Design Document for Savour

Group VB_4

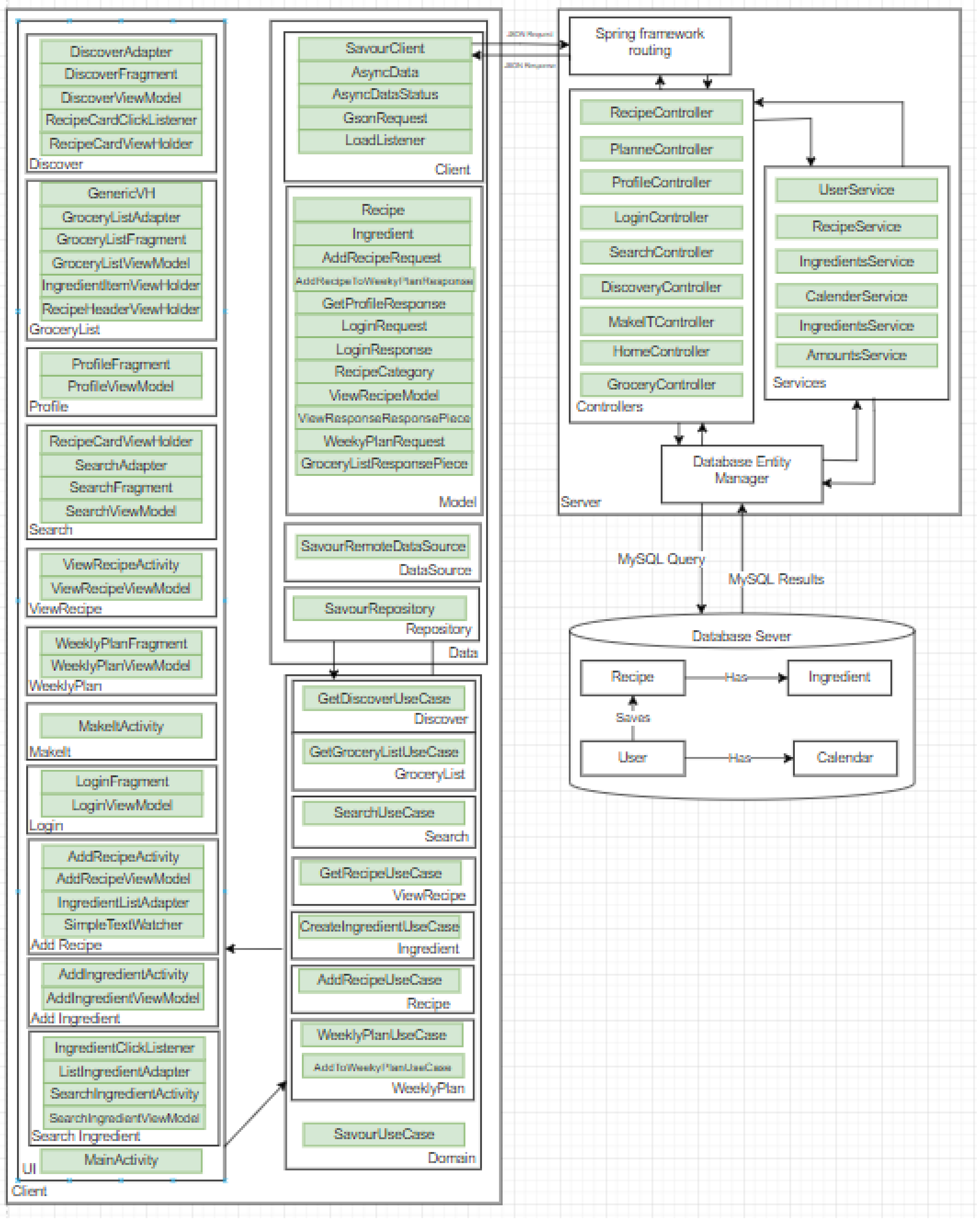
Amy Wieland: 25% contribution

Alec Meyer: 25% contribution

Brandon Richards: 25% contribution

Michael Mazur: 25% contribution

Savour VB_4 Amy Wieland, Alec Meyer, Brandon Richards, Michael Mazur

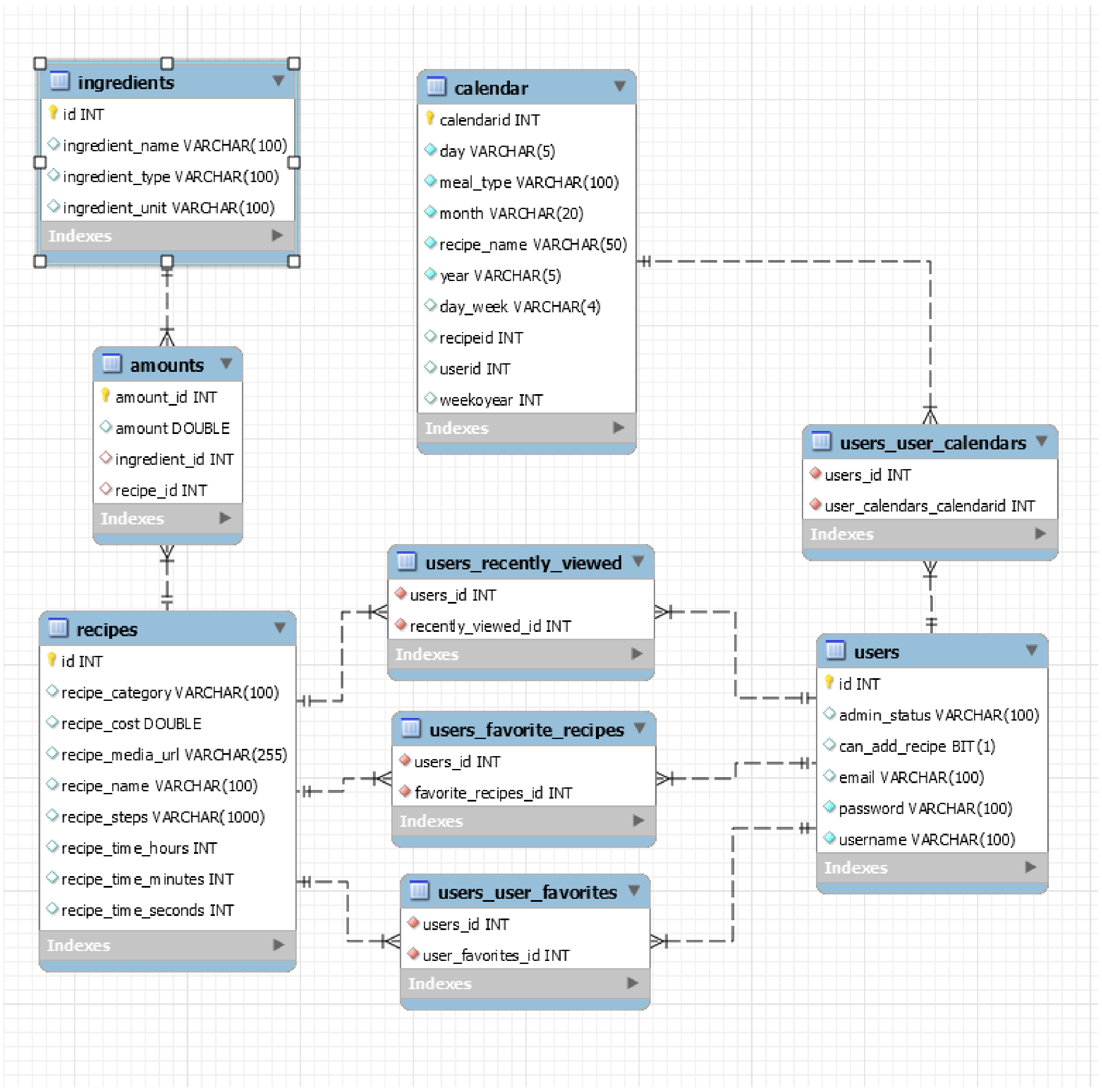


Frontend

The frontend application uses the MVVM architecture. The source code is split into UI, Domain, and Data layers, each in their own package. The UI layer holds all code related to the UI, such as Activities, Fragments, and ViewModels. Activities and Fragments observe live data in their respective ViewModel, and when necessary, the ViewModel will run a Use Case in the domain layer. Use cases handle business logic between the UI and Data layer, such as asking the data layer for data or manipulating data before sending or receiving. Our use cases usually ask the main data layer component, SavourRepository, for the data. The repository checks to see if the requested data is stored in the local data source, and if it is not, uses the remote data source to fetch the data. The remote data source asks SavourClient to request a certain endpoint and expect a JSON response that is then decoded into a POJO. The SavourClient is responsible for the Volley request queue that fetches the data from the remote server.

Backend

The Backend of the Savour application uses the help of Spring-boot's framework in order to handle network interfaces and incoming API requests. These endpoints use RESTful API to communicate the status of request mappings to and from the frontend using ResponseEntities. These Response Entities will send a string, detailing a description of the current state in the backend as well as a HttpStatus on the matter. This will