Pratibodh

A Journal for Engineering

A free and Open Access Journal

Homepage: https://pratibodh.org

RATNA & BM RANVEER Foundation

Analysis on Java Enterprise Web Development Design and Development of Enterprise Web Application

Himanshu Bishnoi[1], Khushveer Singh Gurjar[2],Mohammad Ahmad[3],Neelkamal Chaudhary[4]

Artificial Intelligence and Data Science
Jaipur Engineering College and Research Centre
Sitapura, Jaipur, Rajasthan 302022
[1]himanshubishnoi.ai25@jecrc.ac.in

Abstract

Java, as a versatile programming language, forms the foundation of web application development through its technologies, namely JSP (Java Server Pages), Servlets, and core Java. JSP simplifies dynamic web content creation by seamlessly integrating Java code within HTML pages. Servlets, Java-based server-side components, handle HTTP requests, and enable the execution of server-side logic. These technologies collectively empower developers to create interactive and data-driven web applications efficiently. In parallel, core Java extends its utility beyond web development, serving as a cross-platform language renowned for its portability, security, and extensive ecosystem. This abstract explores the key roles of JSP, Servlets, and core Java, highlighting their significance in building diverse applications, from dynamic web interfaces to robust enterprise systems.

Article Status

Available online

Keywords: JSP,JDBC,SERVLET,IMAGE,PDFCELL.

1. Introduction

In the dynamic and ever-evolving landscape of web development, Java has emerged as a robust and versatile programming language that plays a pivotal role in the creation of feature-rich and scalable web applications. With its extensive ecosystem of tools and frameworks, Java empowers developers to craft web solutions that are not only efficient but also capable of handling complex business logic. This research paper delves into the realm of Java web development, focusing on the integration of core Java, JDBC (Java Database Connectivity), Servlets, and MySQL to create compelling web applications.

The combination of Java, JDBC, Servlets, and MySQL forms a formidable stack that equips developers with the tools necessary to build web applications that seamlessly interact with databases, handle user requests, and deliver dynamic content. This paper will explore the individual components of this stack, highlighting their significance in the web development process, and demonstrate how they can be harnessed collectively to build modern, data-driven web applications.

As we progress through this research, we will delve into the fundamental concepts of each technology, exploring how they contribute to the web development process. We will also discuss best practices, design patterns, and practical examples that showcase the synergy between Java, JDBC, Servlets, and MySQL, enabling developers to build efficient and maintainable web applications .Furthermore, this paper will shed light on the importance of database integration, the role of Servlets in handling HTTP requests and responses, the power of core Java in implementing business logic, and the robustness of MySQL as a relational database management system. By the end of this research, readers will have a comprehensive understanding of how these technologies can be harnessed to create dynamic and interactive web applications that meet the demands of today's digital world.

In a rapidly evolving web development landscape, the knowledge and expertise in Java, JDBC, Servlets, Core Java, and MySQL are invaluable assets for developers seeking to build web applications that are not only functional but also scalable, secure, and responsive to user needs. This research paper aims to serve as a valuable resource for developers, students, and enthusiasts looking to master the art of Java web development through the integration of these powerful technologies.

II.Web Application Architecture using MVC Design Pattern:

Using the Model-View-Controller (MVC) design pattern in Java is a common approach for building scalable and maintainable applications. Here's a high-level overview of how you can implement MVC in a Java application:

1.Model:

Define your data structures and business logic. These represent the core functionality of your application.

Create Java classes that encapsulate data and provide methods to interact with that data.

Examples of Models can include classes that represent entities in your application, like User, Product, Order, etc.

2.View:
Implement the user interface (UI) using Java's Swing, JavaFX, or a web-based framework like Java Servlets and JSP (JavaServer Pages).
Views are responsible for rendering data from the Model and displaying it to the user.
They should not contain business logic; their main role is to present data and capture user input.

