**Design**

**System Design**

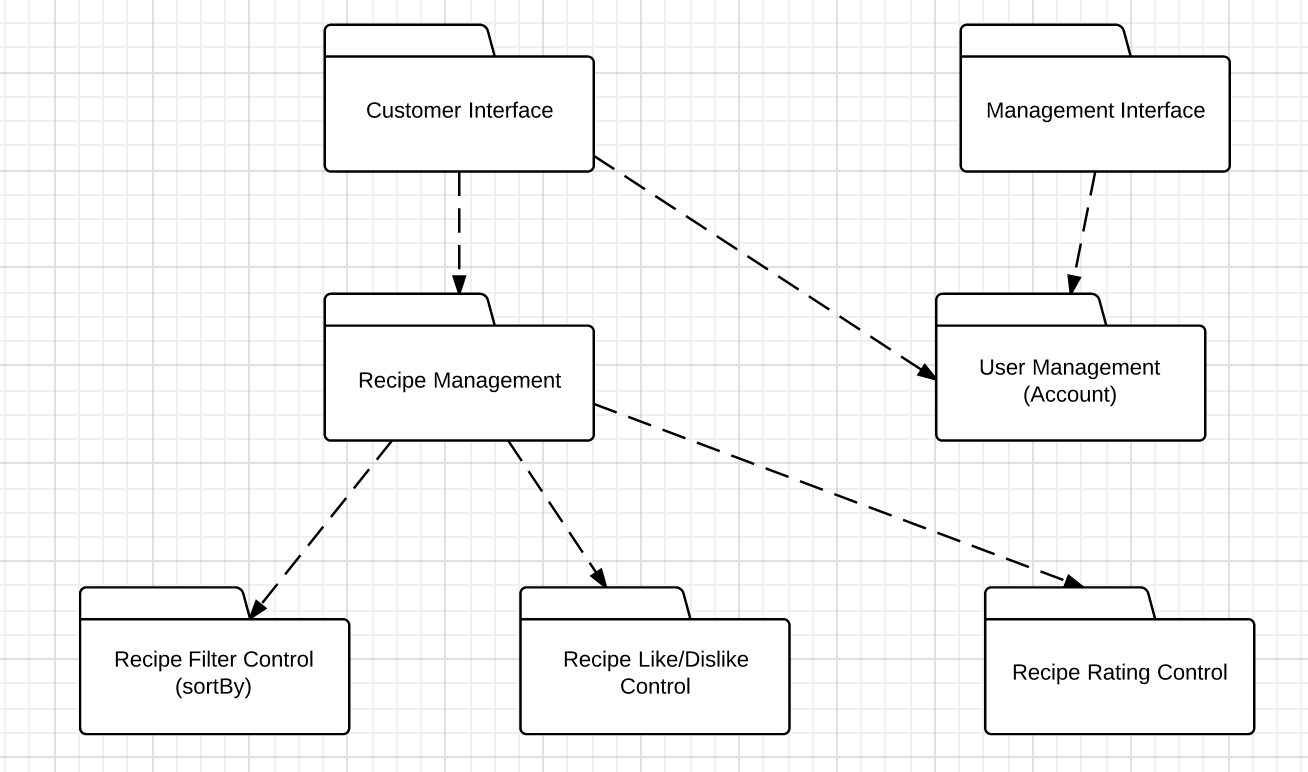
**Subsystem Design**

* Our system will be composed of seven (7) major subsystems being:

1. **Customer Interface**
2. **Management Interface (Administrator and Corporate)**
3. **User Management**
4. **Recipe Management**
5. **Recipe Filter Control (sortBy)**
6. **Recipe Like/Dislike Control**
7. **Recipe Rating Control**

* The approach to designing the seven major subsystems was to break down the subsystems to the lowest level where one subsystem could be implemented by one developer.

