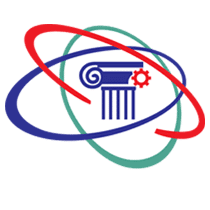
**ACROPOLIS INSTITUTE OF TECHNOLOGY AND RESEARCH, INDORE**

****

**Computer Science and Engineering**

**Departmental Elective**

**Software Architecture (CS-701)**

**IV Year, VII Sem**

Submitted to: Submitted by:

Prof. Shivshankar Rajput Bhavik Mundra

Designation 0827CS201057

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| S. No. | Experiment | Page No. | Remark |
| 1. | Describe your major project architecture. |  |  |
| 2. | Define a use case most suitable for data flow, and design and describe its architecture using the data flow architecture. |  |  |
| 3. | Define a use case most suitable for pipe and filter, and design and describe its architecture using the pipe and filter architecture. |  |  |
| 4. | Describe the J2EE technology stack. |  |  |
| 5. | Document some part of your major project architecture using one of the Architecture Description Languages. |  |  |
| 6. | Document ADLs. Show how and where you will use ADLs in your project. |  |  |
| 7. | Apply CBAM, ATAM, ARID, and ADD to your project and compare each of them with the former. |  |  |
| 8. | Draw basic UML diagrams of your project like Use case, Class, Activity, and Sequence diagram. |  |  |
| 9. | Draw Use case diagrams for the Bus tracking App. |  |  |
| 10. | Draw a Use Case diagram for the MMT App. |  |  |
| 11. | Write a program to demonstrate transaction operations in Hibernate. |  |  |
| 12. | Make a simple web page using AngularJS. |  |  |
| 13. | Write a simple program to demonstrate the use of Servlets and JSP. |  |  |

**4. Describe the J2EE technology stack.**

Java 2 Platform, Enterprise Edition (J2EE) was an older platform for building enterprise-level, scalable, and distributed Java applications. As of my last knowledge update in January 2022, J2EE has been replaced by the Java EE (Enterprise Edition) platform, which is now known as Jakarta EE. However, the core components and concepts of J2EE are still relevant in Jakarta EE. Below is an overview of the J2EE technology stack:

1. Servlets and JSP: Servlets are Java programs that run on the server and handle HTTP requests and responses. JavaServer Pages (JSP) is a technology used to create dynamic web pages. They work together to build web applications in Java.
2. Enterprise JavaBeans (EJB): EJB is a server-side component model for building distributed, transactional, and scalable applications. It provides features like persistence (JPA), messaging (JMS), and session management.
3. Java Naming and Directory Interface (JNDI): JNDI is used for naming and directory services in Java. It allows applications to look up resources and services such as databases and message queues.
4. Java Message Service (JMS): JMS is an API for asynchronous messaging, allowing components to communicate by sending and receiving messages.
5. Java Persistence API (JPA): JPA is a standard for mapping Java objects to relational databases. It provides an object-relational mapping (ORM) framework for simplifying database access.
6. Java Transaction API (JTA): JTA provides a standard API for managing distributed transactions in Java applications.
7. JavaMail API: JavaMail is used for sending and receiving email in Java applications. It allows applications to connect to email servers and send/receive messages.
8. Java Naming and Directory Interface for the Lightweight Directory Access Protocol (JNDI/LDAP): JNDI can also be used to interact with LDAP directories, which are often used for authentication and directory services.
9. RMI-IIOP: This is a bridge between Java's Remote Method Invocation (RMI) and the Internet Inter-ORB Protocol (IIOP), enabling remote method invocation in a distributed environment.
10. XML and Web Services: J2EE supports the use of XML for data exchange and web services for interoperable communication between different systems.
11. Security: J2EE provides a comprehensive security model, including authentication, authorization, and encryption, to secure enterprise applications.
12. Connectivity and Data Sources: J2EE allows the configuration of data sources and connection pooling for efficient and scalable database access.
13. Java API for RESTful Web Services (JAX-RS): JAX-RS is used to create RESTful web services in Java. It allows you to expose Java classes as RESTful web resources.
14. Java API for WebSocket (WebSocket API): WebSocket API enables real-time, full-duplex communication between the client and server.
15. Java Authentication and Authorization Service (JAAS): JAAS is a framework for user authentication and authorization, used to secure Java applications.
16. Web Containers and Application Servers: J2EE applications are deployed in web containers or application servers, such as Apache Tomcat, WildFly (formerly JBoss), and WebSphere, which provide a runtime environment for J2EE applications.

