Natnael Abraha & Josh Barber

Bruce Montgomery

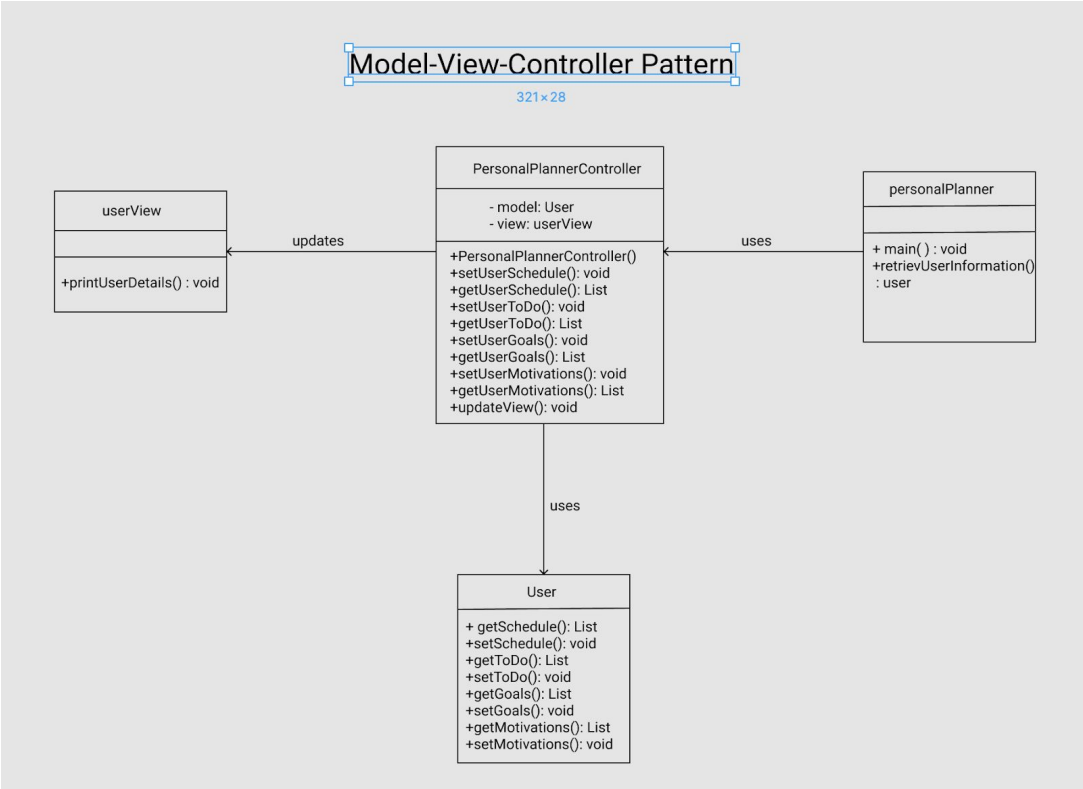
CSCI 4448

9, Dec 2019

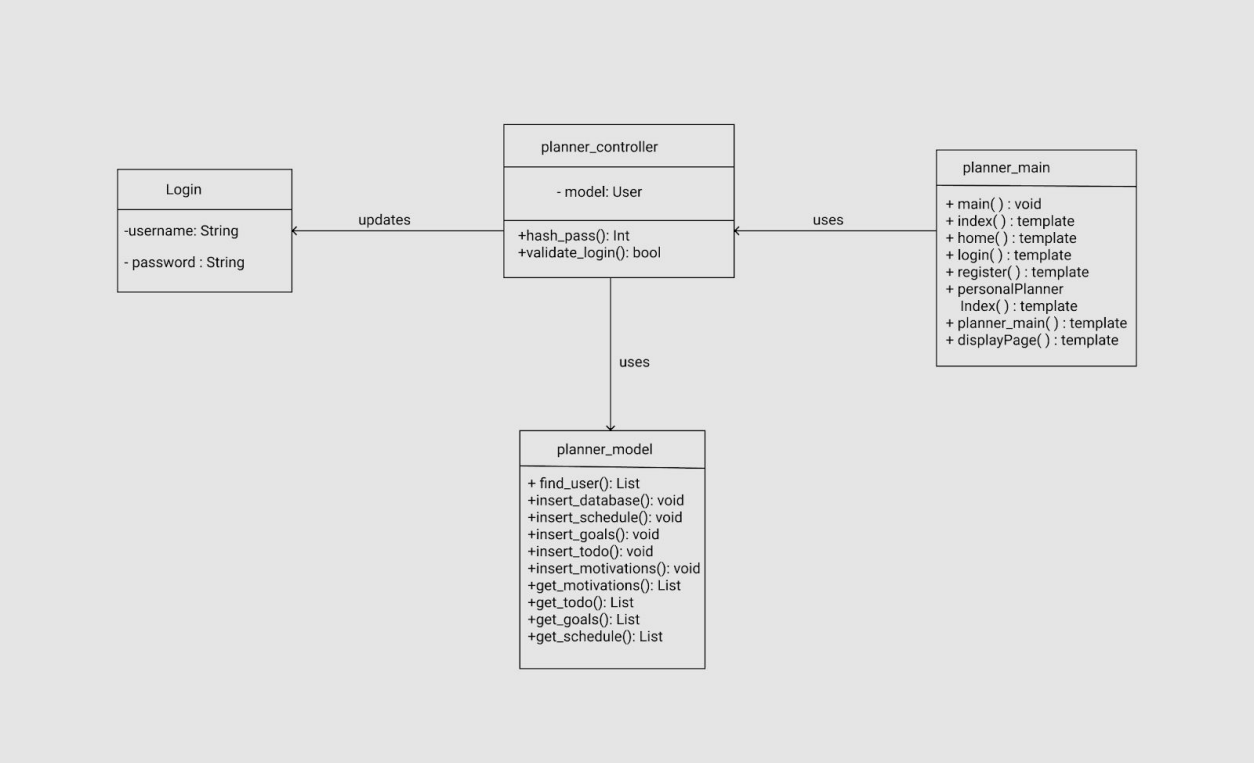
Final State of System Statement: The final state of our system performs exactly as intended. The uses cases that were outlined in Project 4 were able to be met and function properly. The features that were implemented include users being able to register for an account for the website in the database, users being able to login with their account, users being able to document their daily schedule, to-do list, goals, and motivations, as well as make edits to the specified category. All features that were previously planned were able to be implemented.

Final Class Diagram and Comparison Statement:

Class Diagram submitted in Project 4



Final Class Diagram



In our final project, the patterns that we wanted to use and outline was the MVC pattern since our project handles a database and updates the user's view based on the information retrieved from the database. Using Flask, it was difficult to implement the Controller pattern so our project makes use of the MVT pattern (Model, View, Template). In our project, the Model pattern is what sets and uses the schema for our MongoDB database while the View Pattern gets the results from our database and manipulates the data. In our project, the View is what gets the requests and responses, which is page redirects, page errors, and loading templates. The Template pattern is what utilizes our HTML and CSS files for displaying the users information.

In our final diagram we have changes being done to the database to be handled by planner_model, while in our original diagram some of those changes were also being affected by the planner_controller. Inside the planner_controller we've added a hash_pass( ) method which hashes the passwords and a validate_login( ) method which ensures a valid login. Within the planner_main file we have all the routes, with methods that render the pages for our website. We