3.Controller:
Create Java classes that act as controllers. These classes handle user input and coordinate interactions between the Model and View.
Controllers receive input from the View, process it, and update the Model as necessary.
They also update the View to reflect changes in the Model's data.
Controllers act as the glue that binds the Model and View together.

Here's a step-by-step example of how you might shown in (fig1.)

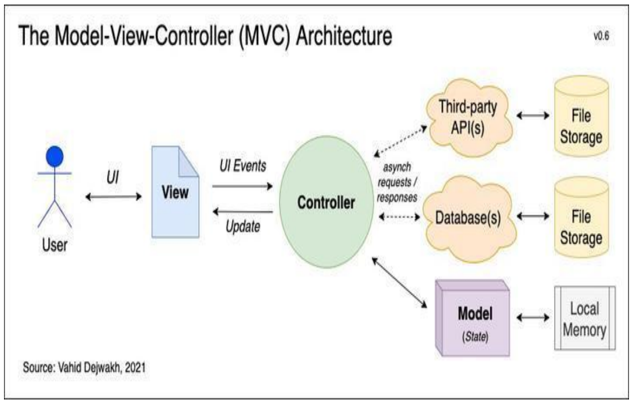


Fig1. MVC Design.

(III).Web Application Architecture in Java EE Edition.

Web application architecture in the Java EE (Enterprise Edition) platform is designed to provide a robust, scalable, and enterprise-grade framework for building large-scale web applications. Java EE offers a set of APIs and services for developing and deploying web applications that can handle high traffic loads and complex business logic[1]. Here's an overview of the key components and concepts in Java EE web application architecture:as shown in (fig2.)

I.Servlets and JSP (JavaServer Pages):

Servlets and JSP are the fundamental building blocks of Java EE web applications.

Servlets handle HTTP requests and generate dynamic content, such as HTML, XML, or JSON, as responses.

JSP is a technology for creating dynamic web pages by embedding Java code within HTML[7] as shown in fig3.

Together, they provide the presentation layer of your application as shown in fig4 & fig5.

II.Enterprise JavaBeans (EJB):

EJBs are server-side components that encapsulate business logic, providing a way to implement the Model layer of your application.

There are three types of EJBs: Session Beans (stateless and stateful), Message-Driven Beans, and Entity Beans (less common in modern Java EE).

Session Beans are used for managing application state and processing business logic.

III.JPA (Java Persistence API):

JPA is a Java EE standard for object-relational mapping (ORM), allowing you to interact with relational databases using Java objects.

JPA simplifies database access and management, enabling you to work with entities and relationships in a more object-oriented manner.

IV. Web Services:

Java EE supports the creation of web services using technologies like JAX-RS (for RESTful services) and JAX-WS (for SOAP-based services).

Web services enable interoperability and communication with other systems, making Java EE suitable for building distributed and SOA (Service-Oriented Architecture) applications.[3]

V. Contexts and Dependency Injection (CDI):

CDI is a powerful Java EE feature that simplifies the management of beans and their dependencies within the application.

It provides a standardized way to define and inject beans, improving modularity and testability.

VI. Security:

Java EE offers robust security features, including authentication, authorization, and secure communication through SSL/TLS.[4]

You can configure security constraints and roles to protect resources and restrict access to specific parts of the application.

VII. Messaging:

Java EE provides messaging capabilities through technologies like JMS (Java Message Service) and Message-Driven Beans, facilitating asynchronous communication within the application or with external systems.

VIII. Transaction Management:

Java EE supports declarative transaction management, allowing you to define transaction boundaries for EJBs and other components.

It ensures ACID (Atomicity, Consistency, Isolation, Durability) properties for database operations.

IX. Packaging and Deployment:

Java EE applications are typically packaged as WAR (Web Application Archive) or EAR (Enterprise Archive) files.

These archives contain all the necessary components, libraries, and configuration files for deployment to a Java EE-compatible server (e.g., Tomcat, Wild Fly, Glass Fish).

X. Scalability and Clustering:

Java EE platforms often support clustering and load balancing to distribute application workloads across multiple servers, improving scalability and fault tolerance.

