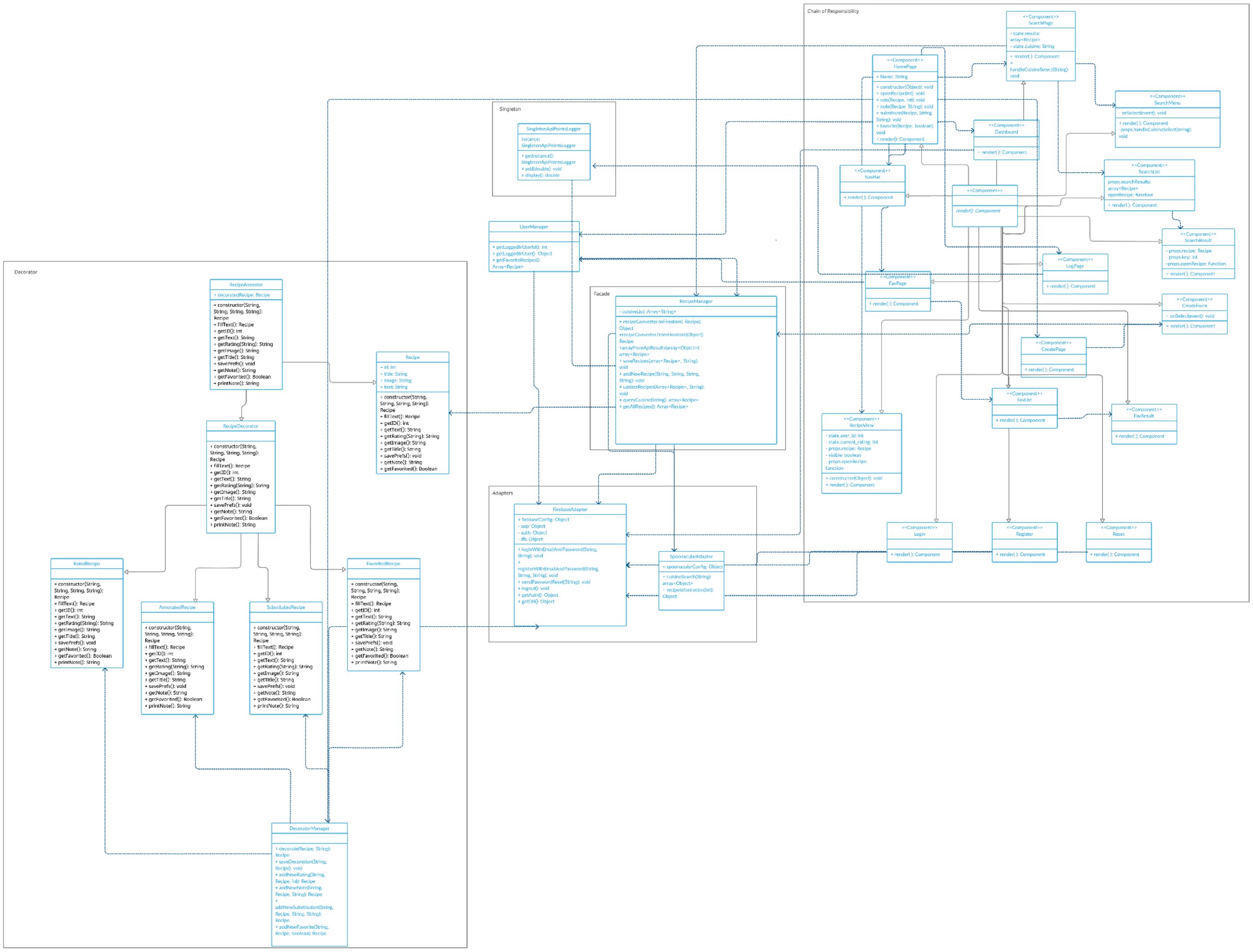
Name of project: World Cuisine App
Team members: Myles Cork, Kai Drumm, Robert Meikle

Final state of system:

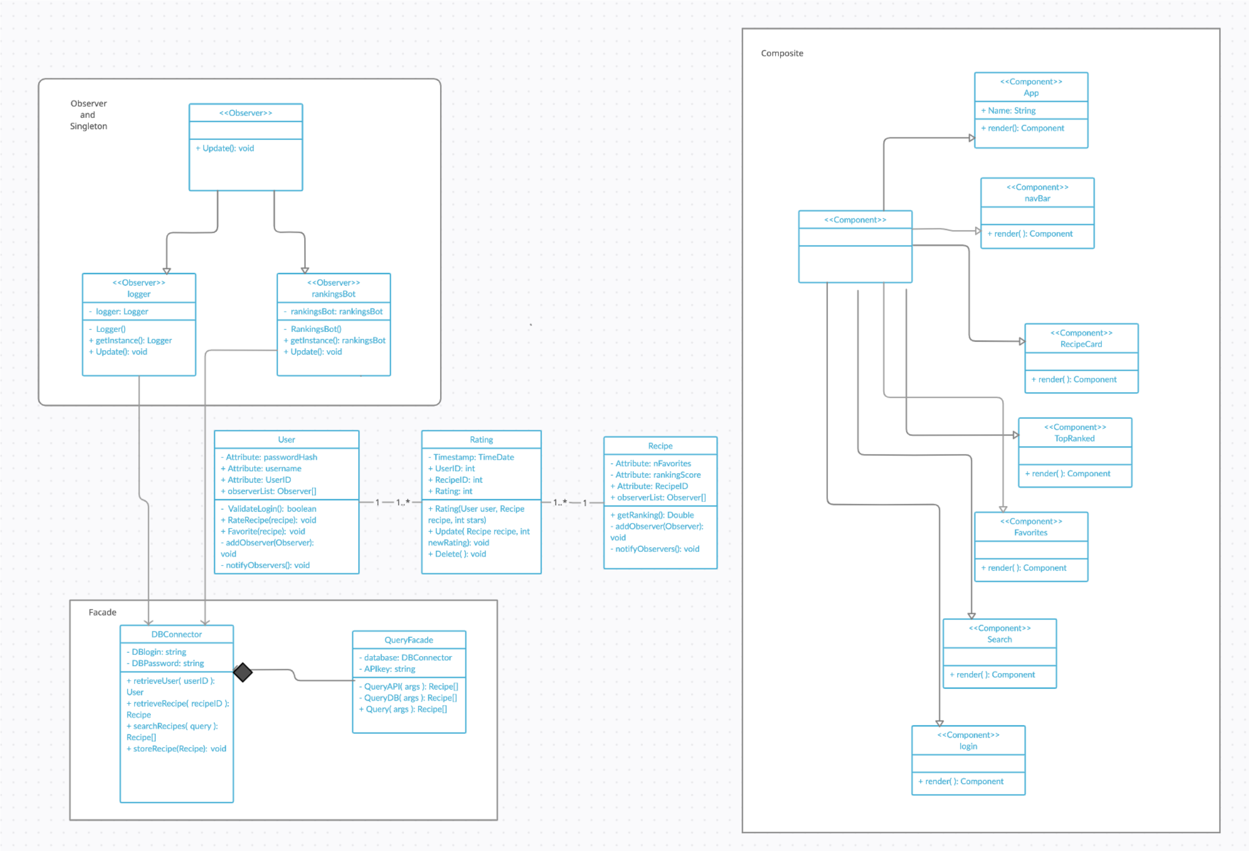
In our web app, users can search for recipes, rate and add notes to recipes, build a favorites list, make ingredient substitutions, and create new recipes. Our web app is locally hosted; it pulls recipe data from the Spoonacular API, and stores recipes and user data in Firebase (NoSQL). There is basic functionality for new users to create an account and for existing users to login into their account. Users can search for recipes by cuisine, view recipes, rate recipes, add notes, and favorite recipes to view later. Users can also specify substitutions for recipes; this is a new feature not planned in projects 5 and 6. We also added the metrics page to keep track of how many Spoonacular API points have been spent, and the ability to create a custom recipe. From projects 5 and 6 we decided to not implement the "top rated recipes" functionality.

Final class diagram:

The main changes between the UML from project 6 are the addition of the RecipeDecorator group and the SingletonApiPointsLogger. The Favorites list and Create page were planned in project 6, but not implemented at that time.