**High-Level Subsystem Design**

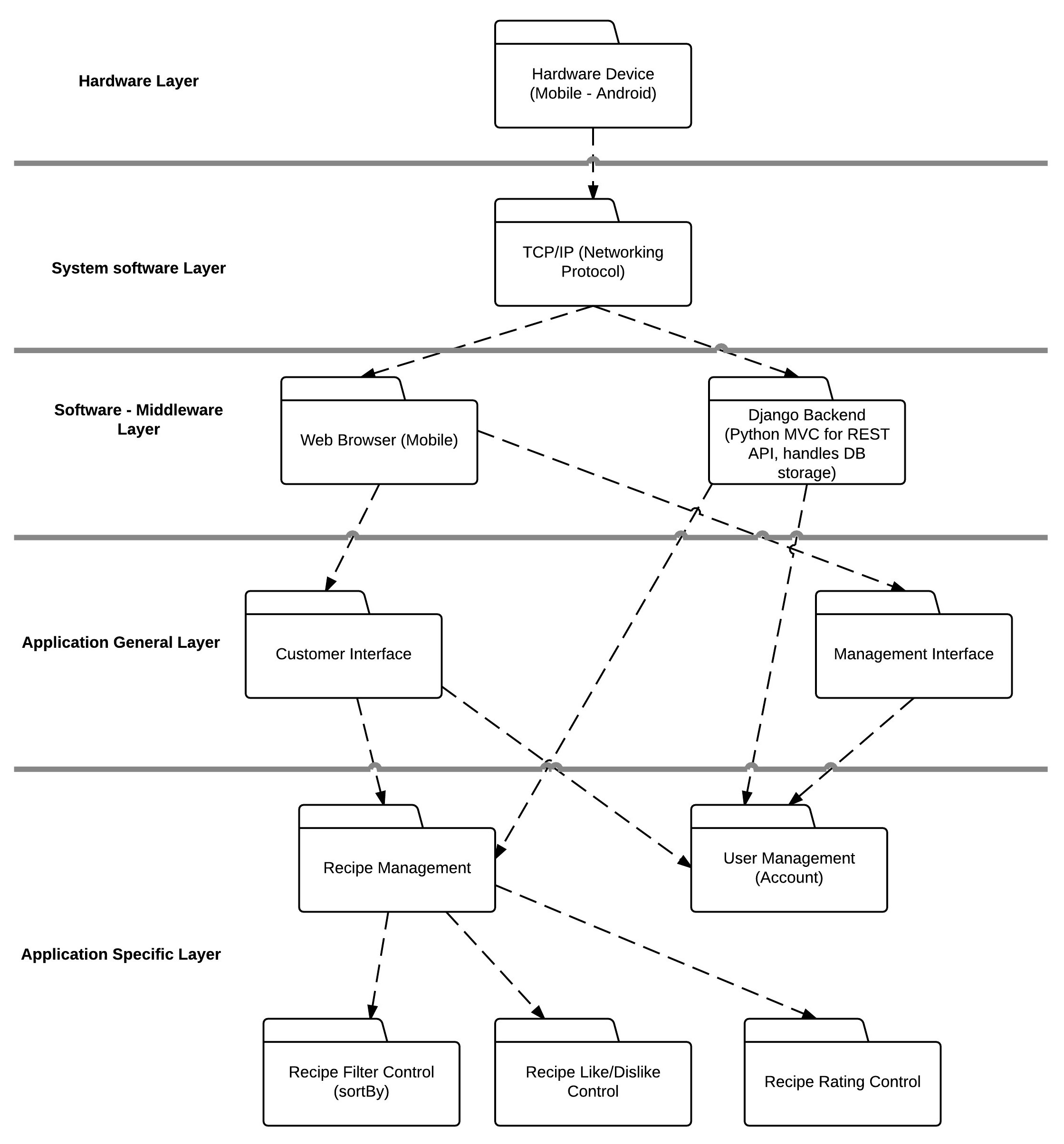


**Subsystem Responsibilities**

1. **Customer Interface**: This subsystem is responsible for the handling all UI design and user interface aspect of the application. Meaning that this interface will complete the link of human-machine interaction (machine being Android or some other mobile device) and will only serve the application’s users. This subsystem will have only features available to users.
2. **Management Interface**: This is the UI and user interface for management and corporate in a real world scenario. This interface provides extended management features such as banning abusive users and deleting recipes that are deemed offensive and abusive in terms of content.
3. **User Management**: This subsystem is responsible for all account related functionality such as editing profile and deactivating your account. This subsystem was also made to be responsible for working parallel with payment management subsystem should that need arise in the future where such a cooking app provides exclusive features who purchase an account subscription.
4. **Recipe Management**: This subsystem is responsible for handling all recipe related functionality such as creating recipes, editing an existing recipe, and deleting a recipe. Other functionality such as rating, like/dislike, and sorting are siphoned off to the subsystems below, as they are major features that require their own subsystems to keep this current one uncluttered.
5. **Recipe Filter Control**: This subsystem is responsible for returning filter/sort by data to the recipe management subsystem based on certain parameters that are predetermined. These parameters being like sort by upload date, popularity, rating, etc.
6. **Recipe Like/Dislike Control**: This subsystem is responsible for handling all the like and dislike related functionality.
7. **Recipe Rating Control**: This subsystem is responsible for handling the rating of a recipe (ex. Rate this recipe 5/5 stars) functionality.

**Dependencies**

* The dependencies are outlined by the **dotted lines** in the high-level subsystem design diagram. The most crucial dependencies in our system design are between **Recipe Management** and **Recipe Filter Control, Recipe Like/Dislike Control, Recipe Rating Control**. The **Recipe Management** subsystem depends on those three subsystems to ensure it has full functionality in the handling of recipes.

