP methods such as GET, POST, PUT, DELETE, and PATCH to perform various operations on resources. These methods help to ensure that the API is standardized and easy to use.

Data Formats: RESTful APIs support a range of data formats, including JSON and XML. These formats help to ensure that data can be easily exchanged between the web application and external services.

Security: Security is a crucial aspect of web applications using RESTful APIs. Secure communication protocols such as HTTPS should be used to protect the data being exchanged between the application and external services. Additionally, authentication and authorization mechanisms should be in place to ensure that only authorized users can access the resources.

Performance: Performance is also an important area of web applications using RESTful APIs. Caching, compression, and load balancing techniques can be used to improve performance and scalability.

Documentation: Documentation is important for any web application, but especially for RESTful APIs. Clear and concise documentation should be provided to help developers understand how to use the API and how to interact with the resources.

2.3 Responsibilities

Developing a web application using REST API involves several responsibilities, which may vary depending on the scope and complexity of the application. However, some common responsibilities of the development team for a web application using REST API include:

1. Designing the API: The development team is responsible for designing the RESTful API, including defining the resources, endpoints, HTTP methods, and data formats. The API design should be flexible and scalable to accommodate future changes and additions.

2. Implementing the API: Once the API design is finalized, the development team is responsible for implementing the API using appropriate technologies such as Java, Spring, and Hibernate. The API should be tested thoroughly to ensure that it is working as intended and meets the requirements.

3. Securing the API: Security is a crucial aspect of a web application using RESTful API, and the development team is responsible for implementing appropriate security measures such as SSL/TLS encryption, authentication, and authorization. The API should be tested thoroughly for vulnerabilities and flaws.

4. Integrating with external services: A web application using RESTful API may need to integrate with external services, such as payment gateways, social media platforms, or other APIs. The development team is responsible for ensuring that the integration is

seamless and secure.

5. Optimizing performance: The development team is responsible for optimizing the performance of the web application and API, using techniques such as caching, compression, and load balancing.

6. Testing and debugging: The development team is responsible for testing and debugging the web application and API thoroughly to ensure that it is working as intended, meets the requirements, and is free from bugs and errors.

7. Documentation and maintenance: The development team is responsible for documenting the web application and API thoroughly, including API documentation, user manuals, and technical guides. Additionally, the team is responsible for maintaining the web application and API, fixing bugs, addressing issues, and releasing updates as needed.

CHAPTER 3

TASK PERFORMED

All tasks performed during the internship program were based on backend (Rest API, RDBMS), UI mocks development. The trainers have assigned some basic tasks to industry standards to make the technology easy to understand.

3.1 REST API’s

REST APIs provide a flexible, lightweight way to integrate applications, and have emerged asthe most common method for connecting components in microservices architectures.
An API, or application programming interface, is a set of rules that define how applications or devices can connect to and communicate with each other. A REST API is an API that conforms to the design principles of the REST, or representational state transfer architecturalstyle. For this reason, REST APIs are sometimes referred to RESTful APIs.
First defined in 2000 by computer scientist Dr. Roy Fielding in his doctoral dissertation, RESTprovides a relatively high level of flexibility and freedom for developers. This flexibility is just one reason why REST APIs have emerged as a common method for connecting components and applications in a microservices architecture.

HTTP Methods for RESTful Services

The HTTP verbs comprise a major portion of our “uniform interface” constraint and provide us the action counterpart to the noun-based resource. The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently. Of thoseless-frequent methods, OPTIONS and HEAD are used more often than others.

Below is a table summarizing recommended return values of the primary HTTP methods in combination with the resource URIs:

HTTP Verb	**CRUD**	**Entire Collection (e.g. /customers)**	**Specific Item (e.g. /customers/{id})**
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

3.2 Spring Boot

Java and the Spring framework

While Java may be easy to use and easier to learn than other languages, the level of complexityto build, debug, and deploy Java apps has escalated to dizzying new heights. This is due to theexponential number of variables modern developers are faced with when developing web appsor mobile apps for common modern technologies such as music streaming or mobile cash payment apps. A developer writing a basic line-of-business app now needs to deal with multiple libraries, plugins, error logging and handling libraries, integrations with web services,and multiple languages such as C#, Java, HTML, and others. Understandably, there is an insatiable demand for any tools that will streamline Java app development, saving thedevelopers time and money.

Enter application frameworks—the large bodies of prewritten code that developers can use and add to their own code, as their needs dictate. These frameworks lighten the developer's load for almost any need—whether they're developing mobile and web apps or working with desktops and APIs. Frameworks make creating apps quicker, easier, and more secure by providing reusable code and tools to help tie the different elements of a software developmentproject all together.

