```
protected void doGet(HttpServletRequest
    req, HttpServletResponse res) throws
    ServletException, IOException {
  // Get data from model
  String studentName = "AHP";
  int studentID = 22002;

  // Set attributes for view
  request.setAttribute("name",
                studentName);
  request.setAttribute("id",
                studentID);

  // Forward to JSP
  request.getRequestDispatcher
      ("studentView.jsp").forward
          (req, res);
  }
}
```

18) Compare and contrast cookies, URL rewriting and HttpSession as methods for session tracking in servlets. Discuss their advantage, limitations and ideal use cases.

Ans. Comparision -

Feature	Cookies	URL rewriting	HttpSession
Storage Location	Client-side	Embedded in URLs	Server-side
Security	Medium (Vulnerable to XSS)	Low	High (Server controlled)
Implementation	response.addCookie()	response.encodeURL()	request.getSession()
Use case	User preferences (lang)	Legacy systems	Sensitive data
Pros	Auto-sent with requests	No client storage	Support objects
Cons	Size limit. Can be disabled	Manual URL handling	Server memory overhead
Persistance	Can be persistant	Only for current session.	Session scoped
Cookie dependency	Required	Not required	Preferred

19) Explain how the HttpSession works across multiple requests and how session time out on invalidation is handled securely.

Ans. HttpSession works in following way:

1) Login: Server creates a session with a unique ID, and sends it as a cookie.

2) Subsequent requests: Browser sends the session ID with each request. Server uses it to retrieve user data.

3) Session storage: User data (like user_id) is stored server-side, linked to session ID.

Session timeout and security:

i) Timeout: Session expire after inactivity (e.g 30 mins). Server deletes expired ~~cost~~ session.

ii) Logout: Server invalidates data session and clears data. Client's session cookie is expired

iii) Protections: It uses secure and block JS access flags for cookies. It regenerate session

IDs often login to prevent fixation attacks.

20) How spring MVC handles an HTTP request from a browser. Describe the role of the @Controller, @RequestMapping and Model Objects in seperating business logic from presentation. Provide a brief flow of example of a login form submission

Ans. When a browser sends request to a Spring MVC application, the flow follows these key steps:

1) Dispatcher Event: It receives the HTTP request and delegates processing to the appropriate controller.

2) @Controller: A class annotated with @Controller defines request-handling methods. It separates businesstogic from authentication

3) @RequestMapping: Maps HTTP requests (login) to specific method.

4) Model object: It carries data between the controller and data view

Example:

```
@Controller
public class LoginController {
    @PostMapping("/login")
    public String handleLogin(
        @RequestParam String username,
        @RequestParam String password,
        Model model) {

    if("admin".equals(username) && "pass
        123".equals(password)) {
        model.addAttribute("message", "Login
            successful!");
        return "dashboard";
    }
    else {
        model.addAttribute("error", "In
            valid Credentials");
        return "login";
    }
}
}
```

21) Spring MVC uses the DispatcherServlet as a front controller. Describe its role in request processing workflow. How does it interact with resolver and handler mappings?

Ans. The ~~role~~ workflow of Spring MVC as a front controller using the dispatcher servlet is given below :-

1) Request Received : The browser sends an HTTP request. Dispatcher Servlet intercepts the request (configured in web.xml or Spring boot auto-configuration).

2) Handler Mapping : ~~Handles~~ Consults HandlerMapping to determine which @Controller method should handle request. Again, it matches URL/method to @RequestMapping annotations (e.g @GetMapping ("/home")).

3) Controller execution : It invokes matched controller method. The method processes the business logic and returns to logical view name.

4) View Resolution :- Delegates to a ViewResolver

(e.g ThymeleafViewResolver) to convert logical viewname into an actual view. (e.g /templates/home.html).

5) Response rendering: Spring boot MVC renders the view (HTML, JSON, etc) and sends the HTTP response back to client.

22) How does PreparedStatement improve the performance and security over statement in JDBC? Write short example to insert a record into a MySQL table using PreparedStatement? [Lab 6]

Ans. PreparedStatement improves performance and security in the following way:

1) PreparedStatement pre-compiles the SQL query. So repeated executions are faster.

