MIAMI

CSC 431

WeCook

System Architecture Specification (SAS)

Team 04

Daniel Moran	Scrum Master
Avani Choudhary	Lead Designer
Alessandro D'Urso	Planning and Architecture Lead

Version History

Version	Date	Author(s)	Change Comments
1.0	03/31/21	AC, AD, DM	First Draft
1.1	04/11/21	AC, AD, DM	Final Draft
1.2	05/05/21	AC, AD, DM	Final Submission

Table of Contents

Table of Figures

1. System Analysis

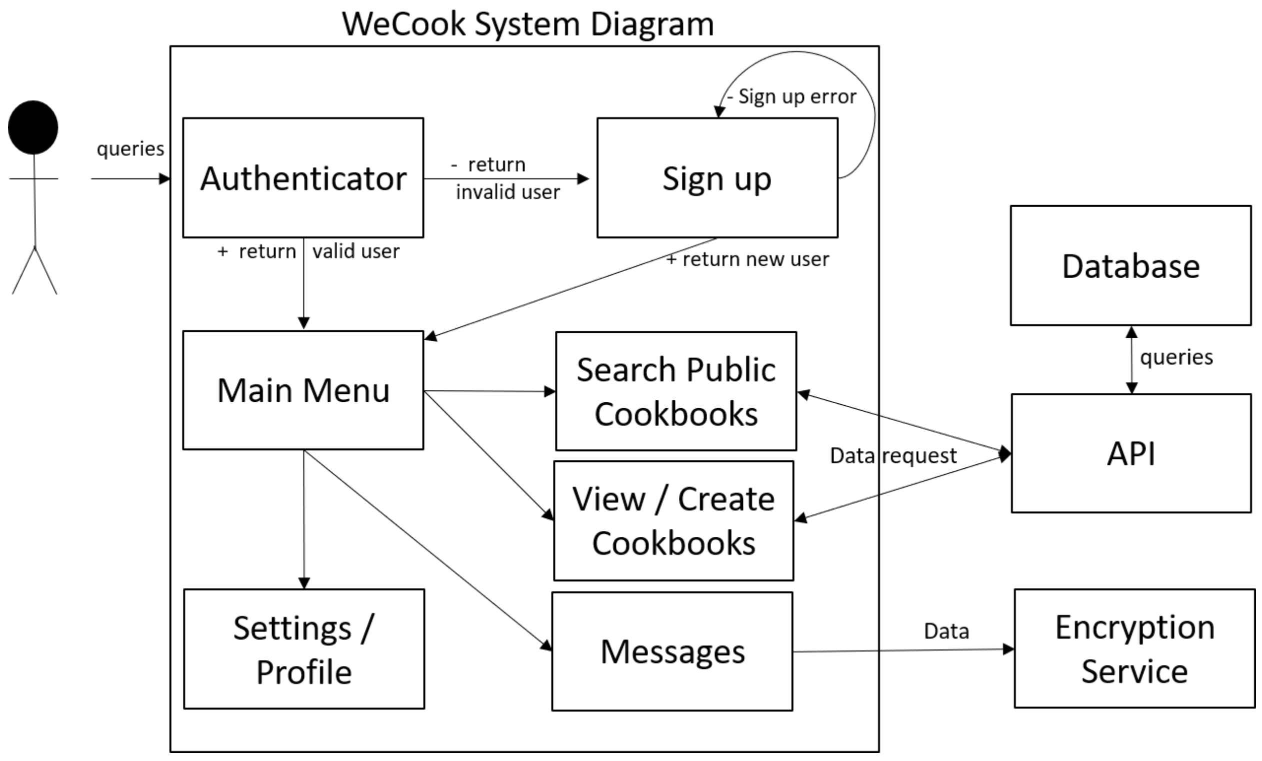
1.1 System Overview

The system will be composed of the following: a database, settings, messaging, and cookbooks (sub-unit = recipes). The database will consist of tables for Public/Private Cookbooks, Recipes, and user account information. Cookbooks and settings will be connected to the database. The database will live in AWS, using Amazon Aurora which is compatible with MySQL, per our System requirements. The Cookbooks will store recipes, as substructures. Messages will be encrypted end-to-end.

We will be using a client-server architecture. The client-server architecture allows the user to request and push changes to the database (server) from their phones (client).

Essentially, the user will be able to view Cookbooks and Recipes (or their settings), which are stored in the database. They will retrieve it from the server, and it will be formatted and displayed client-side. When the user makes changes to existing Cookbooks and Recipes (or settings), or creates new ones, they will be pushed to the database for later retrieval.

1.2 System Diagram



1.3 Actor Identification

There are two different types of human actors involved in the usage of WeCook: users who already have an account, and user's who do not yet have an account. User's with an account are able to message other users, and create, view, edit, publish, and comment on cookbooks. User's without an account are only able to view the welcome screen (login/signup page) until they have created an account. Other actors include an API to query the database, and the database itself.

