CS Enjoyers
CS 3704
Akash Dubey
Max Stroh
Dheer Prajapati
Khoi Lam
Mustafa Hawa

High-Level Design

The kind of architectural pattern that we will be using for our project will be the client-server pattern. This pattern is essentially what it is named: the connection between a server and a client. We will have data stored on a server, which will include information about the user's finances and budget, which can then be requested and accessed by the client. This pattern is most commonly used for many types of websites and even email, but most importantly, it is commonly used for banking. Therefore, this pattern is perfect for the kind of project we are developing.

Low-Level Design

For our financial management application, we need to ensure that our codebase is both flexible and efficient. Creational design patterns are well-suited to these objectives in our project. By using creational design patterns, we can enhance the flexibility of our codebase by providing a structured approach to object creation. Additionally, these patterns promote code reuse and maintainability, leading to more efficient development and easier maintenance in the future.

Pseudocode Representation:

```
InitializeApp()

While AppIsRunning:
   DisplayMainMenu()

   Choice = GetUserChoice()

   If Choice is "SignIn":
      SignInUser()
   Else then create account
```

```
        RegisterNewUser()
    Else just exit
        ExitApp()
    EndIf
EndWhile

# Supporting Functions

Function InitializeApp():
    LoadAppResources()
    ConnectToDatabase()

Function DisplayMainMenu():
    ShowOptions("SignIn", "CreateAccount", "Exit")

Function GetUserChoice():
    Input = GetUserInput()
    Return Input

Function SignInUser():
    Username, Password = GetSignInCredentials()
    If ValidateSignIn(Username, Password):
        User = LoadUserSession(Username)
        DisplayUserDashboard(User)
    Else:
        ShowError("Invalid credentials")

Function RegisterNewUser():
    UserData = GetNewAccountDetails()
    If Not AccountExists(UserData.Email):
        CreateAccount(UserData)
        ShowMessage("Account created successfully")
    Else:
        ShowError("Account already exists")

Function ExitApp():
    CloseAppResources()
    TerminateApp()

Function DisplayUserDashboard(User):
    ShowDashboardOptions("ViewBudgets", "ViewNotifications", "ViewAccount")

    DashboardChoice = GetUserInput()
```

```
    If DashboardChoice is "ViewBudgets":
        ShowBudgets(User)
    ElseIf DashboardChoice is "ViewNotifications":
        ShowNotifications(User)
    ElseIf DashboardChoice is "ViewAccount":
        ShowAccountDetails(User)
    EndIf

# Budget Management

Function ShowBudgets(User):
    Budgets = RetrieveBudgets(User)
    DisplayBudgets(Budgets)

Function CreateBudget():
    BudgetData = GetBudgetInput()
    SaveBudget(BudgetData)
    ShowMessage("Budget created")

Function ShowNotifications(User):
    Notifications = RetrieveNotifications(User)
    DisplayNotifications(Notifications)

Function ShowAccountDetails(User):
    Details = RetrieveAccountDetails(User)
    DisplayAccountDetails(Details)

# Utility Functions (to be implemented)

Function LoadAppResources():
    # Load necessary resources for the app

Function ConnectToDatabase():
    # Establish a database connection

Function GetSignInCredentials():
    # Prompt for and return sign-in credentials

Function ValidateSignIn(Username, Password):
    # Check credentials against the database

Function LoadUserSession(Username):
    # Create a user session from stored data
```

Function GetNewAccountDetails():
    # Collect new account details from user input