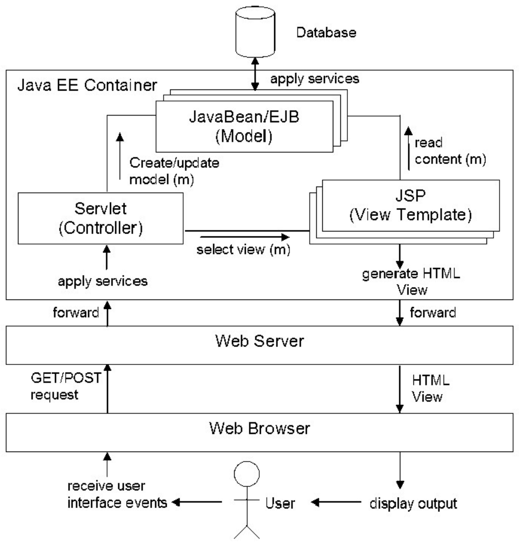


Fig2. Web Architecture In Java EE.

IV. How Dynamic Content Display? Backend Approach:

JSP (Java Server Pages) is a technology used for developing web applications in Java. It is particularly useful for displaying dynamic content on web pages as shown fig5. Here's how JSP is useful in achieving this:

1.Integration**:** Seamlessly integrates Java code within HTML pages.

2.Dynamic Content: Allows server-side generation of dynamic content. Code shown in Fig12.

3.Reusability: Supports creation of reusable components for easier maintenance.

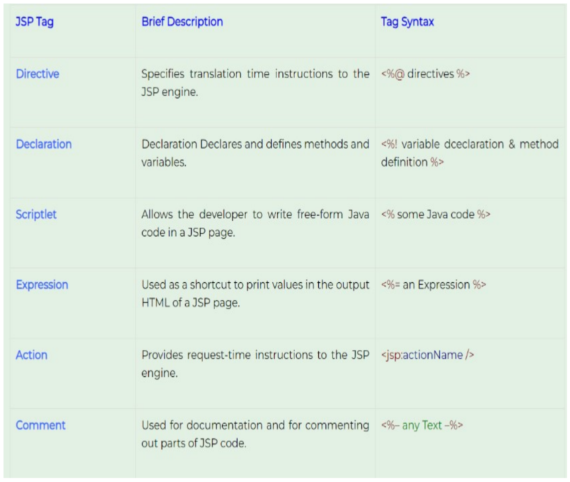
4.Server-Side Processing: Ensures secure and controlled data processing on the server.

5.Java EE Integration: Works well with Java EE for building robust web apps.

6.Session Management: Provides session handling for user-specific data.

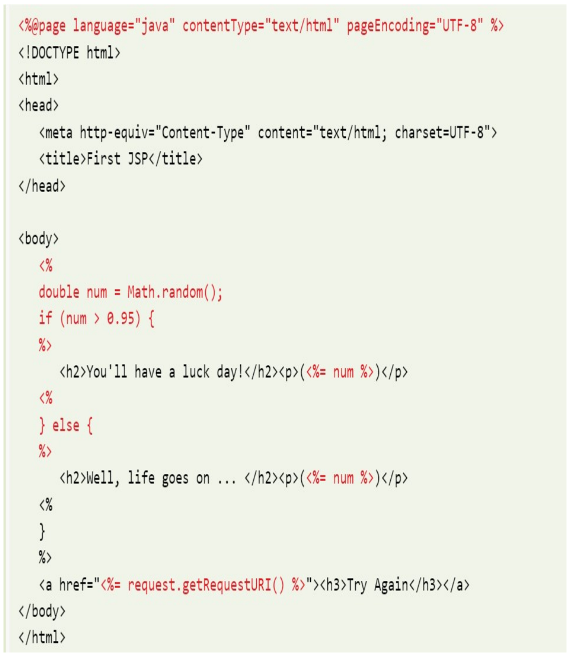
7.Expression Language (EL): Simplifies dynamic data insertion.

