# **CSC 431** FundsBook System Architecture Specification (SAS)

**Team 16**

|  |  |
| --- | --- |
| Luis Diaz | Programmer |
| Temuulen Ganbold | Programmer |
| Julio Ojalvo | Programmer |

# Version History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author(s) | Change Comments |
| 1 | 3/31/2021 | Luis Diaz, Temuulen Ganbold, Julio Ojalvo | Wrote up system analysis, created functional and structural design diagrams |
| 2 | 4/12/2021 | Luis Diaz, Temuulen Ganbold, Julio Ojalvo | Added detail and corrected grammar in the system analysis, fixed functional design, and added detail to structural design |
|  |  |  |  |
|  |  |  |  |

# Table of Contents

[1.](#_3dy6vkm) System Analysis 5

[1.1](#_1t3h5sf) System Overview 5

[1.2](#_4d34og8) System Diagram 5

7[1.3](#_2s8eyo1) Actor Identification 6

[1.4](#_17dp8vu) Design Rationale 6

[1.4.1](#_3rdcrjn) Architectural Style 6

[1.4.2](#_26in1rg) Design Pattern(s) 6

[1.4.3](#_lnxbz9) Framework 7

[2.](#_35nkun2) Functional Design 7

[2.1 Sequence](#_1ksv4uv) Diagram 8

[3.](#_44sinio) Structural Design 9

# Table of Figures

[1 System Diagram](#_3dy6vkm) 5

2 [Sequence Diagram](#_3dy6vkm) 8

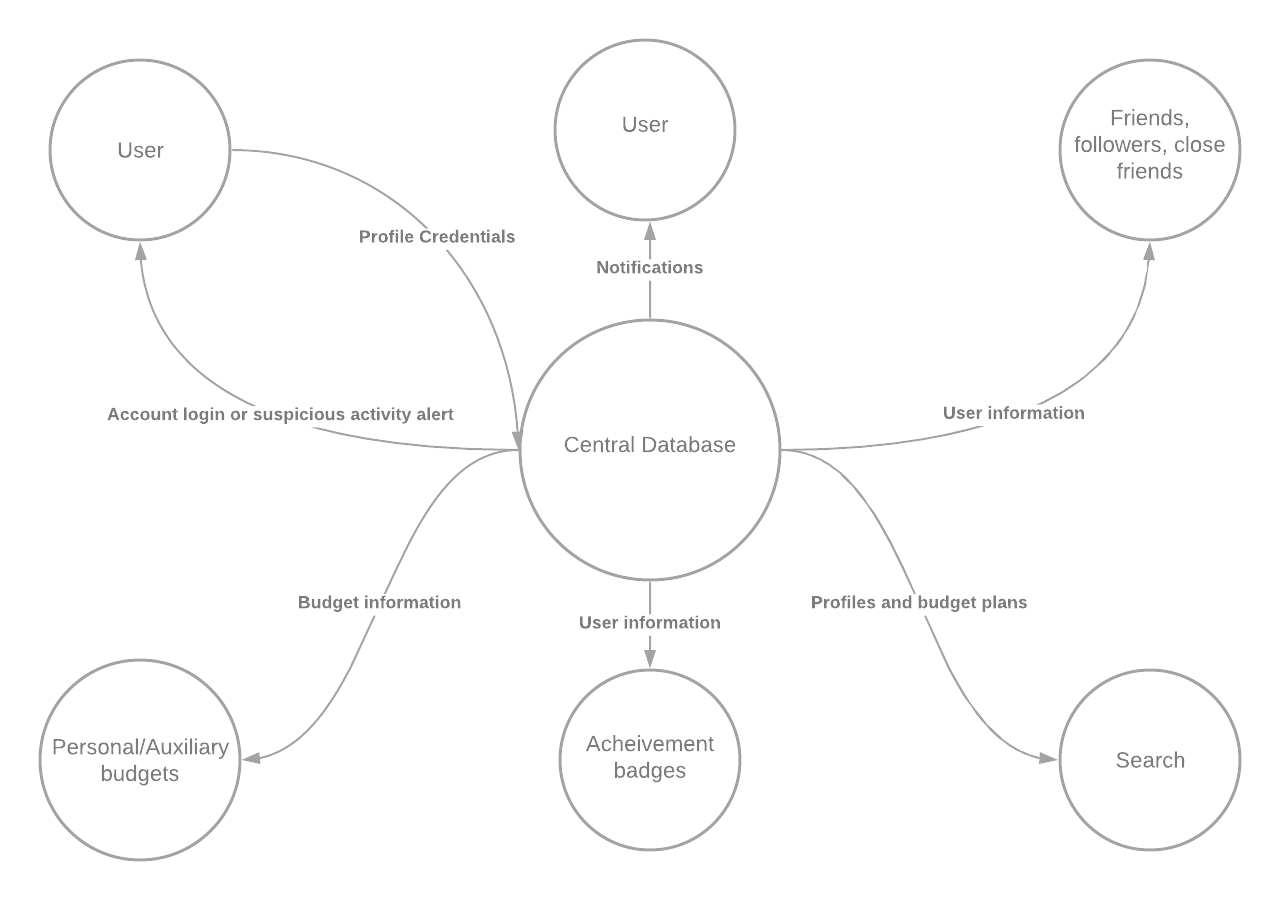
3 [Class Diagram](#_3dy6vkm) 9

### 1. System Analysis

#### 1.1 System Overview

Our system will be a database-centric one where a centralized database containing the budget plans with their tags, user profile credentials, notification information, and user profile information including friends, close friends, display name, profile and banner photo, and badges will be accessible to other components and services. These include the search function, notifications, private messaging, account login, suspicious activity detection, and bank account linking. Bank account information will be stored locally on the user’s device so it is safer and will not need to be entered every time the application is used. If a different device is logged into, the user must verify identity again and re-enter banking information. They will also be notified that banking information on the other device will be deleted.

#### 1.2 System Diagram

**

#### 1.3 Actor Identification

* Primary Actors
  + Users - Calls on the system to deliver a service (Log in, search, etc.)
  + Devices - Initiates a use case and requires the system to achieve its goal
    - Any android devices
    - Any IOS device
* Secondary Actors
  + Central database system - Provides data for the other use cases to complete their functions
  + client and server - Provides a way to share data among users for use cases to work

#### 1.4 Design Rationale

##### 1.4.1 Architectural Style

This will be a database-centric architecture style. A central database will store all necessary information except private banking information which will be stored locally. All functional and non-functional requirements will access this central database if necessary for their process.