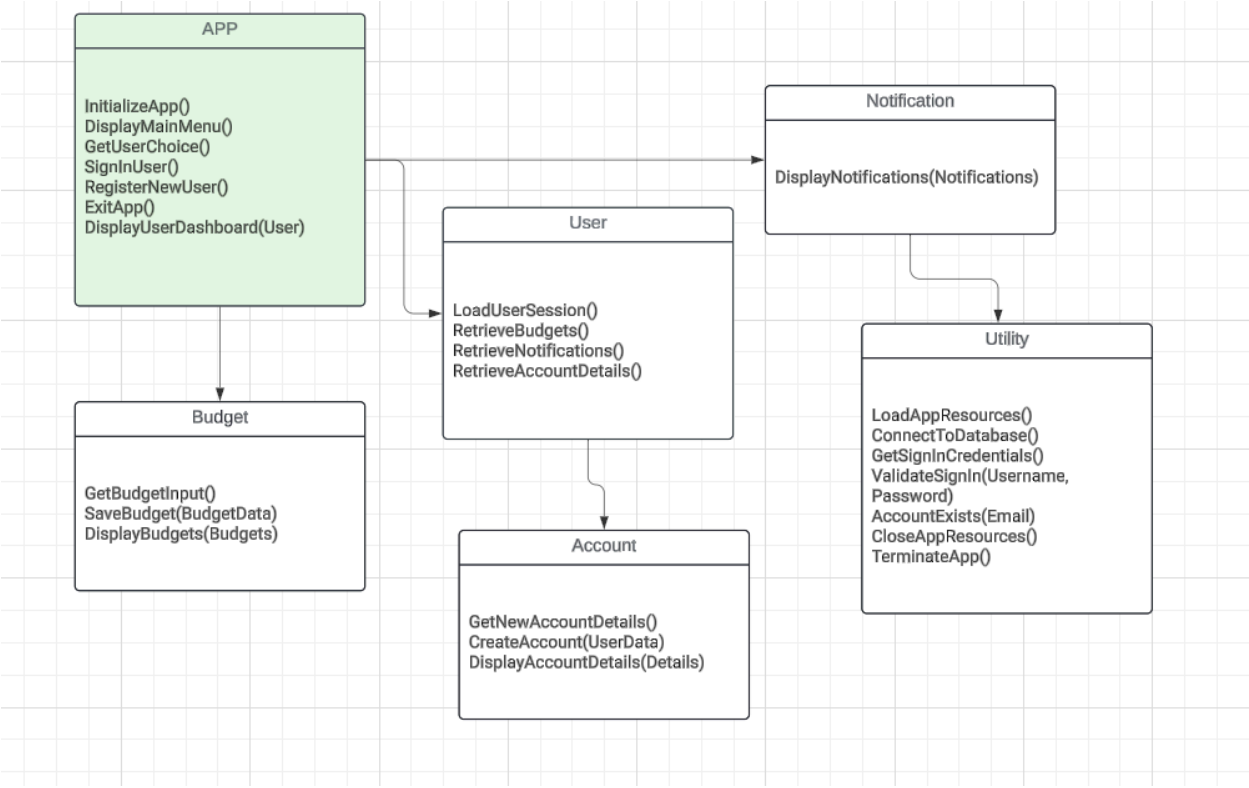
Function AccountExists(Email):
    # Check if an account with the given email already exists

Function CreateAccount(UserData):
    # Save the new account to the database

Function CloseAppResources():
    # Clean up and release resources

Function TerminateApp():
    # Close the application

Informal Class Diagram:



Kanban

For our process deliverable, we have chosen to do a kanban again. However, we have made some updates since our last kanban, which shows new goals that we have and are prioritizing. Bolded tasks represent tasks that are new and updated.