1.4 Design Rationale

1.4.1 Architectural Style

We identified the Client-Server Architecture style to be the best fit for the functionality of WeCook. Our original reasoning was that our application would be database-centric, but since registered users must be able to message and store data on their local device, the system must take the client into account.

1.4.2 Design Pattern(s)

We have decided to use the "Factory" Method design pattern. Essentially, this pattern consists of the designing of the interface(s) to create objects, but allows for the instantiation of classes on a per class basis. Essentially, we will have larger parent classes, such as "Cookbook", which will have inheriting sub-classes such as "Private Cookbook" or "Public Cookbook".

We will also use the "command" design pattern, to allow for the parametrization of user requests into objects that contain all information about the requests, particularly to the database for information about cookbooks. This will allow for the passing of requests as method arguments, delaying or queuing of user requests, and also to support operations that may need to be undone.

1.4.3 Framework

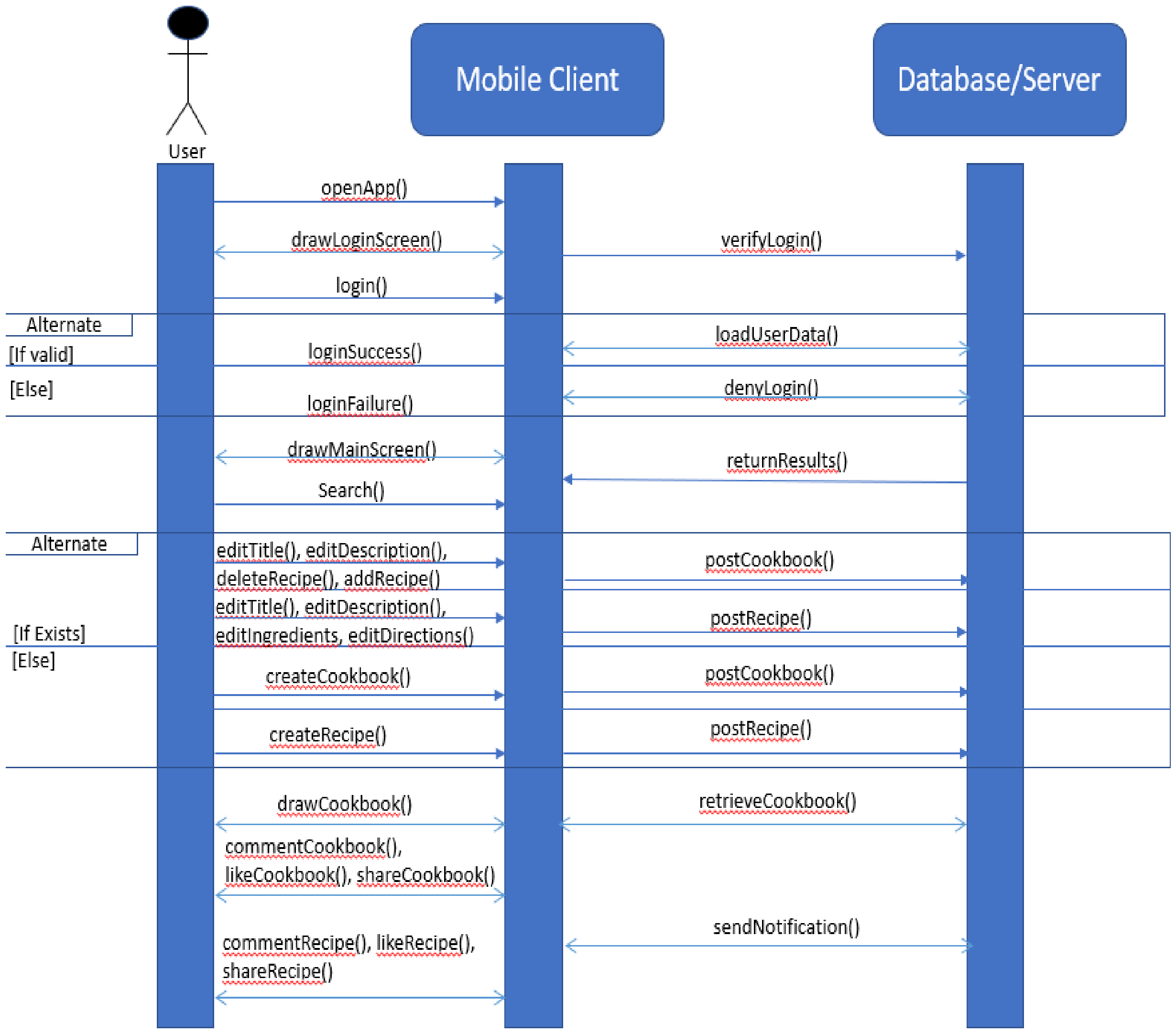
We will be using Apache Cordova and Node.js for the creation of our mobile applications, as well as HTML, CSS & JS. We chose this framework on account of the fact that it allows for the wrapping of Web applications into mobile applications. This streamlines our development process, as we can eventually port the application into a web-based application for desktop computers, as per our evolutionary requirements. It also eliminates the need to learn native iOS and Android development techniques/languages such as Swift or Kotlin.

