17. In a Java EE aplication, how does a servlet controller manage the flow between the model and the view? Provide a brief example that demonstrates forwarding data from a servlet to a JSP and rendering a response.

In a Java EE application, a servlet controller manages the flow between the model and the view by:

1. Receiving requests from the client
2. Calling model classes to process data or fetch from the database.
3. setting attributes on the request scope.

Flow overview:

Client → Servlet (controller) → Model → JSP(view) → client.

Example: Student Info

1. Model Class

```
Public class Student{
    private String name;
    private int age;
Public Student (String name, int age){
    this.name = name;
    this.age = age;  }
```

```
public string getName(){ return name;}
public int getAge(){
                    return age;}
```

2. JSP(View):

```
<%@ page imports="your.package.student" %>
<%
 Student student=(Student)request.getAttribute("student
                                                Data");
%>
<html>
 <head><title>Student Info</title></head>
<body>
   <h2>Student details</h2>
   <p> Name: <%= Student.getName() %></p>
   <p> Age: <%= Student.getAge() %></p>
</body>
</html>
```

3. Controller(Servlet):

```
@WebServlet("/student")
public class StudentServlet extends HttpServlet{
protected void doGet(HttpServletRequest request, Http
                                    ServletResponse response)
throws ServletException, IOException{
 Student student= new Student("Afita", 22);
 request.setAttribute("studentData",student);
```

```
RequestDispatcher dispatcher = request.getRequestDispatcher
                    ("student.jsp");
dispatcher.forward(request, response); } }
```

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

The comparison table is given below:

Feature	Cookies	URL rewriting	HttpSession
Mechanism	Store session id or data in browser cookies.	Appends session id to the URL as a parameter.	Stores data on server side with session ID in client
Storage Location	Client	Client	Server
Persistence	Can persist beyond browser session	only active during session	Exists for session duration
Automatic Handling	Yes	No	Yes
Security	Medium	Low	High
Data Capacity	Limited	Limited	High

Advantages, limitations and ideal use cases of each methods is given below:

Cookies

- Small piece of data stored on the client side.
- Sent with each HTTP request.
- Used to track user info.

Advantages:

- Persistent across sessions.
- Easy to implement.

Limitations:

- Limited in size.
- Security risks.

Ideal use case:

- Remembering user preferences.

URL rewritting

- Session ID is manually added to every URL as parameter.
- Useful when cookies are disabled.

Advantages

- No reliance on browser cookies.
- Simple for all small apps.

Limitations:

- Security risk
- Bookmarked URL's leaked session info.

Ideal use case:

Public web apps where cookies can't be used.

HTTPSession

Advantages

- Secure
- Scalable
- Automatic management by servlet container

Limitations

- Requires server memory
- Data lost if server restarts.

Ideal use case

- Authenticated user sessions, carts, profiles etc.

19. A web application stores user login info using HttpSession. Explain how the session works across multiple requests and how session timeout or invalidation is handled securely.

In a Java web application, HttpSession is a powerful mechanism that allows you to track and manage user specific data across multiple HTTP requests - like login state, cart contents or user preferences.

How HTTPSession Works Across Requests:

i) ~~How~~ User Logs IN!

- A new session is created
- The server generates a unique Session ID.
- This ID is usually sent to the browser via a cookie.

ii) Subsequent Requests:

- The browser automatically sends the JSESSIONID cookie.
- The server uses this ID to retrive the existing session object.

Session Timeout and Invalidation

- **Session Timeout**

1. A session will expire automatically after a period of inactivity.
2. This is configured in web.xml.
3. After the timeout, the session object is destroyed, and the session ID becomes valid.

- **Invalidation**

1. You can explicitly invalidate a session using →
   request.getSession().invalidate();

Secure Handling of Session

1. Use HttpOnly and Secure Cookies
2. Use HTTPs
3. Regenerate Session on Login
4. Validate Session Attributes
5. Short Timeout for sensitive Data.

20. Explain 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 example of a login form submission.

In spring MVC, handling an HTTP request involves a well structured flow that follows the Model-View-Controller pattern. This architecture seperates business logic, request handling, and presentation making the application modular, maintainable and testable.