Decorator
RecipeAncestor
+ decoratedRecipe: Recipe
+ constructor(String, String, String, String): Recipe
+ fillText(): Recipe
+ getID(): int
+ getText(): String
+ getRating(String): String
+ getImage(): String
+ getTitle(): String
+ savePrefs(): void
+ getNote(): String
+ getFavorited(): Boolean
+ printNote(): String
Recipe
+ id: int
+ title: String
+ image: String
+ text: String
+ constructor(String, String, String, String): Recipe
+ fillText(): Recipe
+ getID(): int
+ getText(): String
+ getRating(String): String
+ getImage(): String
+ getTitle(): String
+ savePrefs(): void
+ getNote(): String
+ getFavorited(): Boolean
RecipeDecorator
+ constructor(String, String, String, String): Recipe
+ fillText(): Recipe
+ getID(): int
+ getText(): String
+ getRating(String): String
+ getImage(): String
+ getTitle(): String
+ savePrefs(): void
+ getNote(): String
+ getFavorited(): Boolean
+ printNote(): String
RatedRecipe
+ constructor(String, String, String, String): Recipe
+ fillText(): Recipe
+ getID(): int
+ getText(): String
+ getRating(String): String
+ getImage(): String
+ getTitle(): String
+ savePrefs(): void
+ getNote(): String
+ getFavorited(): Boolean
+ printNote(): String
AnnotatedRecipe
+ constructor(String, String, String, String): Recipe
+ fillText(): Recipe
+ getID(): int
+ getText(): String
+ getRating(String): String
+ getImage(): String
+ getTitle(): String
+ savePrefs(): void
+ getNote(): String
+ getFavorited(): Boolean
+ printNote(): String
SubstitutedRecipe
+ constructor(String, String, String, String): Recipe
+ fillText(): Recipe
+ getID(): int
+ getText(): String
+ getRating(String): String
+ getImage(): String
+ getTitle(): String
+ savePrefs(): void
+ getNote(): String
+ getFavorited(): Boolean
+ printNote(): String
FavoritedRecipe
+ constructor(String, String, String, String): Recipe
+ fillText(): Recipe
+ getID(): int
+ getText(): String
+ getRating(String): String
+ getImage(): String
+ getTitle(): String
+ savePrefs(): void
+ getNote(): String
+ getFavorited(): Boolean
+ printNote(): String
DecoratorManager
+ decorate(Recipe, String): Recipe
+ saveDecoration(String, Recipe): void
+ addNewRating(String, Recipe, int): Recipe
+ addNewNote(String, Recipe, String): Recipe
+ addNewSubstitution(String, Recipe, String, String): Recipe
+ addNewFavorite(String, Recipe, boolean): Recipe
Singleton
SingletonApiPointsLogger
instance: SingletonApiPointsLogger
+ getInstance(): SingletonApiPointsLogger
+ add(double): void
+ display(): double
UserManager
+ getLoggedInUserId(): int
+ getLoggedInUser(): Object
+ getFavoriteRecipes(): Array<Recipe>
Facade
RecipeManager
- cuisineList: Array<String>
+ recipeConverter.toFirestore(Recipe): Object
+recipeConverter.fromFirestore(Object): Recipe
+arrayFromApiResults(array<Object>): array<Recipe>
+ saveRecipes(array<Recipe>, String): void
+ addNewRecipe(String, String, String, String): void
+ updateRecipes(Array<Recipe>, String): void
+ queryCuisine(String): array<Recipe>
+ getAllRecipes(): Array<Recipe>
Adapters
FirebaseAdapter
+ firebaseConfig: Object
- app: Object
- auth: Object
- db: Object
+ logInWithEmailAndPassword(String, String): void
+ registerWithEmailAndPassword(String, String, String): void
+ sendPasswordReset(String): void
+ logout(): void
+ getAuth(): Object
+ getDB(): Object
SpoonacularAdapter
+ spoonacularConfig: Object
+ cuisineSearch(String): array<Object>
+ recipeInformation(int): Object
Chain of Responsibility
<<Component>> SearchPage
- state.results: array<Recipe>
- state.cuisine: String
+ render( ): Component
+ handleCuisineSelect(String): void
<<Component>> HomePage
+ Name: String
+ constructor(Object): void
+ openRecipe(int): void
+ rate(Recipe, int): void
+ note(Recipe, String): void
+ substitute(Recipe, String, String): void
+ favorite(Recipe, boolean): void
+ render(): Component
<<Component>> SearchMenu
- onSelect(event): void
+ render( ): Component
-props.handleCuisineSelect(string): void
<<Component>> Dashboard
+ render( ): Component
<<Component>> NavBar
+ render( ): Component
<<Component>>
render(): Component
<<Component>> SearchList
props.searchResults: array<Recipe>
openRecipe: function
+ render( ): Component
<<Component>> SearchResult
- props.recipe: Recipe
- props.key: int
-props.openRecipe: function
+ render( ): Component
<<Component>> LogPage
+ render( ): Component
<<Component>> FavPage
+ render( ): Component
<<Component>> CreateForm
+ onSelect(event): void
+ render( ): Component
<<Component>> CreatePage
+ render( ): Component
<<Component>> FavList
+ render( ): Component
<<Component>> FavResult
+ render( ): Component
<<Component>> RecipeView
- state.user_id: int
- state.current_rating: int
- props.recipe: Recipe
- visible: boolean
- props.openRecipe: function
+ constructor(Object): void
+ render( ): Component
<<Component>> Login
+ render( ): Component
<<Component>> Register
+ render( ): Component
<<Component>> Reset
+ render( ): Component

