System Design

Team: Lions

Table of Contents

CRC Cards - Sprint 1 + Sprint 2 + Sprint 3 + Sprint 4

Class name:	**CreateRecipeActivity**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Acts as the View for the activity that handles creating a recipe - Detects user interaction with screen and passes data to the Presenter called CreateRecipePresent
Collaborators:	- CreateRecipeContract - CreateRecipePresent - CreateRecipeModel

Class name:	**CreateRecipeContract *(Interface)***
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Coordinates the CreateRecipeActivity class, CreateRecipeModel and CreateRecipePresent class
Collaborators:	

Class name:	**CreateRecipePresent**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Acts as the presenter for the activity that handles creating a recipe - Takes in data from the associated view class and deals with the data appropriately.
Collaborators:	- CreateRecipeActivity - CreateRecipeContract - CreateRecipeModel - Recipe - Ingredient

Class name:	**CreateRecipeModel**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Inserts a Recipe into the database - Checks if a given String already exists as a recipe name in the database
Collaborators:	- CreateRecipePresent - Recipe

Class name:	**CookBookContract *(Interface)***
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Coordinates the CookBookActivity class, CookBookModel and CookBookPresent class
Collaborators:	

Class name:	**CookBookObserver *(Interface)***
Parent class (if any):	
Classname Subclasses (if any):	- CookBookPresent
Responsibilities:	- Defines Observer/Observable behaviour between Model and Presenter
Collaborators:	

Class name:	**CookBookActivityObserver *(Interface)***
Parent class (if any):	
Classname Subclasses (if any):	- CookBookActivity
Responsibilities:	- Defines Observer/Observable behaviour between Activity and Presenter
Collaborators:	

Class name:	**CookBookActivity**
Parent class (if any):	
Classname Subclasses (if any):	

Responsibilities:	- Acts as the view for the activity that handles viewing the user's cookbook - Detects if a user clicks on a recipe. If so, the recipe title gets passed on to the associated presenter for this activity
Collaborators:	- CookBookContract - CookBookPresent - CookBookModel

Class name:	**CookBookPresent**
Parent class (if any):	- CookBookObserver
Classname Subclasses (if any):	
Responsibilities:	- Acts as the presenter for the activity that handles viewing the user's cookbook. - Takes in recipe title clicked from the View, fetches the associated data from the database, and ensures the user is brought to a screen that displays the recipe that was clicked.
Collaborators:	- CookBookActivity - CookBookContract - CookBookModel

Class name:	**CookBookModel**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Obtains a Livedata List of Recipe titles from the Repository, and sends it to the Presenter so that they can be displayed correctly
Collaborators:	- CookBookPresent - CookBookObserver

Class name:	**ViewRecipeActivity**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Displays the recipe the user has clicked on the CookBook screen.

	- This should populate the recipe title, ingredients, steps, and serving size.
Collaborators:	- ViewRecipeContract - ViewRecipeModel - ViewRecipePresent - Recipe

Class name:	**ViewRecipeContract <*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Coordinates the ViewRecipeActivity, ViewRecipeModel, ViewRecipePresent classes.
Collaborators:	

Class name:	**ViewRecipePresenter**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Responds to user interaction with the ViewRecipe screen o Includes scaling feature clicked by user
Collaborators:	- ViewRecipeContract - ViewRecipeActivity - ViewRecipeModel - Recipe - Ingredient

Class name:	**ViewRecipeModel**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Retrieve Recipe clicked on by user and passes it to the ViewRecipePresenter
Collaborators:	- ViewRecipeContract - ViewRecipeActivity - ViewRecipePresent - Recipe - Ingredient

Class name:	**Ingredient**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Stores information of a single ingredient in a recipe. - Takes in the ingredient name, quantity, and quantity_type, and able to modify and update these values
Collaborators:	

Class name:	**Recipe**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Stores information of a single recipe. - Takes in the recipe name, username who created the recipe, serving_size, list of ingredients, list of tags and list of steps, and able to modify and update these values
Collaborators:	

Class name:	**CreateAccountActivity**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Displays the registration page - Takes in user inputs for email and password, phone number, first name and last name.
Collaborators:	- CreateAccountPresent - CreateAccountModel

