CO566
Mobile Systems

CW1

21611431

12th May 2018

Computing & Web Development
Buckinghamshire New University

Summary

Part A

Background

Section 1

Background to the App Idea

I have many times tried to control my budget and find ways to manage my money more efficiently and tried different methods to do so, from the basic pen and paper to apps that use APIs to connect to my bank account to get information about how I spend my money.

Oftentimes I found the idea of an app like that very good at the beginning and set my goals for my budget but always end up either stopping using the app after a little while or not using at all to start with.

With that in mind, I have gotten the idea to build my own budget planner app as the project for this module. It won't be anything extremely complex like the app I once used that connects to my bank account's API, but that will hopefully help me keep track of my own expenses in an efficient manner.

My main ideas for the app are the following:

1. Allow the user to input and customize their budget
2. Allow the user to enter, modify and delete their expenses
3. Allow the user to input their income and their pay day
4. Tell the users how much they have been saving, in a daily, monthly and overall basis
5. Separate the expenses into different categories so the user can adjust budgets more adequately
6. Track daily expenses by calculating a daily spend budget and reducing the expenses made that day from it

Section 2

Project Planning and Management

In order to manage my progress in this project, I am using some Scrum concepts applied to the way GitHub manages project management, since it is where I am hosting the code for the report.

GitHub uses Kanban boards to help teams keep track of the tasks that need to be completed, in a section of the repository called Projects. I used an automated Kanban board that would move my cards around as I got around completing the tasks.

Tasks can be defined in GitHub through the creation of issues. Issues can contain anything from feature implementation and basic project tasks to bugs that need fixing. I would use a issue to note a big task, for example, design the UI of the app, and use the checkboxes available to the writing of an issue to add smaller subtasks that can be considered as objectives to get the bigger task done, such as, for example, draw the sketches of the UI.

I also used the concept of Sprints from the Scrum methodology to set myself a deadline for a certain number of tasks that I needed to complete that were related (for example Design for all the tasks related to the UI of the app, from sketches and wireframes to the actual implementation of the app UI). In GitHub, we can implement Sprints by setting milestones with a title, description and date for completion and then assigning the tasks to the milestone that we want it to be related to.

For my project, I set myself the following milestones and tasks. The plan will be organized in the following way:

- Milestone
  - Task
    * Subtask

2.1 Sprints and tasks

- Design and Planning
  - Decide on the features to be implemented
    * Brainstorm features for the app
    * Select most basic features
    * Narrow down list of optional features to the three most important
  - Plan implementation of the app
    * Write down all the tasks that need to be done for the completion of the app
    * Plan sprints
  - Design the app
    * Design first sketches
    * Implement sketches in wireframe and review
    * Rework on the sketches and re-implement if and as necessary
- Implementation
  - Prepare UI and Layout of the app (for the following features)
    * Management and customization of budget
    * Input of expenses
    * Input of income
    * Tell users how much they have been saving
    * Separate expenses into different categories
    * Track daily expenses
  - Implementation of features related to the UI (Implement processes that happen when the user interacts with the UI for the following features)
    * Management and customization of budget
    * Allow input of expenses
    * Allow input of income
    * Tell users how much they have been saving
    * Separate expenses into different categories
    * Track daily expenses
  - Implement background features
    * Reset budget and money when payday has been reached
    * Update total savings when payday comes through
    * Track daily expenses (reset money spent for the day)
- Deployment and testing
  - Deploy the app to a local emulator

    * Deploy the app into the emulator provided with App Inventor
  - Deploy the app on your own mobile device
    * Make sure that your device is available
    * Deploy the app into the device
  - Test the app in the local emulator (things to test)
    * Interaction with the design (the application does what it should when the user interacts with it)
    * All the data is being stored and retrieved
    * All the calculations are correct
  - Test the app on your own mobile device
    * Interaction with the design (application does what it should when the user interacts with it)
    * All the data is stored and retrieved
    * All the calculations are correct
    * Make sure it doesn't crash

Another good thing about planning the progress of the project with GitHub is that it allows me to review and modify the issues (or tasks) whenever I want, so if there is new things to implement or things that need to be done I can quickly review my project plan, but it also allows me to post comments related to the issue that I am working on, allowing me to track and log my progress, difficulties and sucesses when doing my work.

Section 3

Deciding on Features

When I tackled the task of deciding on which features to implement, I started by concentrating for about five minutes and write down all the features that I thought I would like to see on the app.

When I first started thinking about this project, I also wanted the app to be able to remind the users to input their expenses and to encourage users to keep on tracking their expenses by using notifications to congratulate them on how much they have been saving so far, but, unfortunately, due to time constraints, I had to give up on that idea. However, that idea also inspired me to get some more original ideas of features for the application and it probably wouldn't be the same as what it is now if I didn't have the idea of trying to encourage the user to use the app as much as possible.