Here's where Spring comes in: Spring is an open-source project that provides a streamlined, modular approach for creating apps with Java. The family of Spring projects began in 2003 asa response to the complexities of early Java development and provides support for developingJava apps. The name, Spring, alone usually refers to the application framework itself or the entire group of projects, or modules. Spring Boot is one specific module that is built as an extension of the Spring framework.

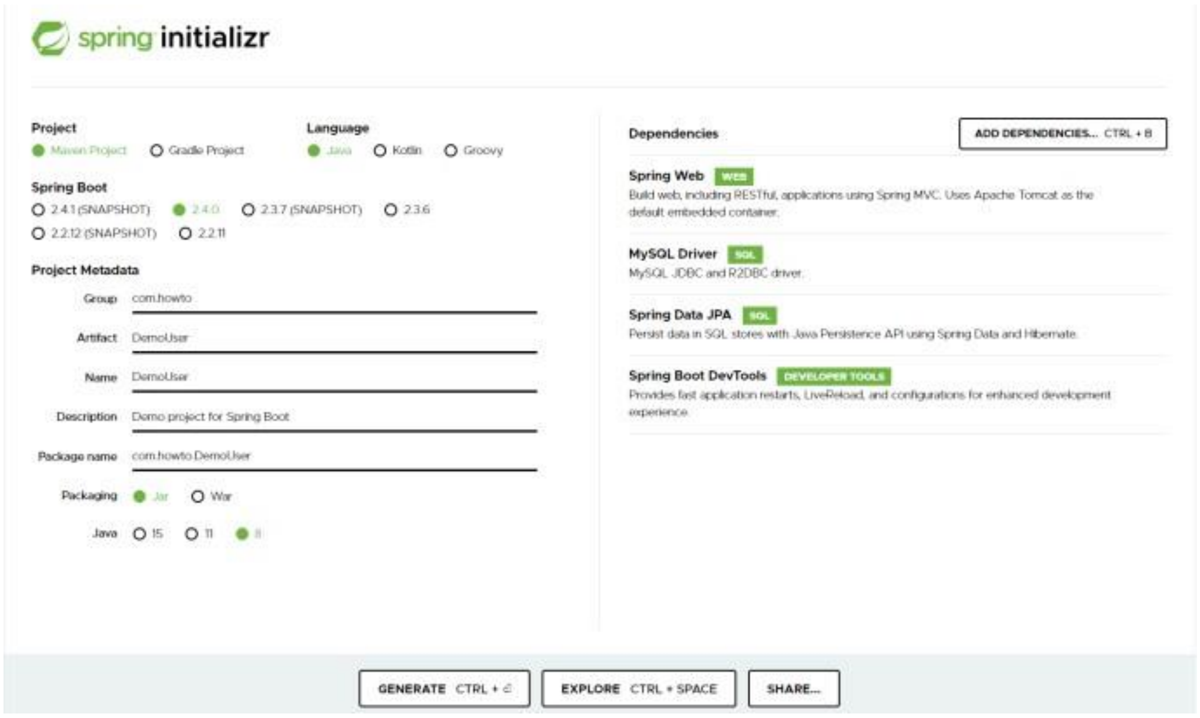
So, with that background on how the Spring framework, Spring Boot, and Java work together,here's the definition of Spring Boot—the tool that streamlines and speeds up web app and microservices development within the Java framework, Spring.

Building a Spring Boot API in JAVA

Step 1: Initializing a Spring Boot Project

To start with **Spring Boot REST API**, you first need to initialize the Spring Boot Project. Youcan easily initialize a new Spring Boot Project with Spring Initializr.

From your Web Browser, go to **start.spring.io**. Choose **Maven** as your Build Tool andLanguage as **Java**. Select the specific version of Spring Boot you want to go ahead with.



Project: [✓ Maven Project] Gradle Project

Language: [✓ Java] Kotlin Groovy

Spring Boot: 2.4.1 (SNAPSHOT) [✓ 2.4.0] 2.3.7 (SNAPSHOT) 2.3.6 2.2.12 (SNAPSHOT) 2.2.11

Project Metadata

Group	com.howto
Artifact	DemoUser
Name	DemoUser
Description	Demo project for Spring Boot
Package name	com.howto.DemoUser
Packaging	[✓ Jar] War
Java	15 11 [✓ 8]

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

MySQL Driver SQL
MySQL JDBC and R2DBC driver.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

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

Fig 3.2: Spring Initializer

You can go ahead and add a few dependencies to be used in this project.

- **Spring Data JPA** – Java Persistence API and Hibernate.
- **Spring Web** – To include Spring MVC and embed Tomcat into your project.
- **Spring Boot DevTools** – Development Tools.
- **MySQL Driver** – JDBC Driver (any DB you want to use).

