**Bank Management System (ATM Machine Project) Documentation**

***1. Introduction***

***Project Overview***

*The Bank Management System (ATM Machine Project) is a web-based application designed to simulate the core functionalities of an ATM. This project is implemented using Java with a backend database connection via JDBC and front-end technologies including HTML, CSS, JavaScript, and Bootstrap. The purpose of this project is to simulate real-world banking features such as user registration, account management, balance checking, and transaction processing. It includes both a user interface for interactions and a backend to manage user data securely.*

***Key Features***

* *User registration and profile management*
* *ATM authentication (login/logout)*
* *Balance checking and money withdrawal*
* *Transaction history*
* *Error handling and validation*
* *Secure data handling via JDBC*

**2. Technologies Used**

Backend:

* Java: Core language used to build the system's logic and functionality.
* JDK: Java Development Kit used to compile and run the Java application.
* JDBC: Java Database Connectivity for connecting to a relational database (e.g., MySQL or PostgreSQL).
* Servlets: Java servlets used to handle HTTP requests and responses.
* JSP (Java Server Pages): Used for displaying dynamic content in the user interface.

Frontend:

* HTML: Structure and layout of the user interface.
* CSS: Styles the HTML templates to improve the user experience.
* Bootstrap: Front-end framework used for responsive design and styling.
* JavaScript: Used for client-side validation of forms and inputs.

Database:

* MySQL (or any other RDBMS): The database to store user data such as accounts, transaction history, etc.

Testing:

* JUnit: Unit testing framework to test DAO classes and services.

**3. Project Setup**

**Prerequisites**

* **JDK** installed on the system.
* **IDE** (e.g., Eclipse, IntelliJ IDEA) to write and compile Java code.
* **MySQL** database server (or any other relational database).
* **Apache Tomcat** or similar servlet container to deploy the application.
* **Maven** (optional) for managing project dependencies.

**Steps to Setup the Project**

1. **Clone the Repository:** If you are using version control (e.g., Git), clone the repository containing the project files:

bash

git clone <repository\_url>

1. **Import the Project into IDE:**
   * Open your IDE (e.g., Eclipse).
   * Import the project as a new Java project.
   * Ensure that the correct JDK version is set.
2. **Configure the Database:**
   * Install and configure MySQL (or your database of choice).
   * Create a new database and necessary tables (such as users, transactions).
   * Update the DBUtil.java (or equivalent file) with the correct database credentials.
3. **Maven Dependencies (if applicable):** If using Maven, include necessary dependencies for JDBC and Servlet API in the pom.xml file.
4. **Run the Application:**
   * Deploy the project to a servlet container like Apache Tomcat.
   * Access the application through http://localhost:8080 in your browser.

**4. Code Breakdown**

**Database Layer (DAO)**

* **DatabaseConnection.java:** Establishes a connection to the database using JDBC.

public class DatabaseConnection {

public static Connection getConnection() {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

return DriverManager.getConnection(DB\_URL, USER, PASSWORD);

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

}

* **UserDAO.java:** Contains methods for interacting with the users table (insert, update, select).

public class UserDAO {

public boolean registerUser(User user) {

Connection conn = DatabaseConnection.getConnection();

// SQL query for inserting a new user

// Execute query and handle result

}

public User getUserByUsername(String username) {

Connection conn = DatabaseConnection.getConnection();

// SQL query for selecting a user by username

// Execute query and return user object

}

}

**Service Layer**

* **UserService.java:** Provides business logic for user registration, login, and other operations.

public class UserService {

private UserDAO userDAO = new UserDAO();

public boolean registerUser(User user) {

return userDAO.registerUser(user);

}

public User loginUser(String username, String password) {

return userDAO.getUserByUsername(username);

}

}

**Servlet Layer**

* **UserServlet.java:** Handles HTTP requests and responses for user login, registration, and profile management.

@WebServlet("/user")

public class UserServlet extends HttpServlet {

private UserService userService = new UserService();

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

// Handle GET request (e.g., display user profile)

}

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

// Handle POST request (e.g., user registration or login)

}

}

**JSP Pages**

* **login.jsp:** The login page where users input their credentials.

<form action="user" method="post">

<input type="text" name="username" placeholder="Username" required>

<input type="password" name="password" placeholder="Password" required>

<button type="submit">Login</button>

</form>

* **profile.jsp:** Displays user profile information retrieved from the database.

<h1>Welcome, ${user.username}</h1>

<p>Balance: ${user.balance}</p>

**5. Features Implementation**

**User Registration and Profile Management**

* User registers through a form.
* The registration details are sent via POST request to the server.
* The server validates the data and stores it in the database.

**Login and Authentication**

* Users log in using their username and password.
* Authentication is done by checking the credentials against the database.

**Balance Checking and Transactions**

* Users can view their current balance.
* Money can be withdrawn from the balance (provided there are sufficient funds).

**Transaction History**

* A history of all transactions is stored in the database and displayed to the user.

**Validation**

* JavaScript is used for client-side form validation (e.g., checking for empty fields).
* Server-side validation ensures that the user inputs are sanitized to prevent SQL injection and other attacks.

**6. Testing**

**Unit Testing**

JUnit tests were written for the service and DAO layers. Below is an example of a test case for the UserDAO class:

java

Copy code

public class UserDAOTest {

private UserDAO userDAO;

@Before

public void setUp() {

userDAO = new UserDAO();

}

@Test

public void testRegisterUser() {

User user = new User("johnDoe", "password123");

boolean result = userDAO.registerUser(user);

assertTrue(result);

}

}

**Manual Testing**

* Verified all forms (login, registration, withdrawal) for correct input validation.
* Tested edge cases such as invalid login credentials and negative balance scenarios.

**7. Conclusion**

**Project Summary**

This Bank Management System provides a simple but effective simulation of an ATM's core functionalities. It uses Java for the backend logic, MySQL for data storage, and JSP/HTML for the user interface. The application demonstrates how front-end and back-end technologies can work together to create a seamless user experience.

**Future Improvements**

* Implement more advanced security features such as password hashing.
* Add features like account transfer, deposit functionality, and ATM transaction limits.
* Improve the user interface with more sophisticated design patterns.

**8. References**

* Official Java Documentation: <https://docs.oracle.com/javase/>
* MySQL Documentation: <https://dev.mysql.com/doc/>
* Servlet and JSP Documentation: https://jakarta.ee/specifications/