**Subsystem to Hardware Mapping**

The diagram depicted above shows the subsystems as layered and mapped to the corresponding **middleware**, **software**, and **hardware**. As per specific subsystem mapping to the hardware the Android hardware will be using TCP/IP networking protocol will be communicating directly with the web browser which will render the front end interface and it will be communicating with our backend (which handles the application specific subsystems) to perform functions such as adding recipes, user logins, and etc.

**Access Control**

The access control is limited between each subsystem. In specific to the *Application Specific Layer* the there is access between *Recipe Management* and the *Recipe Filter Control*, *Recipe Like/Dislike Control*, and *Recipe Rating Control*. There is also access between the *Recipe Management* and *User Management* however, the access is restricted in this partition level. There is only read access in this partition level between subsystems. The write and execute access for data is restricted to each particular subsystem itself.

**Data Storage**

Data storage for the *Application Specific Layer* will be done through the *Django* backend. *Django* is a Python MVC web framework that will handle all data communicate and data modification via a RESTful API. *Django* will create a link between itself and *PostgreSQL*. *PostgreSQL* is an object-relational database management system (ORDBMS) will manage and store the data. *Django* will also handle certain *Application Specific Layer* data that will be queried very frequently so some sort of caching is required to ensure API call response times are less than **1.5s**. This data will be handled and stored in an in-memory key value store like *Redis* or *memcached* to ensure high-speed data retrieval.

**Design Rationale**

We as a group agreed on a system design, which is a combination of both layering and partitioning.

We chose layering because we needed hierarchical structure from layering to define parent child relationship for certain classes to avoid class cluttering and over complication. We follow Object Oriented Programming (OOP) so we definitely needed the hierarchical structure. We have an open architecture meaning that some layers will access layers at deeper levels (ex. Django backend [Middleware layer] accesses *Recipe Management* and *User Management* [Application Specific Layer]).

As per partitioning, we chose this because we needed some subsystems to work directly and communicate with other subsystems in that *partition*. For ex. *Recipe Management* subsystem communicates with *User Management* in parallel to determine whether someone has the sufficient permissions to post a recipe or edit a recipe (we don’t want someone editing a recipe that doesn’t belong to them). Thus, the following reasons why we went with a combination of both layering and partitioning when choosing a system design.