Class name:	**CreateAccountModel**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Creates new Firebase User if valid email and password is provided - Stores username, and user's first name, and user's last name into database under "users" - Return true if successful
Collaborators:	- Create AccountPresent

Class name:	**CreateAccountContract <*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Coordinates the CreateAccountActivity, CreateAccountModel, CreateAccountPresent classes.
Collaborators:	

Class name:	**CreateAccountPresent**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Acts as the presenter for the activity that handles user's action on account creation.
Collaborators:	- CreateAccountActivity - CreateAccountModel - CreateAccountContract

Class name:	**LoginActivity**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Displays the login page - Takes in user inputs for email and password
Collaborators:	- LoginModel - LoginPresent - LoginContract - SessionManager - PreferencesProvider

Class name:	**LoginModel**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Logs in user if provided username and password match an existing account's login credentials
Collaborators:	- LoginPresent

	- LoginObserver

Class name:	**LoginPresent**
Parent class (if any):	- LoginObserver
Classname Subclasses (if any):	
Responsibilities:	- Acts as the presenter for the activity that handles user's action on Login. - Directs the user to the registration page
Collaborators:	- LoginModel - LoginActivity - LoginContract - SessionManager

Class name:	**LoginContract <*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Coordinates the LoginActivity, LoginModel, LoginPresent classes.
Collaborators:	

Class name:	**LoginObserver <*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	- LoginPresent
Responsibilities:	- Coordinates the LoginPresent and LoginActivity to implement the observe/observable pattern
Collaborators:	

Class name:	**PrivateUserProfileContract <*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Coordinates the PrivateUserProfileModel, PrivateUserProfilePresent, and PrivateUserProfileActivity classes
Collaborators:	

Class name:	**PrivateProfileActivityObserver<*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	- PrivateProfileActivity

Responsibilities:	- Coordinates the PrivateProfileActivity and PrivateProfilePresent classes to implement observe/observable pattern
Collaborators:	

Class name:	**PrivateProfileObserver <*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	- PrivateProfilePresent
Responsibilities:	- Coordinates the PrivateProfileModel and PrivateProfilePresent classes to implement observe/observable pattern
Collaborators:	

Class name:	**PrivateUserProfileModel**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Knows currently signed in user's email, username, fullname, phone number, and list of their recipes' names - Notifies PrivateUserProfilePresent of changes to the currently signed in user's profile information
Collaborators:	- PrivateProfileObserver

Class name:	**PrivateUserProfileActivity**
Parent class (if any):	- PrivateProfileActivityObserver
Classname Subclasses (if any):	
Responsibilities:	- Displays and updates all current logged-in user's information and list of recipes that the user has created
Collaborators:	- PrivateUserProfileModel - PrivateUserProfilePresent

Class name:	**PrivateUserProfilePresent**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Coordinates between PrivateUserProfileModel and PrivateUserProfileActivity, pass the data obtained from model to the activity

Collaborators:	- PrivateUserProfileModel - PrivateUserProfileActivity - PrivateProfileActivityObserver

Class name:	**PreferencesProvider**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- A "local storage" class implemented using SharedPreferences that gets initialized once when the app starts running. - Stores current user's login state and email - Provides the SharedPreferences object to SessionManager class
Collaborators:	

Class name:	**SessionManager**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Manages/maintains user's login state - Clears the current user's info if user logs out
Collaborators:	- PreferencesProvider

Class name:	**PublicUserProfileContract <*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Coordinates the PublicUserProfileModel, PublicUserProfilePresent, and PublicUserProfileActivity classes
Collaborators:	

Class name:	**PublicProfileObserver <*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	- PublicProfilePresent
Responsibilities:	- Coordinates the PublicProfileModel and PublicProfilePresent classes to implement observe/observable pattern
Collaborators:	

Class name:	**PublicProfileActivityObserver <*Interface*>**
Parent class (if any):	
Classname Subclasses (if any):	- PublicProfileActivity
Responsibilities:	- Coordinates the PublicProfileActivity and PublicProfilePresent classes to implement observe/observable pattern
Collaborators:	

