System Design

Team: Lions

Table of Contents

CRC Cards - Sprint 1 + Sprint 2

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 title in the database
Collaborators:	- RecipeRepository - 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:	**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):	
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:	- RecipeRepository - CookBookPresent

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:	**RecipeDao *(Interface)***
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- contains function declarations for inserting a new recipe and getting all recipes stored in the database
Collaborators:	

Class name:	**RecipeDatabase*(abstract)***
Parent class (if any):	- RoomDatabase
Classname Subclasses (if any):	
Responsibilities:	- Instantiate a database object
Collaborators:	- RecipeDao - RecipeListConverter

Class name:	**RecipeListConverter**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- a utility class that is responsible for converting types between strings and string lists for the ingredients
Collaborators:	

Class name:	**RecipeRepository**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- able to get all recipes from the database - able to insert a new recipe into the database
Collaborators:	- RecipeDao - RecipeDatabase

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

Class name:	**CreateAccountModel**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Connects to the database - Return true if user's email and password are successfully put into the database
Collaborators:	

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

Class name:	**LoginModel**
Parent class (if any):	
Classname Subclasses (if any):	
Responsibilities:	- Checks users' login credentials
Collaborators:	

Class name:	**LoginPresent**
Parent class (if any):	
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

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

System Interaction with Environment

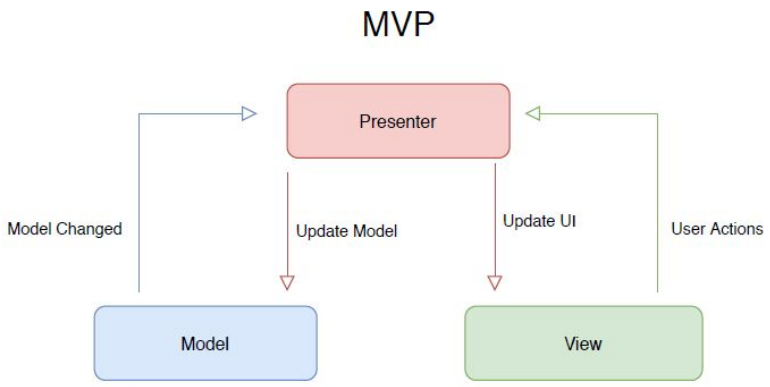
OS	Android Marshmallow 6.0.1 and greater
Programming Language	Java
Emulators	Emulator OS: Marshmallow 6.0.1 Emulator Device: Pixel 2
Database	Firebase/ Room DB (local)
Dependencies	roomDB, data binding, MaterialUI style themes, Firebase
Assumptions	● The device the app runs on has an OS API level of 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
- RecipeDatabase.java – a Room Database
- RecipeDao.java - Contains the methods to access the database
- RecipeRepository.java - is a class for the repository in the viewModel + repository framework for roomDB
- RecipeListConverter.java

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.