8.Tag Libraries: Offers pre-built tags for common tasks and better code organization.



JSP Tag	Brief Description	Tag Syntax
Directive	Specifies translation time instructions to the JSP engine.	<%@ directives %>
Declaration	Declaration Declares and defines methods and variables.	<%! variable dceclaration & method definition %>
Scriptlet	Allows the developer to write free-form Java code in a JSP page.	<% some Java code %>
Expression	Used as a shortcut to print values in the output HTML of a JSP page.	<%= an Expression %>
Action	Provides request-time instructions to the JSP engine.	<jsp:actionName />
Comment	Used for documentation and for commenting out parts of JSP code.	<%– any Text –%>

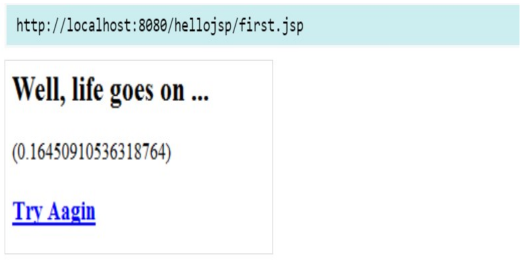
Fig3. Various Tags Used in jsp between Html Tags.



```
<%@page language="java" contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>First JSP</title>
</head>

<body>
   <%
   double num = Math.random();
   if (num > 0.95) {
   %>
      <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
   <%
   } else {
   %>
      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
   <%
   }
   %>
   <a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>
</body>
</html>
```

Fig4.How it is used in html block.

RESULT



http://localhost:8080/hellojsp/first.jsp

Well, life goes on ...

(0.16450910536318764)

Try Aagin

Fig5.Result of following Code

V. Server Side Handling Using Servlet:
A **servlet** is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.[3]For such applications, Java Servlet technology defines HTTP-specific servlet classes.

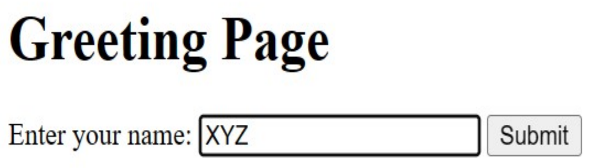
(JSP Page greet.jsp) as shown in Fig6. Jsp Page(client side).[1]

```
<!DOCTYPE html>
<html>
<head>
  <title>Greeting Page</title>
</head>
<body>
<h1>Greeting Page</h1>
<form action="greet" method="post">
  Enter your name: <input type="text" name="name" />
  <input type="submit" value="Submit" />
</form>
<p>
  <%
    String greeting = (String)
request.getAttribute("greeting");
    if (greeting != null) {
      out.println(greeting);
    }
  %>
</p>
</body>
</html>
```

(Sever Side Page Greeting.java) as shown result (fig7.)

```
package com.pkg.falcon;
import java.io.IOException;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/greet") // Use the @WebServlet
annotation to define the servlet mapping
public class GreetingServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {
    // Get the user's name from the request parameter
    String name = request.getParameter("name");

    // Create a greeting message
    String greeting = "Hello, " + name + "!";

    // Set the greeting message as an attribute in the
request
    request.setAttribute("greeting", greeting);

    // Forward the request to the JSP for displaying the
greeting

request.getRequestDispatcher("/greet.jsp").forward(req
uest, response);
  }
}
```

