**Project Documentation: Traffic Management System**

**Project Overview**

The **Traffic Management System** is a web application designed to manage and display traffic data along with user information. It integrates JSP and Servlets to provide a dynamic and interactive user experience. The application demonstrates core Java EE concepts such as MVC architecture, JSTL, EL, and robust error handling.

**Table of Contents**

1. [Features](https://chatgpt.com/c/677a65a2-f600-8004-8adb-156605006e53#features)
2. [System Requirements](https://chatgpt.com/c/677a65a2-f600-8004-8adb-156605006e53#system-requirements)
3. [Project Structure](https://chatgpt.com/c/677a65a2-f600-8004-8adb-156605006e53#project-structure)
4. [Core Components](https://chatgpt.com/c/677a65a2-f600-8004-8adb-156605006e53#core-components)
5. [Setup and Deployment](https://chatgpt.com/c/677a65a2-f600-8004-8adb-156605006e53#setup-and-deployment)
6. [Testing](https://chatgpt.com/c/677a65a2-f600-8004-8adb-156605006e53#testing)
7. [Future Enhancements](https://chatgpt.com/c/677a65a2-f600-8004-8adb-156605006e53#future-enhancements)
8. Conclusion

**Features**

* **Traffic Status Display**: Displays the current traffic status and average speed.
* **User Management**: Lists registered users with details such as name and email.
* **JSP Integration**: Utilizes JSP pages for dynamic rendering of data.
* **JSTL and EL Usage**: Simplifies data rendering and looping in JSP.
* **MVC Architecture**: Separates business logic, presentation, and control.
* **Robust Error Handling**: Ensures graceful handling of unexpected scenarios.

**System Requirements**

**Software Requirements:**

* Java Development Kit (JDK) 8 or higher
* Apache Maven
* Apache Tomcat 9.0 or higher
* Web Browser (Google Chromeetc.)

**Hardware Requirements:**

* Processor: 2 GHz or faster
* RAM: 4 GB or higher
* Disk Space: Minimum 200 MB

**Project Structure**

project/

├── src/

│ └── main/

│ ├── java/

│ │ └── com/

│ │ └── trafficmanagement/

│ │ ├── TrafficManagementServlet.java

│ │ └── User.java

│ ├── resources/

│ └── webapp/

│ ├── WEB-INF/

│ │ ├── views/

│ │ │ └── dashboard.jsp

│ ├── styles/

│ │ └── styles.css

├── pom.xml

**Core Components**

**1. TrafficManagementServlet**

* Handles HTTP GET and POST requests.
* Retrieves traffic and user data.
* Forwards data to JSP for rendering.

**2. User JavaBean**

* Represents user data with properties: id, name, and email.
* Implements constructors and getters for data encapsulation.

**3. JSP Pages**

* **dashboard.jsp**: Displays traffic status and user data dynamically using JSTL and EL.

**4. Stylesheets**

* Adds basic styling for improved UI/UX.

**5. Dependencies**

* Managed using Apache Maven.
* Example dependency in pom.xml:
* <dependency>
* <groupId>javax.servlet</groupId>
* <artifactId>javax.servlet-api</artifactId>
* <version>4.0.1</version>
* </dependency>

**Setup and Deployment**

**1. Clone the Repository**

git clone <repository\_url>

cd project

**2. Build the Project**

mvn clean install

**3. Deploy to Tomcat**

* Copy the generated WAR file (target/project.war) to the webapps directory of your Tomcat server.
* Start the Tomcat server.
* Access the application at http://localhost:8080/project/traffic.

**Testing**

**Manual Testing:**

1. **Traffic Data Display**:
   * Verify the traffic status and average speed render correctly.
   * Example: Moderate Traffic, 40 km/h.
2. **User Data Display**:
   * Confirm all user details (ID, Name, Email) are displayed in the table.
3. **Error Handling**:
   * Simulate scenarios like missing data to check error page rendering.

**Automated Testing:**

* Add unit tests for DAO and service layers (if applicable).
* Example JUnit test:

@Test

public void testUserBean() {

User user = new User(1, "Test User", "test@example.com");

assertEquals(1, user.getId());

assertEquals("Test User", user.getName());

assertEquals("test@example.com", user.getEmail());

}

**Future Enhancements**

* **User Registration**: Add forms to register new users dynamically.
* **Database Integration**: Store traffic and user data in a database.
* **AJAX Updates**: Implement real-time traffic updates using AJAX.
* **Responsive Design**: Improve UI for mobile and tablet devices.

**Conclusion**

This project successfully demonstrates the integration of JSP and Servlets, providing a functional and dynamic web application. With clean code, modular architecture, and comprehensive testing, the

Traffic Management System is a robust solution ready for future enhancements.