Spring MVC Request Handling Flow

When a browser spends a request -

1. The DispatcherServlet receives the request
2. It looks for a controller method mapped to the request URL.
3. The matched @Controller method process the request.
4. It uses a model object to add attributes for the view.

5. The method returns a viewname and the DispatcherServlet renders it.

The role of business logic in seperating business logic is given below:

1. @Controller →
   - Keeps the business logic in services
   - Connects request routing to backend processing.

2. @RequestMapping →
   - Directs with controller method responds to a given URL.
   - Allows clean Routing seperate from both business logic and UI rendering.

3. Model Object →
   - Avoids embedding data processing logic in views.
   - Keeps UI and backend logic loosely coupled.

Example of login form submission –

```
@controller
public class LoginController{
    @PostMapping ("/login")
    public String handleLogin(
        @RequestParam String username;
        @RequestParam String password.
```

```
    Model model) {
if ("admin".equals(username) && "pass".equals(Password))
    model.addAttribute("user", username);
    return "loginSuccess"; }
else {
    model.addAttribute("error", "Invalid credentials");
    return "loginform"; }
} }
```

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

Dispatcher servlet Interaction Diagram:

Browser Request

↓

DispatcherServlet

↓

HandlerMapping → Controller Method

↓

Model → View Name

↓

ViewResolver

↓

Render view with model

↓

Browser receives HTML response

Role of Dispatcherservlet in Request Processing:

1. Receives request:

- All requests mapped to spring MVC or are first received by Dispatcherservlet.

2. Delegates to HandlerMappings:

- The servlet consults HandlerMapping beans to determine which controller method should handle the request.

3. Invokes the controller

- The identified controller method is invoked with any parameters.

4. Processes Return Value

- The controller method returns a logical view with data.

5. Resolves the view:

- Dispatcherservlet uses a ViewResolver to convert the logical view name to an actual view.

6. Renders the view.

22. What is a ResultSet in JDBC and how it is used to retrive data from a MYSQL database? Briefly explain the use of next(), getString() and getInt() methods with an example.

In JDBC, a ResultSet is an object that holds the data returned from executing a select query on a relational data such as: MYSQL. It represents a table of data. Each row in the result set can be accessed one at a time using a cursor that move forward.

How it retrives data is given below:

(. rs.next() moves the first row.)

- Establish a database connection
- create a statement
- Execute the query
- Iterate through the ResultSet
- Retrive column values.
- Process the retrived data.
- Close Resources.

Use of common methods with example:

- next(): Moves the cursor to the next row. Returns false when there are no more rows.
- getString(): Retrives the values of the specified column as a string.
- getInt(): Retrives the value of the specified column as an int.

Example:

```
String query = "select id, name, marks from students";
ResultSet rs = stmt.executeQuery(query);
while (rs.next()){
  int id = rs.getInt("id");
  String name = rs.getString("name");
  int marks = rs.getInt("marks");
System.out.println(id + " | " + name + " | " + marks);
}
```

23. How does JPA manage the mapping between Java objects and relational tables? Explain with an example using @Entity, @Id and @Generated Value annotations. Discuss the advantage of using JPA over raw JDBC.

JPA manage the mapping between Java objects and relational databases tables through a process called Object-Relational Mapping.

Here's how JPA handle the mapping:

Entity definition

- Java classes are marked as JPA entities using the @Entity annotation. This indicates that the class represents a table in the database

Field to column Mapping

- By default, JPA maps fields in an entity class to columns in the corresponding database table with the same time.

Relationship mapping

JPA provides annotations to manage relationships between entities, which correspond to relationships between tables in the database.

Example, with @Entity, @Id and @GeneratedValue annotation:

```
import jakarta.persistance.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.GeneratedValue;
@Entity
public class Student{
  @Id
    private int id;
    private String name;
public int getId(){
            return id; }
public void setId(int id){
          this.id = id; }
public String getName(){
          return name; }
public void setName(String name){
          this.name = name; }
}
```

The advantages of using JPA over raw JDBC for database interaction Java applications -

i) Object - Relational Mapping

ii) Reduced Boilerplate Code

iii) Database Independence

iv) Improved maintainability and Readability

v) Enhanced Productivity

24. Design a simple CRUD application using Spring B and MySQL to manage student records. Describ how each operation would be implemented using a repository interface.