##### 1.4.2 Design Pattern(s)

**Creational Design Pattern**

* Prototype: This design pattern would be beneficial for our project as it would allow us to create a prototypical instance of an object, in this case a budget plan, and clone it to produce new budget plans. This would allow us to create multiple budget plans with different priorities, in a faster and more effective way compared to other design patterns such as the factory method pattern that would require multiple subclasses, making it much more expensive. This also allows a user to create a budget plan before they know all the details of it, editing it as they learn more about budgeting.

**Structural Design Patterns**

* Facade: Since we are planning on having a social media type pf budgeting platform, where if you are close friends with another user you are able to see multiple components such as the user’s friends, other close friends, followers, both personal and auxiliary budget plans, achievements as well as how well they’re doing with their budgets. Since all this information can be seen on a single close friend’s profile, we want to be able to have a design pattern that allows us to see all this information in the simplest way possible. Since Facade provides us with a “simpler, unified interface to often complex underlying code with diverse subsystem components and gives us limited but straightforward access to other subsystems” (Profesor Aguiar), then it would be beneficial to implement this pattern in our application
* Adapter: When it comes to design patterns, adapter would be one of the best and most beneficial to implement on our application. As we had mentioned before, we plan on making the application for both iOS and Android mobile device, so using Adapter would be helpful as it would allow us us to use “different components with incompatible interfaces to talk to each other without modifying the source code in either component” (Professor Aguiar, Lecture 10).
* Decorator: Having the feature of followers, friends, and close friends in a way are tied together as you essentially get to look at other users budget plans, of course with different restrictions and features depending on whether you’re a follower, a friend, or a close friend. Using decorator would allow us to add functionalities without modifying or having to completely rewrite a separate code for each feature. Decorator would essentially allow us to add functionality to already existing components in our application without us having to modify our source code. (Professor Aguiar, Lecture 10).

**Behavioral Design Patterns**

* Command: Since we are working with multiple users at the same time who may be trying to access the same information or execute the same task, we want to have a design pattern that allows the users to get all the information they need with simple single commands, but also have a system that may delay these requests even if it's just milliseconds to prevent conflict with data retrieval. Since the command behavioral design allows us to “ encapsulate a command request as an object, letting us parameterize clients with different requests, queue or log requests, and support undoable operations'' (Professor Aguiar, Lecture 10), the command behavioral design pattern would be of best fit to our application.

