Name: **Piyusha Rajendra Supe**

Enrolment Number: **23CO315**

# Web Technology – Case Study

**Title:** Transform the blogging application from a loose collection of various resources (servlets, HTML documents, etc.) to an integrated web application that follows the MVC paradigm

## 1. Introduction

In the evolving world of web development, the need for scalable, maintainable, and well-structured applications is more pressing than ever. Traditional web applications often begin as fragmented systems—a mixture of HTML pages, embedded Java code in JSPs, and independent servlets. While this structure may suffice for smaller applications, it becomes increasingly difficult to manage as the application grows.

This case study presents a systematic transformation of such a loosely coupled blogging application into a robust, modular, and professionally structured web system. The architectural redesign adopts the **Model-View-Controller (MVC)** paradigm—an industry-standard design pattern that cleanly separates the data, logic, and presentation layers of a web application.

## 2. Background

The initial version of the blogging system consisted of:

- JSP pages embedded with JDBC code
- Direct SQL calls in the presentation layer
- HTML layout and Java logic mixed together
- No clear separation of responsibility

As the application was extended with new features like database updates, content filtering, and styling enhancements, the code became increasingly complex and hard to debug or maintain. This served as a strong motivation to refactor the application using MVC architecture, making it scalable, testable, and future-proof.

## 3. Objective of the Project

The main objective was to **refactor and modernize** the existing blogging application to:

- Follow the **MVC (Model-View-Controller)** architectural design pattern
- Promote **separation of concerns** between business logic, data access, and user interface
- Make the application **more maintainable, reusable, and extendable**
- Enhance **code readability** and **modularity**
- Create a strong foundation for future enhancements

## 4. Understanding MVC Architecture

The **MVC** pattern divides an application into three interconnected components:

**Model:**

Responsible for managing the data, logic, and rules of the application. In this project, the model includes:

- A **Blog.java** class representing blog entries
- A **Blog.java** class handling database operations (e.g., fetching blog posts)

**View:**

Handles the visual representation of the data. This includes the HTML and CSS used to present blog content to users. In this project, it's implemented in:

- **blog_view.jsp**, which dynamically displays the blog data passed from the controller

**Controller:**

Acts as an intermediary between Model and View. It processes incoming HTTP requests, calls the appropriate model methods, and forwards the results to the view. Here, it's implemented as:

- **BlogController.java** (a Servlet)

This architectural separation simplifies both development and maintenance, allowing individual parts of the application to evolve independently.

## 5. Implementation Overview

**Model Layer**

- **Blog.java**: A simple JavaBean class representing a blog post with attributes: name, content, and posted date.
- **Blog.java**: Manages database connection, SQL execution, and returns a list of Blog objects fetched from the database.

**View Layer**

- **blog_view.jsp**: A clean, structured JSP page displaying blog entries with dynamic content rendering. It uses HTML/CSS for styling and avoids embedding any SQL or JDBC logic.

**Controller Layer**

- **BlogController.java**: A servlet that handles HTTP GET requests, interacts with Blog to retrieve data, and forwards the blog list to the view using RequestDispatcher.

## 6. Benefits of the MVC-Based Transformation

| Feature | Before (Loose Structure) | After (MVC Architecture) |
|---|---|---|
| **Code Organization** | Mixed Java, SQL, and HTML in JSP | Clear division between data, logic, view |
| **Maintainability** | Hard to trace or fix bugs | Easier to isolate and resolve issues |
| **Reusability** | Difficult to reuse code blocks | Modular components reused across the app |
| **Extensibility** | Adding new features is risky | New features added without disrupting others |
| **Scalability** | Poorly scalable structure | Scalable design for growing requirements |

## 7. Technologies Used

- **Java EE (Servlets, JSP)**
- **JDBC** for backend database communication
- **MySQL** for persistent data storage
- **Apache Tomcat** for application deployment
- **HTML/CSS** for frontend styling and layout

## 8. Outcome

The refactored application is now a well-structured, professional-grade blogging platform that:

- Fetches and displays blog entries from a MySQL database
- Follows a true **MVC pattern** with clean component responsibilities
- Is easier to maintain, extend, and debug
- Serves as a strong example for learners and developers adopting best practices in web technology

## 9. Future Scope

This MVC foundation can easily support the addition of new features such as:

- User authentication and comment systems
- Blog categories and search functionality
- RESTful APIs for mobile integration
- Admin panel for post management
- AJAX for dynamic user experiences

## 10. Conclusion

This case study successfully demonstrates how a fragmented web application can be transformed into an organized, modular, and maintainable system using the **MVC architecture**. By adhering to the MVC design pattern, the blogging platform has not only improved in terms of performance and structure but also laid a strong groundwork for future development.