Project overview:

i) Add a student (create)

ii) Get all or single student (read)

iii) Update a student's data (update)

iv) Delete a student (delete)

① Database Table

```
create database studentdb;
use studentdb;
```

② Entity Class - Student.java

```
import jakarta.persistence;
@Entity
public class student;
```

```java
@Id
@GeneratedValue
  private long id;
  private string name;
  ---- -- ---
  --- -- -- - .
}
```

3. Repository Interface

```java
import org.springframework.data.jpa.repository.
                                   JpaRepository;
public interface StudentRepository extends JpaRe-
              pository< Student, Long >{
- - - . -
}
```

This interface gives built in methods like:

- save()
- findById(), findAll()
- deleteById()

4. Controller class

```java
import org.springframework.web.bind.annotation;
import java.util.List;
@RestController
@RequestMapping(" /students")
```

```
Public class StudentController{
  private final StudentService service;
Public StudentController(StudentService service)
    this.service = service;
}
    //create
@PostMapping
public student addStudent(@RequestBody Student st
{
 return service.createStudent(student);
}
// add
// update
// delete
}
```

25. How does spring Boot simplify the development of RESTful services? Describe how to implement a REST controller using @RESTController, @GetMapping and @PostMapping. including JSON data handling..

spring Boot greatly simplifies the development of Restful services using :

i) Auto configuration
- Automatically configures web servers and JSON converters with minimal setup.

ii) Embeded server:
- No need to deploy WARs to an external server. Just run your app like a Java application.

iii) Spring web starter
- Includes all required dependencies for buiding REST API's.

iv) Reduced Boilerplate
- Annotations like @RESTController, @GetMapping, @PostMapping etc. simplify request handling.

Impletmentation of REST controller using annotations and JSON data handling :

1)@REST Controller Annotation:

- Marks the class as a REST API controller.

- It combines @Controller and @ResponseBody

2) @GetMapping

- @GetMapping is used to map HTTP Get requests
- Typically used to retrive data from server.

3) @POSTMapping

- @POSTMapping maps HTTP post requests.
- Used to send data to server.
- Conversion here is handled using @RequestBody

④ JSON data handling

- The JSON is sent by the client
- Spring converts this JSON into a Java object using Jackson.
- This is done by making the method parameter with @RequestBody.

26. How does PreparedStatement improve performance and security over statement in JDBC? Write a short example to insert a record into a MySQL table using PreparedStatement.

How PreparedStatement improves Performance & security over statement in JDBC is given below:

Security:

- PreparedStatement automatically escapes user input making it safe from SQL injection attacks.
- With Statement, user input is directly concatenated into the SQL string.

Performance:

- PreparedStatement allows the database to pre-compile the SQL statement and reuse it with diff. parameters.
- This reduces parsing time and improves efficiency.

Example

Inserting a Record Using PreparedStatement

```
import java.sql.*;
public class InsertExample{
```

```
public static void main(String[] args){
 String url="jdbc:mysql://localhost:3306/studentdb"
 String username="maisha";
 String pass = "root-1236";
String insertSQL="insert into students(name,age)
                  values(?,?)";
try(Connection conn=DriverManager.getConnection(
    url,username, password);
 pstmt.setString(1,"Afifa");
 pstmt.setInt(2, 22);
int rowInserted = pstmt.executedUpdate();
  if(rowsInserted>0)
    sout("Successful insertion");
}
 catch(SQLException e){
    e.printStackTrace();
} }
  }
```

27. Describe the difference between the EntityManager's persist(), merge() and remove() operations. When would you use each method in a typical database transaction scenario.

Difference table is given below:

Method	Purpose	Return value	When to use
persist()	Makes a new entity managed and scheduled it for insertion	void	When inserting a new record
merge()	copies the state of a detached entity into a managed entity	Managed entity	When updating an entity
remove()	Delets a managed entity from the database	void	When deleting a record.

Detailed explanation and used scenarios!

1. persist (object entity):

- Used for saving new entities to the db.
- Adds the object to the persistence context and inserts it into the database at commit.
- Entity must be transient.