**Object Design**

**Object Descriptions**

**Entity Objects**

* User
  + This object stores both a username and password for login validation and profile information such as recipes added by user, personal profile information, recipes liked and user status (active or banned). The login control object and the main control object use this object.
* Recipe
  + This object stores information regarding a recipe such as the recipe name, the prep and cook time, the ingredients list, the cuisine category, step by step directions, the difficulty rating, a photo of the dish, the number of servings, the username of the user who posted the recipe, and any extra notes. The main controller object to add or pull query results from the recipe database uses this object.

**Boundary Objects**

**Button Classes**

Button classes are used in the application to notify the main controller of a user’s intent to perform an action. Each button object is derived from the abstract class Button.



Sign In Button

* An instance of this button is contained by the Main Page object. When clicked, the Main Controller object displays the Login Page object and creates an instance of a Login Controller object.

Login Button

* An instance of this button is contained by the Login Page Object. When clicked, the Login Controller object queries the User Database to verify the inputted user information.

Cuisine Dropdown Menu

* An instance of this button is contained by the Main Page Object. When a specific cuisine button is clicked, the Main Controller object queries the database for corresponding recipe instances.

Like Recipe Button

* An instance of this button is contained by Recipe Page Objects. When clicked, the Main Controller object makes corresponding changes to the User in the User Database.

Rate Recipe Button

* An instance of this button is contained by Recipe Page Objects. When clicked, the Main Controller object makes corresponding changes to the Recipe in the Recipe Database.

Add Recipe Button

* An instance of this button is contained by User Profile Page Object. When clicked, the Main Controller object displays an Add New Recipe Form.

Delete Recipe Button

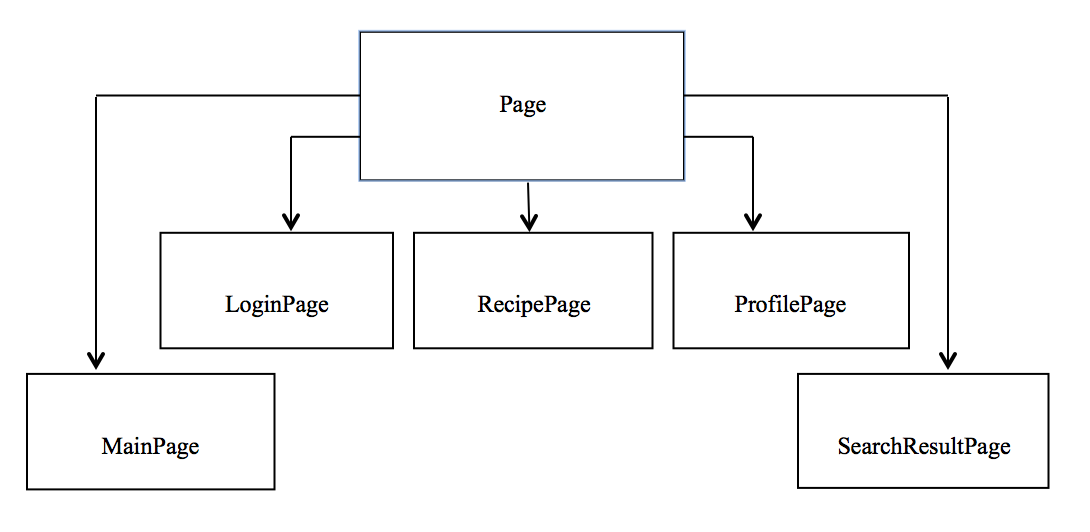
* Instances of this button are contained by the User Profile Page Object. When clicked, the Main Controller object deletes the corresponding recipe from the Recipe Database and makes necessary changes to the User Database

Logout Button

* An instance of this button is contained in every instance of a Page. When clicked, the Main Controller object logs out the user.

Page Classes

* Page classes are used in the application whenever a user navigates to the corresponding page. Each page consists of information regarding the other objects used on the page and various buttons that perform functions or link to other pages. Each page class is a child derived from the abstract class, Page.



MainPage Class

* This is the main page the user sees when opening the application. This page accesses the recipe database and displays a few recently added recipes. Clicking on any of these will create an instance of the RecipePage class for that recipe. This page also contains an instance of the sign in button. When clicked, the main controller creates an instance of the LoginPage class.