allow effective communication with the front end. We set up the backend so that each panel in the frontend has an accompanying RESTController with its respective request mappings. This makes it easier to debug, reduces GOD classes, and makes it easier to read other people's code. In order to accompany these controllers are repos, services, and MySQL tables. The repos will store data on the MySQL tables, making it easy for the frontend to retrieve/save important data. In order to add further functionality and support to the repository interfaces, we have implemented services in order to aid in complex database relationships between separate tables, and to add further functionality while writing separating the logic in a different layer than the @RestController class files.


ingredients
id INT
ingredient_name VARCHAR(100)
ingredient_type VARCHAR(100)
ingredient_unit VARCHAR(100)
Indexes
calendar
calendarid INT
day VARCHAR(5)
meal_type VARCHAR(100)
month VARCHAR(20)
recipe_name VARCHAR(50)
year VARCHAR(5)
day_week VARCHAR(4)
recipeid INT
userid INT
weekoyear INT
Indexes
amounts
amount_id INT
amount DOUBLE
ingredient_id INT
recipe_id INT
Indexes
users_user_calendars
users_id INT
user_calendars_calendarid INT
Indexes
users_recently_viewed
users_id INT
recently_viewed_id INT
Indexes
recipes
id INT
recipe_category VARCHAR(100)
recipe_cost DOUBLE
recipe_media_url VARCHAR(255)
recipe_name VARCHAR(100)
recipe_steps VARCHAR(1000)
recipe_time_hours INT
recipe_time_minutes INT
recipe_time_seconds INT
Indexes
users
id INT
admin_status VARCHAR(100)
can_add_recipe BIT(1)
email VARCHAR(100)
password VARCHAR(100)
username VARCHAR(100)
Indexes
users_favorite_recipes
users_id INT
favorite_recipes_id INT
Indexes
users_user_favorites
users_id INT
user_favorites_id INT
Indexes