1. Introduction

Our app is a mobile budgeting service designed to support people who struggle with creating and sticking to a budget, for example, college students living on their own for the first time. Our app allows users to create their own custom budgets, and track their expenses to see how their actual spending compares to the goals they have set. In addition to helping users in managing their finances, the app also helps users discover local events and restaurants where they can spend any of their excess money from all of their responsible spending,

Students and budget-conscious users often struggle to keep track of spending while finding affordable ways to enjoy local activities. Most budgeting tools don't connect with lifestyle choices, leading to missed social opportunities. Our app combines easy budget management with real-time, location-based suggestions on restaurants and events. This way, users can manage their money better and still find experiences that fit their finances.

The app is designed for users who need help managing both everyday expenses and extra spending. With a focus on these needs, it offers a simple, user-friendly interface that helps users take control of their finances while staying connected to affordable options in their community.

2. User Research

Video Resources:

https://www.youtube.com/watch?v=ijtX98vlwWs

https://www.youtube.com/watch?v=0BPcguxoubQ

https://www.youtube.com/watch?v=vuyYYRqFbVc

https://www.youtube.com/watch?v=4Eh8QLcB1UQ

A piece of feedback that we received in our initial report was that we did not address all of the points brought up by some of the videos. "For example, I noticed your video mentions ways to earn money or find a job. There are also considerations regarding payback to parents, but this information isn't reflected in the insights provided." When creating our insights we were only focusing on parts that pertained to creating and managing a budget, we felt that certain points in the video were outside of the scope of what we were attempting to create and so did not address them in the user insights.

User Insights:

1. Balancing College Life and Finances - College Students often feel pressured to spend money to enjoy their time fully. This app could help them find a balance by allowing them to see how much money they can spend, while also showing nearby events and restaurants.

2. Budgeting Basics - Many students are new to budgeting or have never tried before. A simple budgeting tool with common expense categories could make managing finances easier.

3. Avoiding Financial Stress - Students often feel financially stressed. Spending analysis and simple saving tips with an app could help them feel more in control.

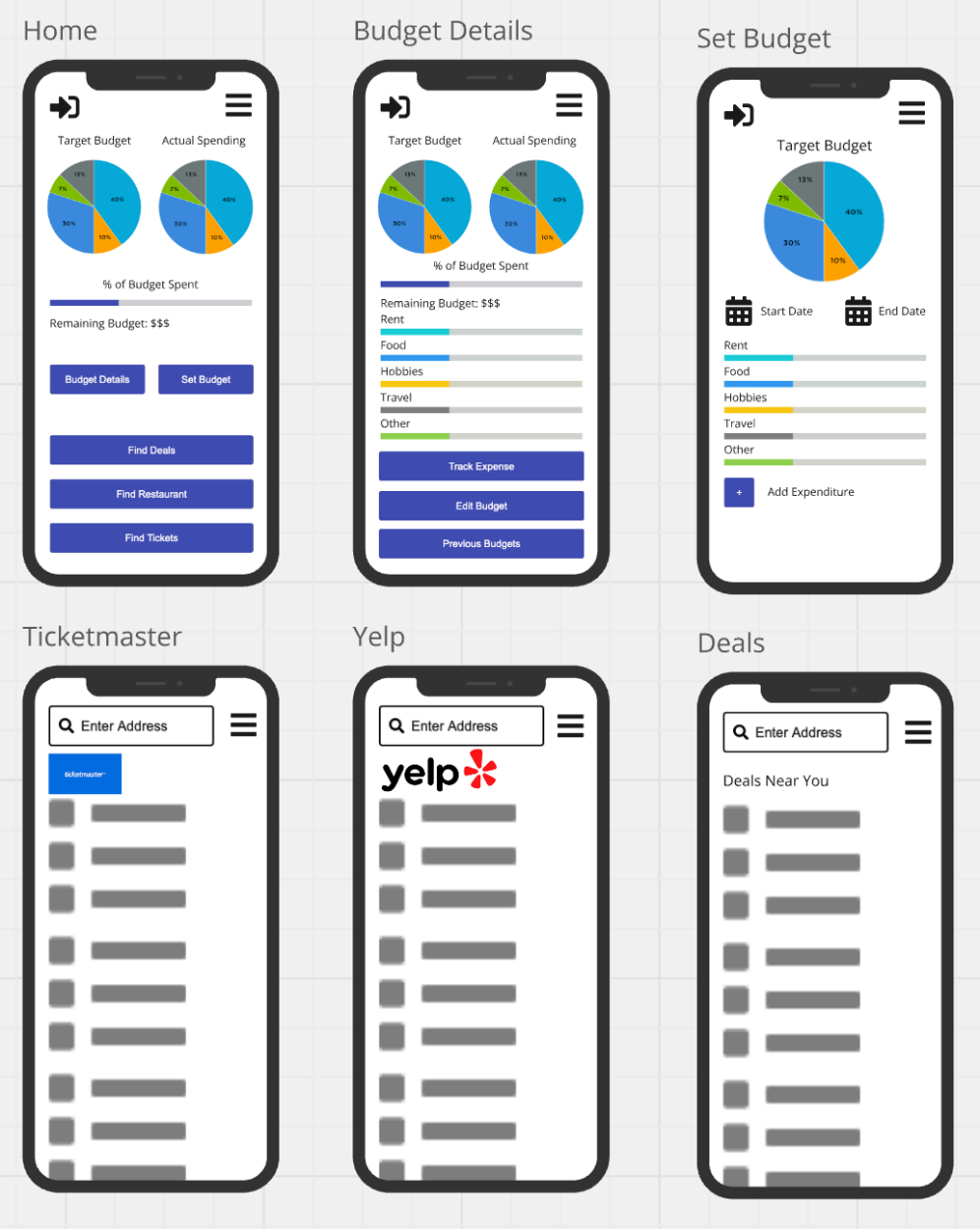
4. Learning for the Future - Individuals could benefit from setting simple, long-term goals while also practicing good spending habits.

