Our web system designed by using MCV architecture

Our hotel reservation management system has been meticulously designed using the Model-View-Controller (MVC) architecture. This architecture provides a structured and modular approach to system development, ensuring separation of concerns and promoting maintainability, scalability, and flexibility.

In our system's design, the Model represents the data and business logic. It encompasses the core functionality of the reservation management system, such as guest information management, room availability tracking, and payment handling. The Model encapsulates the data structures and algorithms required to perform these operations efficiently.

The View layer focuses on the presentation and user interface aspects of the system. It handles the visual representation of information, allowing users to interact with the system through an intuitive and user-friendly interface. The View layer in our hotel reservation management system showcases room availability, facilitates online booking processing, and presents multimedia features to guests, ensuring a seamless and engaging user experience.

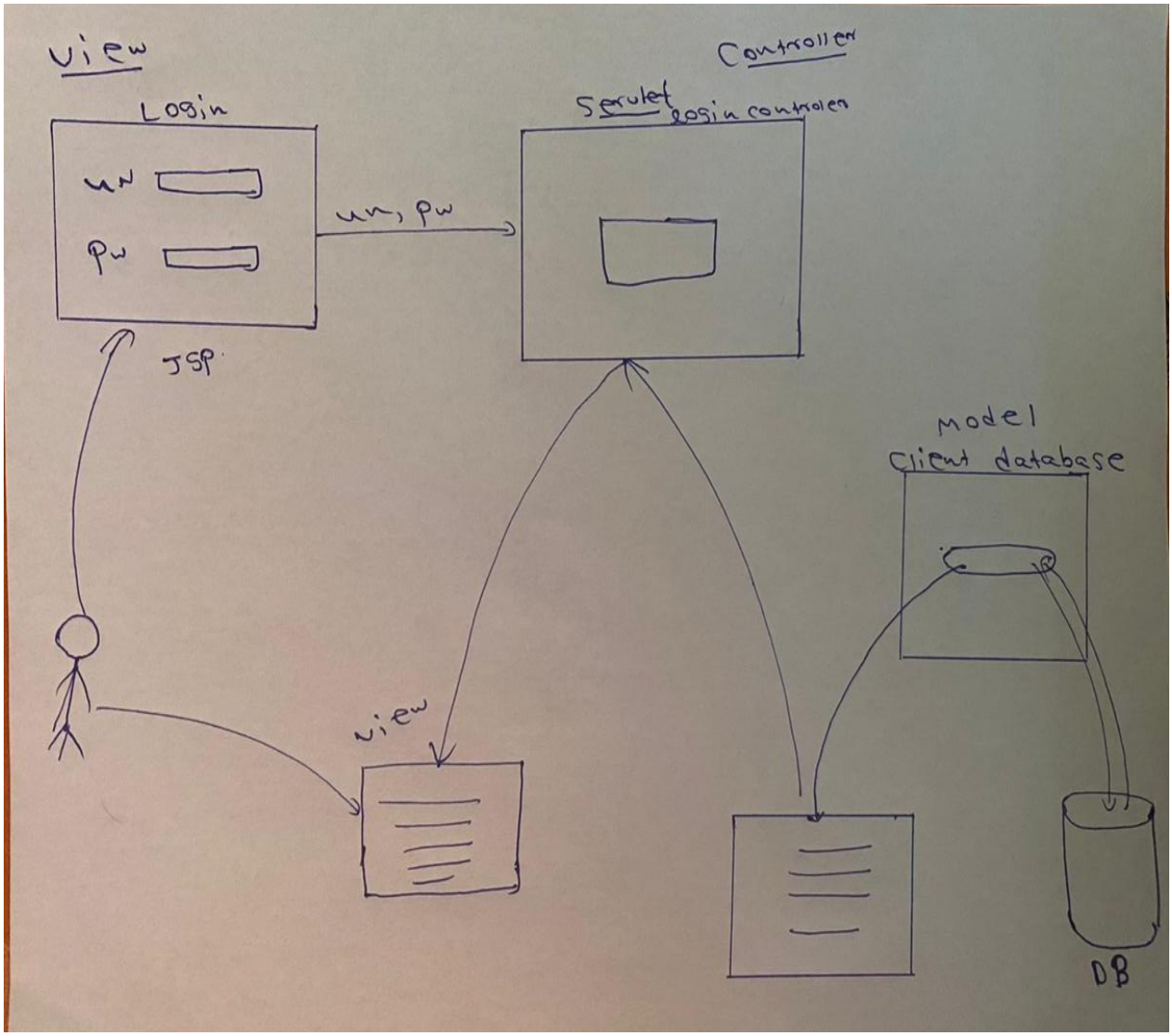
The Controller acts as the intermediary between the Model and the View. It receives user input from the View and communicates with the Model to perform the necessary operations. The Controller ensures that the system responds to user actions appropriately, updating the Model and updating the View accordingly. In our hotel reservation management system, the Controller handles tasks such as assigning rooms, managing reservations, and processing payment transactions.

The MVC architecture used in our hotel reservation management system allows for modular development and easy integration of new features or enhancements. The separation of concerns between the Model, View, and Controller enables developers to work on different components independently, promoting code reusability and maintainability.

Furthermore, the MVC architecture facilitates scalability, as each component can be scaled independently based on specific requirements. For example, if the system needs to handle a larger volume of reservations, the Model and Controller components can be optimized and scaled without impacting the View layer.

our hotel reservation management system follows the MVC architecture to ensure a well-structured, modular, and maintainable design. By leveraging this architecture, we have created a system that effectively manages hotel reservations, enhances the guest experience, and provides a solid foundation for future enhancements and scalability.

Our login system designed by using mvc architecture



In our login system, we have implemented the Model-View-Controller (MVC) architecture to ensure a well-structured and modular design. Let's break down the roles of the Model, View, and Controller components in our system:

Model: The Model component in our system represents the data and business logic. In this case, it includes the client database where user information, such as usernames and passwords, is stored. The Model is responsible for validating user credentials by searching the database for a matching username and password combination. It ensures the integrity and security of user data and performs the necessary operations to authenticate user login attempts.

View: The View component is the user interface that users interact with to enter their login credentials. In our system, the login page serves as the View. It presents the username and password fields, allowing users to input their login information. The View is responsible for capturing user input and forwarding it to the Controller for further processing.

Controller: The Controller component handles the logic and flow of the system. In our case, the Controller is created using a servlet, a Java-based server-side component. When a user submits their login credentials from the View (login page), the Controller receives this data. It then interacts with the Model to validate the user's credentials by querying the client database. Based on the validation result, the Controller determines the next step in the login process. If the credentials are valid, it redirects the user to the next page, typically the user's dashboard or a designated landing page. If the credentials are invalid, appropriate error messages can be displayed to the user.

By using the MVC architecture in our login system, we ensure a clear separation of concerns and maintainability. The Model handles the data and business logic, the View presents the user interface, and the Controller manages the flow and interactions between the Model and the View. This design approach allows for modularity, scalability, and easier maintenance and enhancement of the system in the future.