2) It supports batch() and executeBatch() for bulk insert.

3) It uses parameterized queries (?) so user input is treated as data not executable SQL.

4) Special characters (",") are escaped safely.

Example Code:

```
import java.sql.*;

public class InsentExample {

    public static void main(String[] args) {

    String url="jdbc:mysql://localhost:
            3306/mydb";
        String user = "Atif";
        String password= "1234";

    try(Connected conn=DrivenManagen.get
            Connection(url, user, password);

        PreparedStatement pstmt= conn.
                prepareStatement(sql)){

        pstmt.setString(1,"Atif");

        pstmt.setString(2,"atif544@
                gmail.com");

        int rowsAffected=pstmt.execute
                Update();
```

```
            System.out.println(rowsAffected + " row(s)
                inserted.");
        }
        catch(SQLException e){
            e.printStackTrace();
        }
    }
}
```

23) What is ResultSet in JDBC and how it is used to retrieve data from MySQL database? Briefly explain the use of next(), getString(), and getInt() methods with an example. [Lab 7]

Ans: A ResultSet is an object that represents data returned from a SQL query (e.g SELECT). It acts as a cursor that moves through rows of data, allowing us to retrieve values column by column.

Methods of ResultSet:

Method	Purpose
next()	Moves to the next row. Returns true if row exists, false otherwise
getString(col)	Get a String value from the specified column
~~geet~~ getInt(col)	Gets an Initial value from specified column

Code:

```
import java.sql.*;

public class ResultSetExample {

    public static void main(String[]
                             args) {

String url = "jdbc:mysql://localhost:
                3306:/mydb";

    String user = "Atif";

    String password = "1234";

    String sql = "SELECT id, name, age
                  FROM users";
```

```
try (Connection conn = DriverManager.getConnection(url, user, password);
     Statement stmt = conn.createStatement();
     ResultSet rs = conn.createStatement();

     while (rs.next()) {
         int id = rs.getInt("id");
         String name = rs.getString(2);
         int age = rs.getInt("age");
         System.out.println("ID: " + id +
             ", Name: " + name + ", Age: " + age);
     }
 }
 catch (SQLException e) {
     e.printStackTrace();
 }
}
```

29) How does JPA manage the mapping between Java objects and relational database? Explain with an example using @Entity, @Id and @GeneratedValue annotations. Discuss the advantage

JPA over raw JDBC:

Ans. JPA (Java Persistance API) simplies database interactions by automatically mapping Java classes to database tables using annotations.

Annotations:

i) @Entity: Makes a Java class as a persistable entity

ii) @Id: Specify primary key of entity.

iii) @GeneratedValue: Configures auto-generation of primary keys.

Example:

```
import javax.persistence.*;
@Entity
@Table(name="users")
public class User{
 @Id
@GeneratedValue(strategy = GenerationType.
                    IDENTITY)
    private long id;
    @Column(name="user_name")
```

```
private String name;
private int age;
public User() {}
public User(String name, int age) {
    this.name=name;
    this.age=age;
}
}
```

Database Table (Auto-created by JPA):

```
CREATE TABLE users (
    id bigint auto_increment primary key,
    user_name varchar(255),
    age int
);
```

25] Describe the difference between the Entity Manager's persist(), merge() and the remove() operations. When would you use each method in a typical database transaction scenario?

Ans. Use cases of methods:-

1) Persist(): Inserts a new entity into the database

- Turns new object into a persistent state.
- Throws an exception if entity already exists.

Ex-

```
User newUser = new User("ATF", 23);
entityManager.persist(newUser);
```

2) merge(): Updates or reattaches a detached entity. It works like:

- If the entity already exists in DB then it copies state to a managed instance and returns it.
- If the entity doesn't exist then it inserts as a new record.

Ex-

```
User detachedUser = new User(1L, "Finoj", 30);
User managedUser = entityManager.merge(detachedUser);
```

3) remove(): Deletes an entity from the database.

- Transitions in a managed entity to removed state.
- Throws exception if entity is detached on now.