This transformation reflects the core principles of modern web technology—**modularity, clarity, scalability, and maintainability**—and fulfils the objective of evolving from a loose collection of servlets and JSPs to a unified, integrated web application.

To convert your existing JSP-based blog viewer into an **integrated MVC (Model-View-Controller)** web application, we need to **modularize the code into three clear layers**:

**MVC Structure**

**1. Model (Blog.java & Blog.java)**

Encapsulates database access and data objects.

**Blog.java (Model class for a blog post):**

```
public class Blog {
    private String name;
    private String content;
    private String postedOn;
    public Blog(String name, String content, String postedOn) {
        this.name = name;
        this.content = content;
        this.postedOn = postedOn;
    }

    public String getName() { return name; }
    public String getContent() { return content; }
    public String getPostedOn() { return postedOn; }
}
```

**Blog.java (Handles DB operations):**

```
import java.sql.*;
import java.util.*;
public class Blog {
    private String jdbcURL = "jdbc:mysql://localhost:3306/blogdb";
    private String jdbcUser = "root";
    private String jdbcPass = "";
    public List<Blog> getAllBlogs() {
        List<Blog> blogs = new ArrayList<>();
```

```
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(jdbcURL, jdbcUser, jdbcPass);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM blogs ORDER BY posted_on
DESC");
            while (rs.next()) {
                blogs.add(new Blog(
                    rs.getString("name"),
                    rs.getString("content"),
                    rs.getString("posted_on")
                ));
            }
            rs.close(); stmt.close(); conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return blogs;
    }
}
```

**2. Controller (BlogController.java)**

Handles incoming requests and passes data to the view.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.List;
public class BlogController extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        Blog blog = new Blog();
        List<Blog> blogList = blog.getAllBlogs();
        request.setAttribute("blogs", blogList);
```

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

**3. View (blog_view.jsp)**

Only responsible for presenting data. No Java DB logic.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.*, model.Blog" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Blog Posts</title>
    <link rel="stylesheet" href="css/style.css" />
</head>
<body>
    <div class="container">
        <h2>Blog Posts</h2>
        <%
            List<Blog> blogs = (List<Blog>) request.getAttribute("blogs");
            for (Blog blog : blogs) {
        %>
            <div class="blog-entry">
                <div class="blog-meta"><strong><%= blog.getName() %></strong> | <%= blog.getPostedOn() %></div>
                <div class="blog-content"><%= blog.getContent() %></div>
            </div>
        <%
            }
        %>
    </div>
</body>
```

```
</html>
```

**web.xml Configuration (Servlet Mapping)**

```
<web-app>
    <servlet>
        <servlet-name>BlogController</servlet-name>
        <servlet-class>controller.BlogController</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>BlogController</servlet-name>
        <url-pattern>/blogs</url-pattern>
    </servlet-mapping>
</web-app>
```