These are the features that I came up with on my brainstorm session:

- Set automatic budget based on income
- Allow management and customization of budget
- Separate expenses into different categories
- Allow input of expenses
- Allow modification and removal of entered expenses (categories and retification of amounts)
- Calculate daily expenses based on budget
- Track daily expenses
- Remind users to enter their expenses (at opportunate times)
- Tell users how much they have been saving
- Alert users when daily budget is about to be reached

- Allow separation of expenses that count towards daily expenses or not
- Allow users to enter their income in a way that accomodates the different ways of getting paid (per hour or annual salary) and when they get paid (weekly, fortnightly or monthly)

After the brainstorm session, I separated all the features that I thought about into two types: the ones that would provide the basic functionality of the app and the ones that were optional for the implementation of a budget planner. When I first did this separation, I was still planning on getting the application to motivate the user through notifications.

3.1 Basic Features

1. Management and customization of budget
2. Allow input of expenses
3. Allow input of income
4. Remind users to enter their expenses
5. Tell users how much they have been saving

3.2 Optional Features

1. Set automatic budget based on income
2. Separate expenses into different categories
3. Allow modification and removal of entered expenses
4. Calculate daily expenses based on budget
5. Track daily expenses
6. Alert users when daily budget is about to be reached
7. Allow separation of expenses that count towards daily expenses or not
8. Allow users to enter their income in a way that accomodates the different ways of getting paid (per hour or annual salary based) and when they get paid (weekly, fortnightly or monthly)

When I separated all these expenses, I had to reduce my list of optional features to implement. For me it was pretty straight forward that all the basic features had to be implemented and that only a couple of the optional features would end up being implemented. At the time of writing this section of the report, I am now realising that I ended up implementing more optional features than what I had initially intended because of the way I designed the app and because it seemed like the application wouldn't be complete without them.

3.3 Final features

1. Management and customization of budget
2. Allow input of expenses
3. Allow input of income
4. Separate expenses into different categories
5. Allow removal of entered expenses
6. Calculate daily expenses based on budget
7. Track daily expenses
8. Tell users how much they have been saving

Part B

Design

Section 4

Design Process

After planning what features I would implement on the app and planning the tasks that I would do in order to get the app ready on time for the presentation, I tackled the design of the application by starting to design the UI of the app in paper, then moving to wireframes and review them, reworking both if necessary and then implement the UI in App Inventor.

Since I didn't know much about how App Inventor worked and how to build applications with it when I started designing the application, I decided to dive in into research of how to make applications with App Inventor and starting implementing the features as I learned about the platform and then rework on the final design taking into account the constraints of building in App Inventor.

When designing the User Interface, I assumed that the application would be used with the smartphone screen being in a portrait orientation and took into consideration that the elements would need enough screen space to allow the user to interact with them. This task was made easy for me considering that App Inventor automatically allocates a generous amount of space for the elements as a default, but it was still something that I had to take into account when modifying the original proportions of the elements in order to achieve a certain design.

Considering that most of the functionality of the application would be triggered by events resulting of the interaction of the user with the User Interface, I started planning the design of the application by deciding on what the User Interface would be.

Section 5

User Interface Design

5.1 Overview and Introduction Screens

When I started designing the user interface, I had set myself the goal of making the use of the app as intuitive as possible, so that it would take as less effort as possible from the user to get to learn the interface and how to use the application.

I also started thinking about how I was going to tackle the implementation of the data in the system and how to make the application do all the things that I had planned for it to do.

When I was planning the application and the features that it would have, I was planning with the vision of it being used by a single person for personal use and since the application didn't hold any information that would be valuable to steal, such as credit card numbers or bank account numbers, that it wasn't necessary to implement any supplementary security measures other than the ones that users implement in Android itself (such as screen lock) or to create any accounts to access the application.

However, I knew that I wanted the application to ask the user for certain data when lauching the application for the first time in order to get a base for the rest of the functionality. I planned then for the application to have three introductory screens:

- An introduction saying the information that would be required from the user in order to get the application running
- A screen to enter the initial budget seperated by categories
- A screen to enter the monthly income of the user

5.2 Home Screen and Expenses-related screens

After deciding on what the introduction screens would contain, I started thinking about what the user would want to see when they would enter the application every single time in the form of a home screen. Since the goal of the application was to encourage the user to enter their expenses on a daily basis, I decided on making the home screen display the expenses that the user has made on the current day, as well as the money that the user has left until the next pay check and for the day and to let the user add expenses to the system with the help of a button clearly displayed on top of the list of expenses for the day.