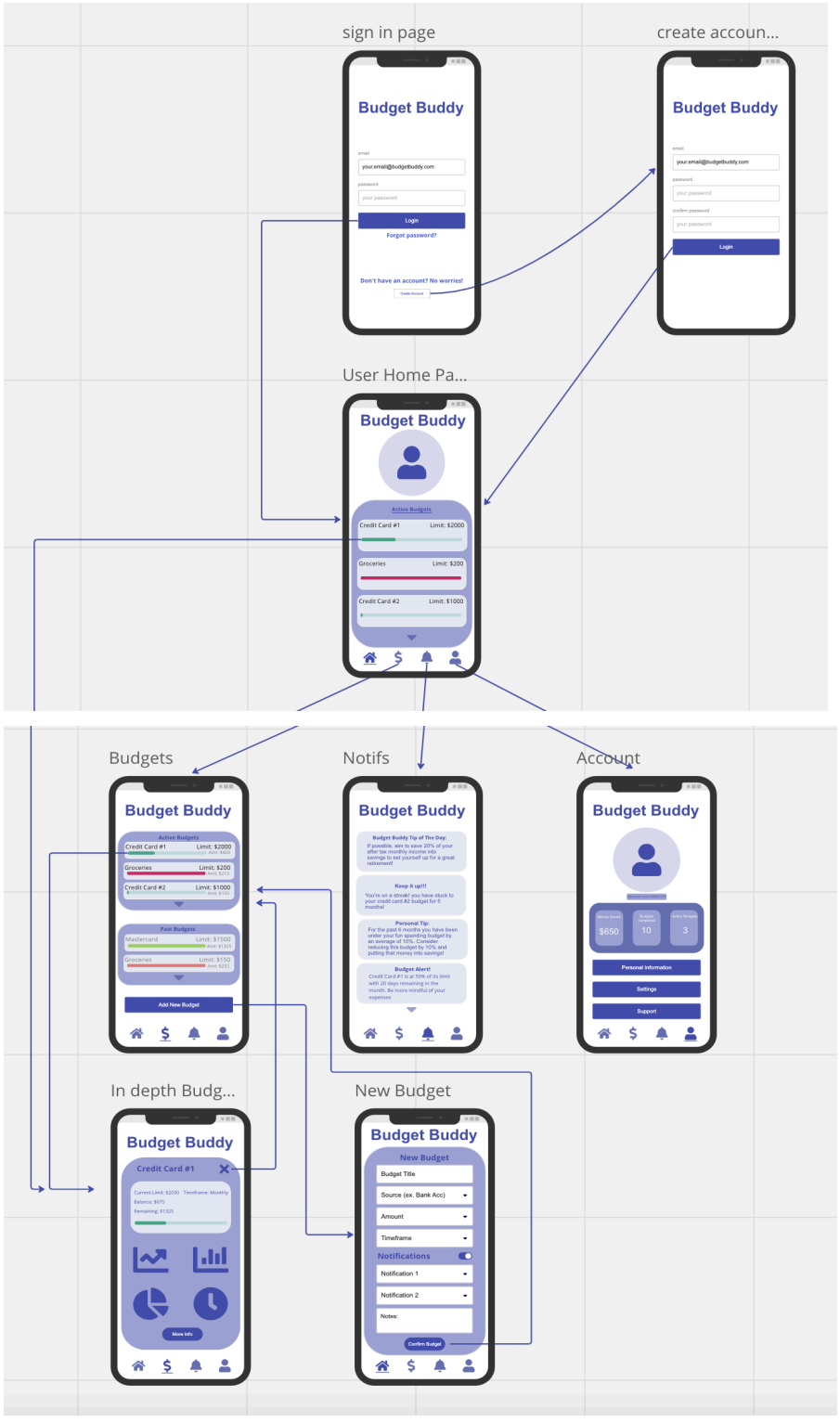
Priority List:

1. Establishing a deadline to work towards
    a. A deadline to work towards is crucial to getting our product or app out into the real world. Without a deadline, our team would have nothing to work towards and would eventually slow down how fast everyone works. With the deadline, we can work towards it and finish quicker.
2. Establishing clear communications between all members/teams.
    a. We need to establish clear communication between all team members and teams because it is essential that our various teams communicate with each other about issues that might affect each team. For example, if the front end is implementing something that would need the backend's help, like a login system, they would need to let the backend know how the login system works and what features it has.
3. Meet with backend developers to discuss plans for the backend structure.
    a. This is essential to creating the app since we need to always know what the backend team has been doing or planning, so we try to get all perspectives when it comes to developing the app in terms of the backend.
4. Meet with front-end developers to discuss plans for the front-end structure.
    a. This is also essential to creating the app since we need to always know what the front-end team has been doing or planning, so we try to get all perspectives when it comes to developing the app in terms of the front end. After forming a plan with the front end, we can successfully get a vision of how the app will look.
5. **Finish the project before the deadline**
    a. As the due date approaches, it is one of our highest priorities that we make sure we can meet the final deadline and submit our finished project. We should always be thinking about where we are and where we would like to be in relation to the final deadline.