Once you’re done with the configuration, click on “**Generate**”. A ZIP file will then be downloaded. Once the download is finished, you can now import your project files as a MavenProject into your IDE (such as **Eclipse**) or a text editor of choice.

Step 2: Connecting Spring Boot to the Database

Next, you need to set up the Database, and you can do it easily with Spring Data JPA.

Add some elementary information in your **application.properties** file to set up the connection to your preferred Database. Add your JDBC connection URL, provide a username and password for authentication, and set the **ddl-auto** property to **update**.

Hibernate has different dialects for different Databases. Hibernate automatically sets the dialect for different Databases, but it's a good practice to specify it explicitly.

```
spring.datasource.url = jdbc:mysql://localhost:3306/user
spring.datasource.username = user
spring.datasource.password = user
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

Step 3: Creating a User Model

The next step is to create a **Domain Model**. They are also called **Entities** and are annotated by **@Entity**.

Create a simple User entity by annotating the class with **@Entity**. Use **@Table** annotation tospecify the name for your Table. **@Id** annotation is used to annotate a field as the id of an entity. Further, set it as a **@GeneratedValue** and set the **GenerationType** to **AUTO**.

```
@Entity
@Table(name = "user")
public class User {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private long id;
   private String name;
}
```

Now, this class (entity) is registered with Hibernate.

Step 4: Creating Repository Classes

To perform CRUD (Create, Read, Update, and Delete) operations on the **User** entities, you'llneed to have a **UserRepository**. To do this, you'll have to use the **CrudRepository** extensionand annotate the interface with **@Repository**.

```
@Repository
public interface UserRepository extends CrudRepository<User, Long> {}
```

Step 5: Creating a Controller

You've now reached the Business Layer, where you can implement the actual businesslogic of processing information.

@RestController is a combination of **@Controller** and **@ResponseBody**.Createa **UserController** as shown below.

```
@RestController
@RequestMapping("/api/user")
public class UserController {

   @Autowired
   private UserRepository userRepository;

   @GetMapping
   public List<User> findAllUsers() {
       // Implement
   }

   @GetMapping("/{id}")
   public ResponseEntity<User> findUserById(@PathVariable(value = "id") long id) {
     // Implement
   }

   @PostMapping
   public User saveUser(@Validated @RequestBody User user) {
       // Implement
   }
}
```

You've to **@Autowired** your **UserRepository** for dependency injection. To specify the type of HTTP requests accepted, use the **@GetMapping** and **@PostMapping** annotations.

Let's implement the **findAll()** endpoint. It calls the **userRepository** to **findAll()** users andreturns the desired response.

```
@GetMapping
public List<User> findAllUsers() {
   return userRepository.findAll();
}
```

To get each user by their **id**, let's implement another endpoint.

```
@GetMapping("/{id}")
public ResponseEntity<User> findUserById(@PathVariable(value = "id") long id) {
   Optional<User> user = userRepository.findById(id);

   if(user.isPresent()) {
      return ResponseEntity.ok().body(user.get());
   } else {
      return ResponseEntity.notFound().build();
   }
}
```

If the **user.isPresent()**, a **200 OK** HTTP response is returned, else,a **ResponseEntity.notFound()** is returned.

Now, let's create an endpoint to save users. The **save()** method saves a new user if it isn'talready existing, else it throws an exception.

```
@PostMapping

public User saveUser(@Validated @RequestBody User user) {

   return userRepository.save(user);

}
```

The **@Validated** annotation is used to enforce basic validity for the data provided about the user. The **@RequestBody** annotation is used to map the body of the **POST** request sent to theendpoint to the **User** instance you'd like to save.

Step 6: Compile, Build and Run

You can change the port of Spring Boot from your **application.properties** file.

```
server.port = 9090
```

8080 is the default port that Spring Boot runs in.

It's time to run the Spring Boot REST API you've created. To run the application, directly execute **./mvnw spring-boot:run** on the command line from your base project folder where **pom.xml** is located.

If your application has successfully run, you'll be able to see these audit logs at the end of yourcommand line.

```
2020-11-05 13:27:05.073  INFO 21796 --- [  restartedMain]
```

```
2020-11-05 13:27:05.108  INFO 21796 --- [  restartedMain]

o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http)
withcontext path ''

2020-11-05 13:27:05.121  INFO 21796 --- [  restartedMain]
```

Step 7: Testing the Spring Boot REST APIs

Your Spring Boot REST API is now up and running on

http://localhost:8080/. Use your browser, **curl,** or **Postman** to test the

endpoints.

To send an HTTP GET request, go to **http://localhost:8080/api/user** from your browser andit will display a JSON response as shown.

```
[
  {
            "id": 1,
            "name":"John"
  },
  {
            "id": 2,
            "name":"Jane"
  },
  {
            "id": 3,
            "name": "Juan"
  }
]
```

Now, send an HTTP POST request to add specific users to your Database.

