



NEAR ITPB, CHANNASANDRA, BENGALURU-560067

Affiliated to VTU, Belagavi

Approved by AICTE

Recognized by UGC under 2(f)

Accredited by NBA&NAAC

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SEMESTER – VI

CLOUD COMPUTING FULL STACK APPLICATION

DEVELOPMENT LABORATORY

MVJ22CS61

ACADEMIC YEAR 2024– 2025(EVEN)

## LABORATORY MANUAL

NAME OF THE STUDENT : \_\_\_\_\_

BRANCH : \_\_\_\_\_

UNIVERSITY SEAT No. : \_\_\_\_\_

SEMESTER & SECTION : \_\_\_\_\_

BATCH : \_\_\_\_\_

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING****VISION:**

To create an ambiance in excellence and provide innovative emerging programs in Computer Science and Engineering and to bring out future ready engineers equipped with technical expertise and strong ethical values.

**MISSION:**

1. **Concepts of computing discipline:** To educate students at undergraduate, postgraduate and doctoral levels in the fundamental and advanced concepts of computing discipline.
2. **Quality Research:** To provide strong theoretical and practical background across the Computer Science and Engineering discipline with the emphasis on computing technologies, quality research, consultancy and training.
3. **Continuous Teaching Learning:** To promote a teaching learning process that brings advancements in Computer Science and Engineering discipline leading to new technologies and products.
4. **Social Responsibility and Ethical Values:** To inculcate professional behavior, innovative research Capabilities, leadership abilities and strong ethical values in the young minds so as to work with the commitment for the betterment of the society.

**Programme Educational Objectives (PEOs):**

- PEO01: Current Industry Practices:** Graduates will analyze real world problems and give solutions using current industry practices in computing technology.
- PEO02: Research & Higher Studies:** Graduates with strong foundation in mathematics and engineering fundamentals will pursue higher learning, R&D activities and consultancy.
- PEO03: Social Responsibility:** Graduates will be professionals with ethics, who will provide industry growth and social transformation as responsible citizens.

**Programme Outcomes (POs):**

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

### **Program Specific Outcome:**

**PSO1: Programming:** Ability to understand, analyse and develop computer programs in the areas related to algorithms, system software, multimedia, web design, DBMS, and networking for efficient design of computer-based systems of varying complexity.

**PSO2: Practical Solution:** Ability to practically provide solutions for real world problems with a broad range of programming language and open-source platforms in various computing domains.

### **COURSE OBJECTIVES :**

1. Understand the core principles and concepts of cloud-native application development
2. Identify effective strategies for designing and architecting cloud-native applications
3. Design scalable and resilient cloud-native applications using Spring Boot, RESTful APIs, Hibernate, Docker, Kubernetes and microservices.

4. Build and deploy a cloud-native application using Spring Boot, Hibernate, REST API, Docker, Kubernetes, and microservices.
5. Understand the fundamentals of microservices architecture, the relevance of Spring and the cloud, and apply the best practices and patterns for developing, routing, securing, logging, and deploying microserv

**PREREQUISITES:**

1. Java.
2. Database.
3. API's Creation.
4. Docker
5. Kubernetes

**COURSEOUTCOMES(CO's):**

CONo	CO's
C60.1	Understand the essential principles and concepts of cloud-native application development, including microservices, containerization, orchestration, and cloud-based security and scalability
C606.2	Develop a migration strategy for legacy applications to cloud-native architect
C606.3	Implement service discovery, configuration management, and monitoring using Spring Cloud and Kubernetes
C606.4	Design, Develop, and deploy a scalable, resilient, and secure cloud-native application using Spring Boot, Hibernate, REST API, Docker, Kubernetes, and microservices, leveraging containerization, orchestration, and CI/CD pipelines to ensure efficient and reliable operation in
C606.5	Apply best practices and patterns to develop, route, secure, log, and deploy microservices, enabling them to build scalable, resilient, and cloud-ready application

**Software Requirement :**

1. Eclipse IDE
2. Spring initializer
3. Postman
4. Docker Desktop

**Operating System :**

1. Windows/ Linux.

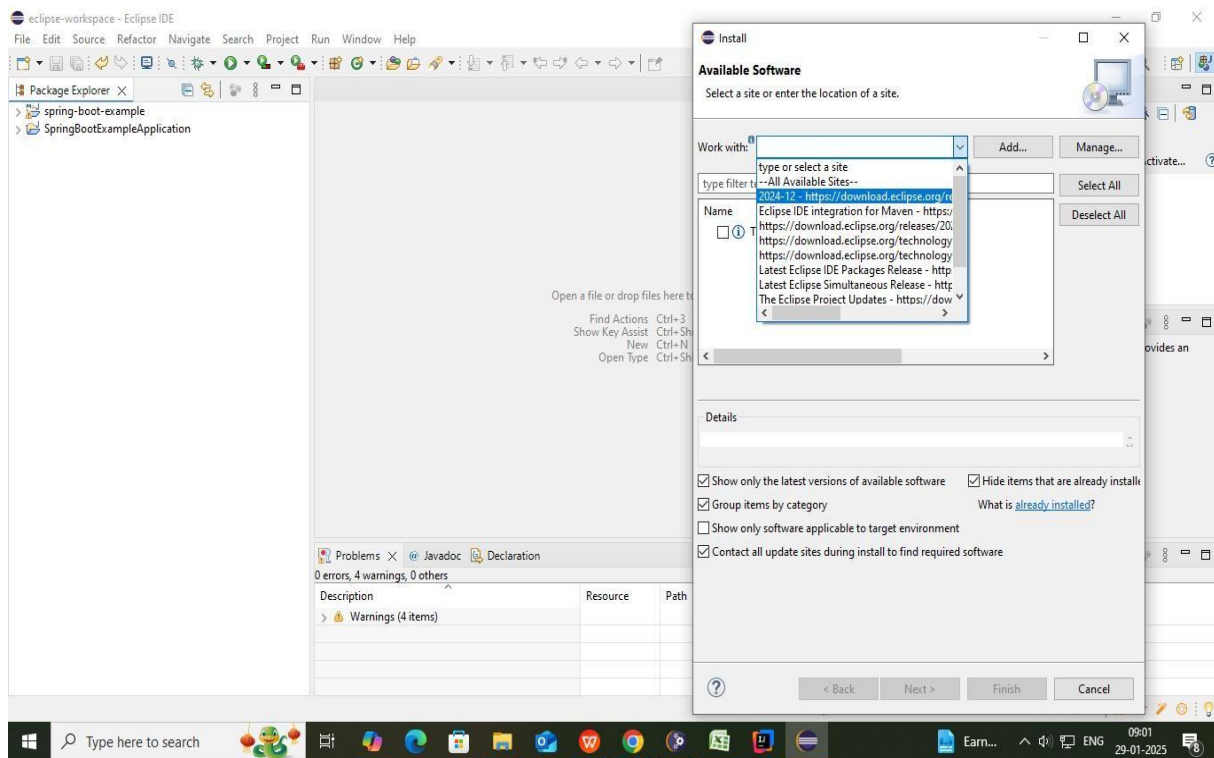
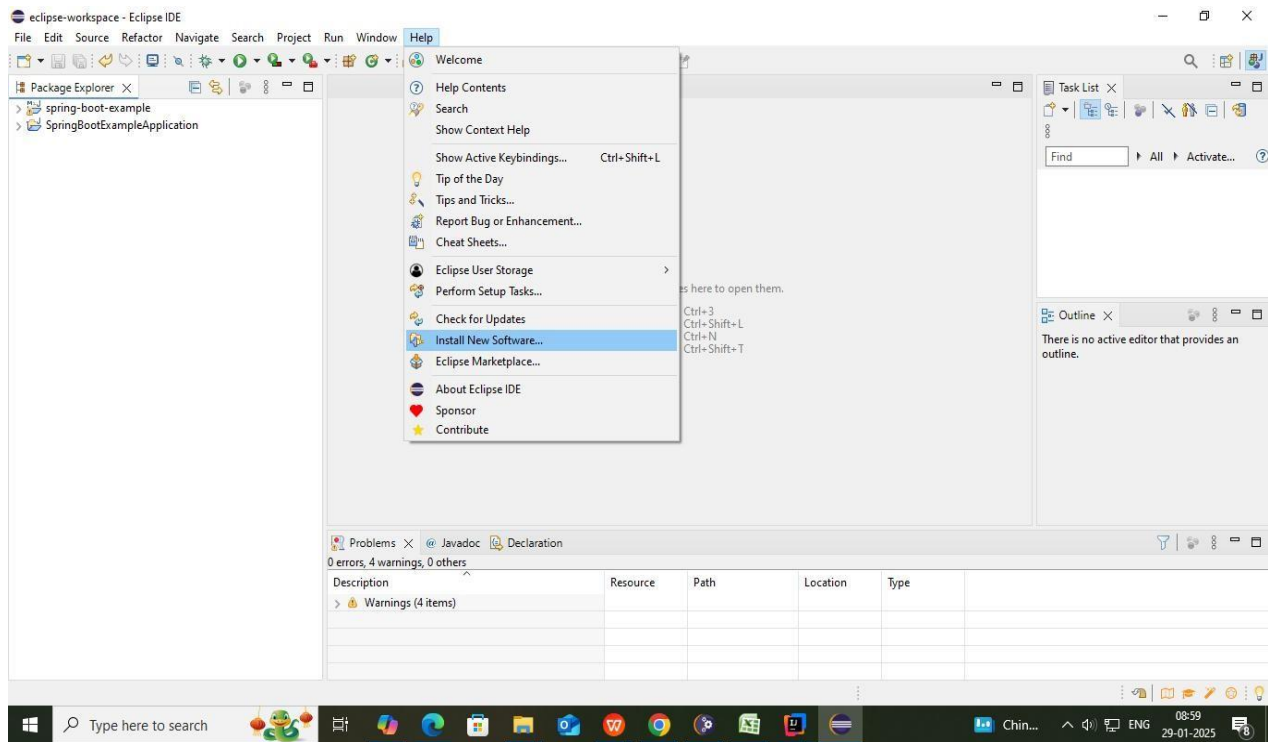
## CONTENTS

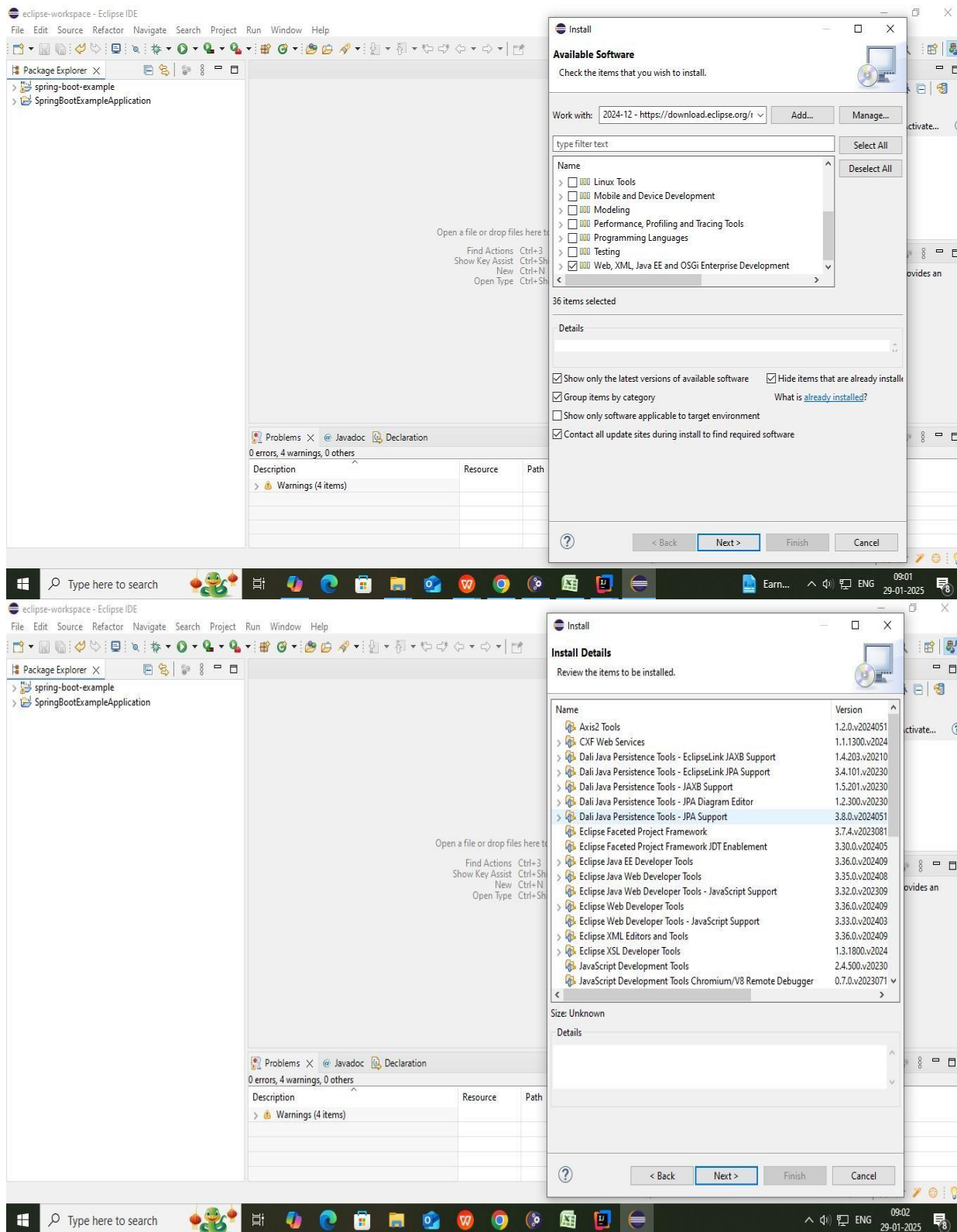
Exp 1:	Demonstrate Dependency Injection using annotation based using Spring boot	2 hrs
Exp 2:	Demonstrate Dependency Injection using constructor based using Spring boot	2 hrs
Exp 3:	Create a Spring Boot Application using Maven Plugin - Write a sample REST Controller API using Spring Annotations - Using Postman invoke the REST Controller to demonstrate end to end working.	2 hrs
Exp 4:	Write a sample REST App to demonstrate the below Concepts with a use-case of your choice. - GET, PUT, POST, DELETE.	2 hrs
Exp 5:	Write a sample REST App to Validate the REST API POST & PUT request. - Design a custom response with appropriate validation errors to the caller	2 hrs
Exp 6:	Write a Java application using Hibernate to insert data into Student DATABASE and retrieve info based on particular queries (For example update, delete, search etc...)	2 hrs
Exp 7:	Demonstrate Spring Data JPA integration in a Spring Boot application using Hibernate	2 hrs
Exp 8:	Demonstrate using Spring Boot: Complete the docker setup on your Sandbox. – Download a docker image from Docker Hub and deploy the same on your docker server – Build a sample custom image for any of the App of your choice and run the app image as a container.	2 hrs
Exp 9:	Using a docker compose file, deploy multiple apps/containers(eg: MySql, SpringBoot) onto the docker server (Project based)	2 hrs
Exp10:	Demonstrate with Spring Boot: Setup a Kubernetes development Env on your Sandbox (use Docker Desktop or Minicube) (Project based)	2 hrs