Links:

- https://nodejs.org/
- https://cordova.apache.org/

2. Functional Design

Fig. 2.1: WeCook Sequence Diagram

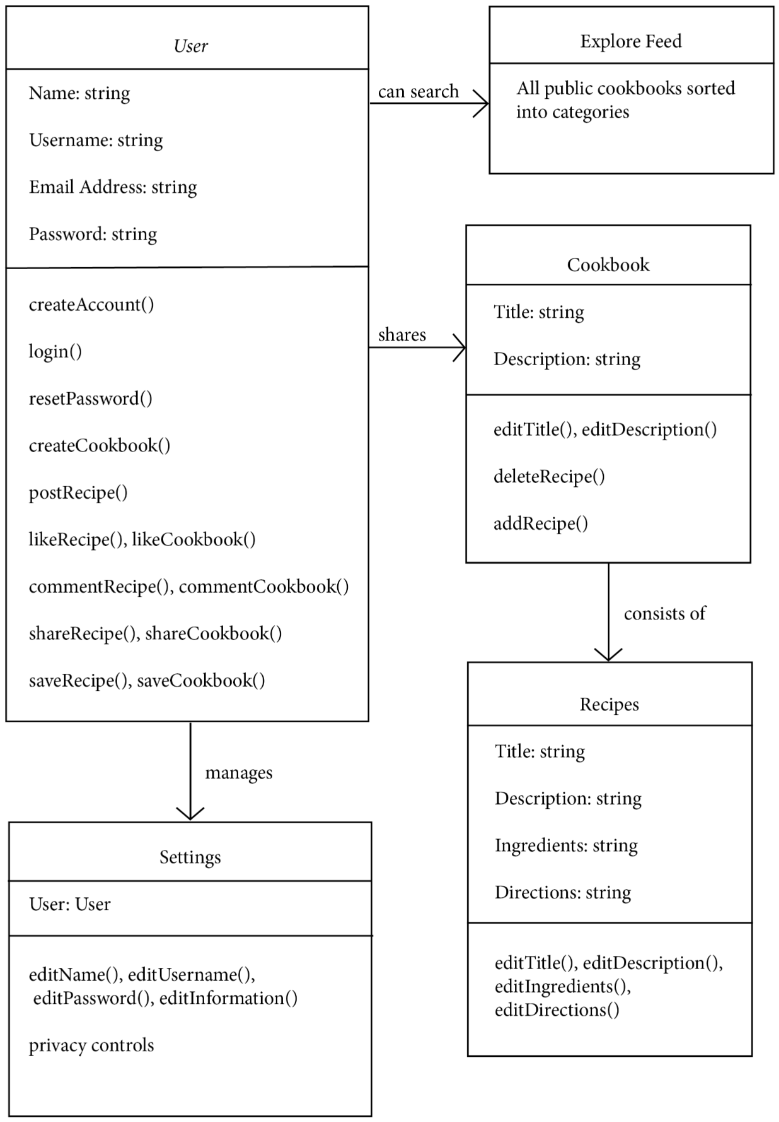


- After submitting the login page, the login is verified and user data is loaded if the login was a success.
    - If the login was not a success, login is denied and a failure message is displayed.
- Users can request screens from the web server: search view, account view, or local cookbook view.
- Once a view is selected, the corresponding screen is opened.
- On the search screen, users can send a request to the database for public user profiles and cookbooks, save cookbooks and recipes on their local device, and publish comments on other's public cookbooks and recipes.
- On the account screen, users can edit profile details, edit friends list, and change account information.
- On the local cookbook screen, users can create, view, and edit their own cookbooks, as well as change the publishing permissions for each.

- From any screen, users can request to send another user a private message, to which the message view will be opened. From this screen, both users may send messages to each other through peer-to-peer encryption, without interacting or storing data on the main server.
- At the end of use, the application is closed.

3. Structural Design

Fig. 3.1: WeCook Class Diagram


User
Name: string
Username: string
Email Address: string
Password: string
createAccount()
login()
resetPassword()
createCookbook()
postRecipe()
likeRecipe(), likeCookbook()
commentRecipe(), commentCookbook()
shareRecipe(), shareCookbook()
saveRecipe(), saveCookbook()
can search
Explore Feed
All public cookbooks sorted into categories
shares
Cookbook
Title: string
Description: string
editTitle(), editDescription()
deleteRecipe()
addRecipe()
consists of
Recipes
Title: string
Description: string
Ingredients: string
Directions: string
editTitle(), editDescription(), editIngredients(), editDirections()
manages
Settings
User: User
editName(), editUsername(), editPassword(), editInformation()
privacy controls