```
$ curl --location --request POST 'http://localhost:8080/api/user'
--header 'Content-Type: application/json'
--data-raw '{ "id": 4, "name": "Jason" }'
```

The API will return a 200 OK HTTP response with the response body of the persisted user.

```
{
   "id": 4,
   "name": "Jason"
}
```

That’s it, you have now successfully tested your Spring Boot REST API.

3.3 POSTMAN

Postman is a standalone tool that **exercises web APIs by making HTTP requests from outside the service**. When using Postman, we don't need to write any HTTP client infrastructure code just for the sake of testing. Instead, we create test suites called collectionsand let Postman interact with our API.

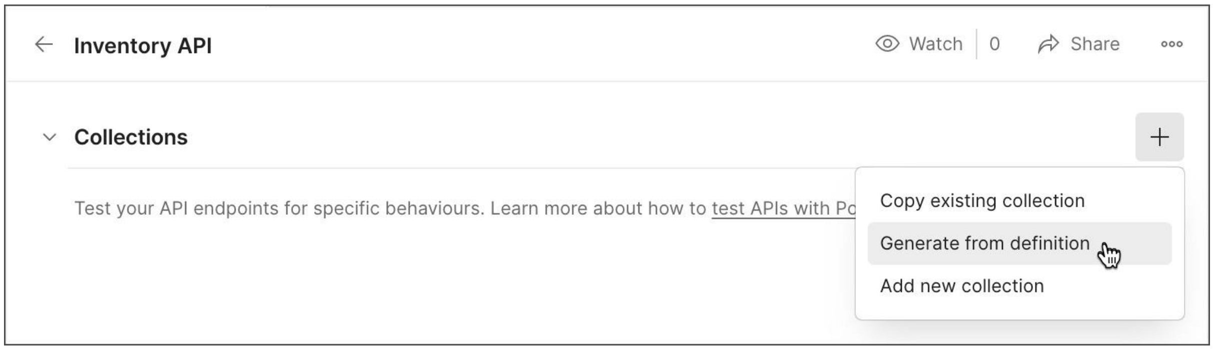
Testing API's using Postman

Adding API tests

You can connect a test collection (a collection containing API tests) to an API you've definedin the Postman API Builder.

