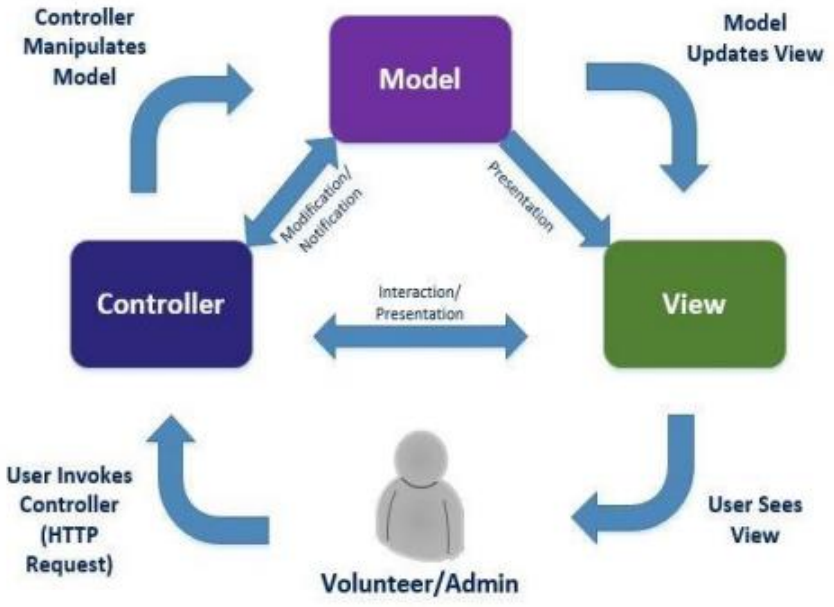
Summary

Model View Controller will be used as the main architectural design for UNF's School of Computing Hours System. The system will also feature architectural components that encapsulate parts of the Client-Server pattern.

Model



Purpose

The project's main architectural design is based on the Model, View, and Controller pattern. Due to the purposes behind the project, the system has a need for all CRUD elements. Users accessing the system will need to create reports upon which faculty members will need to be able to read these reports. A system administrator will also need to be able to update and delete reports as needed. Users/clients utilizing the system will have to make these requests to the server through the web API which in turn will act upon the database, inserting/deleting based on the respective user. Any user that uses the system will be expected to access it via most web browsers on any mobile or static device given that it has access to the internet. Users will be constantly updating their profiles with every report they submit, and hours and reports will be logged into the system. At the admin level, they will be able to filter through these submitted reports and manipulate this recorded data within the database. MVC Architecture is key in this process and allows for this whole process to be cohesive and efficient for the purposes behind our system. Users will utilize the views to use buttons and click links to call controller methods which in turn manipulate the model in order to access and display specific reports/data back to the view for users to utilize. All these processes interact with each other in specific patterns to ensure all data flows correctly and efficiently.

Architectural Rationale

Divide and Conquer

- The Model, View, and Controller architecture embodies the idea of dividing and conquering. Each competent of this architecture is separated by three main logical components: model, view and controller. The user interacts with views which render the controller logic as interactions within the system which in turn manipulate data within the model.

Increased Cohesion

- The Model, View and Controller architecture is separated into three partitions of logic. However, each section relies on one another to create a responsive and overall, more cohesive structure. Each part of the structure flows perfectly to create our web application and ensure everything runs smoothly; this is only accomplished because of the MVC architecture.

Reduce Coupling

- Fundamentally the idea of MVC reduces the threat of coupling in code bases. Coupling is prevented by isolating components and preventing entities from propagating from one to another.

Increase Reuse

- Model, View, and Controller architecture allows for better reuse of code. While maintaining high cohesion of code, MVC allows for the reusability of code in the future. Because of the separation of these logical structures, it makes them easier to reuse.

Design for Flexibility

- The way MVC is structured may seem rigid due to the separation of these key components but, it creates a more flexible environment. If a new view is to be created based on an older controller it should be created with minimal problems. Controllers do not have to be overcomplicated and the redesigning of Views and Models alike will not be required with changes to older controllers.

Design for Testability

- The MVC architecture allows for excellent testability and debugging. Unit testing allows for singular views to be tested alongside the coinciding models and controllers associated with them. In many cases this is used alongside the entire project and makes debugging a easy process to hash out small bugs in subdivisions of the project.