Ex-

```
User user = entityManagen.find(User.class,
                    1L);
entityManagen.remove(user);
```

Transaction output of database be like:

After
1) persist();

```
insert into users(name,email) values
        ("AHf","ahf594@gmail.com");
```

2) After merge():

```
Update users set name="Anoj", email=
    "anoj123@gmail.com" where Id=1;
```

3) After remove:

```
delete from users where id=2;
```

26) Design a simple CRUD application using Spring boot and mySQL to manage student records. Describe how each operation (Create, Read, Update, Delete) would be implemented using a repository interface [Lab 7]

Ans- Simple CRUD application using Spring boot and mySQL is developed below:

1) application.properties

```
spring.datasource.url = jdbc:mysql://localhost:3306/student_db?CreateDatabaseIfNotExist=true

spring.datasource.username = Atif
spring.datasource.password = 1234
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show_sql=true
```

2) Student Entity (Student.java)

```
package com.example.studentcrud.model;
import jakarta.persistence.*;
@Entity
```

```
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private long id;

    private String name;
    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 String getEmail() { return email; }
    public void setEmail(String email) {

        this.email = email; }

}
```

3) Student Repository

```
import package com.example.studentcrud.
        repository;

import com.example.studentcrud.model.
                Student;
```

```
import org.springframework.data.jpa.repo-
    sitory.JpaRepository;
public interface StudentRepository extends
        JpaRepository<Student, Long> {
}
```

4. StudentController (Crud)

```
package com.example.studentcrud.controller;
import com.example.studentcrud.model.
        Student;
import com.example.studentcrud.repository
        StudentRepository;
import org.springframework.web.bind.
        annotation.*;
import java.util.List;
@RestController
@RequestMapping("/students")
public class StudentController {
    private final StudentRepository
            repo;
```

```
public StudentController(StudentRepository
        repo) {
    this.repo = repo;
}
@PostMapping
public List<Student> getAllStudents() {
    return repo.findAll();
}
@GetMapping("/{id}")
public Student getStudent(@PathVariable
        Long id) {
    return repo.findById(id).orElse(null);
}
@PutMapping("/{id}")
public Student updateStudent(@PathVariable Long id, @RequestBody
        Student updated) {
    return repo.findById(id).map(student
    -> {student.setName(updated.get
```

```
Name());
student.setEmail(updated.getEmail());
return repo.save(student);
}).orElse(null);
@DeleteMapping("/{id}")
public String deleteStudent(@PathVariable Long id){
    repo.deleteById(id);
    return "Deleted student with ID:" +id;
}
}
```

5) Main Application :

```
package com.example.studentcrud;
import org.springframework.boot.autoconfigure.
SpringBootApplication;
@SpringBootApplication
public class StudentCrudApplication {
    public static void main(String[] args){
SpringApplication.run(StudentCrudApp
lication.class,args);
}
}
```

TOPIC NAME DAY TIME DATE / /

27) How does spring boot simplify the development of RESTful services? Describe how to implement a ~~result~~ REST controller using @RestController, @GetMapping and @PostMapping including JSON data handling [Lab 8]

Ans. Spring boot greatly simplifies RESTful service development by:

1) Providing auto-configuration for web applications.
2) Embedding services (Tomcat, Jetty, etc.) so ~~you~~ we don't need to deploy WAR files.
3) Offering starter dependencies that include all needed libraries.
4) Simplifying configuration with sensible defaults.
5) Including built-in JSON support
6) Reducing boilerplate code through annotations

GOOD LUCK

Implementing REST controllers:

```
import org.springframework.web.bind.
    annotation.*;
import java.util.*;
@RestController
    public class ItemController {
     List<String> items = new ArrayList<>();
@GetMapping("/items")
 public List<String> getItems()
        {
          return items;
        }
@PostMapping
 public String addItem(@RequestBody String item) {
     items.add(item);
    return item;
}
}
```

JSON Example:

1) Get /items → Returns []

2) Post /items (Body: "Apple") → Returns:

"Apple"

3) GET /items now returns: ["Apple"]

29) How Maven manage dependencies and build lifecycle in a spring boot project? Explain the structure of a typical pom.xml and how spring boot starter dependencies simplify development.

