James Elliot
Alekzander Green
Aakansha Arora
Khoa Nguyen

Project Architecture

For our project, we identified two primary architectures that we will be using within our project. These architectures are the Client-Server architecture pattern and the MVC architecture pattern. For this reason, both are covered in the following pages.

Client- Server Architecture

What is Client-Server Architecture

In the client-server architecture, servers and clients interact to provide services and share resources over a network. In the chatroom application the client interacts with the central server, which handles the message routing and storage.

How will we apply Client-Server Architecture

We believe this is a potential system architecture for our project for the following reasons. We will also briefly explain how we would implement it.

1. Centralized Control and Network Communication:
   Because our project will need to support multiple connections to the same chatroom, it is beneficial to use server-client architecture to separate the clients and server, allowing for real-time updates to clients. The server and clients will be connected to a network, either global or local. The server will have an IP address or domain to which the client can request to connect. Furthermore:
   - Server Setup: the server will be set up with the required resources and functionality that will provide a client with a specific service. Potential services could be getting messages, sending messages, creating chatrooms, or deleting chatrooms, etc.
   - Listening to Requests: the server will be programmed to listen for client requests on a particular port. To achieve this, the client will send requests to the server IP with port address. Clients initiate communication by sending requests to the server. These requests typically include a particular action or command that describes what the client wants the server to do.
   - Request Processing: on receiving the request from the client, the server will process the request, develop a response, and return a result to the client.
   - Response Handling: on receiving the response from the server, the client will perform an appropriate action based on the request and response.
   - Statelessness: client- server communication is usually stateless, i.e., each request from client to server will be independent. This will be an important feature for a chatroom as each service will be independent from each other like creating new chatrooms, sending messages, opening embedded IDEs, etc.

2. Scalability

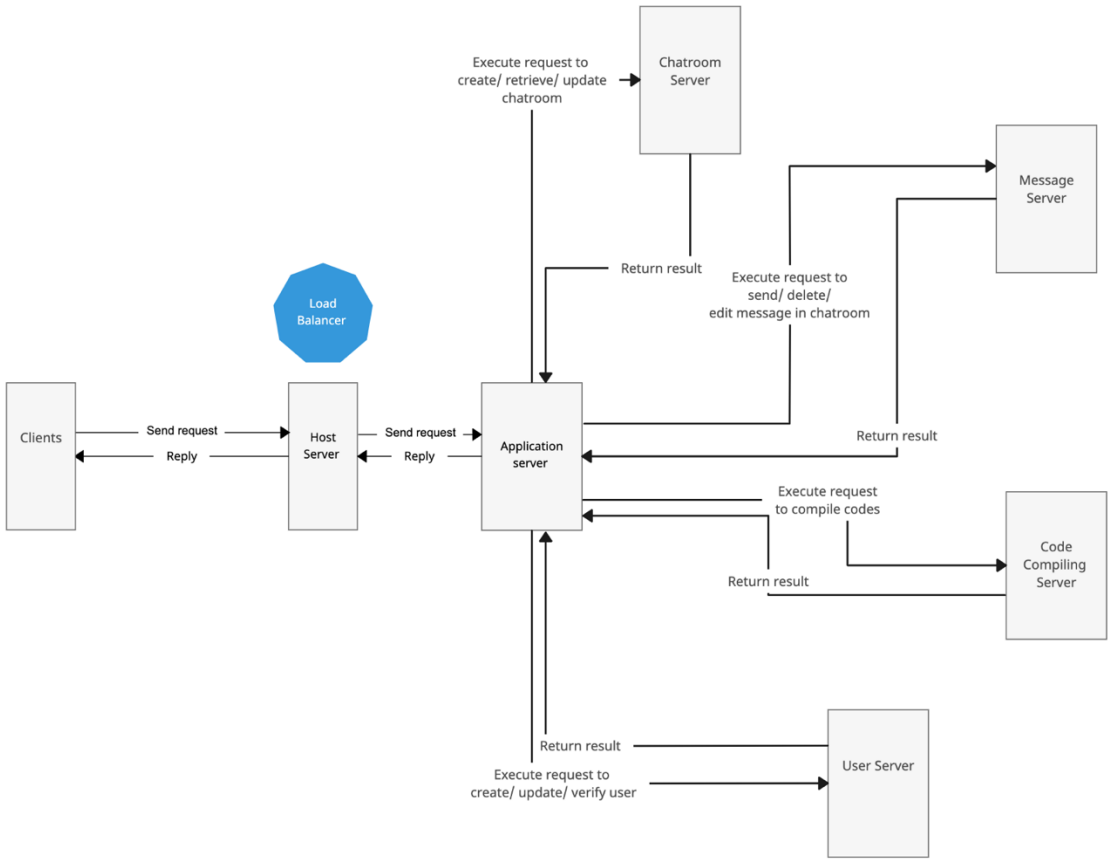
   A client-server architecture is usually used to handle large numbers of clients simultaneously. Load balancing and clustering techniques can be applied to ensure flawless communication with a large number of clients and servers.

