# 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.

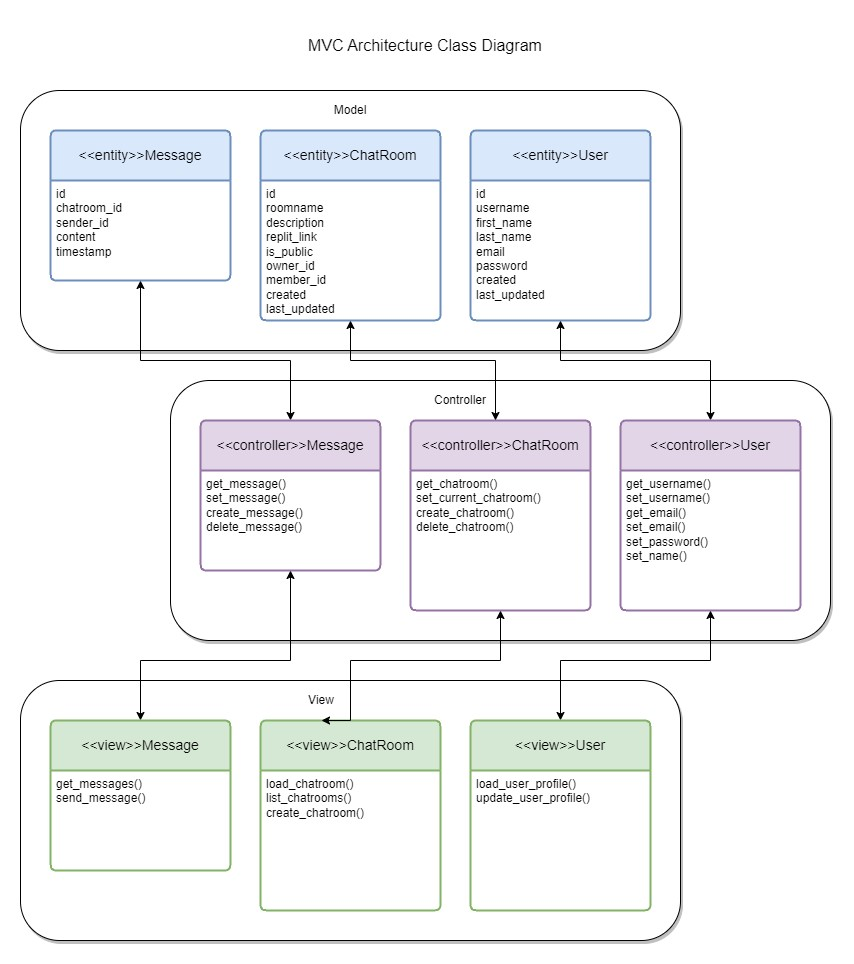
## Advantages of applying MVC to our project

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.

# MVC Diagrams

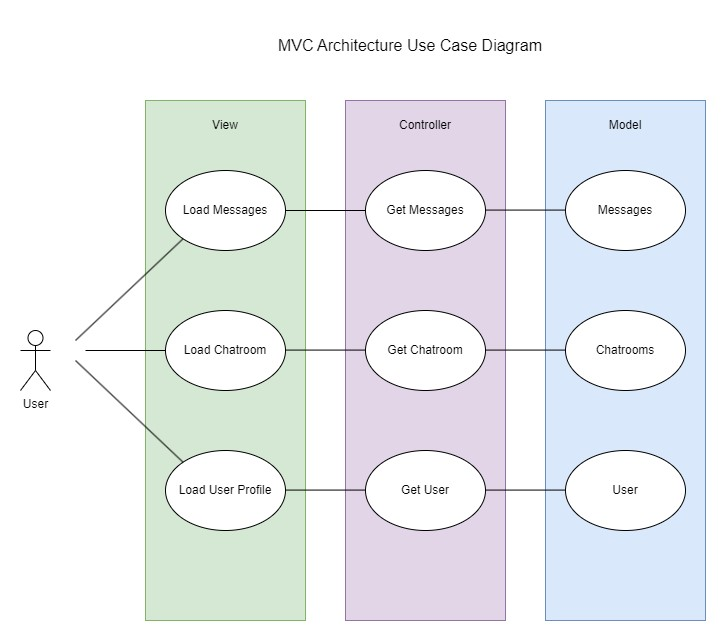
## MVC Class Diagram



## Explanation:

The reason we used the Class Diagram to view 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 the class. For example, we know 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.

## MVC Use Case Diagram



## Explanation:

The reason we used a Use Case Diagram to represent our MVC Architecture so 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.