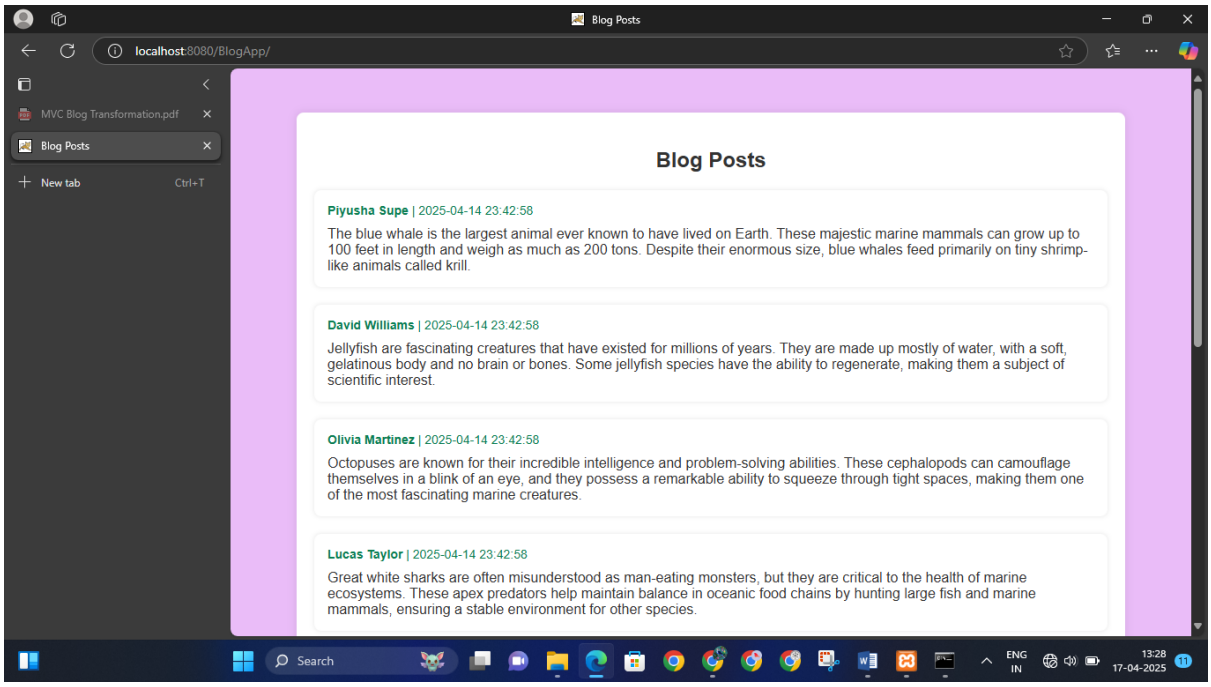
**Style.css**

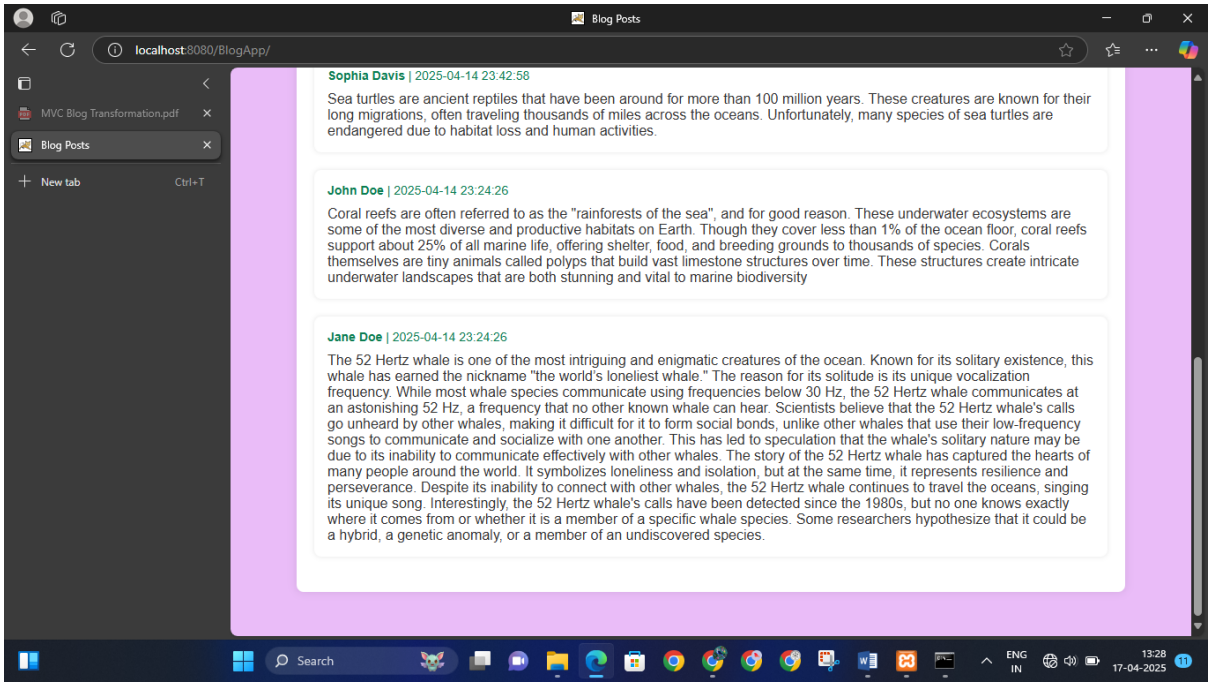
```
body {
    font-family: Arial, sans-serif;
    background-color: #eabcf8;
    margin: 0;
    padding: 0;
}

.container {
    max-width: 900px;
    margin: 50px auto;
    background-color: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h2 {
    text-align: center;
    color: #333;
```

```
}
.blog-entry {
    margin-bottom: 20px;
    padding: 15px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}
.blog-meta {
    font-size: 14px;
    color: #0b7e52;
    margin-bottom: 10px;
}
.blog-content {
    font-size: 16px;
    color: #333;
}
```

**Access the App: URL:**http://localhost:8080/BlogApp/

## Output






# Database Structure – database table