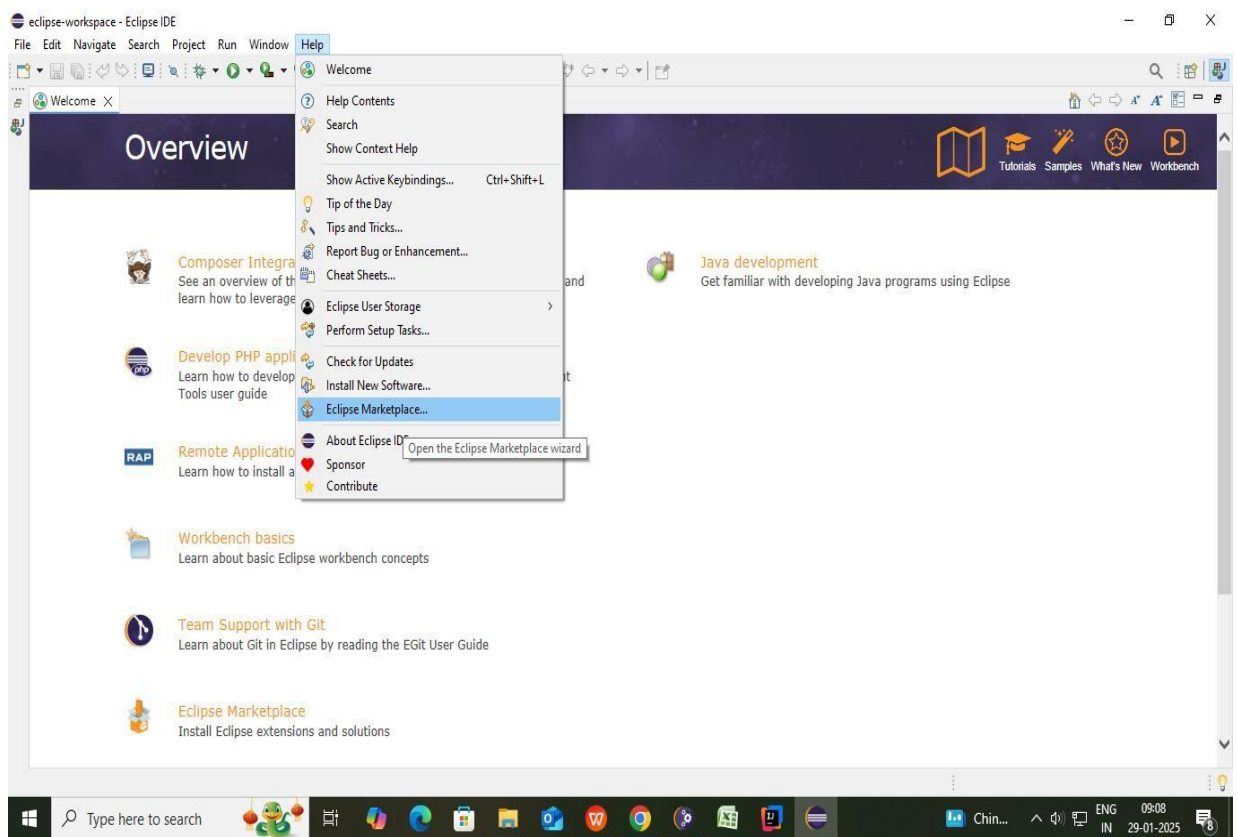
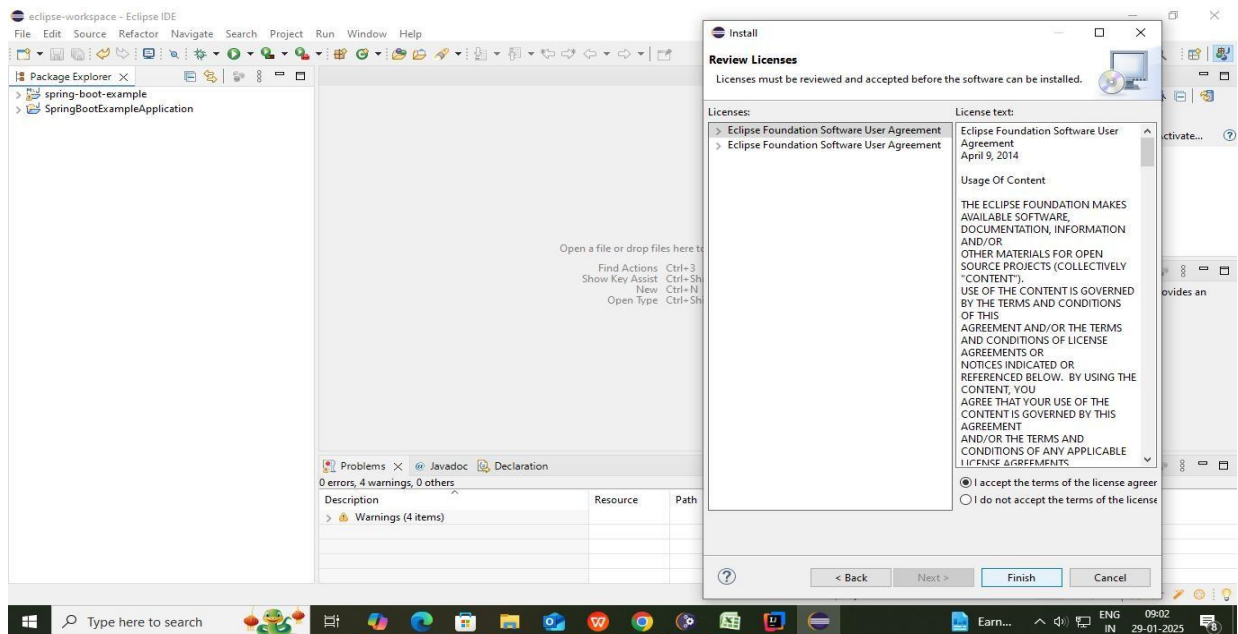
## Creating Spring Boot Web Project:

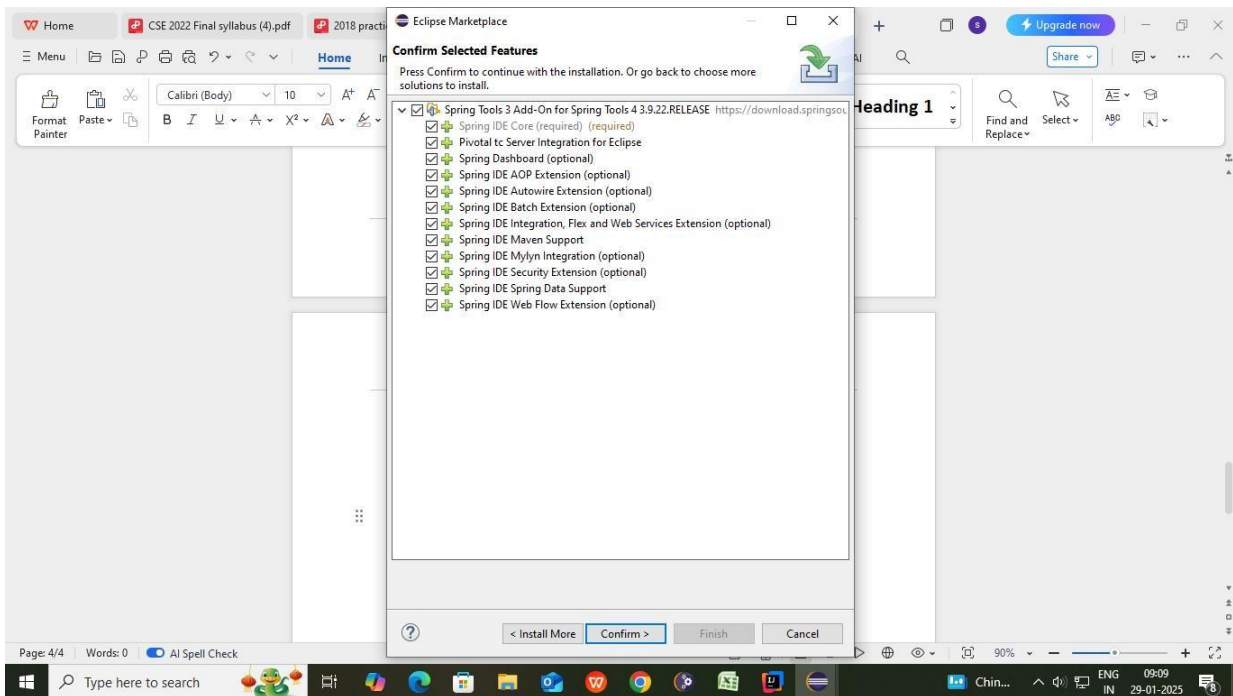
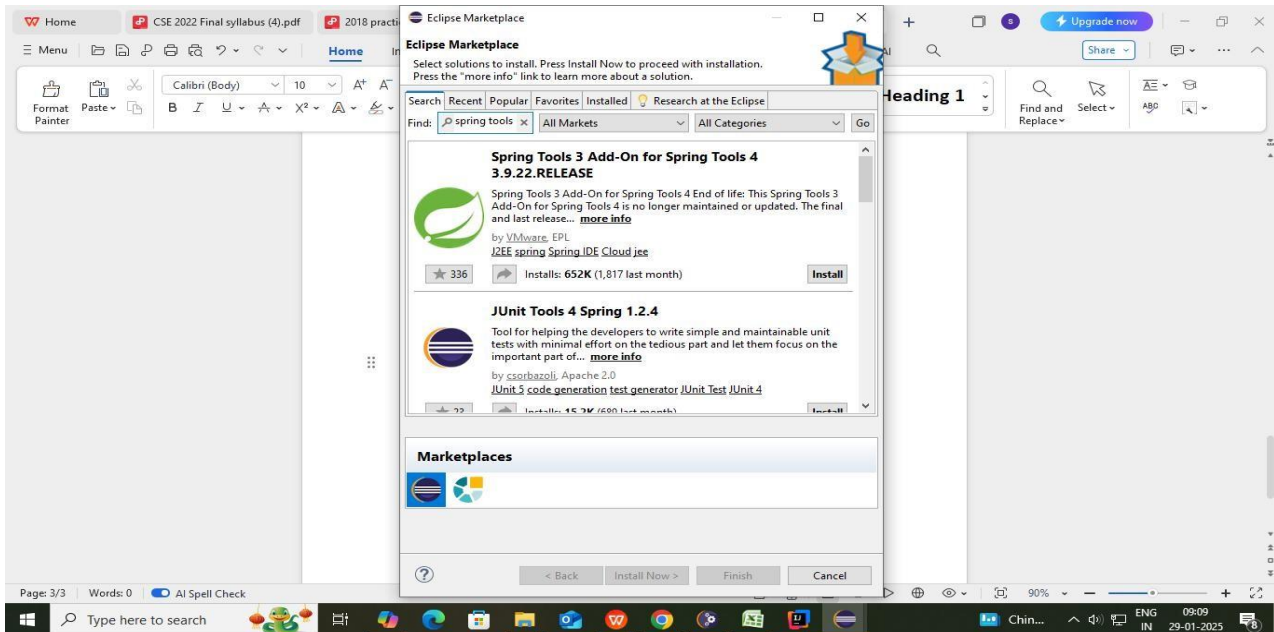
Follow the Steps Given Below

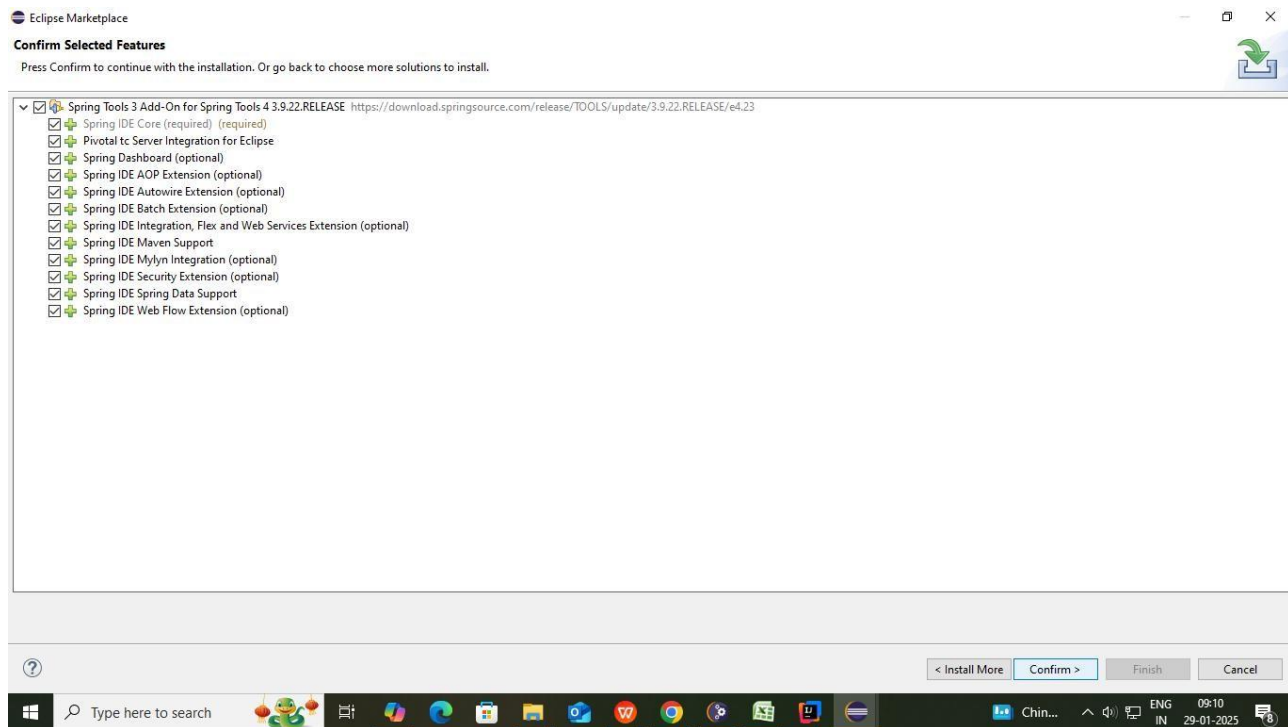
### Open Eclipse



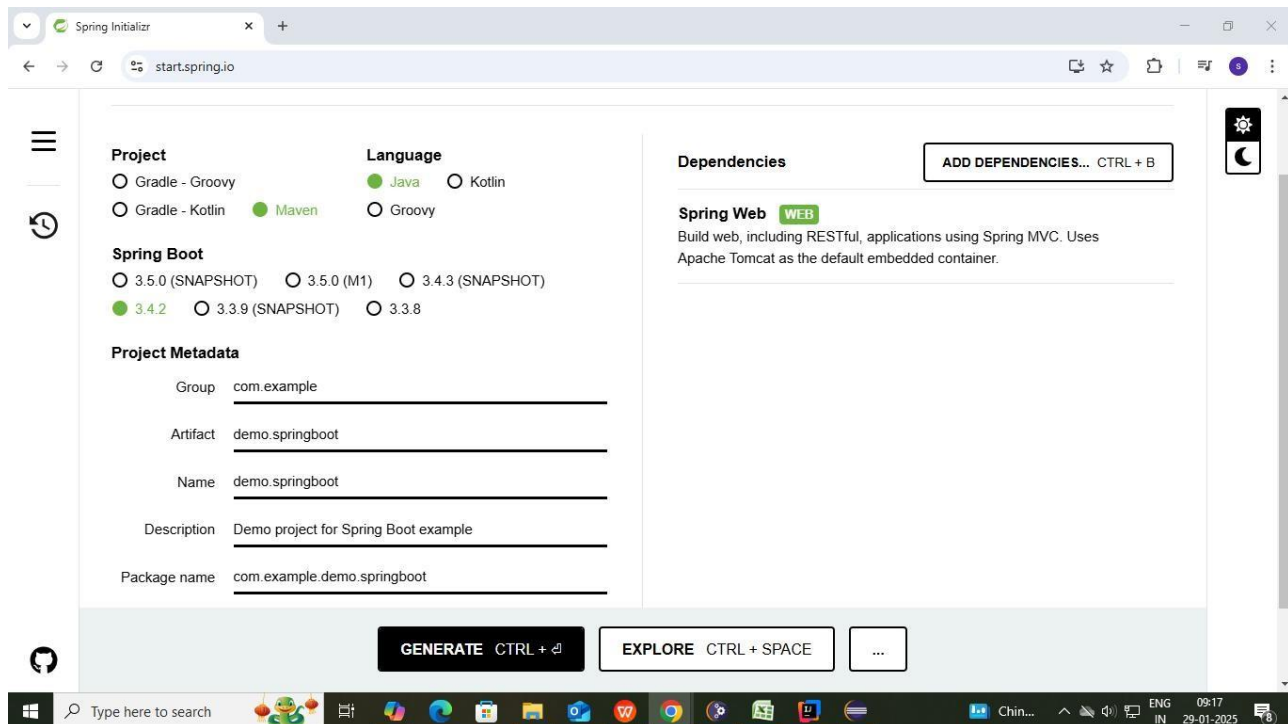




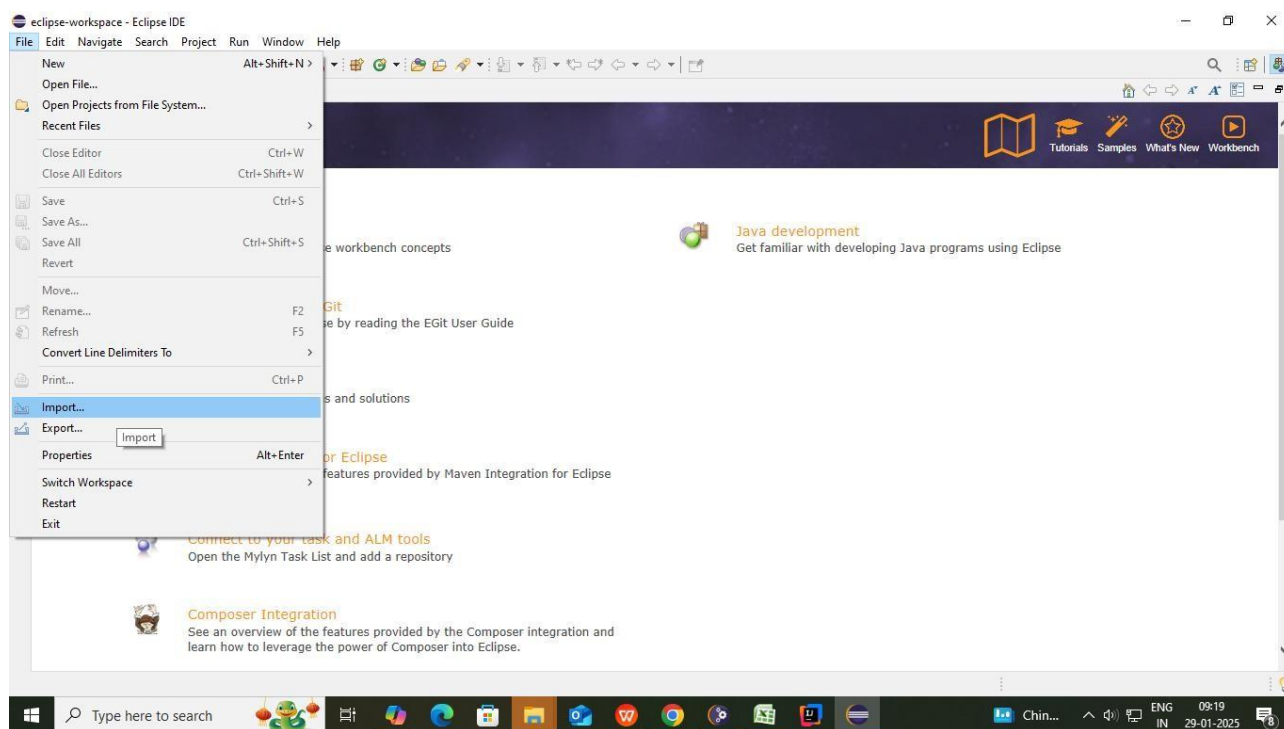
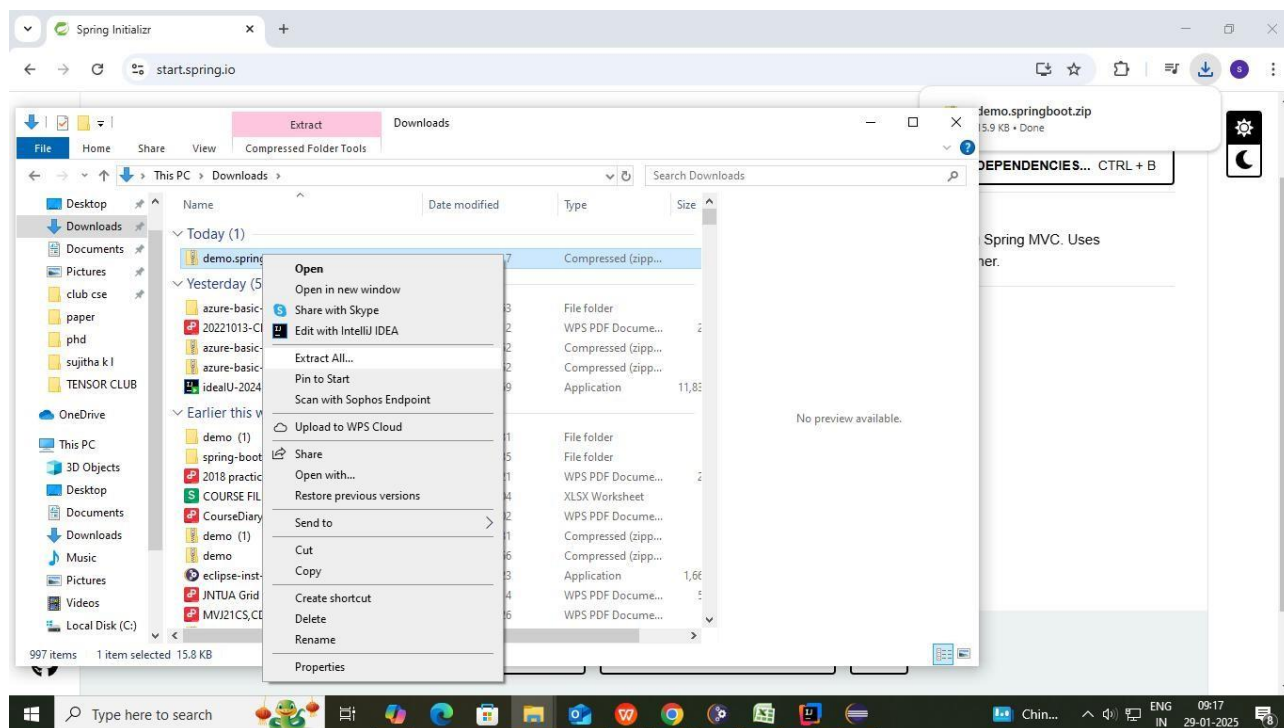