To add a test collection to an API, do the following:

1. Select **APIs** in the sidebar and select an API.
2. Select **Test and Automation**.
3. Next to **Collections**, select **+** and select an option:
   - **Add new collection** - This option creates a new empty collection in the API.You can add your tests to the **Tests** tab.
   - **Copy existing collection** - Select an available collection from the list. A copyof the collection is added to the API.
   - **Generate from definition** - Change any settings to customize the new collection and select **Generate Collection**.

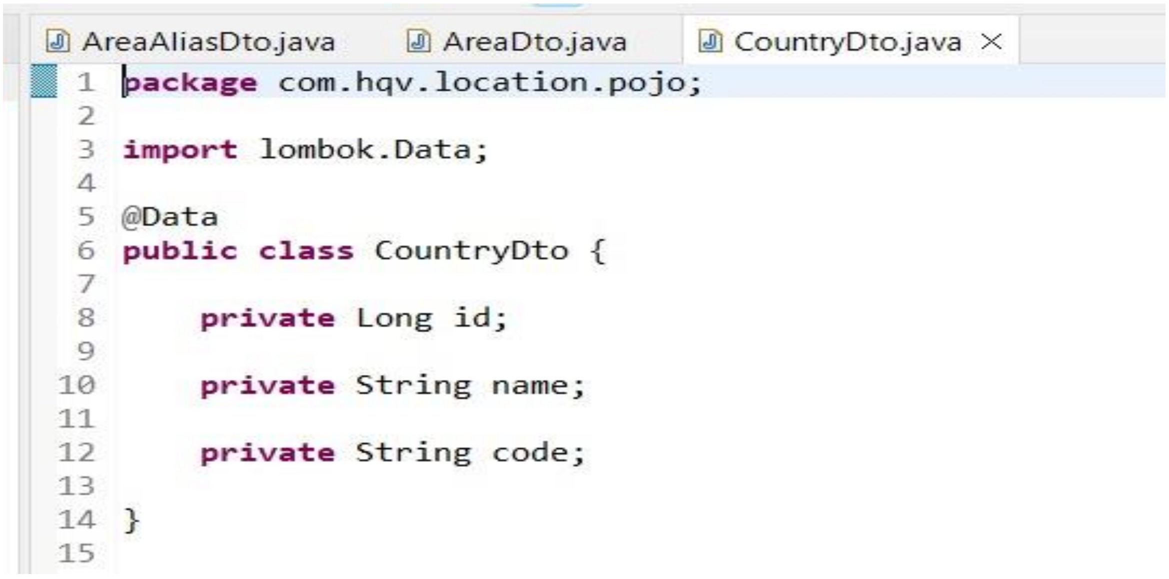


Test your API endpoints for specific behaviours. Learn more about how to test APIs with Po

Fig 3.3: Postman adding tests

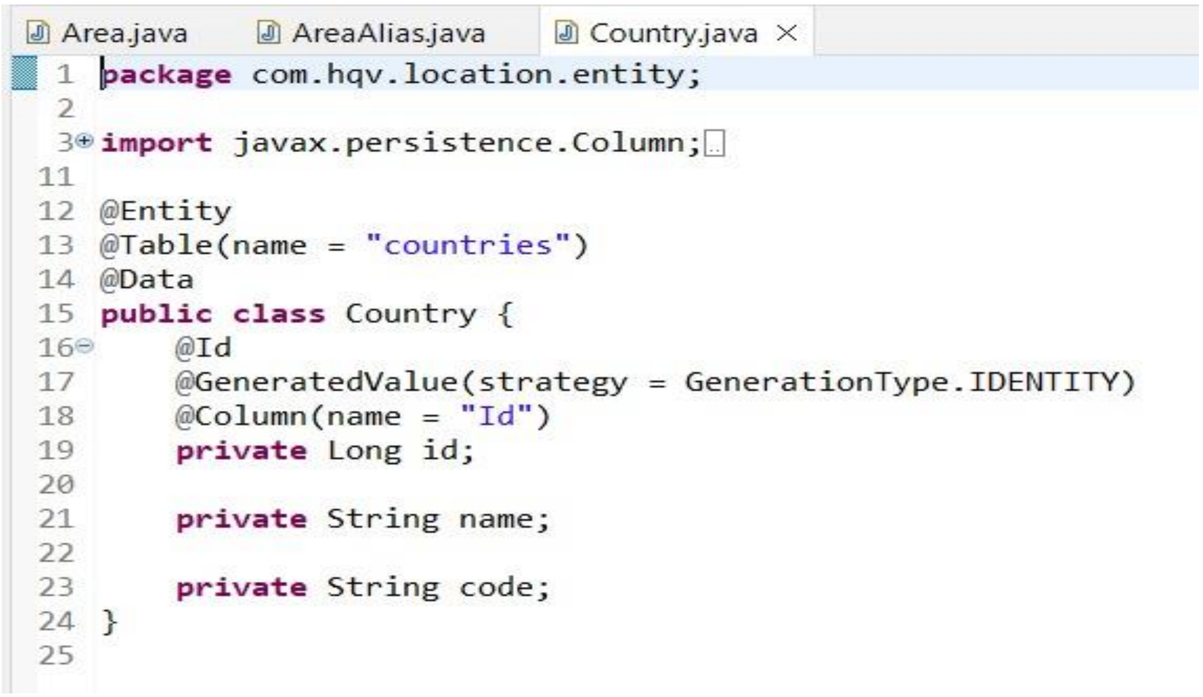
3.4 Project

Location API’s Using Java Spring Boot



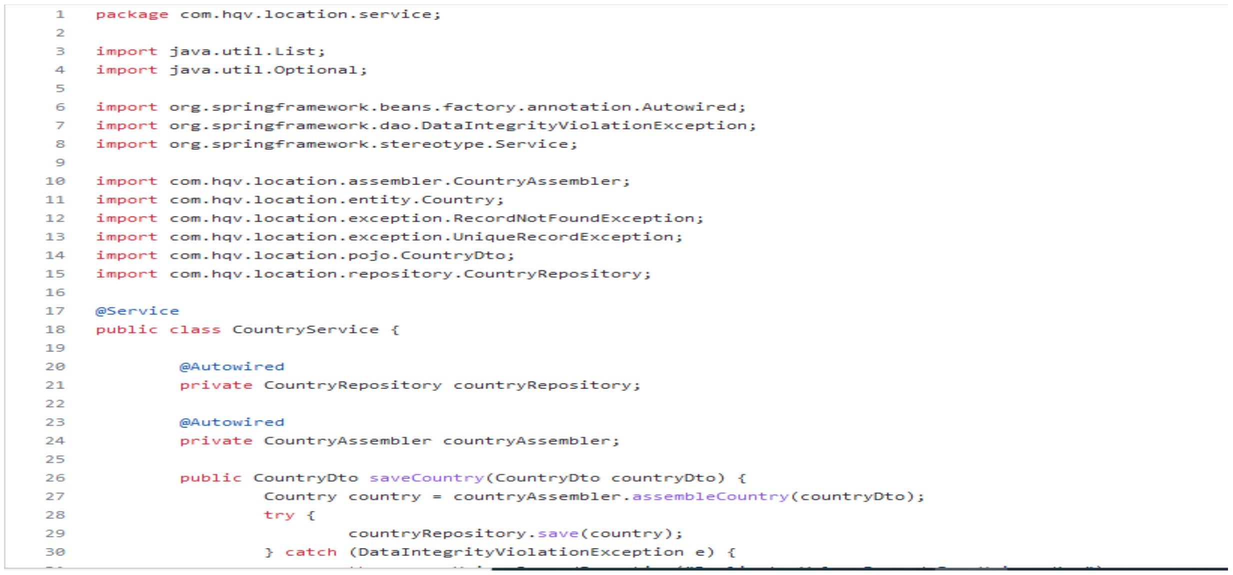
```java
package com.hqv.location.pojo;