Ans. Maven is a build automation and dependency management tool used in the Spring Boot projects. It handles -

1) Dependencies - Dowloading libraries from the dependencies.

2) Build lifecycle (Compiling, testing, packaging).

3) Project Structure

1) Dependency management in Maven:

- Local repository ~~Code~~ Cached libraries
- Central Repository (Maven Central) - Default public repo.
- Spring Boot Starter POMs - Predefined dependency sets.

2) Maven build lifestyle:

i) validate - Checks project structure
ii) compile - Compiles source code
iii) test - Run unit tests
iv) package - Creates JAR/WAR
v) install - Installs Jar to local repo
vi) deploy - Publishes to remote repository.

Structure of POM

Project Structure

GOOD LUCK

mint-app/

```
└── src/
    └── main/
        └── java/
            └── com/example/
                └── DemoApplication.java
            └── resources
                └── application.properties
└── pom.xml
```

pom.xml

```
<?xml version = "1.0" ?>
 <project>
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId> org.springframework.boot
                  </groupId>
      <artifactId> spring-boot-starter-
                  parent</artifactId>
```

```
<version>3.1.0</version>
</parent>
<dependencies>
<dependency><groupId>org.framework.
boot</groupId><anHfactId>
spring-boot-stanter-web</anHfactId>
</dependency>
</dependencies>
</project>
```

30) Compare Maven and Gradle as build tools in spring boot projects. Discuss their syntax, performance and dependency handling with an example of a build.gradle configuration for a simple REST API.

Ans Comparision between Maven and Gradle mentioned below:

Maven	Gradle
1) XML configuration	1) DSL configuration
2) Created in 2002	2) Created in 2008.
3) Fewer breaking changes	3) Faster performance due to build cache.
4) More popular	4) Less popular.
5) Unit of work is goals	5) Unit of works is task.

1) Syntax:

Feature	Maven	Gradle
Configuration	Declarative	Programmatic (Kotlin)
Readability	Verbose	Concise
Example	pom.xml	build.gradle.

2) Performance:

1) Maven is slower. But gradle is faster.

ii) Caching is basic. But gradle is ~~advanced~~ advanced.

3) Dependency management:

i) Resolution is sequential but it is parallel in case of gradle.

ii) Version conflict is manual but that of in gradle is automatic.

Example - build.gradle:

```
plugins {
    id 'org.springframework.boot'
        version '3.1.0'
}
dependencies {
    implementation 'org.spring.
        framework.boot:spring-boot-
            starter-web'
}
```

32) Demonstrate the project I developed with the important codes and Graphical User Interface [Lab 9]

Ans. Project Overview

Project Name: University Hall Menu Management System (MBSTU) hall menu.

Technology:

i) Backend: Spring boot

ii) Frontend: Thymeleaf (~~HTTP~~ HTML)

iii) Database: MySQL

Purpose:

i) To manage daily hall menu efficiently

ii) Student can view menus, book meals and give feedback.

iii) Admin can manage menus (CRUD) and moniton all bookings and feedback.

User Roles

Student:

i) View the available menu.

ii) Book meals for a specific day

iii) View own bookings and total cost.

iv) Give feedback for booked meals.

Admin

i) Create new menu entries (Add)

ii) Update or delete existing menus.

iii) View all bookings with total cost calculation

iv) View all student feedback.

GUI

1) Home page :

i) Welcome message: Welcome to MBSTU hall system

ii) MBSTU University Logo displayed.

iii) Two main buttons: Admin / Student

TOPIC NAME _____ DAY _____

TIME _____ DATE / /

2) Admin Dashboard:

i) Menu list: Display all menu with delete and edit options.

ii) Add/Update menu form: To operate menu CRUD operations.

iii) View bookings: Shows booking Id, Student username, Menu, cost and total cost.

iv) View Feedbacks: Displays student name, menu info, rating and comments.

3) Student Dashboard:

i) View ~~books~~ menus: Shows all daily menus with the option to book.

ii) By bookings: Lists all bookings by logged-in student with total cost.

iii) Give feedback: Students can submit ratings and comments for booked meals.

GOOD LUCK