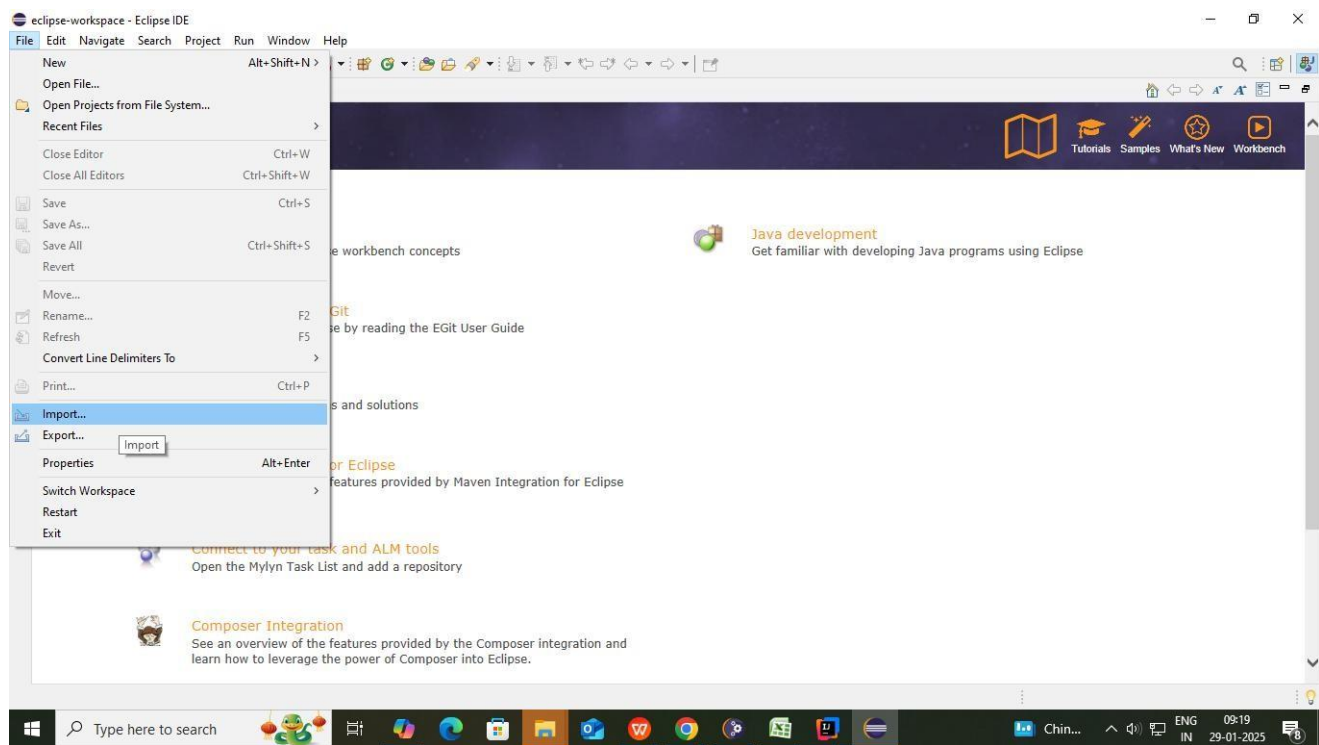
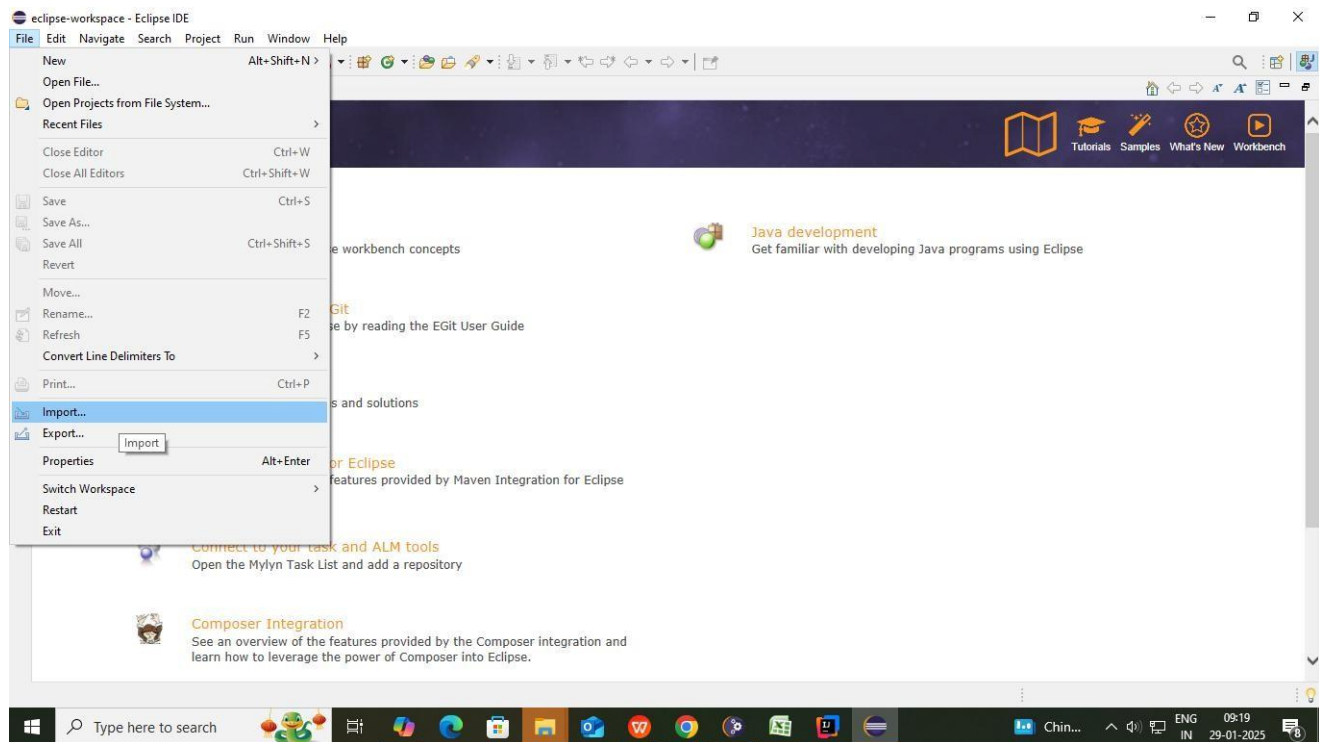


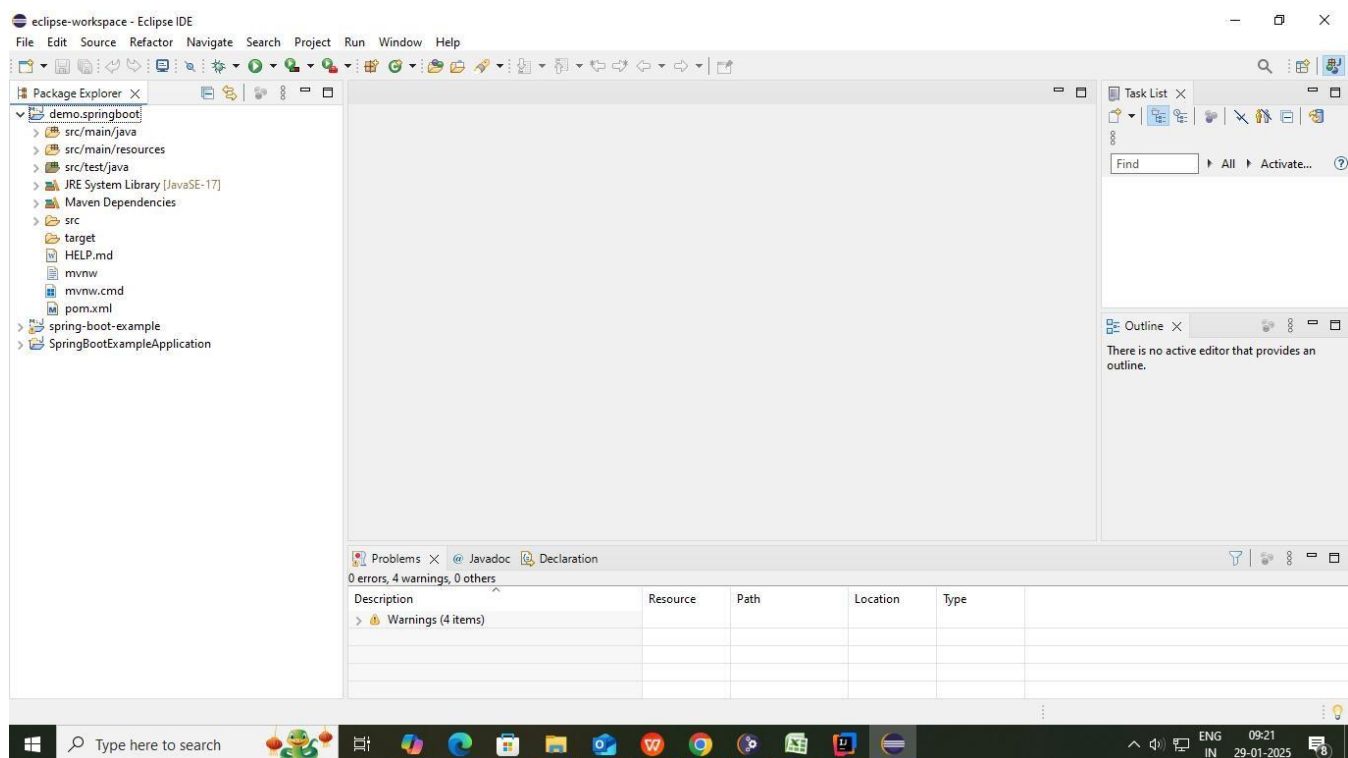
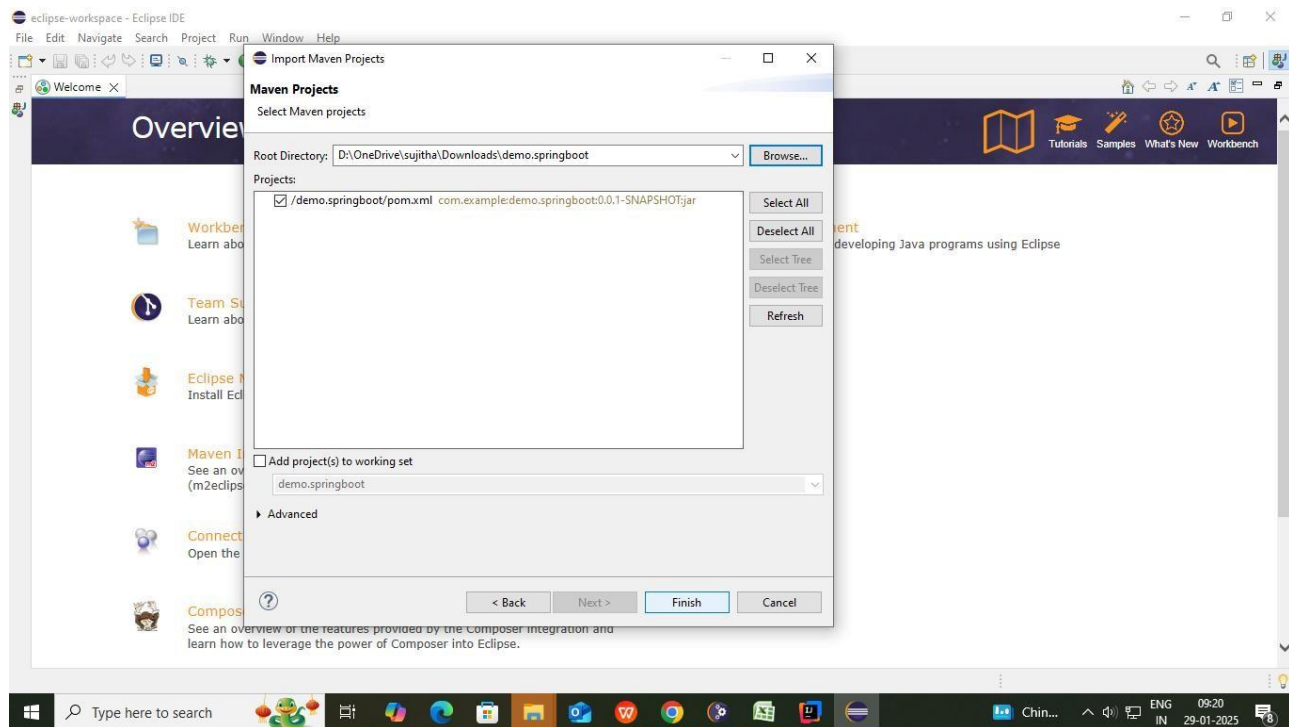
Open [start.spring.io](https://start.spring.io) in web Browser choose options which is selected and click **GENERATE CTRL**