3. Security

   Security measures, such as authentication, authorization, and encryption, are essential components of client-server systems to protect data and ensure that only authorized clients can access the server's resources. This will be an important feature in the chatroom to protect user information.

James Elliot
Alekzander Green
Aakansha Arora
Khoa Nguyen

Client- Server Architecture Diagram



Explanation of the Client-Server Diagram

In this diagram, there are 2 main components: clients and the application server. The client is the front-end, which is responsible for displays and UI, and the application server is the backend, comprising of different layers that each control a specific functionality, such as database operations, data controlling, etc.

When a client makes a request of any kind, it will be passed to the host server then to the application server. The application server will then pass this to a specific server that is responsible for that request. For example, when a new user is registered, it will be passed to the user server to handle that request. Unlike MVC architecture, client-server does not represent the inner structure of backend well.

James Elliot
Alekzander Green
Aakansha Arora
Khoa Nguyen

Model-View-Controller (MVC) Architecture Pattern

What is MVC

In the Model-View-Controller (MVC) architecture pattern, the presentation, interaction, and data systems are separated into three logical components: the view, controller, and model, respectively. This breakdown allows presentations, interactions, and data to be handled independently from one another so that changes in presentation, interaction, or data do not (generally) require updates to all three.

How we will apply MVC

We will use this pattern in our project to represent and interact with records in our database. For instance, we will have message records that will be represented by a Message model; the messages can be interacted with using the Manager controller; and the message model can be displayed using the Message Display view, allowing us to view the message record in a formatted way.

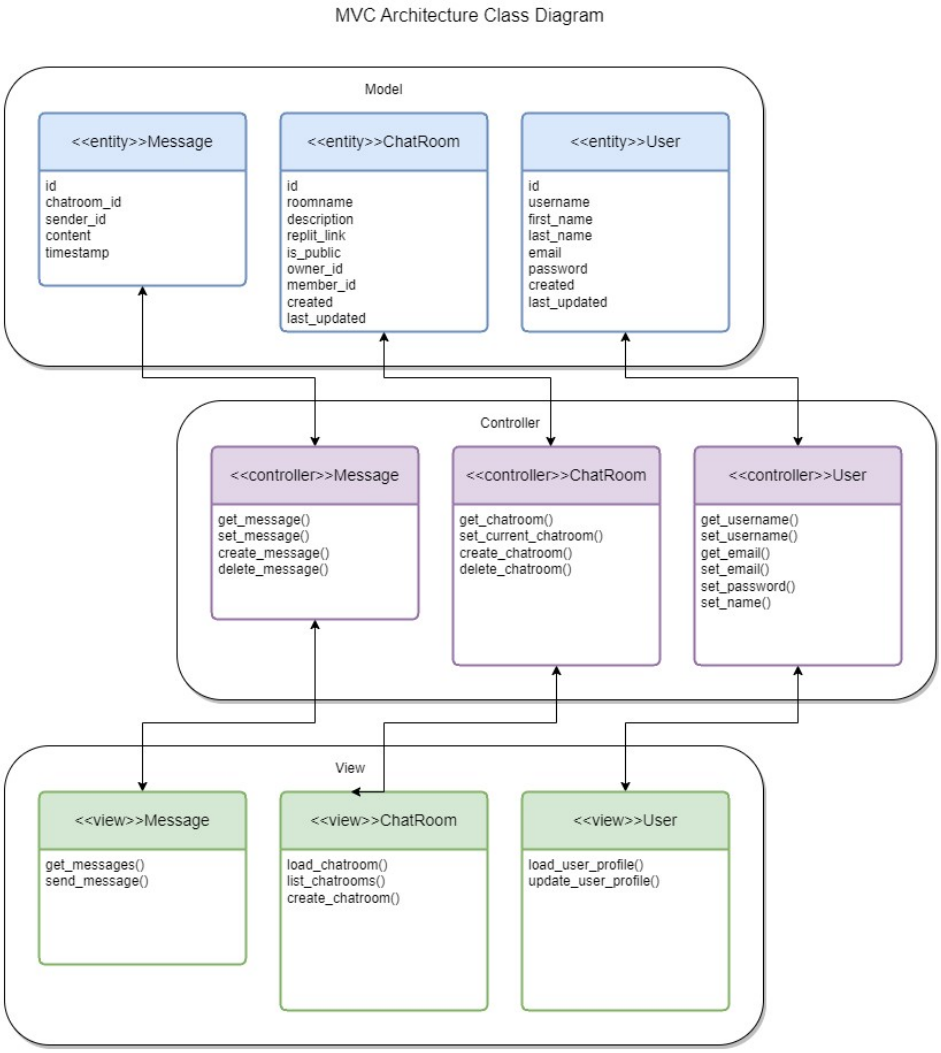
Advantages of applying MVC to our project

One advantage of using this architecture pattern in our project is that since the models and controllers are logically separated from the views, we can have the backend be responsible for the models and controllers and allow the frontend to be responsible for the views. This is important for our project because the backend and frontend use different programming languages.