2. merge ()

- used for updating detached entities
- Return a new managed copy of the entity with its state copied from the input.
- Entity can be detached / transient.

3. remove ():

- Used for deleting an entity from database.
- Marks the entity for deletion, it is removed at commit.
- Entity must be managed.

28. Compare @RestController and @Controller in spring Boot. In a RESTful API for a library system, describe how you would structure the endpoints for books using REST principles.

Comparison between @RestController and @Controller is given below:

Feature	@Controller	@RESTController
Purpose	Used for rendering views	Used for RESTful APIs
Returns	Returns View Name	Returns Respone Body
Requires @ResponseBody	Yes	No
Use Case	Traditional MVC apps	Web services.

Restful API structure for a library system:

HTTP Method	Endpoint	Description
GET	/books	Retrive all books
GET	/books/{id}	Retrive a book by Id.

POST	/books	Add a new book
PUT	/books/{id}	Update an existing book
DELETE	/books/{id}	Delete a book by Id

Benefits given by this structure:

- Intuitive: Easy to understand and use by clients.
- Consistent: Follows uniform interface principle
- Cacheable: GET responses can be cached

29. You are buiding a multi module spring boot application. Explain how you would structure the project using Maven or Gradle and manage inter module dependencies effectively:

In a multi-module Spring Boot Application, the goal is to seperate concerns into modular, reusable, components, improving maintainability, testability and accessability:

Project structure (Maven or Gradle):

```
springboot.multimodule/
├ pom.xml
├ common/
  └ pom.xml
├ repository/
  └ pom.xml
├ service
  └ pom.xml
├ api
  └ pom.xml
```

So here
.api is responsible for exposing REST endpoints and it depends on service.

- service is responsible for business logic and depends on repository.
- repository handles DB access
- common is responsible for DTOs, enums, utilities and depends on nothing.

Example of inter module dependencies:

```
<dependencies>
  <dependency>
    <groupId> com.example </groupId>
    <artifactId> common </artifactId>
    <version> 1.0.0 </version>
  </dependency>
  <dependency>
    <groupId> com.example </groupId>
    <artifactId> repository </artifactId>
    <version> 1.0.0 </version>
  </dependency>
</dependencies>
```

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.

Comparison table of Maven and Gradle:

Feature	Maven	Gradle
Syntax	XML	Groovy/Kotlin DSL
Read ability	verbose	Concise, more flexible
performance	slower.	Faster
Custom Logic	Harder to script	Easily scriptable
Build output	Typically .jar/.war	same, but more flexible
Tooling support	Excellent	Growing
Community support	Mature, very widely adopted	Increasing rapidly

Example for a simple Spring Boot Rest API

Using gradle:

```
plugins{
    id 'java'
    id 'org.springframework.boot' version '3.2.0'
    id 'io.spring.dependency-management' version '1.1.
}
group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'
repositories{
        mavenCentral()
}
dependencies {
    implementation 'org.springframework.boot:spring-
                    boot-starter-web'
    implementation 'com.fasterxml.jackson.core:
                    jackson-databind'
}

test{
    useJUnitPlatform()
}
```

31. How does maven manage dependencies and build lifecycle in a spring boot project. Explain the structure of a typical pom.xml and how the spring boot starter dependencies simplify development.

How Maven manages dependies and build lifecycle is given below:

Dependencies:

- Maven uses the pom.xml file to declare external libraries.
- It downloads required JAR files from Maven central.
- Developers don't need to manually download or manage JARs.

Build lifecycle:

Maven has a predefined build lifecycle consisting of phases.

- validate - checks if the project is correct.
- compile - compiles the source code.
- test - runs until tests.
- package - packages code.

Typical pom.xml structure

```
<modelversion> 4.0.0 </modelversion>
<groupId> com.example </groupId>
<artifactId> student-app </artifactId>
<version> 1.0.0 </version>
<packaging> jar </packaging>
<parent>
  <groupId> org.springframework.boot </groupId>
  <artifactId> spring-boot-starter-parent </artifactId>
  <version> 3.2.0 </version>
  <packaging> <relativepath/>
  </parent>
```

- How spring boot starter dependencies simplify development is given below:

- Spring MVC
- Jackson
- Embeded Tomcat
- Logging