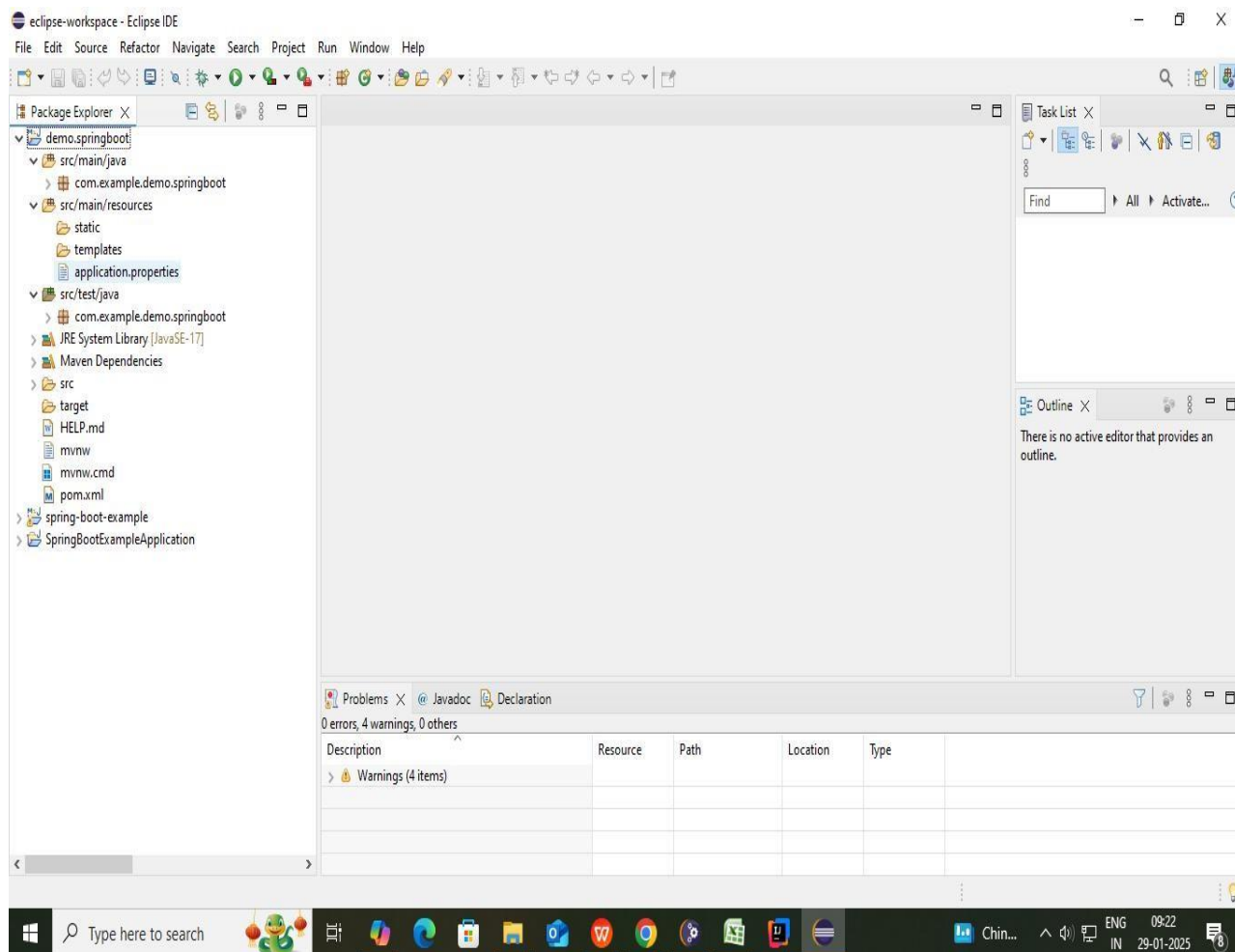


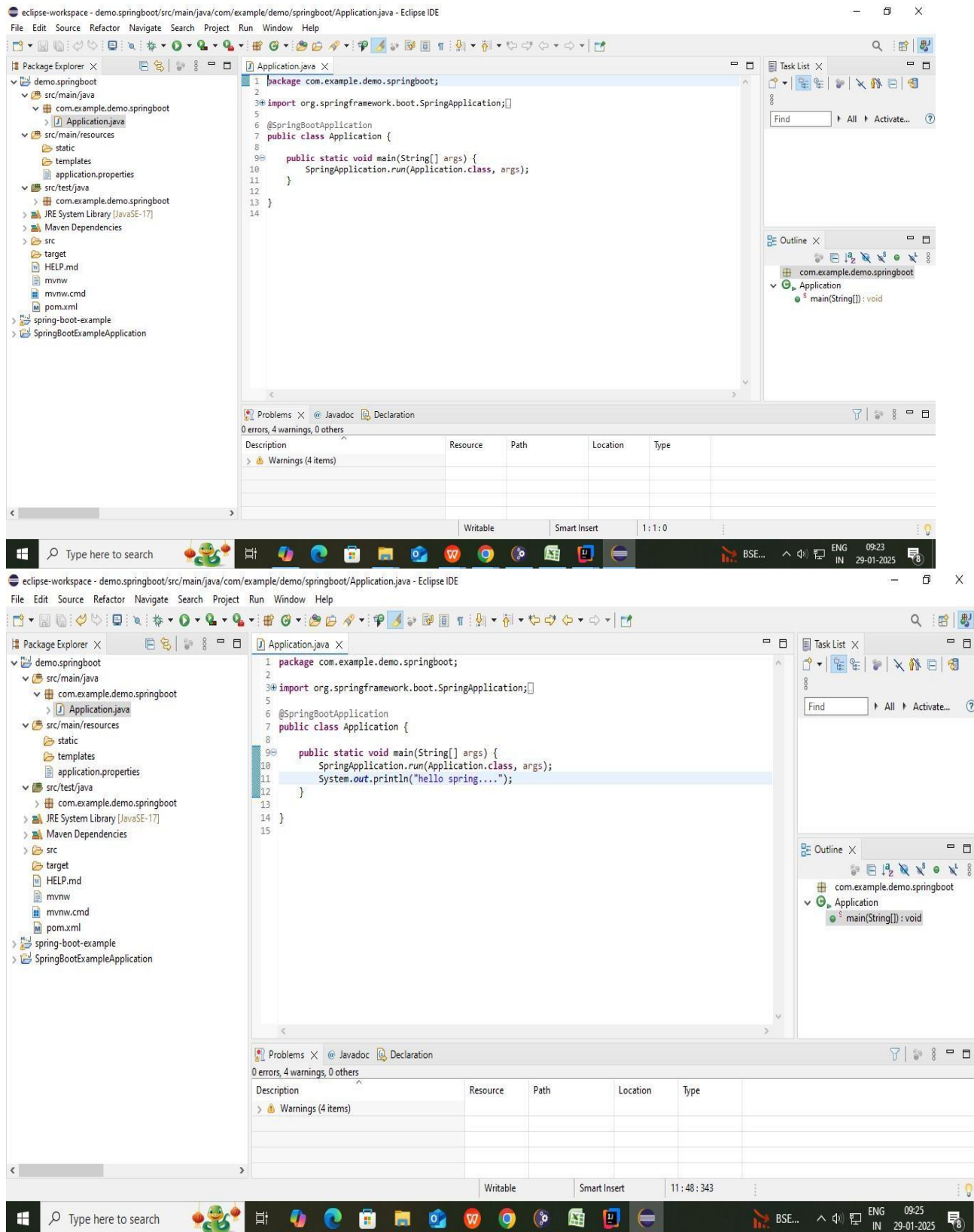
Unzip the folder

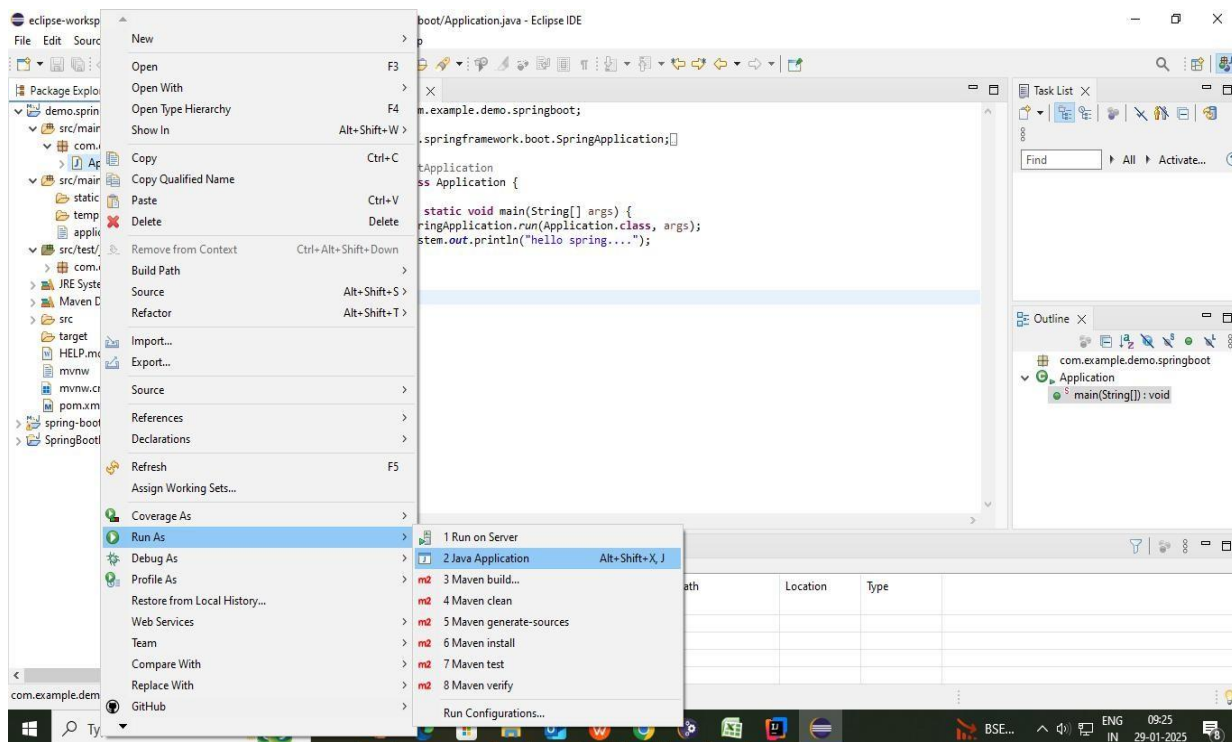




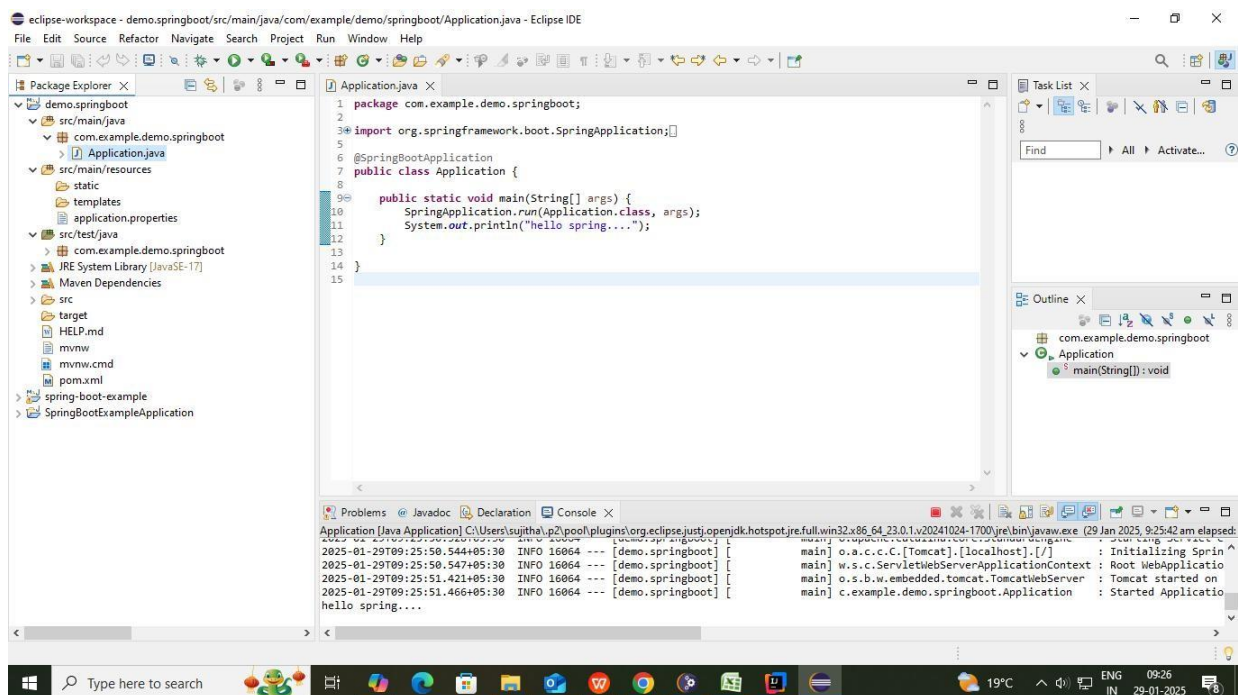


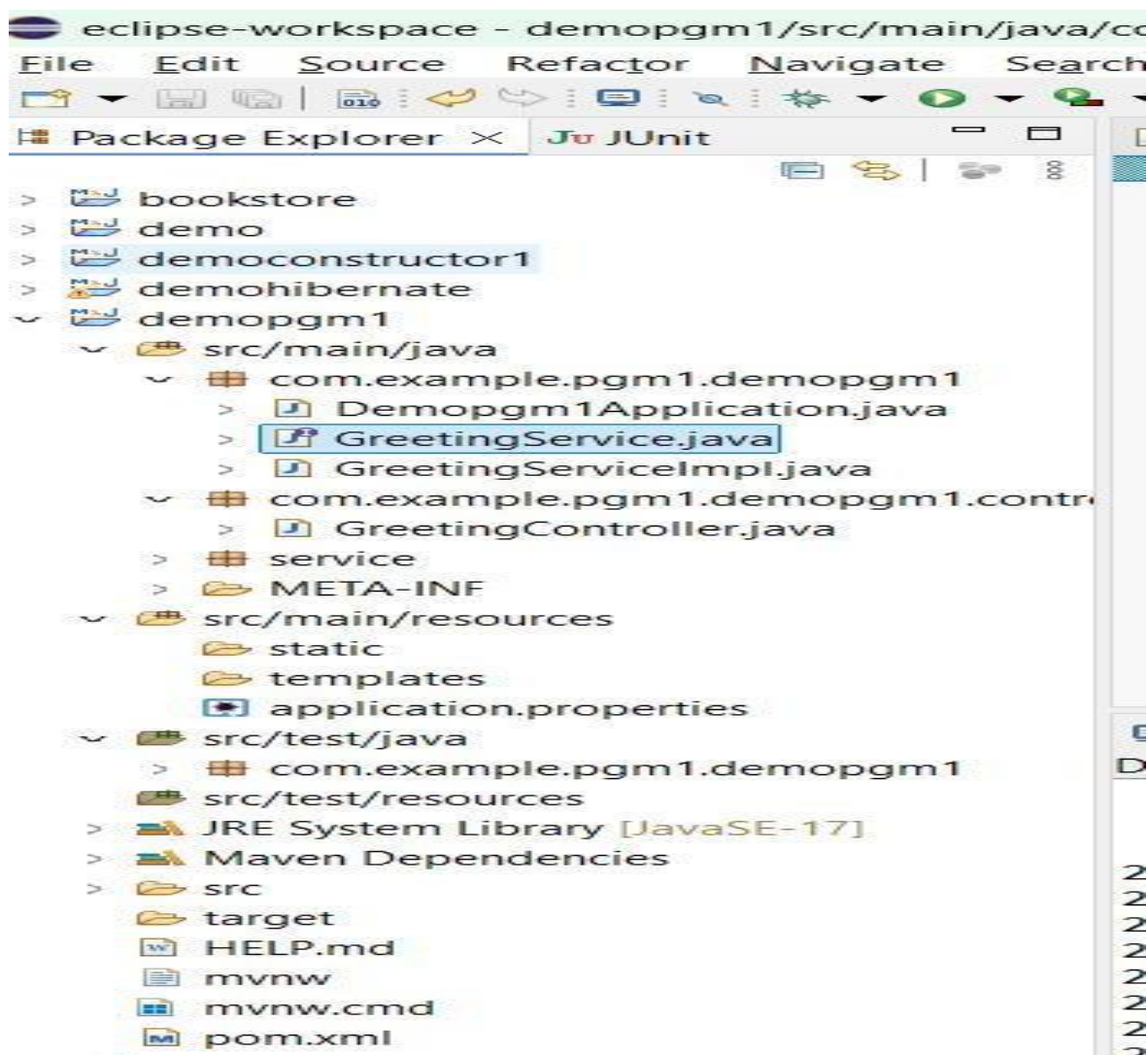






## OUTPUT:



**1. Demonstrate Dependency Injection using annotation based using Spring boot**

## 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 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.4.2</version>
<relativePath/><!-- lookup parent from repository -->
</parent>
<groupId>com.example.pgm1</groupId>
<artifactId>demopgm1 </artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>demopgm1 </name>
<description>Demo project for Spring Boot</description>
<url/>
<licenses>

<license/>
</licenses>
<developers>
<developer/>
</developers>
<scm>
<connection/>
<developerConnection/>
<tag/>
<url/>
</scm>
<properties>
<java.version>17</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-devtools</artifactId>
<scope>runtime</scope>
<optional>true</optional>
</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>
```

**Demopgm1Application.java:**

```
package com.example.pgm1.demopgm1;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class Demopgm1Application {

    public static void main(String[] args) {
        SpringApplication.run(Demopgm1Application.class, args);
    }

}
```

**GreetingService.java -interface**

```
package com.example.pgm1.demopgm1;
```

```
publicinterface GreetingService {
    String greet();
}
```

**GreetingServiceImpl.java**

```
package com.example.pgm1.demopgm1;
```

```
import org.springframework.stereotype.Service;
```

```
@Service
publicclass GreetingServiceImpl implements GreetingService {
    @Override
    public String greet() {
        return "Hello, World!";
    }
}
```

**GreetingController.java**