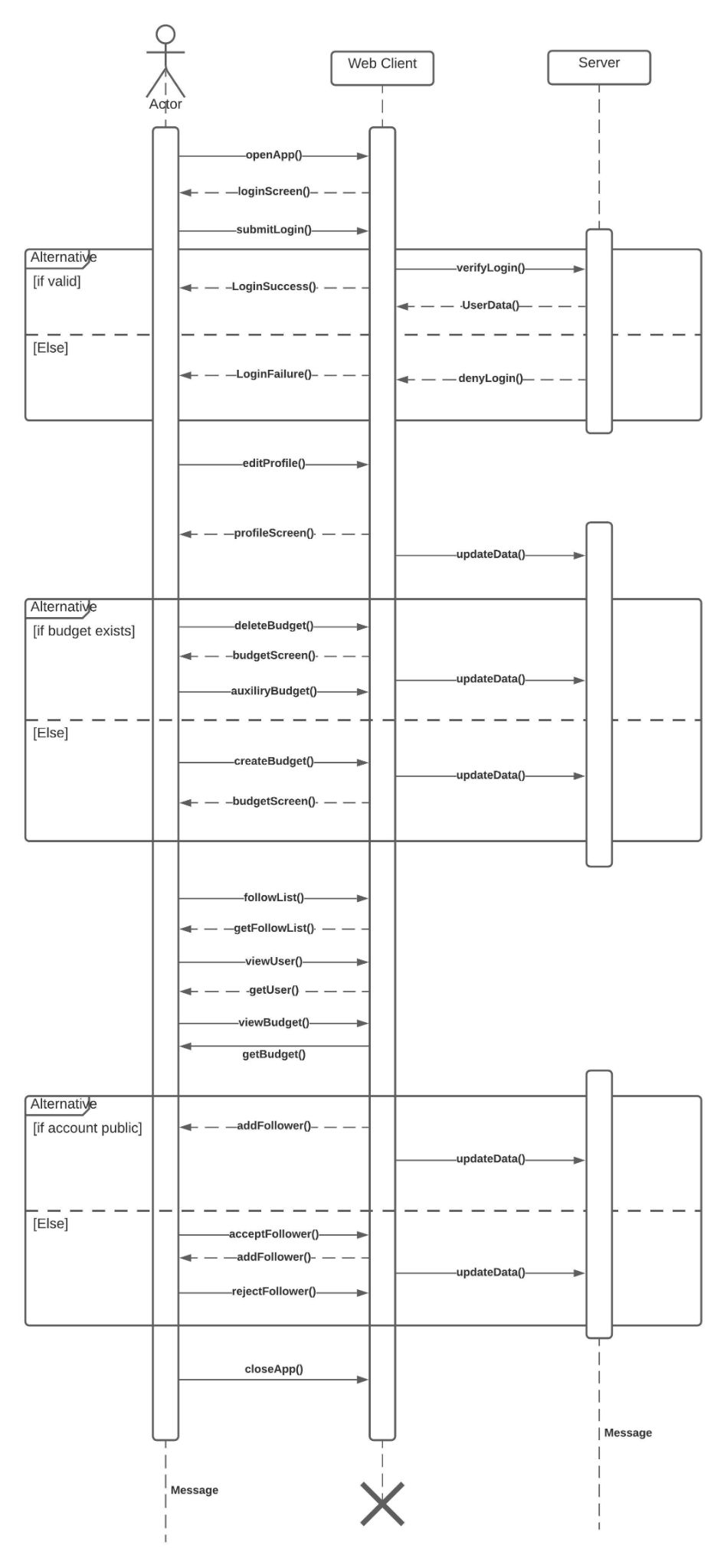
##### 1.4.3 Framework

When it comes to framework, SPRING: Java + Hibernate + MySQL/PostgreSQL would be the best framework for our application. Using this framework would allow us to create our central database and query any information needed as the user requests it. SPRING would also allow us to have a more secure application as its authentication feature allows us “to integrate with industry-standard security schemes and deliver trustworthy solutions that are secure by default” (Spring.io). This is essential in our app as there can be instances where users may link their bank accounts to the application.