User Stories:

1. As a student, I want to set a spending budget so that I can manage my expenses effectively.
2. As a budget-conscious user, I want to view a list of nearby events and restaurants that fit my budget, so that I can still enjoy my time without overspending
3. As a user, I want to categorize my expenses, so that I can better track where I spend my money.
4. As a user new to budgeting, I want to understand budgeting basics, so I can feel more confident in managing my finances.
5. As a user, I want to receive recommendations for budget-friendly options based on my location so that I can eat out without breaking my budget.
6. As a student, I want to set and adjust a personal budget, so that I can track my spending and stay within my limits.

In our first report we received feedback that some of these user stories and insights were too vague. We tried to keep that in mind as we continued development, but for the final report we included the original user stories and insights because we felt it better demonstrated what we were thinking at the time of planning our app. We felt that it would be unfair for us to go back and change them now that the app has been completed because it would no longer reflect how we actually approached development.

3. Wireframe Design



In our original wireframe design, we had 6 screens planned. For the final application we ended up creating several other screens but this was our initial concept on Miro.

1. The Home screen would display the most important information about the user's current budget, as well as offer navigation to other parts of the app.

2. The Budget Details screen would display a more detailed set of information about the user's budget, including the different categories, or expenditures, and how much money had been spent on each.

3. A Set Budget screen, which allowed the user to create a new budget, and plan out the total budget cost, start and end date, and a spending breakdown, where they could create expenditures and input how much of the budget would be allocated to each expenditure.

4. A Ticketmaster screen, which would use Ticketmaster's api to generate a list of events near the user, that the user could spend any excess money from their budget on.

5. A Yelp screen, which would use Yelp's api to generate a list of restaurants near the user that the user could spend any excess money from their budget on.

6. A Deals Screen where local deals would be displayed to help the user save money. (This screen was dropped from the final project on the recommendation of our Professor)

4. Implementation

UI Implementation:

Our application used three activities to achieve the final functionality.

1. The Main Activity contained all of the budget creation and tracking functionality. It contained a top menu, and a fragment container. We created a nav graph to navigate through the different budgeting fragments:

    - The Home fragment: Is the starting position of the nav graph, and can be returned to using the Home button in the menu. It uses text views to display the name of the current budget and the total budget as well as the amount currently spend. There is also a progress bar widget to visually represent how much money has been spent compared to the budget's total target. It also contains buttons to navigate to the Budget Details, New Budget, and Add Expense fragments.

    - The Budget Details fragment: Displays the details of a budget. There are text views to display the budget's name, amount spent, spending target, start date and end date. There is also a recycler view that displays the spending breakdown. Each view item in the recycler view displays an expenditure for that budget, using text views for the name of the expenditure, amount spent for that expenditure, and target for that expenditure. The view item also contains a progress bar representing the amount spent relative to the target. An example of an expenditure would be Rent: $500.