```
package com.example.pgm1.demopgm1.controller;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import com.example.pgm1.demopgm1.GreetingService;
```

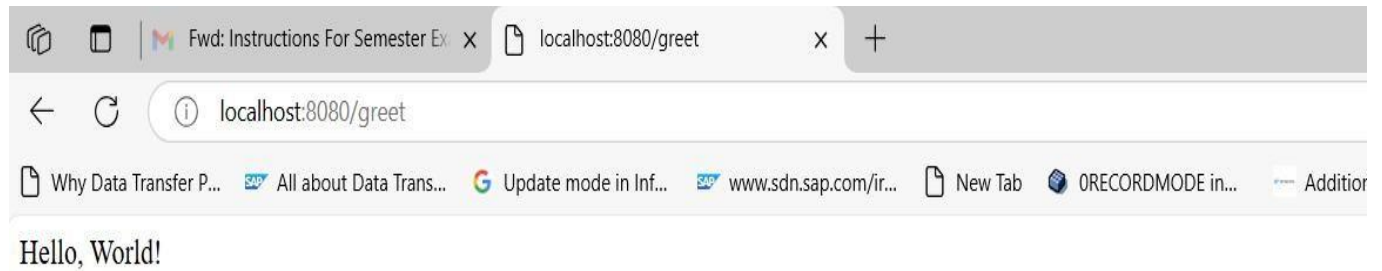
```
@RestController
publicclass GreetingController {

    privatefinal GreetingService greetingService;
```

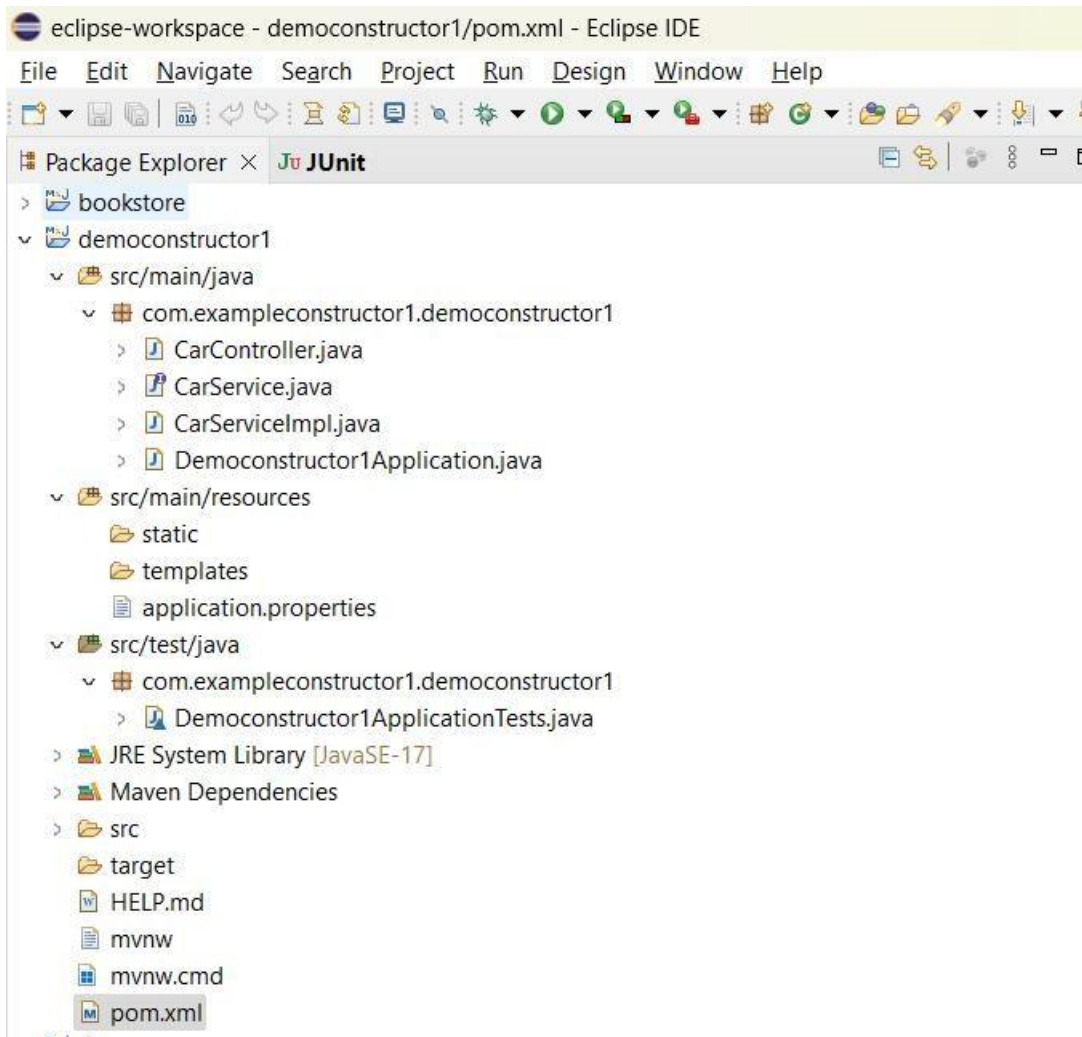
```
@Autowired
public GreetingController(GreetingService greetingService) {
    this.greetingService = greetingService;
}
```

```
@GetMapping("/greet")
public String greet() {
    return greetingService.greet();
}
```

## **OUTPUT:**



## 2. Demonstrate Dependency Injection using constructor based using Spring boot



### 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 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.4.2</version>
<relativePath/><!-- lookup parent from repository -->
</parent>
<groupId>com.exampleconstructor1</groupId>
<artifactId>democonstructor1</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>democonstructor1</name>
<description>Demo project for Spring Boot constructor</description>
<url/>
```

```
<licenses>
<license/>
</licenses>

<developers>
<developer/>
</developers>

<scm>
<connection/>
<developerConnection/>
<tag/>
<url/>
</scm>
<properties>
<java.version>17</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>
```

### **CarController**

```
package com.example.constructor1.democonstructor1;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CarController {

    private final CarService carService;

    @Autowired
    public CarController(CarService carService) {
        this.carService = carService;
    }
}
```

```
@GetMapping("/start")
public String startCar() {
    carService.startCar();
    return "Car started!";
}
}
```

## CarService

```
package com.exampleconstructor1.democonstructor1;
```

```
publicinterface CarService {
void startCar();
}
```

## CarServiceImpl

```
package com.exampleconstructor1.democonstructor1;
```

```
import org.springframework.stereotype.Service;
```

```
@Service
```

```
publicclass CarServiceImpl implements CarService {
```

```
@Override
```

```
publicvoid startCar() {
```

```
// TODO Auto-generated method stub
```

```
}
```

```
publicstaticvoid main(String[] args) {
```

```
// TODO Auto-generated method stub
```

```
System.out.println("Car is starting...");
```

```
}
```

```
}
```

```
package com.exampleconstructor1.democonstructor1;
```

```
import org.springframework.boot.SpringApplication;
```

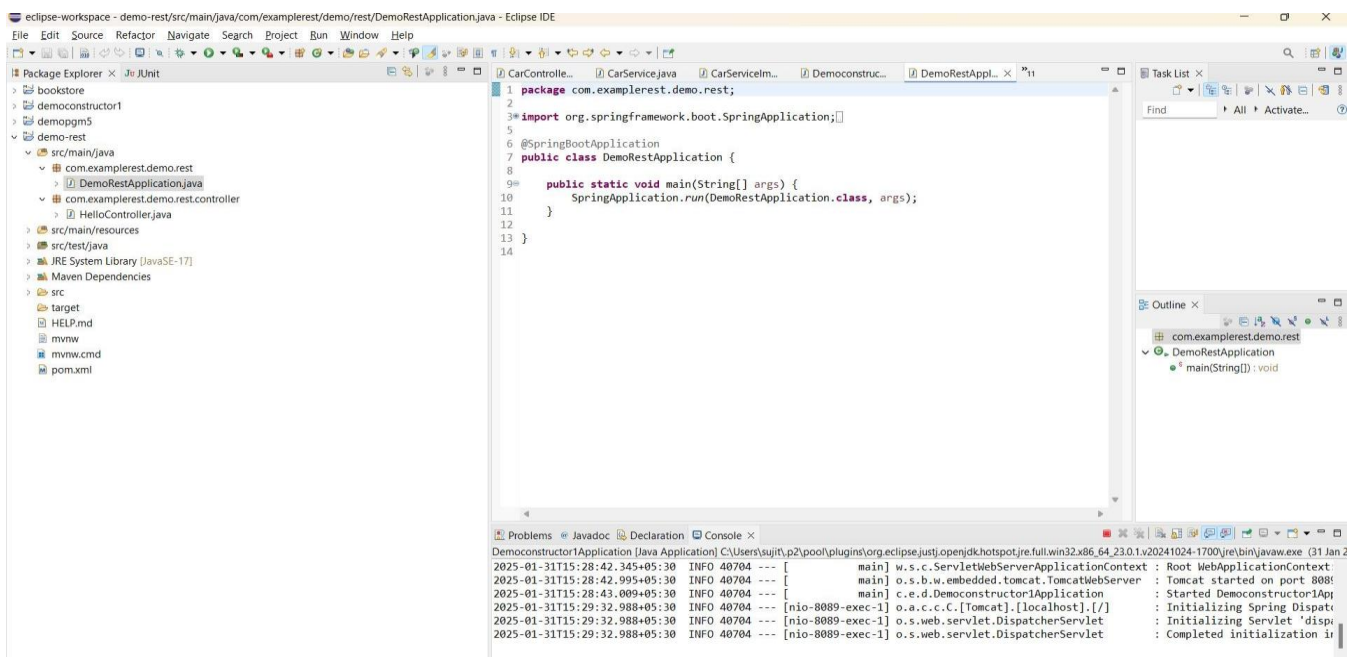
```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class Democonstructor1Application {

    public static void main(String[] args) {
        SpringApplication.run(Democonstructor1Application.class, args);
    }
}
```

### **3. Create a Spring Boot Application using Maven Plugin - Write a sample REST Controller API using Spring Annotations - Using Postman invoke the REST Controller to demonstrate end to end working.**

DemoRestApplication.java  
package com.examplerest.demo.rest;



```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

```

```

@SpringBootApplication
public class DemoRestApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoRestApplication.class, args);
    }

}

```

HelloController.java  
package com.examplerest.demo.rest.controller;

```

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

```

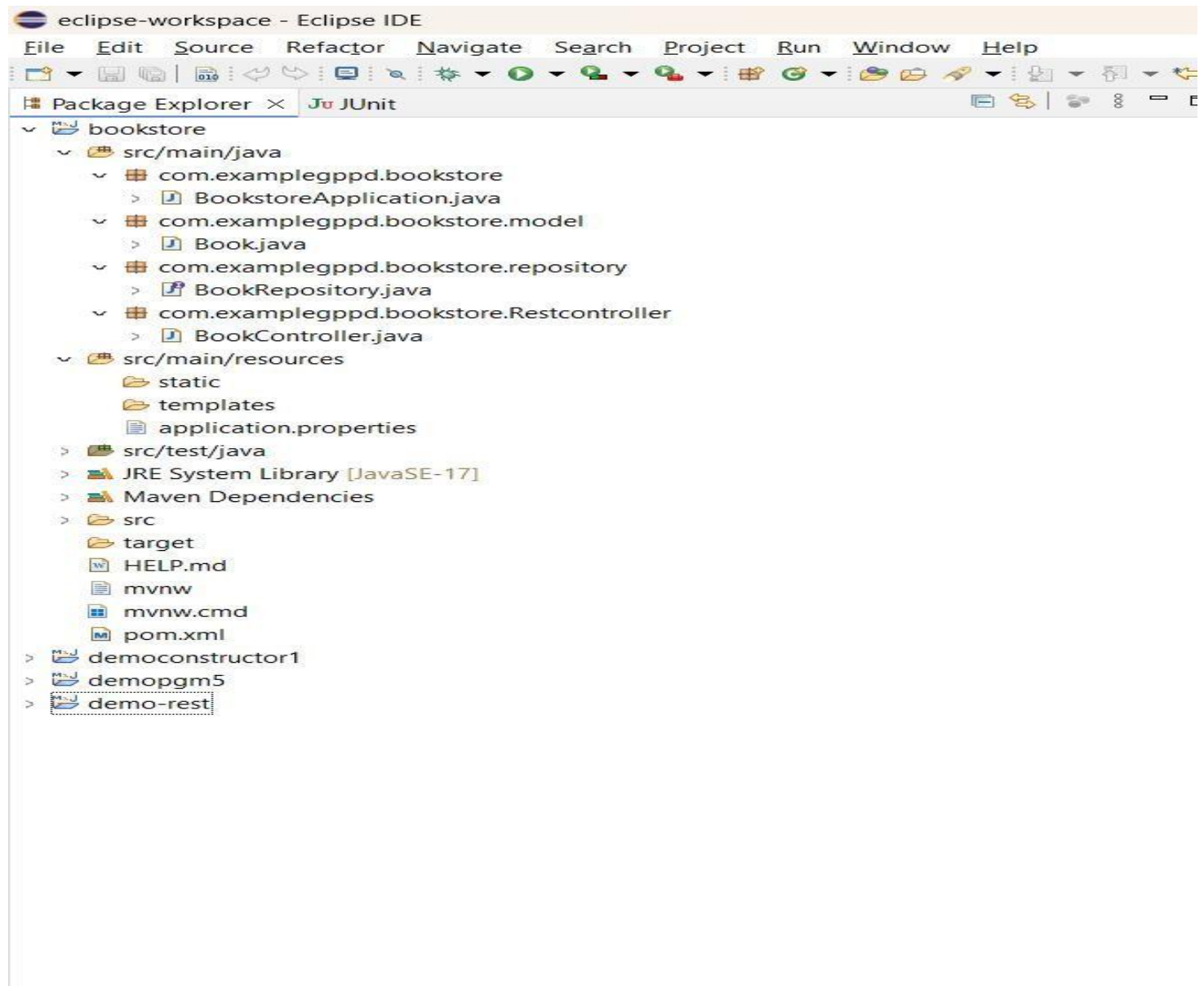
```

@RestController
@RequestMapping("/api")
public class HelloController {

```

```
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, World!";
    }
}
```

**4. Write a sample REST App to demonstrate below Concepts with a use-case of your choice. - GET, PUT, POST, DELETE.**



**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 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.4.2</version>
<relativePath/><!-- lookup parent from repository -->
</parent>
<groupId>com.examplegppd</groupId>
<artifactId>bookstore</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

```
<name>bookstore</name>
<description>Demo project for Spring Boot rest</description>
<url/>
<licenses>
<license/>
</licenses>
<developers>
<developer/>
</developers>
<scm>
<connection/>
<developerConnection/>
<tag/>
<url/>
</scm>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
<scope>runtime</scope>
</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>
```

### BookstoreApplication

```
package com.examplegppd.bookstore;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

### @SpringBootApplication

```
public class BookstoreApplication {

    public static void main(String[] args) {
        SpringApplication.run(BookstoreApplication.class, args);
    }
}
```

```
}
```

Book

```
package com.examplegppd.bookstore.model;
```

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

```
@Entity
```

```
publicclass Book {
```

```
@Id
```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
private Long id;
```

```
private String title;
```

```
private String author;
```

```
public Long getId() {
```

```
returnid;
```

```
}
```

```
publicvoid setId(Long id) {
```

```
this.id = id;
```

```
}
```

```
public String getTitle() {
```

```
returntitle;
```

```
}
```

```
publicvoid setTitle(String title) {
```

```
this.title = title;
```

```
}
```

```
public String getAuthor() {
```

```
returnauthor;
```

```
}
```

```
publicvoid setAuthor(String author) {
```

```
this.author = author;
```

```
}
```

```
}
```

```
// Getters and Setters
```

```
    BookRepository.java
```

```
package com.examplegppd.bookstore.repository;
```

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

```
import com.examplegppd.bookstore.model.Book;
```

```
public interface BookRepository extends JpaRepository<Book,Long> {
```

```
}
```

BookController

package com.examplegppd.bookstore.Restcontroller;

import java.util.List;

import java.util.Optional;

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.examplegppd.bookstore.model.Book;

import com.examplegppd.bookstore.repository.BookRepository;

@RestController

@RequestMapping("/api/books")

public class BookController {

    @Autowired

    private BookRepository bookRepository;

    @GetMapping

    public List<Book> getAllBooks() {

        return bookRepository.findAll();

    }

    @GetMapping("/{id}")

    public ResponseEntity<Book> getBookById(@PathVariable Long id) {

        Optional<Book> book = bookRepository.findById(id);

        return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

    }

    @PostMapping

    public Book createBook(@RequestBody Book book) {

        return bookRepository.save(book);

    }

    @PutMapping("/{id}")

    public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {

        Optional<Book> book = bookRepository.findById(id);

        if (book.isPresent()) {

            Book updatedBook = book.get();

            updatedBook.setTitle(bookDetails.getTitle());

            updatedBook.setAuthor(bookDetails.getAuthor());

            bookRepository.save(updatedBook);

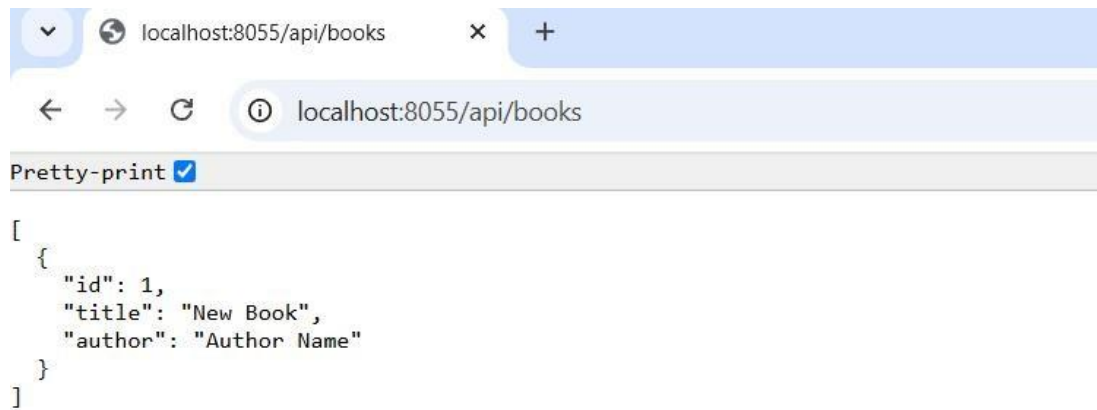
            return ResponseEntity.ok(updatedBook);