Class name:	**PrivateUserProfilePresent**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Coordinates between PublicUserProfileModel and PublicUserProfileActivity, pass the data obtained from model to the activity
Collaborators:	- PublicUserProfileModel - PublicUserProfileActivity - PublicProfileActivityObserver

Class name:	**PublicUserProfileModel**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Given a user's username, knows their fullname and list of their recipes' names - Notifies PublicUserPresent of any changes in user's profile information
Collaborators:	- PublicProfileObserver

Class name:	**PublicUserProfileActivity**
Parent class (if any):	- PublicProfileActivityObserver
Classname Subclasses (if any):	
Responsibilities:	- Displays and updates all current logged-in user's information and list of recipes that the user has created
Collaborators:	- PublicUserProfileModel - PublicUserProfilePresent

System Interaction with Environment

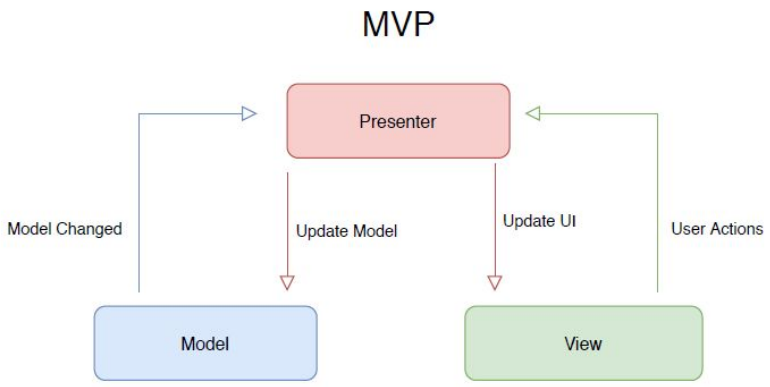
OS	Android Marshmallow 6.0.1 and greater
Programming Language	Java
Emulators	Emulator OS: Marshmallow 6.0.1 Emulator Device: Pixel 2
Database	Firebase
Dependencies	Data binding, MaterialUI style themes, Firebase
Assumptions	● The device the app runs on has an OS API level of 16 or greater. ● The device must have enough storage to save the data.

Architecture of the System

Architecture high-level explanation:

The application follows the Model-View-Presenter(MVP) design pattern, which can be seen to be Android's adaptation of the Model-View-Controller design pattern. A diagram of the architecture is included below. In this architecture, the Presenter is responsible for interpreting any interaction the user has with the view and working with the Model accordingly. Note that the model does not communicate with the View. Our team decided to use this architecture as it is very commonly used among Android developers for its ease of testing capabilities.

Image of the MVP design pattern:



Links that describes the MVP:

- https://android.jlelse.eu/architectural-guidelines-to-follow-for-mvp-pattern-in-android-2374848a0157
- https://www.raywenderlich.com/7026-getting-started-with-mvp-model-view-presenter-on-android
- http://www.gwtproject.org/articles/mvp-architecture.html
- https://www.martinfowler.com/eaaDev/uiArchs.html#Model-view-presentermvp

MVP Components:

- CreateRecipePresent.java, CookBookPresent.java, ViewRecipePresenter.java, LoginPresent.java, CreateAccountPresent.java – the presenter
- ViewRecipeActivity.java, CookbookActivity.java, CreateRecipeActivity.java – the view
- CreateRecipeContract.java, CookBookContract, ViewRecipeContract.java, CreateAccountContract.java – defines contract between view and presenter
- CookBookModel.java, CreateRecipeModel.java, CookBookModel.java, CreateAccountModel.java, ViewRecipeModel.java – the model

Room Components (Part of Model):

- Recipe.java – this is the entity which represents a relational table within the database
- Ingredient.java - This represents an Ingredient Object that is stored inside a Recipe object

Anticipated Errors and Exceptions:

The user is required to fill out all fields when creating a recipe. If they fail to do this a toast notifies them of this.

In the upcoming sprints, we also plan to handle cases where the user inputs the incorrect data type. For example, if the user inputs a String into the textfield intended for a quantity of an ingredient, a toast will inform them that their input is invalid and they must enter a number into that field.