**11. Write a program to demonstrate transaction operations in Hibernate.**

Hibernate is an Object-Relational Mapping (ORM) framework for Java that simplifies database interactions. Demonstrating transaction operations in Hibernate involves setting up a Hibernate configuration, creating a model class, and performing database transactions. Below is an example program that demonstrates basic transaction operations using Hibernate:

1. Set up your Hibernate project:
   * Create a new Java project and add the required Hibernate libraries to the ‘classpath’.
2. Create a Hibernate configuration file (hibernate.cfg.xml) in the ‘src’ directory:

<!-- hibernate.cfg.xml -->

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- JDBC Database connection settings -->

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/database</property>

<property name="hibernate.connection.username">bhavikmundra</property>

<property name="hibernate.connection.password">password</property>

<!-- JDBC connection pool settings -->

<property name="hibernate.c3p0.min\_size">5</property>

<property name="hibernate.c3p0.max\_size">20</property>

<!-- Specify dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- Enable Hibernate's automatic session context management -->

<property name="hibernate.current\_session\_context\_class">thread</property>

<!-- Echo all executed SQL to stdout -->

<property name="hibernate.show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Mapping file -->

<mapping resource="com/software/architecture/Entity.hbm.xml"/>

</session-factory>

</hibernate-configuration>

1. Create a Hibernate mapping file (Entity.hbm.xml) in the src/com/software/architecture directory:

<!-- YourEntity.hbm.xml -->

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="com.software.architecture.Entity" table="deleteThisTable">

<id name="id" type="int">

<column name="id" />

<generator class="native" />

</id>

<property name="name" type="string">

<column name="name" />

</property>

</class>

</hibernate-mapping>

1. Create a Java entity class (Entity.java) in the ‘com.software.architecture’ directory:

package com.software.architecture;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Entity {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

// Constructors, getters, and setters

}

1. Write a Java program to perform Hibernate transactions:

package com.software.architecture;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class HibernateTransactionDemo {

public static void main(String[] args) {

Configuration configuration = new Configuration().configure("hibernate.cfg.xml");

SessionFactory sessionFactory = configuration.buildSessionFactory();

Session session = sessionFactory.openSession();

// Begin a transaction

Transaction transaction = session.beginTransaction();

try {

// Create a new entity and save it to the database

Entity entity = new Entity();

entity.setName("Transaction Test");

session.save(entity);

// Commit the transaction

transaction.commit();

System.out.println("Transaction committed successfully.");

} catch (Exception e) {

// Rollback the transaction in case of an error

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

sessionFactory.close();

}

}

}

1. Execute the program, and it will create a new entity, save it to the database, and demonstrate transaction operations with Hibernate.

**12. Make a simple web page using AngularJS.**

AngularJS is an older JavaScript framework for building dynamic web applications. It's essential to note that AngularJS has been largely replaced by Angular (2+), but I can provide you with a simple example of creating a web page using AngularJS:

First, make sure to include the AngularJS library in your HTML file. You can download it or use a content delivery network (CDN). Here's how to include it using a CDN:

<!DOCTYPE html>

<html>

<head>

<title>Simple AngularJS Web Page</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<div ng-app="myApp" ng-controller="myController">

<h1>{{ greeting }}</h1>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myController', function($scope) {

$scope.greeting = 'Hello, My name is Bhavik Mundra!';

});

</script>

</body>

</html>

In the code above:

* We included the AngularJS library via a CDN.
* We defined an AngularJS module called 'myApp' using ng-app.
* We created an AngularJS controller called 'myController' using ng-controller.
* Inside the controller, we defined a scope variable greeting and set its value to 'Hello, AngularJS!'.
* We used double curly braces {{ greeting }} to display the value of greeting in the HTML.
* Save the HTML file and open it in a web browser. You should see the text "Hello, AngularJS!" displayed on the web page.

This is a simple example of using AngularJS to create a dynamic web page.

**Screenshot:**

****

**13. Write a simple program to demonstrate the use of Servlets and JSP.**

Servlets and JSP (JavaServer Pages) are technologies used for building web applications in Java. Servlets handle the server-side logic, while JSP is used to create dynamic web pages. Here's a simple example of a web application using Servlets and JSP to display a "Hello, World!" message:

1. Create a new Java web project in your favorite Integrated Development Environment (IDE) or build it manually by setting up the project structure.
2. Create a Servlet (HelloWorldServlet.java):

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.IOException;

public class HelloWorldServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<html><head><title>Hello World</title></head><body>");

out.println("<h1>Hello, World!</h1>");

out.println("</body></html>");

out.close();

}

}

1. Create a JSP (hello.jsp):

<!DOCTYPE html>

<html>

<head>

<title>Hello JSP</title>

</head>

<body>

<h1>Hello, World! (from JSP)</h1>

</body>

</html>

1. Configure the Servlet in the “web.xml” file:

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd">

<servlet>

<servlet-name>HelloWorldServlet</servlet-name>

<servlet-class>HelloWorldServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloWorldServlet</servlet-name>

<url-pattern>/HelloWorldServlet</url-pattern>

</servlet-mapping>

</web-app>

1. Create an HTML page (index.html) to access the Servlet:

<!DOCTYPE html>

<html>

<head>

<title>Hello Servlet and JSP</title>

</head>

<body>

<h1>Hello, Servlet and JSP Example</h1>

<p><a href="HelloWorldServlet">Click here for Hello World (Servlet)</a></p>

<p><a href="hello.jsp">Click here for Hello World (JSP)</a></p>

</body>

</html>

1. Deploy the web application to a servlet container like Apache Tomcat.
2. Access the application by navigating to the following URL:

‘http://localhost:8080/your-web-app-context/’.

1. When you visit the URL, you will see an HTML page with links to "Hello World (Servlet)" and "Hello World (JSP)." Clicking these links will display the "Hello, World!" message from both the Servlet and JSP. This example demonstrates the basic usage of Servlets and JSP in a Java web application.