Project 5 class diagram:



Third party code vs original code statement

We used the React and Node frameworks to build the web app. The login page was mostly borrowed from an example online.

https://reactjs.org/docs/getting-started.html
https://firebase.google.com/docs
https://developer.mozilla.org/en-US/docs/Web
https://nodejs.org/en/docs/
https://blog.logrocket.com/user-authentication-firebase-react-apps/

Statement on OOAD process for the project

We started by following React tutorials to build the static views, and then worked on adding interactive functionality. We added the Model and Adapters folders to hold our main object-oriented code. We started by focusing on creating adapter classes to connect to both Firebase and the Spoonacular API in order to cache recipe search results and avoid overusing our API key. We also created a Facade class, RecipeManager, which hides the logic of pulling from Firebase if possible and querying Spoonacular if needed.

We had planned to use the Singleton and Observer patterns together to implement a logger that would dynamically update the average ratings of each recipe as multiple users interact with the app. However, those of us new to Javascript were surprised to realize that the entire code of the app is run within the browser, so only one user will interact with it at a time, anyway. We looked at upgrading FirebaseAdapter to a singleton, because it holds a connection to the database which should only be initialized once per app instance. We were surprised to find that in Javascript, the Singleton pattern is less typical; we found a StackOverFlow post suggesting that we model it after ‘Module’, a Javascript-specific pattern, instead. FirebaseAdapter was working well so we left it in the Module pattern form. We later added a true Singleton class, SingletonApiPointsLogger, as a proof of concept to explore how this pattern would be expressed in Javascript.

As we moved on from querying the API, caching the results and displaying the recipes, we realized that the Decorator pattern would work well for applying user customizations to the recipes. We decided to use it for adding ratings, notes, ingredient substitutions, and marking recipes as favorites. These are stored in Firebase in a simple form, but when they are pulled from Firebase they are used to wrap the Recipe object in a RecipeDecorator, which overrides one of the getter- or display- related functions on Recipe.

Key points:

- Sometimes it’s a bit forced to use OO patterns in a Javascript project - the language has its own patterns, and due to its weak typing characteristics, creating abstract parent classes to provide polymorphism is not really necessary.
- Designing the UML diagram ahead of time was still helpful to point us towards what to implement. However, drawing the UML diagram is slow using online drawing tools. It might be nicer to just sketch it on a whiteboard. The UML also changed as we became more familiar with the React framework.
- Recognizing some of the patterns from class as inherent patterns to how the web framework works (for example, how React Components use the Chain of Responsibility) helped us more rapidly adapt to coding in a language we had little experience with.

Recorded Demonstration

https://drive.google.com/file/d/1_XZRo4o4FsZ4gSUXddkWLMHVGeLpzCJw/view?usp=sharing