### 2.Functional Design

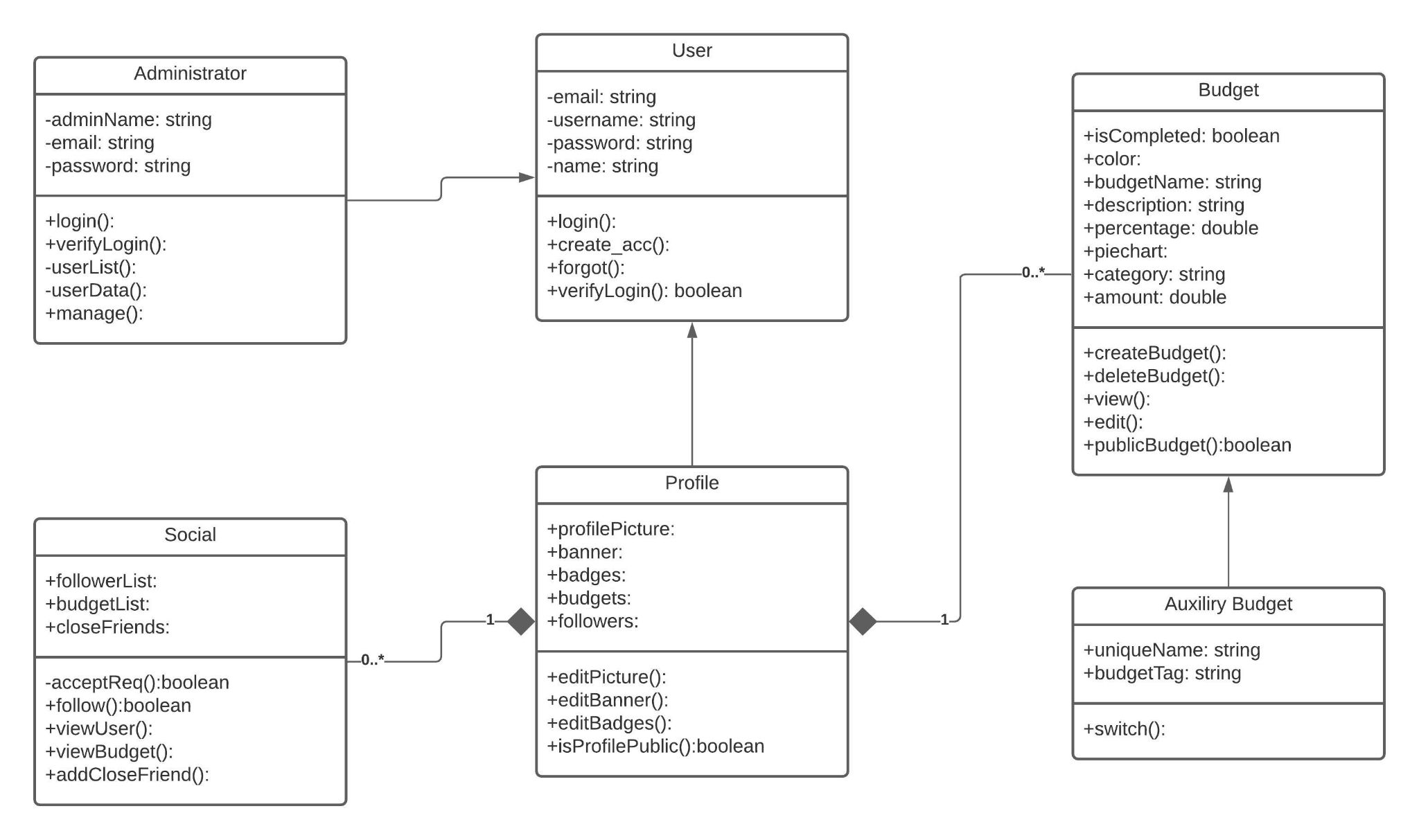
Following is our sequence diagram for Fundsbook

* When the user opens the application, the login screen pops up.
* After entering the login information for your account, it is verified and user profile loads if login information was correct.
* If login information is incorrect, the login request is denied and failure message is displayed.
* Budget screen pops up when you create a budget.
* When a budget is created, auxiliary budgets are available to create.
* If the profile is public, anyone can follow, and anyone can view your profile and budget(s).
* If the profile is private, you can accept or reject follow request(s). Only your followers can view your profile and budget(s).



### 3. Structural Design

Following is our class diagram representing our system:



### 