import lombok.Data;

@Data
public class CountryDto {

    private Long id;

    private String name;

    private String code;

}

```

Fig 3.4.1: Country Dto class



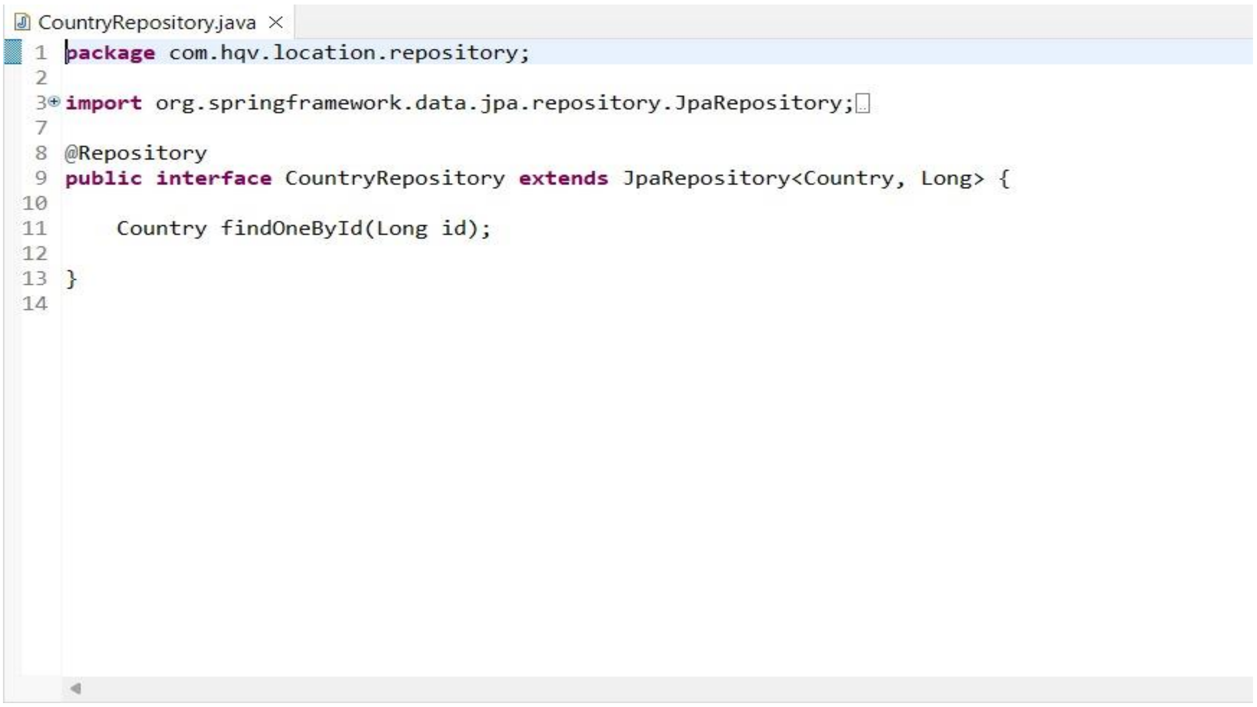
```java
package com.hqv.location.entity;

import javax.persistence.Column;

@Entity
@Table(name = "countries")
@Data
public class Country {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "Id")
    private Long id;

    private String name;

    private String code;
}

```

Fig 3.4.2: Country Entity class



```
package com.hqv.location.service;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.stereotype.Service;

import com.hqv.location.assembler.CountryAssembler;
import com.hqv.location.entity.Country;
import com.hqv.location.exception.RecordNotFoundException;
import com.hqv.location.exception.UniqueRecordException;
import com.hqv.location.pojo.CountryDto;
import com.hqv.location.repository.CountryRepository;

@Service
public class CountryService {

        @Autowired
        private CountryRepository countryRepository;