didn't have those methods on our original diagram because we weren't familiar with the flask framework and what methods we would need to render the pages. Insofar as our planner_model, the methods that we had in our original diagram are the same, but we've also added a find_user( ) method which finds a user in the database.

Statement on the OOAD process for your overall Semester Project:

One of the major problems we faced while working on our semester project was how to integrate the MVC pattern with the flask framework. The python framework Django makes use of the MVT pattern so we modeled our pattern after that. Another major issue that we spent a long time working on was passing html input data into the mongo database using ajax (Jquery). We had a lot of issues with data types and having ajax reach the flask app route methods. After a lot of trials and research we were able to solve this problem and successful input data into our database. Another negative issue we had was learning the python framework Flask to create our project along with PyMongo, a Flask plug-in. Many hours were spent reading different documentations of how the framework operated with the different plug-ins and how to combine Flask with JavaScript.

Third-Party code vs. Original code Statement:

Third-Party Code

We used https://www.w3schools.com for html and css tutorials.

We used https://www.w3schools.com/howto/howto_css_login_form.asp as a tutorial on how to create a view for a login page

We used https://github.com/PrettyPrinted/mongodb-user-login as a guide to set up our flask framework and create a login and registration system.

We used https://www.w3schools.com/howto/howto_js_todolist.asp tutorial for the javascript implementation of outlining a list structure in html for our project

Original Code

We've implemented the controller, model and main/view MVC pattern by modifying the flask framework referenced from https://github.com/PrettyPrinted/mongodb-user-login.

All HTML and CSS files are original in our system.