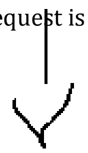
RESULT



Greeting Page

Enter your name: XYZ Submit

Fig 6.By hitting submit request is transfer to servlet



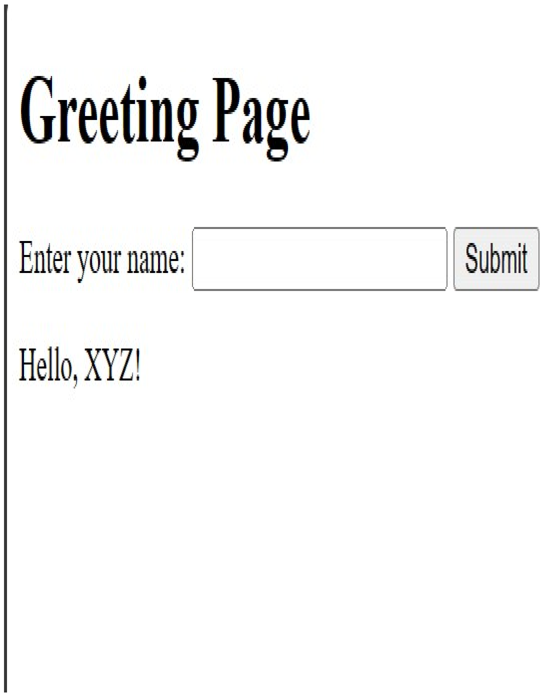


Fig7.Servlet Response

VI. JDBC And My Sql Role in Functioning Backend Part of Project.

JDBC (Java Database Connectivity) and MySQL play essential roles in the backend part of a web application for displaying dynamic content. Let's break down their roles:

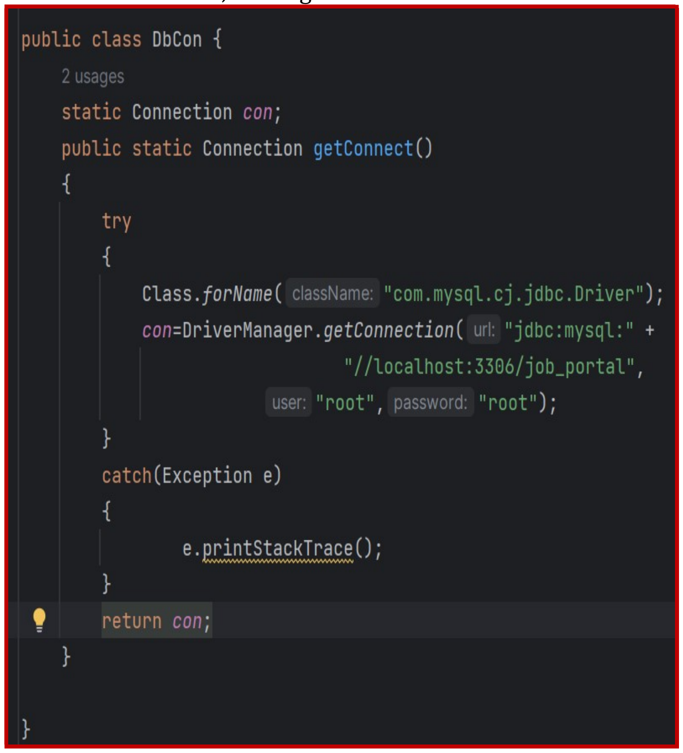
1.MySQL (Database):

MySQL is a relational database management system (RDBMS) that stores and manages data.
It is commonly used as a backend data store in web applications because it provides a structured way to organize and retrieve data.[4]
MySQL stores data in tables with rows and columns, making it suitable for a wide range of applications.

2.JDBC (Java Database Connectivity):