        @Autowired
        private CountryAssembler countryAssembler;

        public CountryDto saveCountry(CountryDto countryDto) {
                Country country = countryAssembler.assembleCountry(countryDto);
                try {
                        countryRepository.save(country);
                } catch (DataIntegrityViolationException e) {
```

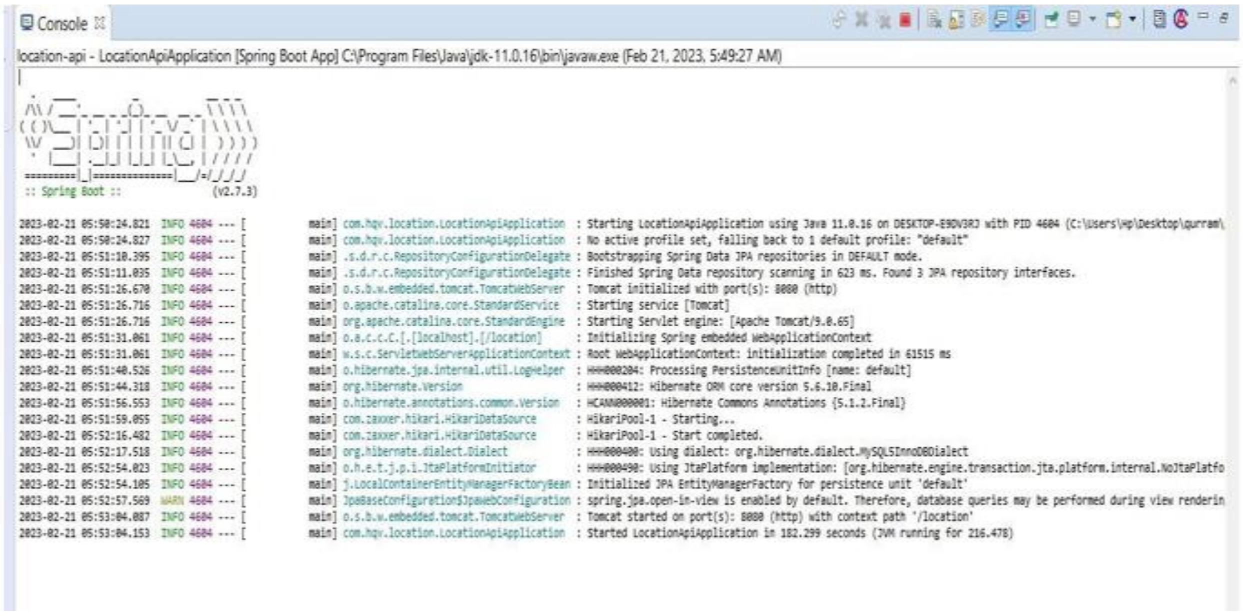
Fig 3.4.2: Country Service class



CountryRepository.java

```
package com.hqv.location.repository;

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

@Repository
public interface CountryRepository extends JpaRepository<Country, Long> {

    Country findOneById(Long id);