        } else {

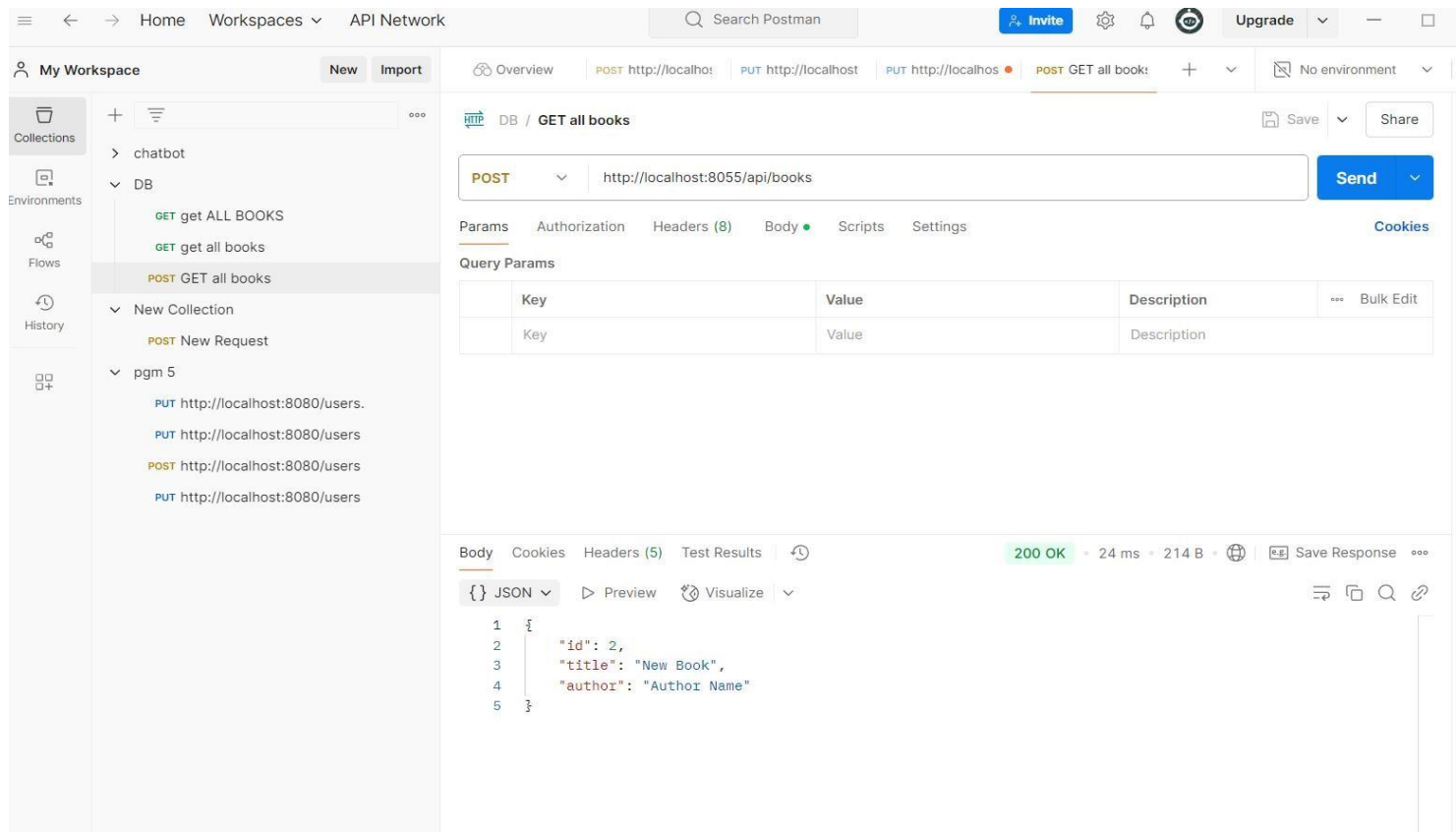
            return ResponseEntity.notFound().build();

```
    }  
}  
  
@DeleteMapping("/{id}")  
public ResponseEntity<Void> deleteBook(@PathVariable Long id) {  
    Optional<Book> book = bookRepository.findById(id);  
    if (book.isPresent()) {  
        bookRepository.delete(book.get());  
        return ResponseEntity.noContent().build();  
    } else {  
        return ResponseEntity.notFound().build();  
    }  
}
```

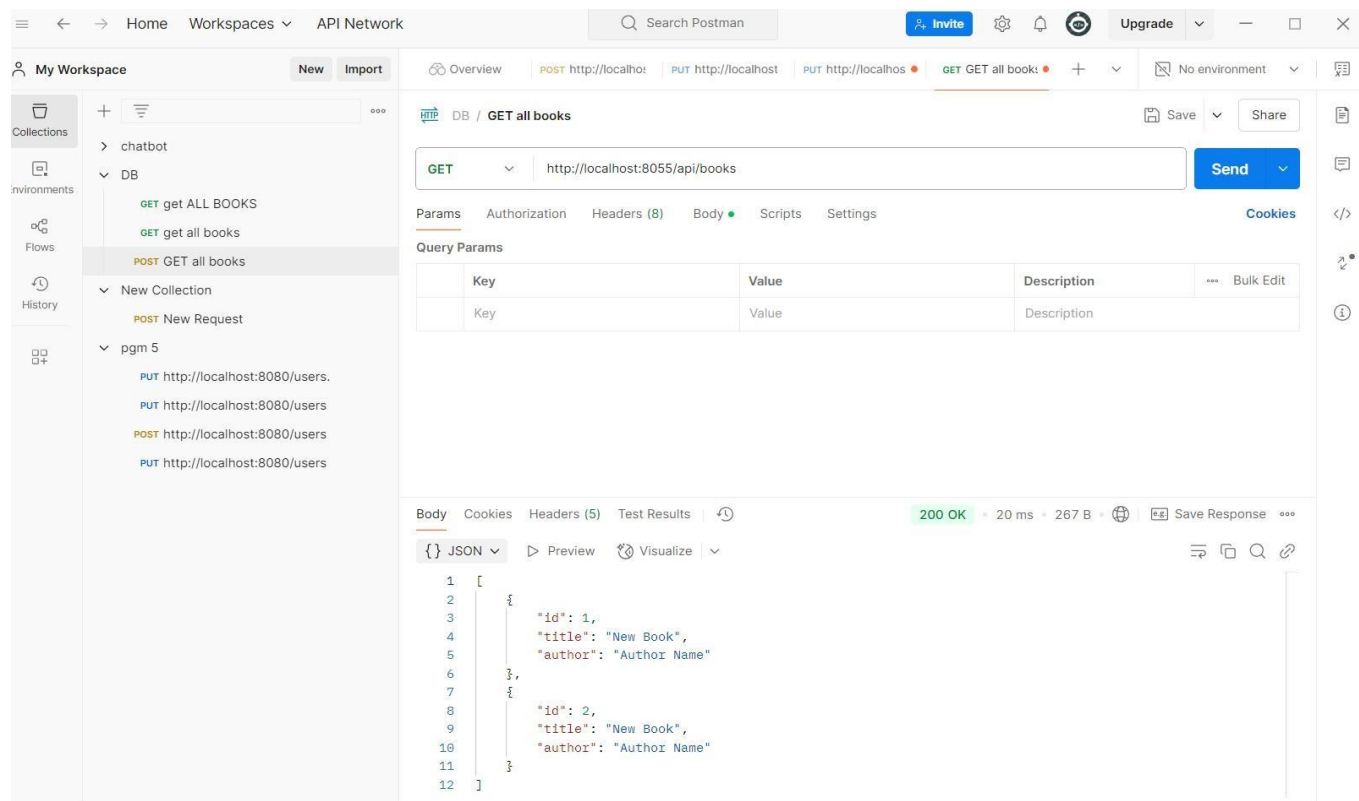
BROWSER:



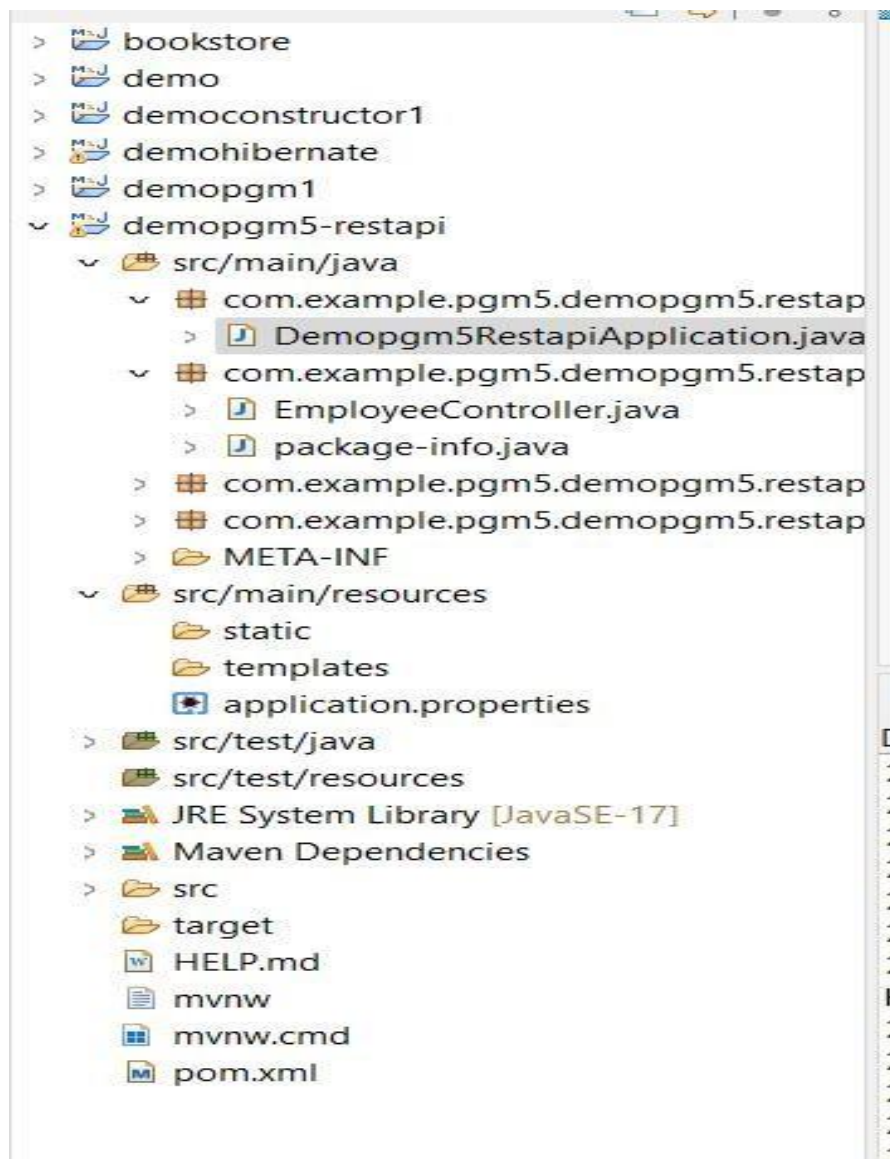
## POSTMAN APP:POSTM



GET:



**5. Write a sample REST App to Validate the REST API POST & PUT request. -Design a custom response with appropriate validation errors to the caller**



```
<?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>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.4.2</version>
<relativePath/><!-- lookup parent from repository -->
</parent>
<groupId>com.example.pgm5</groupId>
<artifactId>demopgm5-restapi</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>demopgm5-restapi</name>
<description>Demo project for Spring Boot</description>
<url/>
```

```
<licenses>
<license/>
</licenses>
<developers>
<developer/>
</developers>
<scm>
<connection/>
<developerConnection/>
<tag/>
<url/>
</scm>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
<scope>runtime</scope>
</dependency>

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

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
<optional>true</optional>
</dependency>
<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</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>
```

```
package com.example.pgm5.demopgm5.restapi;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Demopgm5RestapiApplication {

    public static void main(String[] args) {
        SpringApplication.run(Demopgm5RestapiApplication.class, args);
    }

}

package com.example.pgm5.demopgm5.restapi.controller;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.example.pgm5.demopgm5.restapi.jpaprepository.EmployeeRepository;
import com.example.pgm5.demopgm5.restapi.model.Employee;
import jakarta.validation.Valid;

@RestController
@RequestMapping("/api/employees")
public class EmployeeController {

    @Autowired
    private EmployeeRepository employeeRepository;

    @PostMapping
    public ResponseEntity<?> createEmployee(@Valid @RequestBody Employee employee, BindingResult
result) {
        if (result.hasErrors()) {
            Map<String, String> errors = new HashMap<>();
            for (FieldError error : result.getFieldErrors()) {
```

```
        errors.put(error.getField(), error.getDefaultMessage());
    }
    return new ResponseEntity<>(errors, HttpStatus.BAD_REQUEST);
}
Employee savedEmployee = employeeRepository.save(employee);
return new ResponseEntity<>(savedEmployee, HttpStatus.CREATED);
}

@PutMapping("/{id}")
public ResponseEntity<?> updateEmployee(@PathVariable Long id, @Valid @RequestBody Employee
employee, BindingResult result) {
    if (result.hasErrors()) {
        Map<String, String> errors = new HashMap<>();
        for (FieldError error : result.getFieldErrors()) {
            errors.put(error.getField(), error.getDefaultMessage());
        }
        return new ResponseEntity<>(errors, HttpStatus.BAD_REQUEST);
    }
    return employeeRepository.findById(id).map(existingEmployee -> {
        existingEmployee.setName(employee.getName());
        existingEmployee.setAge(employee.getAge());
        existingEmployee.setEmail(employee.getEmail());
        Employee updatedEmployee = employeeRepository.save(existingEmployee);
        return new ResponseEntity<>(updatedEmployee, HttpStatus.OK);
    }).orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));
}

@GetMapping
public List<Employee> getAllEmployees() {
    return employeeRepository.findAll();
}
}
```

```
package com.example.pgm5.demopgm5.restapi.jparespository;
```

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

```
import com.example.pgm5.demopgm5.restapi.model.Employee;
```

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}
```

```
package com.example.pgm5.demopgm5.restapi.model;
```

```
import jakarta.persistence.Entity;
```

```
import jakarta.persistence.GeneratedValue;
```

```
import jakarta.persistence.GenerationType;
```

```
import jakarta.persistence.Id;
```

```
import jakarta.persistence.Table;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotEmpty;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;

@Entity
@Table(name = "employees")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotEmpty(message = "Name is required")
    @Size(min = 2, message = "Name should have at least 2 characters")
    private String name;

    @NotNull(message = "Age is required")
    private Integer age;

    @NotEmpty(message = "Email is required")
    @Email(message = "Email should be valid")
    private String email;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

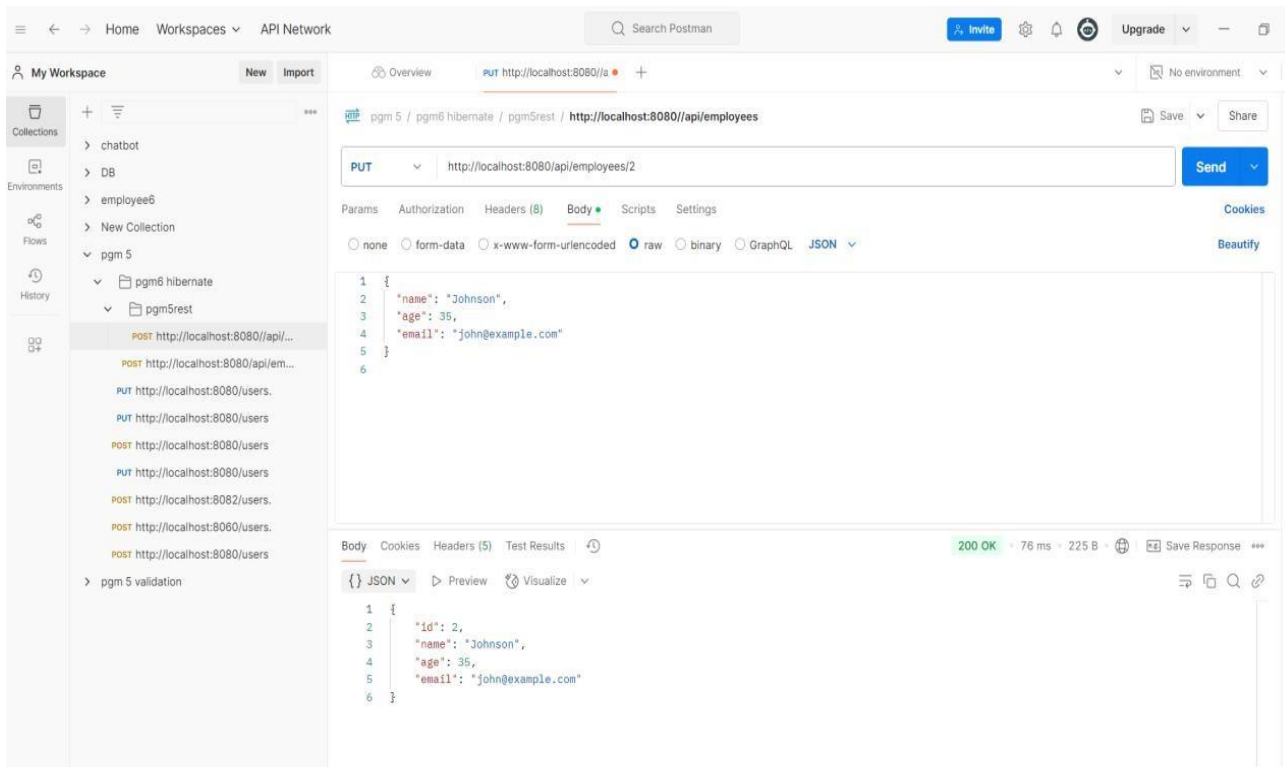
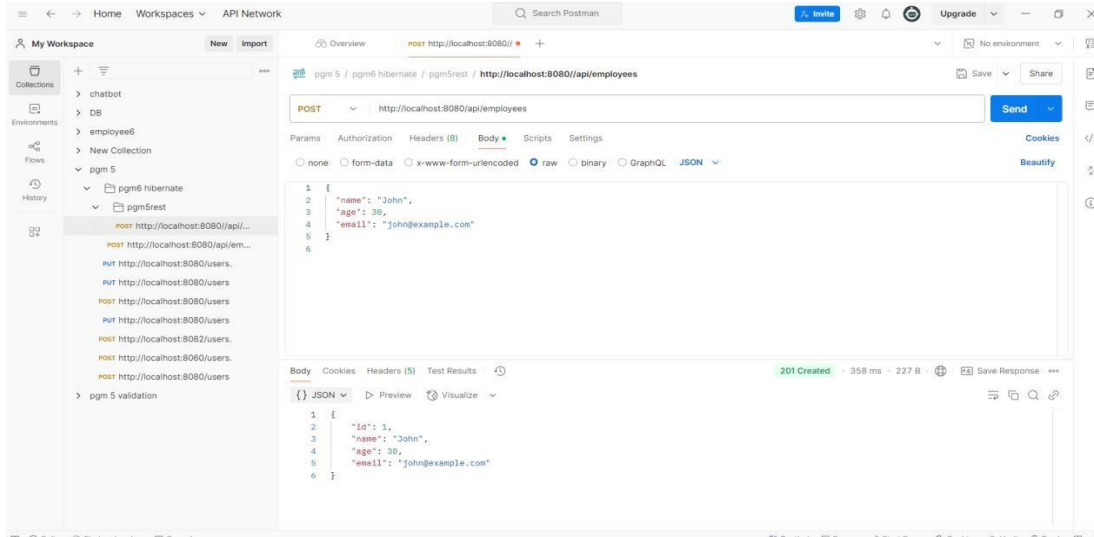
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

```
// Getters and setters
```

```
}
```