    - The New Budget fragment: Allows the user to design an new budget and input the relevant information. There is an edit text field for the budget name, start date and end date, as well as for expenditure name and target. There is an add button that adds an expenditure to a recycler view when its name and target have been filled. There is also a

text field that displays the total target of the budget which is the sum of its expenditures. Finally there is a save button that saves the new budget and all its expenditures, and replaces the last current budget as the new current budget.

- The Add Expense fragment: Allows the user to add expenses to their budget, tracking how much money they have spent. There is a spinner which displays a dropdown of all the expenditures for the current budget. There are edit text fields for entering the cost of the expense, the date, and any additional information. Finally there is a save button that creates a new expense, saves it to the database, and updates the current budget to reflect the new expense.

- The Previous Expenses fragment: Can be reached through the top menu button, All Expenses. It contains a recycler view to display all the previous expense the user has created. The view item in the recycler view contains text views that display the expenditure that expense belongs to, its cost, and its date. Clicking on an expense in the recycler view navigates to the Expense Details fragment and displays the details of that expense.

- The Expense Details fragment: Displays all the details of a given expense. There are text views for the budget and expenditure the expense is associated with, the cost, the date, and memo. There is also a Back button that navigates back to the Previous Expenses fragment.

- The Previous Budgets fragment: Can be reached through the top menu button, All Budgets. It contains a recycler view to display all the previous budgets the user has created. The view item in the recycler view contains text views that display the budget's name, total amount spent, and spending target. Clicking on a budget in the recycler view navigates to the Budget Details fragment and displays the details of that budget.

2. The Yelp activity has an edit text field where the user can input a zip code. There is a recycler view that will display a list of nearby restaurants when the search button is pressed. The view item in the recycler view uses text views to display the name of the restaurant and its address.

3. The Ticketmaster activity has two edit text fields to input longitude and latitude. There is a recycler view that will display a list of nearby events when the search button is pressed. The view item in the recycler view uses text views to display the name, date, and location of the event.

Data and Information Handling:

The data for the budget managing and tracking are collected from the user inputs in the New Budget fragment and the Add Expense fragment. The data for the Restaurants and Events activities are collected from the yelp and Ticketmaster API respectively.

There are four classes used for handling budget data:

- Budget class: Stores a string name, target double, spent double, start date string, end date string.

- Expenditure class: Stores and expenditure name string, an associated budget name string, target double, and spent double.

- Expense class: Stores an associated budget name string, associated expenditure name string, cost double, memo string, and date string.

- Current Budget class: Stores a string that keeps track of the name of the current budget.

Any screen that displayed information about a budget would use a budget object to store that information. Both the Budget Details and the New Budget fragments, utilized ArrayLists of expenditures to display all the expenditures for the budget's spending breakdown in a recycler view. The Previous Expenses fragment utilized an ArrayList of expenses to populate its recycler view, while the Previous Budgets used an ArrayLists of budgets.

View Model and Database:

Our View Model contained seven MutableLiveData variables:

- database: MutableLiveData of our database class for loading and saving all the user budgets, expenditures, expenses, and current budget.

- currentBudget: MutableLiveData of type Budget, used for displaying any information regarding the current budget. For example when expenses are added, the user is given expenditure choices based on the current budget and the cost is added the current budget.

- viewBudget: MutableLiveData of type Budget, used for displaying the details of a budget on the Budget Details fragment. When coming from the home screen the user is shown the current budget, when coming from the All Budgets fragment the viewBudget is set to the budget they selected and the details of that budget are displayed.

- viewExpense: MutableLiveData of type Expense, used for displaying the details of an expense on the Expense Details fragment. When an expense is selected for the Previous Expense recycler view, the viewExpense is set to it so it will display on the Expense Details fragment.

- viewExpenditureList: MutableLiveData of type ArrayList<Expenditure> used to store a list of expenditures to be displayed in the recycler views of the Budget Details fragment and the New Budget fragment.

- viewExpenseList: MutableLiveData of type ArrayList<Expense> used to store a list of expenses to be displayed in the recycler view for the Previous Expenses fragment.

- viewBudgetList: MutableLiveData of type ArrayList<Budget> used to store a list of Budgets to be displayed in the recycler view for the Previous Budgets fragment.

