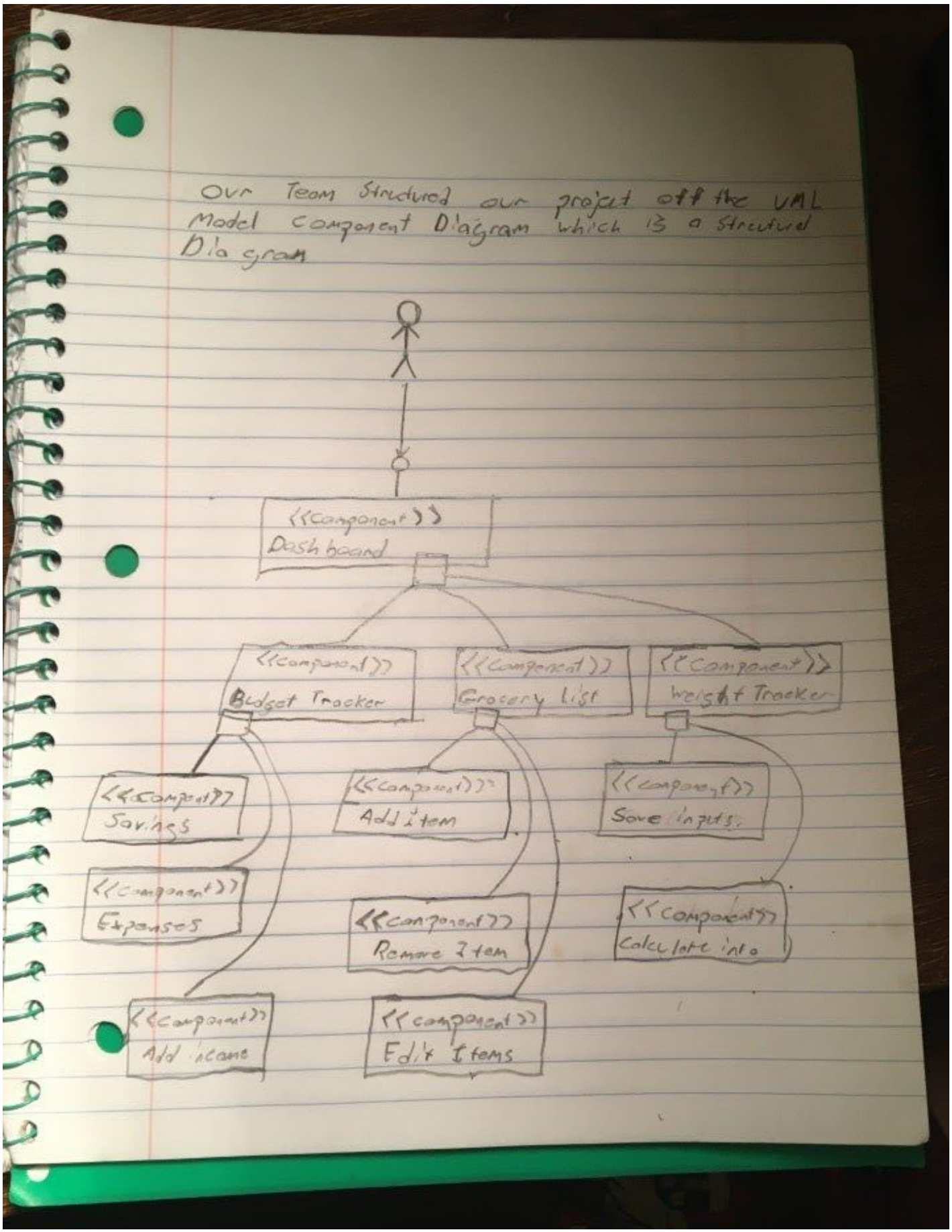
Design Paradigm- Component

For our project we decided to implement the component-level design paradigm. This paradigm best fits our project due to the nature of our project using three distinctly different applications. So our application can then be split into three different "components" that come together to interact with the client. In each of these three main components there are unique implementations, data structures, algorithms, etc. that are used to care out their objectives. Additionally, for each the three applications their objectives can be split into different components. For example, the budget tracker component can be split into three or more components. One of those components can have the main purpose of calculating and keeping track of all the current amount of money in one's bank account and any positive updates (aka deposits/income) that can be made to the account. Then there is a component that adds and keeps track of a list of expense categories that user has and then allow the user to update the amount in those categories by inputting expenses. Finally, there's a component that calculates whether or not the use had met their savings goal based upon their total income and total expenses. Each of these three components in the budget tracker have their own implementations, algorithms, communications, etc. Similar to the overall project, those three components come together to create the budget tracker application. In addition, each of the three main components can be reused in other applications or replaceable in our project by another application because each of the three main applications exist have their own definitions and our independent of one another. Considering that at each level of the project there are separate and independent components that have their own objectives that can be come together to form our project, we decided that the component-level design encapsulates our project's design.

Software Architecture- Pipes and Filters

Our lifestyle management application has several components which interact together. One part of the project consists of a budget tracker which has components that are combined to give a better user experience and a great performance in terms of providing useful data to the user. The information from one component is sent to another, then filtered and then showed to the user according to the person's goals and preferences. The information provided by the user goes through a series of analysis and computations, before a feedback and/or the results of the analysis is/are printed out on the screen. A weight tracker component also follows the same logic. The user gets to pick their goals and the program helps them follow the process of achieving those goals. That is done by tracking the records of previously recorded weights and comparing them together, then comparing with the user's weight goal. The third part of the project is a grocery list. We give the user an interface to create a grocery list and edit it as they wish, then help them figure out the total cost of their list of items. We may implement a relation between budget and grocery lists to help the user find out how much they are spending on each category of groceries. As a result, the Pipes-and-Filters Architecture is the one that fits best with our project. Each component can be implemented and updated independently. We have an easier way of managing interactions with the user. So, the maintenance of the whole project can go smooth.

UML modeling diagram- Component



Design Pattern(s)- Observer, State, Singleton, Façade(maybe)

For our project design, we used Singleton, Façade, Observer, and State design patterns to help figure out the design of our application. First, we utilized the singleton design pattern to create our project. Since our project is a web application we wanted to have one main interface that the user interacts with, so we only gave the user one point of access to our application. We are able to do this by using routing which allows the user to change the feature that is shown on that one main interface when they click on one of the linked tabs. Then, we utilized the Façade design pattern to structure our application because the framework Angular allows you to create a unified interface that utilizes sets of interfaces and sets of services defined in different components of the application. This design then allowed us to build a complex Angular application that can access multiple services and components at any moment. Next, we utilized the observer design to help define how the application's objects change when the user inputs a new value. Since our project is an interactive web application, this pattern helped us define the dependencies between objects when there is a new input into one of our many objects. This then allows all of the dependents of an object to be notified and updated automatically. Lastly, we utilized the state design because we have three different features in our application. Therefore whenever the user changes the feature shown, we can allow the main interface of our application to change its behavior.