LoginPage Class

* This page contains input fields for the user to enter their username and password. It also contains an instance of the login button, which validates the user’s username and password and logs them into the system.

RecipePage Class

* This is the page that is displayed when a recipe is clicked from the main page, the search result page or the profile page. This page contains information about the recipe such as rating, ingredients, cuisine category, etc. This page also contains an instance of the rating button, which allows users to rate the recipe.

ProfilePage Class

* This page contains user information such as name, recipes liked, etc. It also contains a list of recipes added by the user. This page has an instance of the add recipe button and an instance of the delete recipe button so the user can perform those actions.

SearchResultPage Class

* This page displays the list of recipes in the order determined by the search that was performed. They are arranged by date, preparation time, cuisine type or difficulty. When a recipe is clicked, an instance of the RecipePage class is created for that recipe.

Form Class

* Form classes are used when a user is required to enter input. There are a variety of input fields followed by a button to submit the information. Each form class is a child derived from the abstract class, Form.

Form

AddNewRecipeForm

LoginForm

AddNewRecipeForm Class

* This form allows users to enter information pertaining to the recipe they are adding to the database. This form has an instance of the add recipe button which stores the information in the recipe database.

LoginForm Class

* This form allows users to enter their username and password so they can log in to the system. This form has an instance of the login button, which accesses the user database and validates the user’s login information before logging them into the system.

**Control Objects**

* Main Control
  + The main controller is used whenever the user wants to interact with the app, depending on what the user taps or inputs on the screen it is all sent to the main controller where it is processed and necessary actions are performed such as interacting with the recipe database or changing pages.
* Login Control
  + The login controller is used when the user wants to login to his account. When the user inputs his username and password it would access the database to verify if he has a legitimate account that isn’t banned or if the account is an active account. The control is then passed to the main controller object

**Class Diagram**

|  |
| --- |
| User class |
| Username : String  Password : String  Recipe : objects  profileInfo :Array  recipeLiked :Linked List  UserStatus: Boolean |
| Public User(all above attributes ); |

**Method Descriptions**

Public user(all attributes)

* Access user database
* Add new entry to user database using attributes passed

|  |
| --- |
| Recipe class |
| recipeName: string  prepCookTime : integer  IngredientList : list  cusinieCategory: String  step-by-step :String  datein : String  rating : Integer  photo :Picturefile  servings : Integer  extraNotes : String |
| Public Recipe(all above attributes ); |

**Method Descriptions**

Public recipe(all attributes)

* Access recipe Database
* Add new entry to recipe database using attributes passed

|  |
| --- |
| Login class |
|  |
| Public DisplayProfile(array profileInfo);  Private VerifyUser(Username,password); |

**Method Descriptions**

Public DisplayProfile(array profileInfo)

* Create instance of ProfilePage object using profile info to fill out page

Public VerifyUser(string username, string password)

* Access user database
* Search database using username and password
* If username and password match an existing account
* Log user in
* Else
* Login failed

|  |
| --- |
| Main Controller class |
|  |
| Public Main();  Public AddRecipe(recipeName,prepCookTime,IngredientList,cusinineCategory,step-by-step,datein,rating,photo,servings,extraNotes);  Public DeleteRecipe();  Public SortbyCusinine(cusinieCategory);  Public SortbyDifficulty(rating);  Public SortbyDate(datin);  Public SortbyPreptime(prepCookTime);  Public UserStatus(userstatus);  Public CreateLoginControl(username,password);  Public MakeChangeToDatabase(); |

**Cohesion**

* All of our methods (modules) have either Information Cohesion or Functional Cohesion because they either perform a single action/change on/to a single data structure or perform a number of actions/changes on/to a single data structure. Because of this, our methods are easy to reuse and debug.

**Coupling**

* All of our methods (modules) have either Stamp Coupling or Data Coupling because they either operate on most or all of the elements of the passed data structure.

**Gane and Sarsen’s Analysis (Data Flow Diagram [DFD] based on Gane and Sarsen’s rules)**