Our View Model also had three functions, addToExpenditureList, addToExpenseList, and addToBudgetList, which added objects to their respective lists.

For our app we used a room database to create a relational database for storing all of our budget information. All four data classes were stored as their own tables in the database.

- budgetTable
  - name: String (Primary Key)
  - target: Double
  - spent: Double
  - start_date: String
  - end_date: String
- expenditureTable
  - name: String (Compound Primary Key)
  - budget_name (Compound Primary Key)
  - target: Double
  - spent: Double
- expenseTable
  - eid: UUID
  - budget_name: String
  - expenditure_name: String
  - cost: Double
  - date: String
  - memo: String
- currentBudgetTable
  - id = “Current”(Primary Key)
  - name: String

The Budget, Expenditure, and Expense tables would keep track of all the information from user created Budgets, Expenditures, and Expenses. Expenditures would have a budget_name to signify which budget they belonged to, and expenses would have both budget and expenditure names to signify both the budget and expenditure they belonged to. Finally the currentBudgetTable would have one entry, “Current” and would store the name of the current budget, so that on start up the View Model currentBudget could be set to the correct budget from the budgetTable. Finally the BudgetAppDB extends the DAO’s for all the entities which contain all the relevant database queries.

We used two API services in our app, the Yelp API and the Ticketmaster API. The Restaurants Activity uses the Yelp API by sending a request containing a zip code, and the Yelp service returns a list of all restaurants near that zip code. The Events activity uses the Ticketmaster API by sending a request containing a location represented by latitude and longitude. The Ticketmaster API returns a list of events near those coordinates.

5. Evaluation

Design Problems:

In our opinion our biggest design problem is the apps lack of instruction. We believe all the functionality is helpful but are worried that a first time user might not know how to use the app. For example I believe that the process of creating a new budget may not be the most intuitive, and would love to hear user feedback on how to make that better for new users.

An additional design problem is the way we handle dates. Currently they are just stored as strings, the idea being that the app should give users as much freedom as possible, and so they can choose how they want to express the start and end dates in there own budgets. For example if they have a weekly expense, and they just want to mark it down as Monday, instead of mm/dd/yy form they are free to do so. The problem is with this freedom users could enter any strings as dates, resulting in some weird possibilities.

Finally, there is no way to edit budgets after they have been created beyond adding expenses. The missing or wanted features section later will explain the reason that this feature is missing but it does make the app slightly annoying to use since there is no way to fix a miss input like a typo.

Implementation Issues:

Surprisingly, our biggest implementation issue was organizing and sharing the work, more specifically with GitHub. We encountered various errors when trying to collaborate over GitHub culminating in the main branch breaking. For reasons we were never able to fully grasp, the main branch stopped being able to compile, when trying to revert to older versions we simply encountered more errors, from local file paths being changed, to GitHub being unable to merge branches. We eventually abandoned the repository and started using google drive to share zip

files of the project. While we no longer encountered errors with this method, it made it much harder for us to work simultaneously on separate parts of the code, and when we did, we would have to meet up and manually merge our progress into one project.

Another implementation issue came with trying to incorporate Ticketmasters API. The API uses longitude and latitude to determine nearby events, but from a user perspective we found this to be less than ideal. We felt that this was a poor way to express a location because most people do not know their longitude and latitude at any given moment. Perhaps we could have used the gps services of the phone to determine the users current location but that still wouldn't solve the problem of searching other locations. Ultimately, we were unable to figure out a better way to deploy the Ticketmaster API feature than longitude and latitude.

Missing or Wanted Features:

Our app is currently missing a few of the features we originally intended to implement. The most glaring of which are the edit budget functionality and the pie graph displays that were featured in the concept wireframe on Miro. Having never created pie graphs in Android Studio, we decided that it would be best to focus on the functionality that we knew we could implement. Similarly we felt that while the ability to edit a budget would be a useful feature in the app, it was not functionally required. In the case of a mistake the user could always make a new budget, while this is not ideal, we decided it would be a lower priority. We had hoped to implement the edit functionality and pie charts at the end but ended up running out of time.

Suggestions for Future Improvement:

The first thing we would look at for future improvements is the features we were unable to complete. We would want to include pie charts to add another visual representation of the user's spending to compliment the progress bars we have already implemented. We would also like to go back and add an edit budget feature, which would let the user change any of the information in their current budget, allowing them to correct typos, or add new expenditures they may have forgotten.