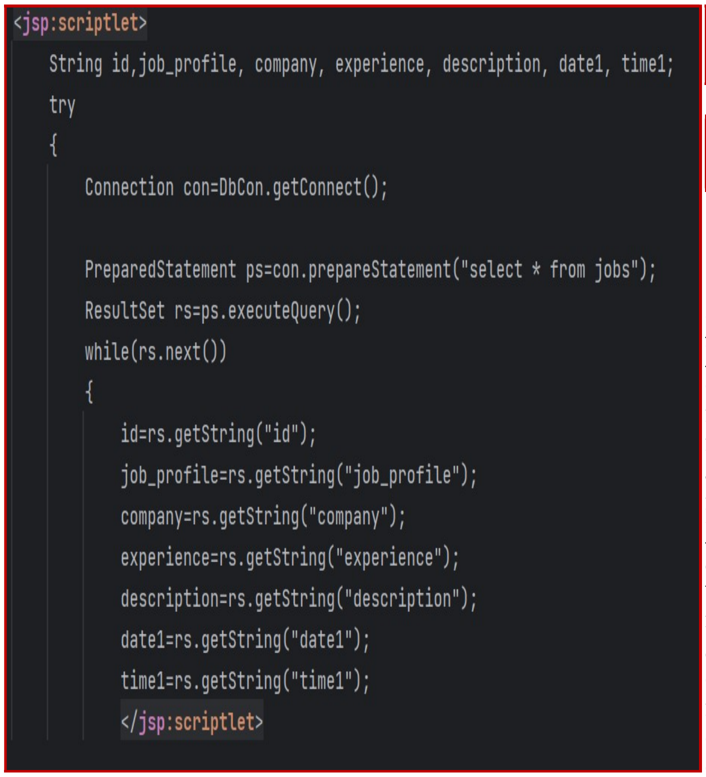
JDBC is a Java-based API that allows Java applications to interact with relational databases, including MySQL.
It provides a standard interface for connecting to databases, executing SQL queries, and processing query results.
JDBC enables Java developers to perform various database operations, such as inserting, updating, deleting, and retrieving data from a MySQL database.[2]
Here's how MySQL and JDBC work together to enable dynamic content in a web application:

1. Data Storage: MySQL serves as the backend data storage where you store information like user profiles, product details, and more.
2. Data Retrieval: JDBC is used in the backend Java code (e.g., servlets or server-side controllers) to connect to the MySQL database, execute SQL queries, and retrieve data based on user requests. For example, you might retrieve user records or product information from the database.[2]As shown in fig8.
3. Data Processing: Once data is retrieved using JDBC, it can be processed and manipulated in Java to prepare it for display. This may include formatting, sorting, and filtering data as needed.
4. Dynamic Content Generation: After processing, the Java code generates dynamic content based on the retrieved data. This content can be in the form of HTML, JSON, XML, or any other format suitable for the web application.
5. Display: The dynamically generated content is then sent to the client-side (e.g., a web browser) as part of an HTTP response.[7]The client-side technology (e.g., HTML, JavaScript, CSS) renders this content, making it visible to the end user.



```
public class DbCon {
    2 usages
    static Connection con;
    public static Connection getConnect()
    {
        try
        {
            Class.forName( className: "com.mysql.cj.jdbc.Driver");
            con=DriverManager.getConnection( url: "jdbc:mysql:" +
                                 "//localhost:3306/job_portal",
                        user: "root", password: "root");
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return con;
    }

}
```

Fig9 JDBC Connection[4]

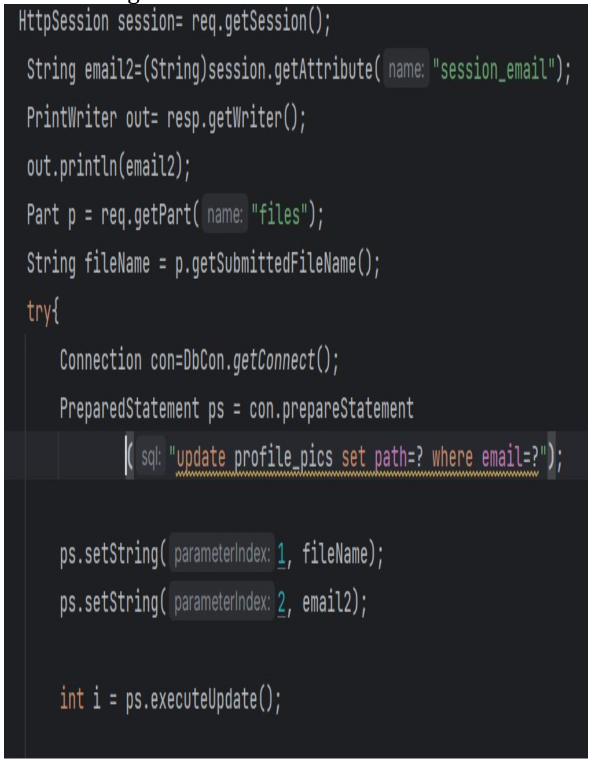


```
<jsp:scriptlet>
    String id,job_profile, company, experience, description, date1, time1;
    try
    {
        Connection con=DbCon.getConnect();

        PreparedStatement ps=con.prepareStatement("select * from jobs");
        ResultSet rs=ps.executeQuery();
        while(rs.next())
        {
            id=rs.getString("id");
            job_profile=rs.getString("job_profile");
            company=rs.getString("company");
            experience=rs.getString("experience");
            description=rs.getString("description");
            date1=rs.getString("date1");
            time1=rs.getString("time1");
            </jsp:scriptlet>
```