}
```

Fig 3.4.4 Country Repository class

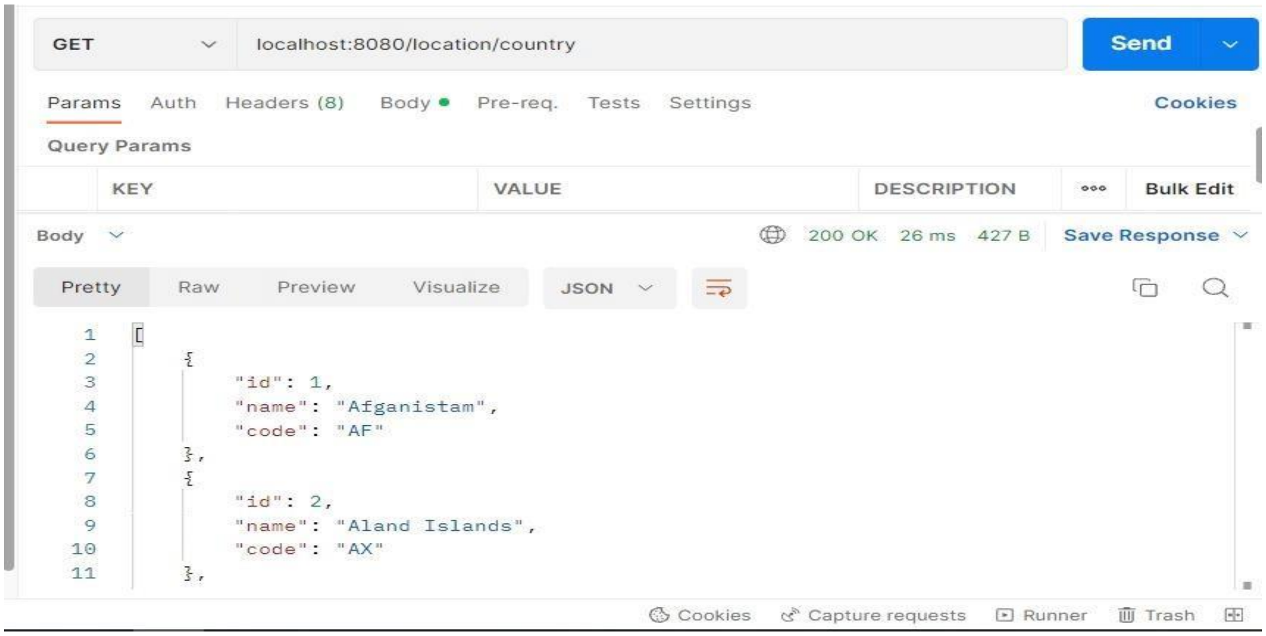


Console

location-api - LocationApiApplication [Spring Boot App] C:\Program Files\Java\jdk-11.0.16\bin\javaw.exe (Feb 21, 2023, 5:49:27 AM)

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.3)

2023-02-21 05:50:24.821  INFO 4604 --- [           main] com.hqv.location.LocationApiApplication  : Starting LocationApiApplication using Java 11.0.16 on DESKTOP-E9DV3RJ with PID 4604 (C:\Users\Hp\Desktop\qurram\
2023-02-21 05:50:24.827  INFO 4604 --- [           main] com.hqv.location.LocationApiApplication  : No active profile set, falling back to 1 default profile: "default"
2023-02-21 05:51:10.395  INFO 4604 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-02-21 05:51:11.035  INFO 4604 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 623 ms. Found 3 JPA repository interfaces.
2023-02-21 05:51:26.670  INFO 4604 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2023-02-21 05:51:26.716  INFO 4604 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-02-21 05:51:26.716  INFO 4604 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
2023-02-21 05:51:31.061  INFO 4604 --- [           main] o.a.c.c.C.[.[localhost].[/location]      : Initializing Spring embedded WebApplicationContext
2023-02-21 05:51:31.061  INFO 4604 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 61515 ms
2023-02-21 05:51:40.526  INFO 4604 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2023-02-21 05:51:44.318  INFO 4604 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.10.Final
2023-02-21 05:51:56.553  INFO 4604 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2023-02-21 05:51:59.055  INFO 4604 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2023-02-21 05:52:16.482  INFO 4604 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2023-02-21 05:52:17.518  INFO 4604 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
2023-02-21 05:52:54.023  INFO 4604 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatfo
2023-02-21 05:52:54.105  INFO 4604 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-02-21 05:52:57.569  WARN 4604 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view renderin
2023-02-21 05:53:04.087  INFO 4604 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path '/location'
2023-02-21 05:53:04.153  INFO 4604 --- [           main] com.hqv.location.LocationApiApplication  : Started LocationApiApplication in 182.299 seconds (JVM running for 216.478)
```

Fig 3.4.5: Console Output



GET localhost:8080/location/country

Body 200 OK 26 ms 427 B

```
[
    {
        "id": 1,
        "name": "Afganistam",
        "code": "AF"
    },
    {
        "id": 2,
        "name": "Aland Islands",
        "code": "AX"
    },
```

Fig 3.4.6: Country HTTP Response

CHAPTER

REFLECTION

The internship from HQV Technologies was very useful. The instruction was useful and showed us how to quickly construct solutions to the issue statements. The most advantageousaspect of internships was the opportunity to get practical knowledge on different technologies. The instructor used a distinct method of instruction, giving us time to apply the principles realistically to ensure that we understood them. We were able to successfullycomplete the exam because to the effective teaching we received throughout the internship.

The idea of internship program sees merit in attempting to shorten the period on training that is often significant duration to orient the trainee or newly inducted person onto the project. The internship covered the concepts of back end development object oriented concepts, working with GitHub , Understanding, developing and testing API’s, and also some UI/UXdesign such as UI Mockup. It is and has great tools, libraries and frameworks.

The internship session has been a great learning journey helping the participants in the internship program to understand the concepts of Web Development. It also helped us improve our logical thinking. It helped us to improve our communication skills. They taughtus to manage the time so that we could code maximum in limited or specified time. We realized that soft skills contribute to a positive work environment and help us maintain an efficient workflow.

CHAPTER

CONCLUSION

The demand for internships is more pressing as self-learning skills are increasingly needed in the workplace on short notice. learned about industry norms, the abilities to study on our own, logical reasoning, and many other skills that are needed tohelp us contribute to and grow with the industry. I now have a better understanding of the professional options available thanks to the internship program. It becomes clearer how important it is to continue learning throughout one's career in order to succeed in the field, as well as how important it is to approach challenges with an open mind.

Interpersonal skills were enhanced by the internship. During this internship, we developed our web page design skills. Our logical reasoning skills were also improved by the trainer.