## OUTPUT:



The image shows a Postman interface for a REST client. The workspace is named "My Workspace" and contains a collection named "pgm 5". The selected request is a POST request to "http://localhost:8080/api/employees". The request body is a JSON object: 

```
{  "name": "",  "age": 35,  "email": "john@example.com"}
```

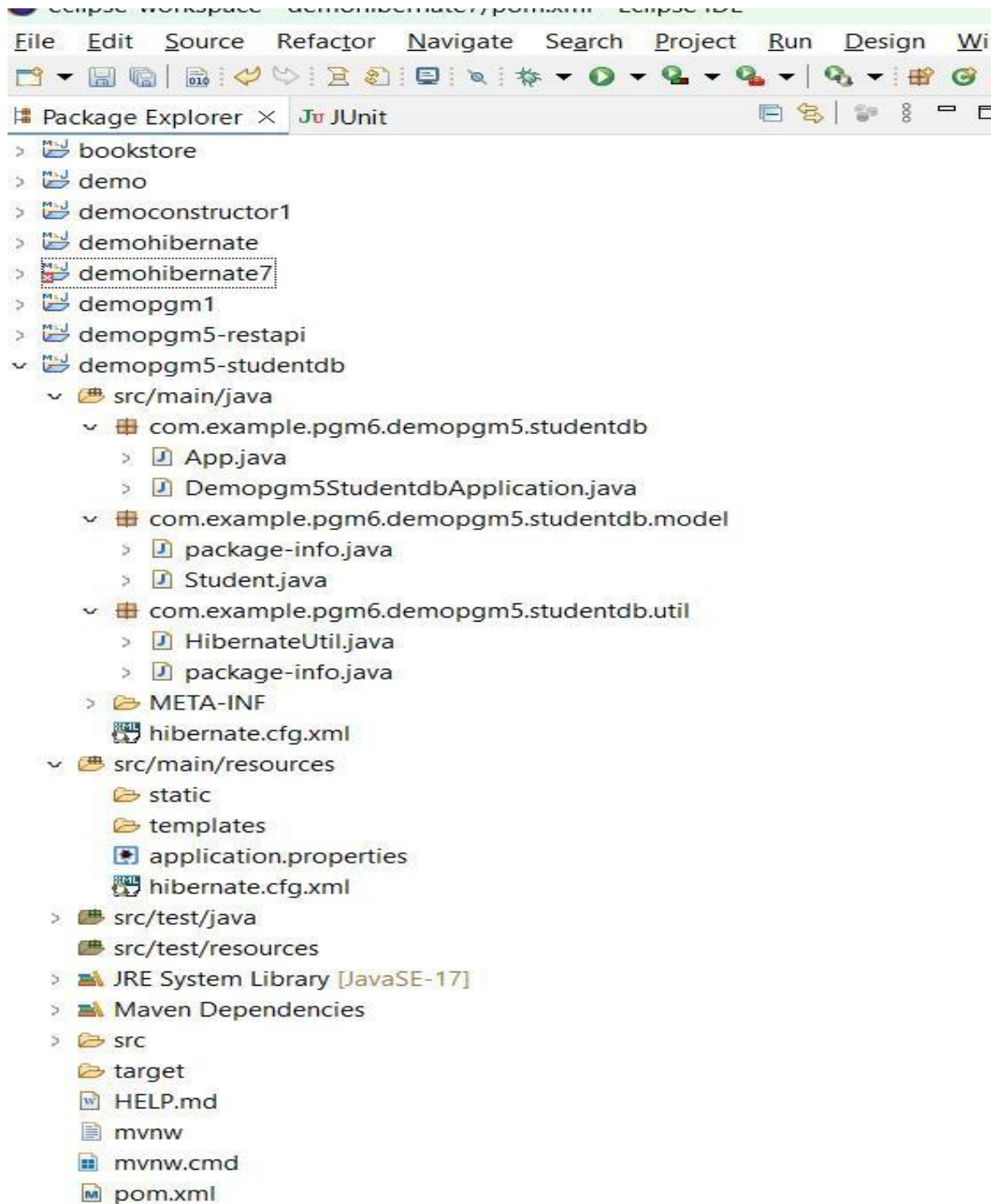
. The response is a 400 Bad Request, with a message: 

```
{  "name": "Name should have at least 2 characters"}
```

. Below the Postman interface, a web browser window shows the URL "localhost:8080/api/employees" and the response body, which is a JSON array of two employee objects: 

```
[  {    "id": 1,    "name": "John",    "age": 30,    "email": "john@example.com"  },  {    "id": 2,    "name": "Johnson",    "age": 35,    "email": "john@example.com"  }]
```

**6. Write a Java application using Hibernate to insert data into Student DATABASE and retrieve info based on particular queries (For example update, delete, search etc...)**



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 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
```

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.4.2</version>

<relativePath/><!-- lookup parent from repository -->
</parent>
<groupId>com.example.pgm6</groupId>
<artifactId>demopgm5-studentdb</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>demopgm5-studentdb</name>
<description>Demo project for Spring Boot</description>
<url/>
<licenses>
<license/>
</licenses>
<developers>
<developer/>
</developers>
<scm>
<connection/>
<developerConnection/>
<tag/>
<url/>
</scm>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-jersey</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
<optional>true</optional>
</dependency>
<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>javax.xml.bind</groupId>
```

```
<artifactId>jaxb-api</artifactId>
<version>2.3.1</version>
</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>

package com.example.pgm6.demopgm5.studentdb;

import org.hibernate.Session;
import org.hibernate.Transaction;

import com.example.pgm6.demopgm5.studentdb.model.Student;
import com.example.pgm6.demopgm5.studentdb.util.HibernateUtil;

public class App {
    public static void main(String[] args) {
        App app = new App();
        Student student = new Student();
        student.setId(1L);
        student.setName("John ");
        student.setAge(20);
        app.saveStudent(student);
        app.updateStudent(student.getId(), "Jane ", 21); // Pass correct arguments
        app.getStudent(1L);
        app.deleteStudent(1L);
    }
    @SuppressWarnings("deprecation")
    public void saveStudent(Student student) {
        System.out.println("Attempting to save student...");
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction transaction = null;
        try {
            transaction = session.beginTransaction();
            session.save(student);
            transaction.commit();
            System.out.println("Student saved successfully!");
        } catch (Exception e) {
            if (transaction != null && transaction.isActive()) {
                transaction.rollback();
            }
            e.printStackTrace();
        } finally {
            if (session != null && session.isOpen()) {
                session.close();
            }
        }
    }
}
```

```
}  
}  
  
@SuppressWarnings("deprecation")  
public void updateStudent(Long id, String name, int age) {  
  
    System.out.println("Attempting to update student...");  
    Session session = HibernateUtil.getSessionFactory().openSession();  
    Transaction transaction = null;  
    try {  
        transaction = session.beginTransaction();  
        Student student = session.get(Student.class, id);  
        if (student != null) {  
            student.setName(name);  
            student.setAge(age);  
            session.update(student);  
            transaction.commit();  
            System.out.println("Student updated successfully!");  
        } else {  
            System.out.println("Student not found with ID: " + id);  
        }  
    } catch (Exception e) {  
        if (transaction != null && transaction.isActive()) {  
            transaction.rollback();  
        }  
        e.printStackTrace();  
    } finally {  
        if (session != null && session.isOpen()) {  
            session.close();  
        }  
    }  
}  
  
public void getStudent(Long id) {  
    System.out.println("Attempting to retrieve student...");  
    Session session = HibernateUtil.getSessionFactory().openSession();  
    try {  
        Student student = session.get(Student.class, id);  
        if (student != null) {  
            System.out.println("Student found: " + student.getName());  
        } else {  
            System.out.println("Student not found with ID: " + id);  
        }  
    } finally {  
        session.close();  
    }  
}  
  
@SuppressWarnings("deprecation")  
public void deleteStudent(Long id) {  
    System.out.println("Attempting to delete student...");  
    Session session = HibernateUtil.getSessionFactory().openSession();  
    Transaction transaction = null;  
    try {  
        transaction = session.beginTransaction();  
        Student student = session.get(Student.class, id);  
        if (student != null) {  
            session.delete(student);  
            transaction.commit();  
            System.out.println("Student deleted successfully!");  
        } else {  
            System.out.println("Student not found with ID: " + id);  
        }  
    }  
}
```

```
catch (Exception e) {
    if (transaction != null&&transaction.isActive()) {
        transaction.rollback();
    }
    e.printStackTrace();
} finally {
    if (session != null&&session.isOpen()) {

        session.close();
    }
}
}
```

```
package com.example.pgm6.demopgm5.studentdb;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Demopgm5StudentdbApplication {

    public static void main(String[] args) {
        SpringApplication.run(Demopgm5StudentdbApplication.class, args);
    }

}
```

```
package com.example.pgm6.demopgm5.studentdb.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "students")
public class Student {
    @Id
    private Long id;
    private String name;
    private int age;
    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
}
```

```
package com.example.pgm6.demopgm5.studentdb.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
```

```
private static SessionFactory sessionFactory;
```

```
static {  
    try {  
        sessionFactory = new Configuration().configure().buildSessionFactory();  
    } catch (Throwable ex) {  
        throw new ExceptionInInitializerError(ex);  
    }  
}  
  
public static SessionFactory getSessionFactory() {  
    return sessionFactory;  
}  
  
public static void shutdown() {  
    getSessionFactory().close();  
}  
}
```

Go to file->new->other->expand hibernate->choose cfg.xml [Hibernate.cfg.xml](#)

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE hibernate-configuration PUBLIC  
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"  
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  
<hibernate-configuration>  
    <session-factory>  
        <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>  
        <property name="hibernate.connection.password">1987</property>  
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mydb</property>  
        <property name="hibernate.connection.username">suji</property>  
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>  
    </session-factory>  
</hibernate-configuration>
```

Application properties:

# Database connection settings

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

spring.datasource.username=suji

spring.datasource.password=yourpassword

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

# Hibernate settings

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Server settings (optional)

server.port=8080

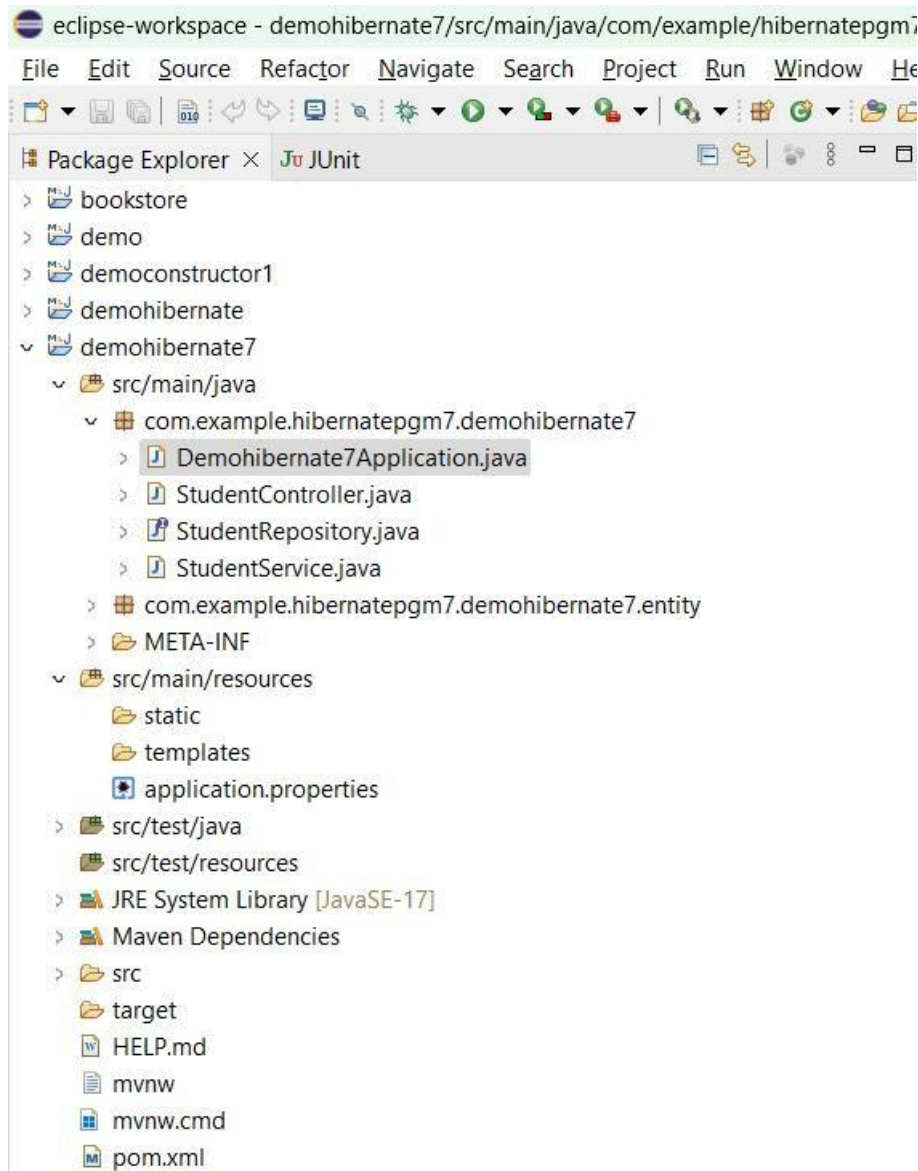
server.error.include-message=always

Custom application settings

(optional)

app.name=HibernateExampleApp

## 7. Demonstrate Spring Data JPA integration in a Spring Boot application using Hibernate



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>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.4.2</version>
<relativePath><!-- lookup parent from repository -->
</parent>
<groupId>com.example.hibernatepgm7</groupId>
<artifactId>demohibernate7</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

```
<name>demohibernate7</name>
<description>Demo project for Spring Boot</description>
<url/>
<licenses>
<license/>
</licenses>
<developers>
<developer/>
</developers>
<scm>
<connection/>
<developerConnection/>
<tag/>
<url/>
</scm>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency><groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.29</version></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>
```

### **Demohibernate7Application**

```
package com.example.hibernatepgm7.demohibernate7;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class Demohibernate7Application {

    public static void main(String[] args) {
        SpringApplication.run(Demohibernate7Application.class, args);
    }

}
```

### **StudentController**

```
package com.example.hibernatepgm7.demohibernate7;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
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.example.hibernatepgm7.demohibernate7.entity.Student;

@RestController
@RequestMapping("/students")
public class StudentController {
    @Autowired
    private StudentService studentService;

    @PostMapping
    public Student createStudent(@RequestBody Student student) {
        return studentService.saveStudent(student);
    }

    @GetMapping("/{id}")
    public Student getStudent(@PathVariable Long id) {
        return studentService.getStudent(id);
    }

    @GetMapping
    public List<Student> getAllStudents() {
        return studentService.getAllStudents();
    }

    @DeleteMapping("/{id}")
    public void deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id);
    }
}
```

```
}
```

### **StudentRepository**

```
package com.example.hibernatepgm7.demohibernate7;

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

import com.example.hibernatepgm7.demohibernate7.entity.Student;

public interface StudentRepository extends JpaRepository<Student, Long> {
}
```

### **StudentService**

```
package com.example.hibernatepgm7.demohibernate7;

import java.util.List;

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

import com.example.hibernatepgm7.demohibernate7.entity.Student;

@Service
publicclass StudentService {
    @Autowired
    private StudentRepository studentRepository;

    public Student saveStudent(Student student) {
        returnstudentRepository.save(student);
    }

    public Student getStudent(Long id) {
        returnstudentRepository.findById(id).orElse(null);
    }