Fig 8.Connection call and Query Fetch

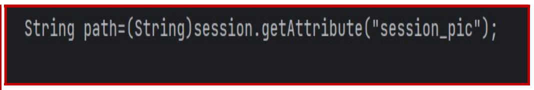
VII. How Jdbc Useful In File Handling.

Some function and class of core java is used to handle the file handling task like uploading profile pic or downloading when the user register first time in any website a default value is passed in databases table for getting the default value for the first time and then it is update by with the help of simple Sql Query[8]. As shown in Fig10 code.



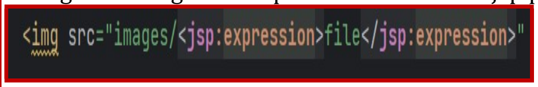
```
HttpSession session= req.getSession();
 String email2=(String)session.getAttribute( name: "session_email");
 PrintWriter out= resp.getWriter();
 out.println(email2);
 Part p = req.getPart( name: "files");
 String fileName = p.getSubmittedFileName();
 try{
     Connection con=DbCon.getConnect();
     PreparedStatement ps = con.prepareStatement
             ( sql: "update profile_pics set path=? where email=?");

     ps.setString( parameterIndex: 1, fileName);
     ps.setString( parameterIndex: 2, email2);

     int i = ps.executeUpdate();
```

Fig10. setting pic path and update in DB.



```
String path=(String)session.getAttribute("session_pic");
```

Fig11.Getting session pic name at another jsp page



```
<img src="images/<jsp:expression>file</jsp:expression>"
```

Fig12. Setting file path in img tag.

VIII. Conclusion

Java technology, encompassing JSP, JDBC, Servlets, and Jakarta EE, is a cornerstone of modern software development. It offers a unique blend of platform independence, making it possible to write code that can run seamlessly across diverse environments. Servlets and JSP facilitate dynamic web content and user interactions, with Servlets handling the core logic and JSPs taking care of the presentation layer. JDBC simplifies database connectivity, allowing developers to work efficiently with relational databases. Jakarta EE, formerly known as Java EE, empowers the creation of scalable and robust enterprise applications, providing essential features like transaction management, messaging, and security[5]. The extensive Java community, vast documentation, and rich library ecosystem foster collaboration and innovation. Furthermore, Java prioritizes security and reliability, employing automatic memory management and robust exception handling. In summary, Java technology's adaptability, reliability, and community support make it a foundational choice for building a wide range of software applications, from web solutions to large-scale enterprise systems.

References

[1]. Smith, J. (2020). Java Web Development: A Comprehensive Guide. Web Development Journal.
[2]. Brown, A., & Davis, C. (2018). Mastering JDBC: Best Practices and Techniques. Database Technologies.
[3]. Web Developer's Association. (2019). Core Java for Web Applications. Web Development Insights.
MySQL Documentation Team. (2021). MySQL: The Definitive Guide. MySQL Publishers.
[5]. Flanagan, D. (2018). Java in a Nutshell: A Desktop Quick Reference. O'Reilly Media.
[6]. Oracle. (2022). Java Persistence API (JPA).