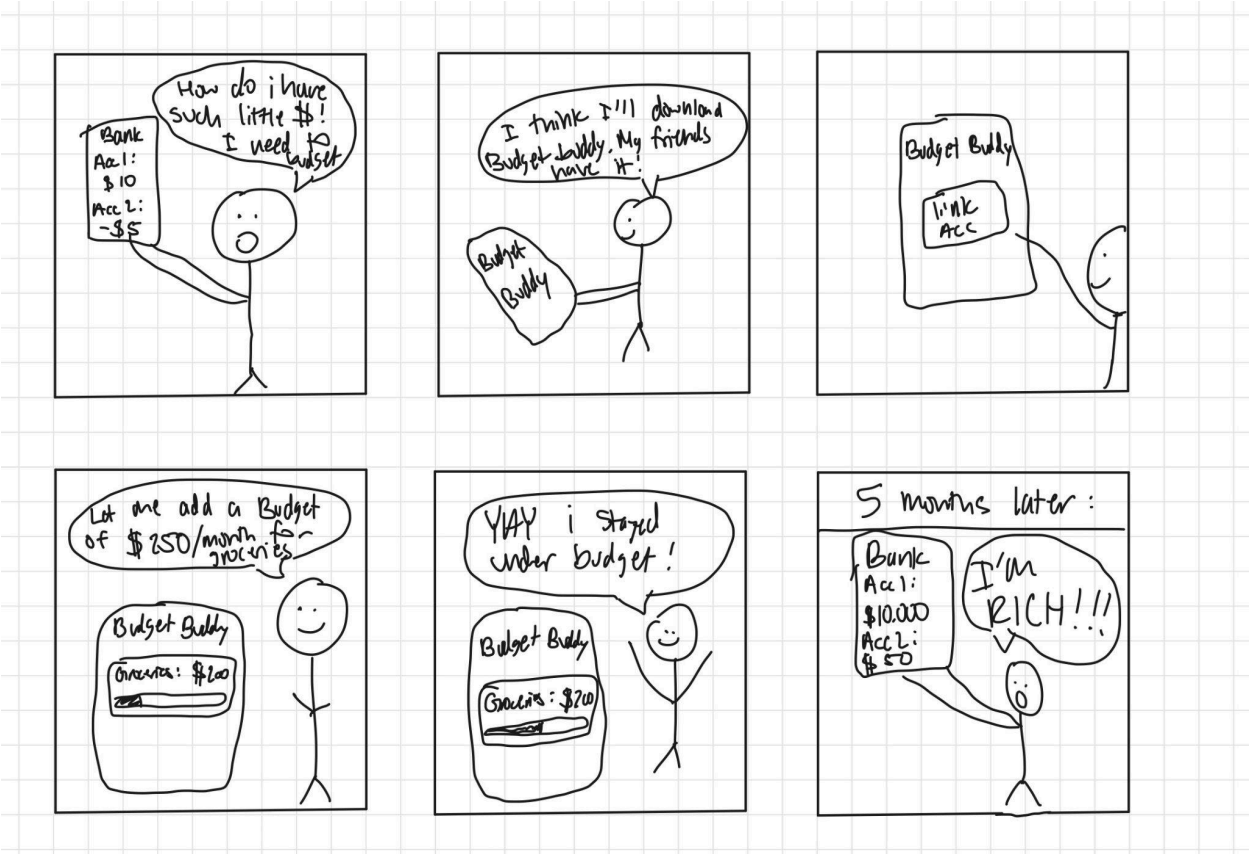
6. **Interpreting test data**
    a. In order for us to make the most out of user feedback from our prototype, it is important that we consider all of our data to further improve upon the project that we have. Doing so will allow us to have a more complete and polished project that we know people will like.
7. **Continue to receive feedback on our project**
    a. Even if one prototyping session has been completed, it is still important that we receive feedback from test subjects in order to completely ensure that our project is the best it can possibly be. We still should be mindful of what others think of our project, and seek to find ways that we can still improve upon it.
8. Meet with backend/frontend to try to incorporate sample data from findings.
    a. Our backend and frontend teams must meet to discuss any developments in their progress. This establishes clear communication and provides transparency to the whole team, which will allow us to be more efficient and successful when working towards our end goal.
9. Update previously written code if necessary
    a. Considering that we are using an adaptive and iterative process, it is important for us to go back and look through our code as we progress forward. We need to be able to adapt our code to any new findings, which will make our app greater and more flexible when developing.
10. Have weekly meetings to discuss progress towards deadlines.
    a. Having clear communication and transparency is extremely important when working as a team, so having weekly meetings that give updates on each team member's progress is essential for having this transparency.

Design Sketch:

Wireframe:



Storyboard:



When creating the idea of this app, our main goals were to make the app easy to use and convenient so that people at any wealth standpoint could use the app with ease. In the storyboard, the scario follows someone who has poor money managing skills and sees that his bank account is showing low amounts of money. He comes to the realization that he needs to start budgeting his money so he downloads Budget Buddy to help him save money. He creates a “grocery” budget to limit his monthly spending on greoceries to $250. After following this budget and any others that he created when he downloaded the app, after 5 months, he checks his bank statement to see that he has saved thousands of dollars by useing budget buddy. As stated earlier ease and convenience were the main focal points of this app which is why in the wireframe, there aren’t too many screens that the user needs to navigate through. The home page shows active budgets and the user can navigate to the money page to see active and previous budgets as well as add new ones, the notification page to see any reminders and tips that the app sends them, and the account page where they can change any settings or account information.