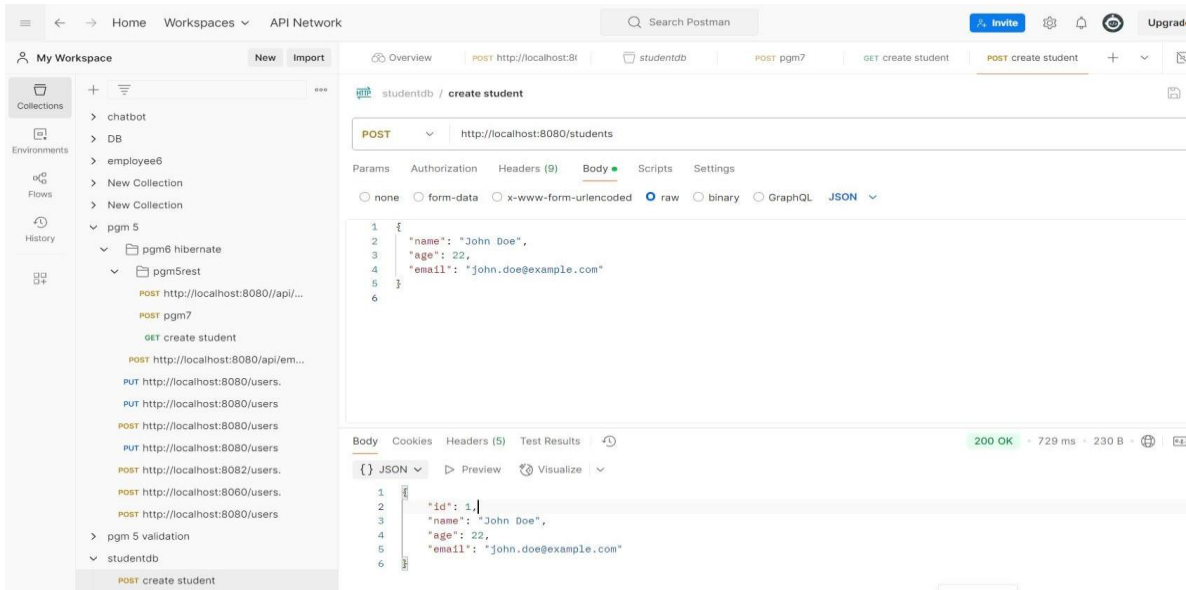
    public List<Student> getAllStudents() {
        returnstudentRepository.findAll();
    }

    publicvoid deleteStudent(Long id) {
        studentRepository.deleteById(id);
    }
}
```

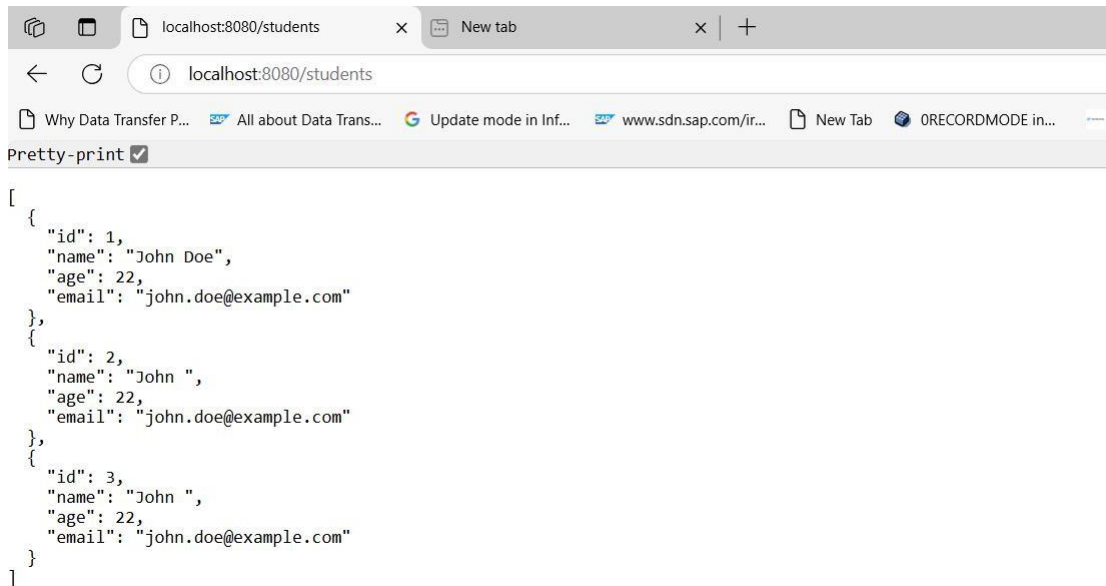
### **application.properties:**

```
spring.datasource.url=jdbc:mysql://localhost:3306/studentdb
spring.datasource.username=suji
spring.datasource.password=Suji1987@#
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
server.port=8080
logging.level.org.springframework=DEBUG
```

## Using post man:



## USING BROWSER:



**8. Demonstrate using Spring Boot: Complete the docker setup on your Sandbox.**

- Download a docker image from Docker Hub and deploy the same on your docker server
- Build a sample custom image for any of the App of your choice and run the app image as a container

Here's a step-by-step guide to demonstrating the use of Docker with Spring Boot:

**Step 1: Install Docker**

If you haven't already, install Docker on your sandbox environment. You can download the Docker installer from the official Docker website.

**Step 2: Pull a Docker Image from Docker Hub**

Open a terminal and run the following command to pull the official MySQL Docker image from Docker Hub:

```
docker pull mysql:latest
```

This will download the latest MySQL Docker image from Docker Hub.

**Step 3: Run the MySQL Container**

Run the following command to start a new container from the MySQL image:

```
docker run -d --name mysql-server -e MYSQL_ROOT_PASSWORD=password -p 3306:3306 mysql:latest
```

This will start a new container named "mysql-server" and map port 3306 on the host machine to port 3306 in the container.

**Step 4: Create a Spring Boot Application**

Create a new Spring Boot application using your preferred method, such as using Spring Initializr or manually creating the project structure.

**Step 5: Create a Dockerfile**

Create a new file named "Dockerfile" in the root directory of your Spring Boot project. Add the following contents to the file:

```
FROM openjdk:8-jdk-alpine
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

This Dockerfile uses the official OpenJDK 8 image as a base, copies the Spring Boot application JAR file into the container, and sets the entrypoint to run the JAR file using Java.

#### Step 6: Build the Docker Image

Run the following command to build the Docker image:

```
docker build -t my-spring-boot-app .
```

This will create a new Docker image with the name "my-spring-boot-app".

#### Step 7: Run the Docker Container

Run the following command to start a new container from the "my-spring-boot-app" image:

```
docker run -d --name my-spring-boot-app -p 8080:8080 my-spring-boot-app
```

This will start a new container named "my-spring-boot-app" and map port 8080 on the host machine to port 8080 in the container.

#### Step 8: Verify the Application

9. Demonstrate using Spring Boot: Complete the docker setup on your Sandbox.

**OPEN END ASSINGMENT (PROJECT BASED):**

1	Using a docker compose file, deploy multiple apps/containers (eg: MySql, Spring Boot) onto the docker server ( Project based)
2	Demonstrate with Spring Boot: Setup a Kubernetes development Env on your Sandbox (use Docker Desktop or Mini kube) ( Project based)

**VIVA QUESTION & ANSWERS:**

1. What is Cloud Computing?

Answer: Cloud Computing is a model for delivering computing services over the internet, where resources such as servers, storage, and applications are provided as a service to users.

2. What are the main characteristics of Cloud Computing?

Answer: The main characteristics of Cloud Computing are on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.

### Cloud Service Models

1. What are the three main Cloud Service Models?

Answer: The three main Cloud Service Models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

2. Can you explain the differences between IaaS, PaaS, and SaaS?

Answer: IaaS provides virtualized computing resources, PaaS provides a platform for developing and deploying applications, and SaaS provides software applications over the internet.

### Cloud Deployment Models

1. What are the four main Cloud Deployment Models?

Answer: The four main Cloud Deployment Models are Public Cloud, Private Cloud, Hybrid Cloud, and Community Cloud.

2. Can you explain the differences between Public Cloud, Private Cloud, Hybrid Cloud, and Community Cloud?

Answer: Public Cloud is a multi-tenant environment, Private Cloud is a single-tenant environment, Hybrid Cloud is a combination of Public and Private Cloud, and Community Cloud is a shared environment for specific industries or organizations.

## Cloud Security

1. What are some of the key security concerns in Cloud Computing?

Answer: Some of the key security concerns in Cloud Computing include data breaches, unauthorized access, and denial-of-service attacks.

2. How can organizations ensure the security of their data in the Cloud?

Answer: Organizations can ensure the security of their data in the Cloud by using encryption, access controls, and monitoring and auditing mechanisms.

## Cloud Providers

1. Who are some of the main Cloud providers?

Answer: Some of the main Cloud providers include Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), and IBM Cloud.

2. Can you compare and contrast the services offered by AWS, Azure, and GCP?

Answer: AWS, Azure, and GCP offer a range of services, including computing, storage, and database services. Each provider has its own strengths and weaknesses, and the choice of provider will depend on the specific needs of the organization.

## Case Studies

1. Can you describe a real-world example of a company that has successfully implemented Cloud Computing?

Answer: For example, Netflix has successfully implemented Cloud Computing using AWS, which has enabled the company to scale its services and improve its customer experience.

2. What are some of the benefits and challenges that organizations may face when implementing Cloud Computing?

Answer: Benefits may include increased scalability, flexibility, and cost savings, while challenges may include security concerns, vendor lock-in, and the need for new skills and training.

3. What is Cloud Computing and how does it relate to Spring Boot?

Answer: Cloud Computing is a model for delivering computing services over the internet, and Spring Boot is a Java framework that can be used to build cloud-native applications.

#### 4. How does Spring Boot support Cloud Computing?

Answer: Spring Boot provides a range of features and tools that support Cloud Computing, including auto-configuration, embedded servlet containers, and support for cloud-based services such as AWS and Azure.

### Cloud Deployment

#### 1. How would you deploy a Spring Boot application to a cloud platform such as AWS or Azure?

Answer: You can deploy a Spring Boot application to a cloud platform using a range of tools and services, including AWS Elastic Beanstalk, Azure App Service, and Cloud Foundry.

#### 2. What are some of the benefits and challenges of deploying a Spring Boot application to a cloud platform?

Answer: Benefits may include scalability, flexibility, and cost savings, while challenges may include security concerns, vendor lock-in, and the need for new skills and training.

### Cloud Native Applications

#### 1. What is a cloud-native application and how does Spring Boot support this concept?

Answer: A cloud-native application is an application that is designed to take advantage of cloud computing models, and Spring Boot provides a range of features and tools that support this concept, including support for microservices, containers, and serverless computing.

#### 2. How would you design a cloud-native application using Spring Boot?

Answer: You would design a cloud-native application using Spring Boot by following a range of principles and patterns, including the use of microservices, containers, and serverless computing, as well as the use of cloud-based services such as AWS Lambda and Azure Functions.

### Spring Cloud

#### 1. What is Spring Cloud and how does it relate to Spring Boot?

Answer: Spring Cloud is a set of tools and frameworks that provide a range of cloud-based services, including configuration management, service discovery, and circuit breakers, and Spring Boot is a Java framework that can be used to build cloud-native applications.

2. How would you use Spring Cloud to build a cloud-native application using Spring Boot?

Answer: You would use Spring Cloud to build a cloud-native application using Spring Boot by following a range of steps, including the use of Spring Cloud Config for configuration management, Spring Cloud Netflix for service discovery and circuit breakers, and Spring Cloud Stream for event-driven architectures.

### Case Studies

1. Can you describe a real-world example of a company that has successfully used Spring Boot to build a cloud-native application?

Answer: For example, Netflix has successfully used Spring Boot to build a range of cloud-native applications, including its content delivery network and its recommendation engine.

2. What are some of the benefits and challenges that organizations may face when using Spring Boot to build cloud-native applications?

Answer: Benefits may include increased scalability, flexibility, and cost savings, while challenges may include security concerns, vendor lock-in, and the need for new skills and training.

1. What is Hibernate and what problem does it solve?

Answer: Hibernate is an Object-Relational Mapping (ORM) tool that simplifies the interaction between Java applications and relational databases.

2. How does Hibernate map Java classes to database tables?

Answer: Hibernate uses annotations and XML configuration files to map Java classes to database tables.

3. What is the difference between Hibernate's Session and SessionFactory?

Answer: A Session is a single-threaded, short-lived object that represents a conversation between the application and the database, while a SessionFactory is a thread-safe, long-lived object that creates and manages Sessions.

4. How does Hibernate handle transactions?

Answer: Hibernate provides a transaction management system that allows applications to manage database transactions programmatically.

5. What are some of the benefits and limitations of using Hibernate?

Answer: Benefits include simplified database interaction, improved portability, and reduced boilerplate code, while limitations include a steep learning curve, potential performance overhead, and limited support for certain database features.

## Kubernetes

1. What is Kubernetes and what problem does it solve?

Answer: Kubernetes is a container orchestration system that automates the deployment, scaling, and management of containerized applications.

2. How does Kubernetes manage containerized applications?

Answer: Kubernetes uses a combination of pods, replica sets, deployments, and services to manage containerized applications.

3. What is the difference between a pod and a container in Kubernetes?

Answer: A pod is a logical host for one or more containers, while a container is a runtime instance of a Docker image.

4. How does Kubernetes provide high availability and scalability for applications?

Answer: Kubernetes provides high availability and scalability through the use of replica sets, deployments, and horizontal pod autoscaling.

5. What are some of the benefits and limitations of using Kubernetes?

Answer: Benefits include improved application scalability, high availability, and simplified management, while limitations include a steep learning curve, potential complexity, and limited support for certain application types.

## Docker

1. What is Docker and what problem does it solve?

Answer: Docker is a containerization platform that simplifies the packaging, shipping, and running of applications by providing a lightweight and portable way to deploy applications.

## 2. How does Docker containerization work?

Answer: Docker containerization works by packaging an application and its dependencies into a single container that can be run on any Docker-compatible host.

## 3. What is the difference between a Docker image and a Docker container?

Answer: A Docker image is a read-only template for creating containers, while a Docker container is a runtime instance of a Docker image.

## 4. How does Docker provide isolation and security for containers?

Answer: Docker provides isolation and security for containers through the use of kernel namespaces, control groups, and SELinux or AppArmor.

## 5. What are some of the benefits and limitations of using Docker?

Answer: Benefits include improved application portability, simplified deployment, and increased efficiency, while limitations include potential security risks, limited support for certain application types, and potential complexity.

## Introduction to Microservices

### 1. What are Microservices and how do they differ from Monolithic architecture?

Answer: Microservices are a software development technique that structures an application as a collection of small, independent services that communicate with each other using APIs. Unlike Monolithic architecture, Microservices allow for greater scalability, flexibility, and resilience.

### 2. What are the benefits of using Microservices?

Answer: Benefits include improved scalability, flexibility, and resilience, as well as faster development and deployment cycles, and the ability to use different programming languages and technologies for each service.

## Microservices Architecture

### 1. What is the role of APIs in Microservices architecture?

Answer: APIs play a crucial role in Microservices architecture, enabling communication between services and allowing them to exchange data and requests.

## 2. How do Microservices handle service discovery and registration?

Answer: Microservices use service discovery mechanisms, such as DNS, load balancers, or service registries, to register and discover services.

## 3. What is the role of containers in Microservices architecture?

Answer: Containers, such as Docker, provide a lightweight and portable way to deploy Microservices, enabling greater efficiency and scalability.

## Microservices Communication

### 1. What are the different types of communication between Microservices?

Answer: Microservices can communicate using synchronous or asynchronous communication, including RESTful APIs, message queues, and event-driven architecture.

### 2. How do Microservices handle errors and failures?

Answer: Microservices use fault-tolerant mechanisms, such as circuit breakers, bulkheads, and retry policies, to handle errors and failures.

### 3. What is the role of service meshes in Microservices communication?

Answer: Service meshes, such as Istio or Linkerd, provide a layer of infrastructure that enables secure, reliable, and observable communication between Microservices.

## Microservices Deployment and Management

### 1. How do Microservices handle deployment and scaling?

Answer: Microservices use container orchestration tools, such as Kubernetes, to automate deployment, scaling, and management.

### 2. What is the role of monitoring and logging in Microservices?

Answer: Monitoring and logging play a crucial role in Microservices, enabling developers to detect issues, debug problems, and optimize performance.

### 3. How do Microservices handle security and compliance?

Answer: Microservices use a range of security mechanisms, including authentication, authorization, and encryption, to ensure security and compliance.

## Case Studies and Real-World Examples

1. Can you describe a real-world example of a company that has successfully implemented Microservices?

Answer: For example, Netflix has successfully implemented Microservices, using a range of technologies, including Java, Python, and Node.js, to build a scalable and resilient architecture.

2. What are some of the challenges and lessons learned from implementing Microservices in a real-world scenario?

Answer: Challenges may include managing complexity, ensuring communication and collaboration between teams, and handling errors and failures. Lessons learned may include the importance of automation, monitoring, and logging, as well as the need for a culture of continuous learning and improvement.

## SAFETY INSTRUCTIONS

### **Do's :**

1. Do wear ID card and follow dress code.
2. Be on time to Lab sessions.
3. Do log off the computers when you finish.
4. Do ask the staff for assistance if you need help.
5. Do keep your voice low when speaking to others in the LAB.
6. Do ask for assistance in downloading any software.
7. Do make suggestions as to how we can improve the LAB.
8. In case of any hardware related problem, ask LAB in charge for solution.
9. If you are the last one leaving the LAB , make sure that the staff in charge of the LAB is informed to close the LAB.
10. Be on time to LAB sessions.
11. Do keep the LAB as clean as possible.

### **Dont's :**

1. Do not use mobile phone inside the lab.
2. Don't do anything that can make the LAB dirty (like eating, throwing waste papers etc).
3. Do not carry any external devices without permission.
4. Don't move the chairs of the LAB.
5. Don't interchange any part of one computer with another.
6. Don't leave the computers of the LAB turned on while leaving the LAB.
7. Do not install or download any software or modify or delete any system files on any lab computers.
8. Do not damage, remove, or disconnect any labels, parts, cables, or equipment.
9. Don't attempt to bypass the computer security system.
10. Do not read or modify other user's file.
11. If you leave the lab, do not leave your personal belongings unattended. We are not responsible for any theft.