We would also like to make another attempt at making the Ticketmaster API more user friendly. Instead of the user needing to input a latitude and longitude, we would want them to be able to input something more commonly known, for example a zip code like in the Yelp API.

Finally the app does not enforce the proper number of decimal points for money. Instead of displaying $100.00 it will display $100.0. It also allows for users to enter fractions of cents, for example 87.877. By the time we realized this as an issue, we did not have the time to correct it. There were too many places where doubles were converted to strings to fix all of them. However, if given more time, it would be one of our highest priorities, in terms of improving the overall look of the app.

Key Code Snippets:

Budget display for Home fragment:

```
// current budget display
viewModel.currentBudget.observe(viewLifecycleOwner){
    progressBar = view.findViewById(R.id.home_progressBar)
    val targetBudget = viewModel.currentBudget.value?.target // Total budget
    val amountSpent = viewModel.currentBudget.value?.spent   // Amount spent so far

    // Calculate percentage spent
    val percentage = ((amountSpent!! / targetBudget!!) * 100).toInt()

    // Update ProgressBar and TextView
    progressBar.progress = percentage

    // update name spent and target
    budgetNameText = view.findViewById(R.id.home_budget_name)
    var name_string = "Current Budget: " + viewModel.currentBudget.value?.name
    budgetNameText.text = name_string

    spentText = view.findViewById(R.id.home_budget_spent)
    var spent_string = "Spent: $" + amountSpent.toString()
    spentText.text = spent_string

    targetText = view.findViewById(R.id.home_budget_target)
    var target_string = "Target: $" + targetBudget.toString()
    targetText.text = target_string
}
```

Setting up the recycler views for Expenditures:

```
// set up recycler
expenditure_recyclerView = view.findViewById(R.id.details_budget_recycler)
recyclerViewManager = LinearLayoutManager(context)
expenditure_recyclerView.layoutManager=recyclerViewManager

recyclerViewAdapter = ExpenditureRecyclerViewAdapter(viewModel.viewExpenditureList.value!!)
expenditure_recyclerView.adapter = recyclerViewAdapter
```

Error Messages for invalid inputs when creating a new Budget:

```
// save button on click
save_button.setOnClickListener(){
    if (budget_name_input.text.isEmpty()) {
        Toast.makeText(requireContext(), text: "Please enter a Budget Name!", Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }
    if (budget_start_input.text.isEmpty()) {
        Toast.makeText(requireContext(), text: "Please enter a Start Date!", Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }
    if (budget_end_input.text.isEmpty()) {
        Toast.makeText(requireContext(), text: "Please enter an End Date!", Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }
    if (target_tracker == 0.0){
        Toast.makeText(requireContext(), text: "Please add an Expenditure!", Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }
    // makes sure budget name doesnt exist already
    if (viewModel.database.value!!.BudgetDAO().bExists(budget_name_input.text.toString())){
        Log.d( tag: "Database Call",  msg: "budget already exists" + budget_name_input.text.toString())
        Toast.makeText(requireContext(), text: "Please choose new Budget Name!", Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }
    // create newBudget
```

Yelp API interface:

```
interface YelpApi {
    @GET("businesses/search")
    fun searchRestaurants(
        @Header("Authorization") authHeader: String,
        @Query("location") location: String
    ): Call<YelpResponse>
}
```

Creating a new budget, and inserting it and its expenditures into the database:

```
// create newBudget
var newBudget = Budget()
newBudget.name = budget_name_input.text.toString()
newBudget.target = target_tracker
newBudget.start_date = budget_start_input.text.toString()
newBudget.end_date = budget_end_input.text.toString()
viewModel.currentBudget.value = newBudget
// update currentBudget in db
var updated = CurrentBudget()
updated.name = newBudget.name
viewModel.database.value?.CurrentBudgetDAO()?.insert(updated)
// add new budget to db and budget list
viewModel.addToBudgetList(newBudget)
viewModel.database.value?.BudgetDAO()?.insert(newBudget)
// set all expenditure names to budget name
for (exp in viewModel.viewExpenditureList.value!!){
    exp.budget_name = newBudget.name
    // add expenditures to db
    viewModel.database.value?.ExpenditureDAO()?.insert(exp)
}
// go home
findNavController().navigate(R.id.action_global_homeFragment)
```