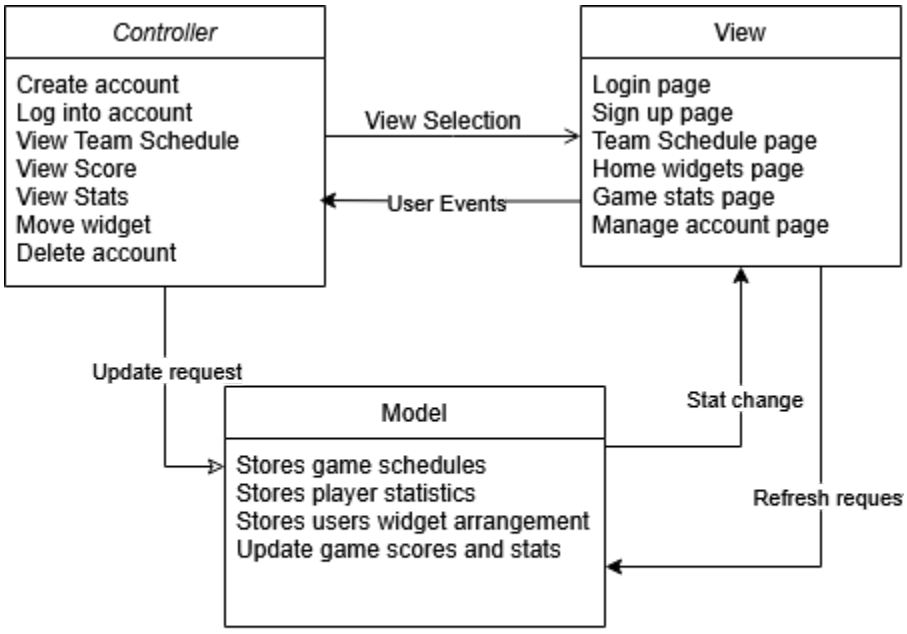
Team: Dudes

Architecture Design:

For designing the architecture of Black Bear Bulletin, we designed a Model-View-Controller architecture. We chose an MVC architecture because it separates the program into multiple more digestible parts. By separating the architecture of BBB into the three parts, controller, view, and model, it allows for easier separation of tasks and allows for more flexibility. This MVC architecture will ensure that multiple components do not directly interact with each other, allowing for more flexibility throughout the application.

Architecture Design Diagram:



Design Description:

The architectural pattern consists of three components, the view, the controller and the model.

The view for our application starts with a login page, where the user can continue as a guest, login in using their account, or move onto a sign up page for making a new account. Once a user is signed in, the home page will be displayed. The home page will display widgets which the user can add, delete, and move around. From the home page, the user can move to both the Team schedule page and the Game stats page. The Team schedule page will show the schedules for a selection of teams made by the user. The Game stats will show what is happening in the games of said selected teams. The user can access an account management page from any of the logged in pages, where the user can edit the settings of their account. React and bootstrap will be the libraries in use for the view.

Starting with the Login page, if the user presses create account, the controller will set the view to the Sign Up page. Once the user confirms their information on the Sign Up page, the controller will send that information to the Model, and move the view to a new Home page. If the user presses the login button, the view changes to the Login screen. Once the user enters their

username and password, the controller passes that information to the model, and if the model returns a user profile the controller tells the view to open that user's Home page. When the user edits their widgets, the controller sends those changes to the user profile in the model. If the user switches to the Team Schedule, the controller tells the view to display the Team Schedule page, then tells the model to update the view on what teams to display and their schedules. When the Game Stats page is selected, the controller tells the view to switch to the Game Stats page, and tells the model to continuously update the view as to what games are going on and what the scores are. When the user clicks off the Game Stats page, the model is told to stop updating the view. When the Manage Account button is clicked, the controller gets the view to switch to the Manage Account button, and when the user hits the save button the controller sends any changes made to the model.

The model stores user profiles in a secure database. When the model receives an update from the Sign Up page, it sets up a new user profile with the username and password the user chose, then sets all widgets and profile settings to default and the following teams list to an empty list. When a log in update is received, the model finds the given username, then checks to see if the password matches. If they don't, the model sends back an incorrect password error. If the username can't be found, the model sends a null user error. If the username is found and the password matches, the model sends the widget settings from the user profile to the view. When the model receives a Team Schedule update, it checks for the schedule of all of the teams in the following teams list from the NCAA scraper library. It then sends the following teams list and the schedule of those teams to the view. When a Game Stats update is received, the model uses the NCAA scraper library to find any games that are involving a team from the following teams list, then send those games to the view, and continually scrapes and updates the scores of said games until the control tells it to stop. When any updates to the widget or user profile settings are received, the model makes those changes in the user profile, then sends those updates to the view.