Another advantage of using this architecture pattern is that we can develop our frontend and backend separately, allowing us to build models on the backend and views on the frontend independently, only changing the controller when the view or model changes.

James Elliot
Alekzander Green
Aakansha Arora
Khoa Nguyen

MVC Class Diagram

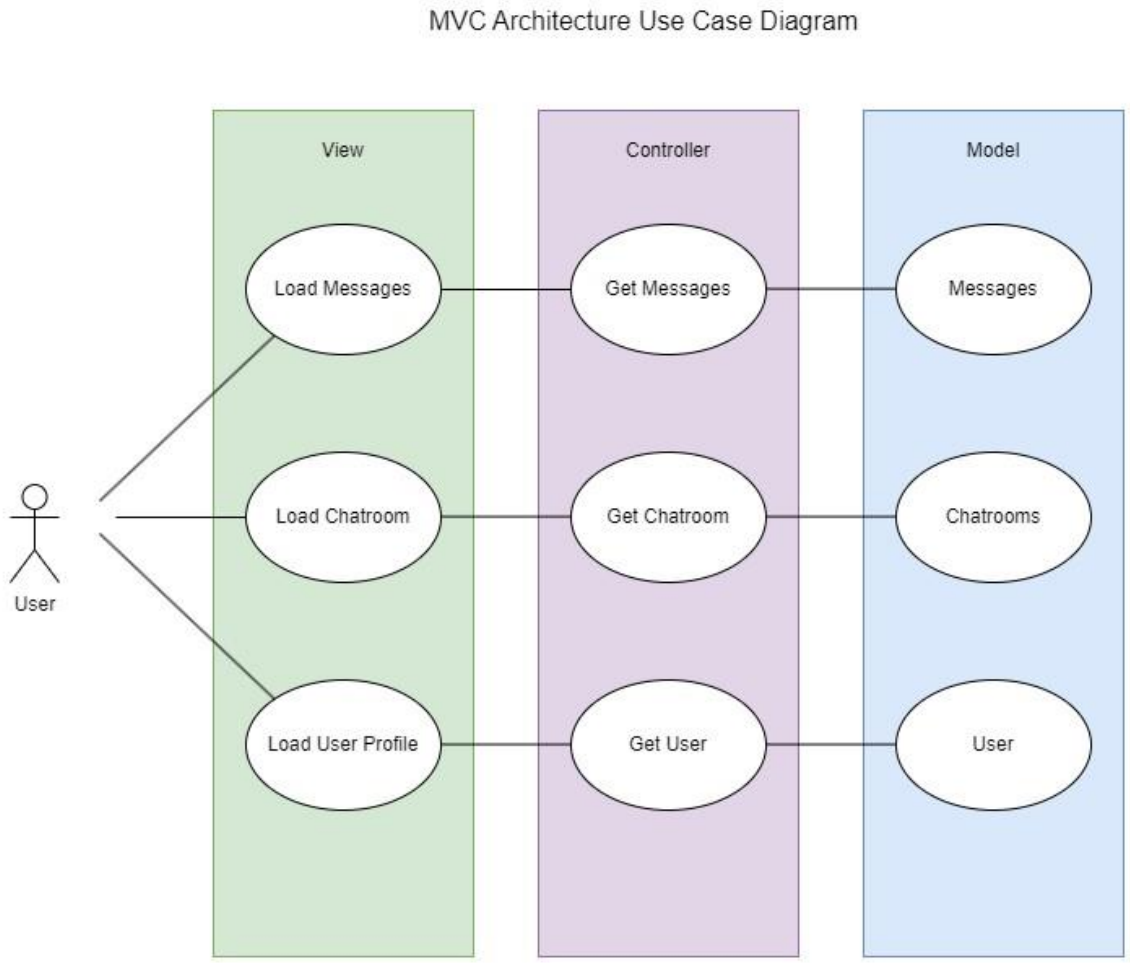


Explanation of the MVC Class Diagram

The reason we used the Class Diagram to visualize the MVC Architecture pattern is that it allows for the organization of large web applications into a View, Mode, or Controller. It also captures and defines the structure of classes or classifiers, and it shows common roles or responsibilities that define each class. For example, we know that the Message class will be handled by the Message controller and displayed using the Message view. This principle can be applied to the rest of the classes for the Chat application.

James Elliot
Alekzander Green
Aakansha Arora
Khoa Nguyen

MVC Use Case Diagram



Explanation of the MVC Use Case Diagram

The reason we used a Use Case Diagram to represent our MVC Architecture is because it allows for the goals of the system and users can be easily represented. Someone can look at this diagram and understand the role of the user, view, controller, and model. This is also good for representing the flow of events when it comes to user interaction. One thing to note about Use Case diagrams is that they align with business requirements. When you mix the MVC Architecture pattern with the Use Case diagram, it allows for easy planning and implementation.