When I was thinking of a way to list the expenses, I had to make a decision on what kind of information I wanted the expenses to contain. I came to the conclusion that since the expenses would be entered by the user into the system, that they would want to give them a name, as well as a description in case the expense contains many things (for example, when doing the shopping for the week), but also that some more basic information should be associated with them, such as the store or place where the expense was made, to which category it belonged and how much it was. That was quite a lot of information to be contained inside a home screen. In order to unclutter the home screen and make the information as accessible as possible to the user but still allow them to see the most important information, I planned on making the elements in the list of expenses clickable, and, when clicked, lead the user to another screen that would then display all the information of the expense to the user in a clear and concised way.

After making that decision, I had then to decide which data to present to the user in the expenses list. When designing the sketches and wireframes, I had in mind to present the name of the expense, the place where it was made and the amount of money,but, due to constraints inside App Inventor, what I ended up implementing was only the name and amount of the expense.

After deciding what the home screen would contain, I was left with the dilema of how to manage navigation inside the application. For that purpose, I had different options, such as a collapsible menu or a navigation bar. I decided to implement a navigation bar that would lead the user to the main functions of the application, and then inside the main pages, if necessary, lead the user to additional screens through buttons or list elements, just like for the expenses list on the main page or the button to add expenses.

After designing the home screen, I started designing the view that would display the information for an expense. My goal for that view was to give as much space for the display of information as possible and to present it clearly and properly separated so that the user could grasp it easily. When designing it, I made sure that the user also knew what was the information that was presented to them by the means of labels explaining what the following information was. I also wanted the user to be able to do something with that page other than consult the information related to that expense, so I planned on adding two buttons at the button of the screen, one that would allow the user to modify

the information of that expense and another to delete the expense from the system. In the final product, however, considering that the user can add an expense to any date they wish, I decided to remove the modify button from that screen and only leave the delete button.

The next screen that I designed was the screen to add expenses. My intent with that screen was to make it in the shape of a form. An horizontal space would be reserved for each of the information required from the user for the expense, allowing the screen to not be overflowed with elements, and at the right bottom of the screen would be a button for the user to save the information in the system.

5.3 Button placement

Since I just evoked buttons and their placement, I have placed most of the buttons in the application on the firstly on the bottom of the screen but then also to the right of the screen since most people are right-handed and tend to hold their phone only with their right hand and use their thumb to interact with the screen.

5.4 Category-related screens

For the categories views, I decided to use a similar approach to the one for the expenses views. The user would be able to see a list of the categories and the amount of money that has been allocated to each of them thorugh a list on a categories main page that would be accessible though the navigation bar. Clicking on any of the items of the list would then lead to a page with the details of the category and a button that would lead the user to a page displaying all the expenses that were assigned to that category in a list, just like the one from the home screen.

The view presenting the user with the details for the category would be very similar to the one displaying the details for an expense, and would also allow the user to change the amount of money allocated to that expense. Some of the information displayed for that category would also be what should be associated type of expenses should be associated with that category and an estimate of how much of the income should be allocated to said category, according to NimbleFins (n.d.).

5.5 Budget, Income and Settings views

The last views that I designed were the views for the budget, the income and the settings.

The budget view allows the user to both check and change the amount of money that is set to be spent on a monthly basis as well as check how much is set to be spent on a daily basis, check how much the user is saving on a monthly basis if they manage to spend according to budget and how much they have saved so far with the application.

The income view allows the user to see and change their monthly income and their pay day, as well as check how much money they have left for the month. By knowing when the user gets paid, it is possible to do background checks to see if the user has gotten paid and then reset the amount of money they have to spend, how much they have spent for the month and also know exactly how much they have saved in the previous month.

The settings view is a very simple view. In the beginning of the development of this view, I didn't have that many settings that could be changed in the application other than the currency used for the application and the option to reset the app to the beginning. When implementing the feature, however, I ended up giving up on the setting to change the currency, since most of the elements that require the input from the user don't have any currency signs in order to allow the enforcement of numerical data entry only.

All of these views are accessible though the navigation bar of the application for ease of access for the user.

5.6 Sketches

With all of the thought process that helped me design the application explained, I can now present the sketches that helped me design the user interface and decide on how to implement the features of the application:

5.7 Wireframes

When converting the sketches to wireframes, I used an open-source GUI prototyping tool called (n.d.) and elements from collections that contained common Android user interface elements to give them a more realistic look and feel.

5.8 App Inventor Implementation

The implementation of the user interface was pretty straightforward, however, there were some things that I needed to take into consideration and work around with when building the interface:

Figure 5.1: Sketches for Budget Planner

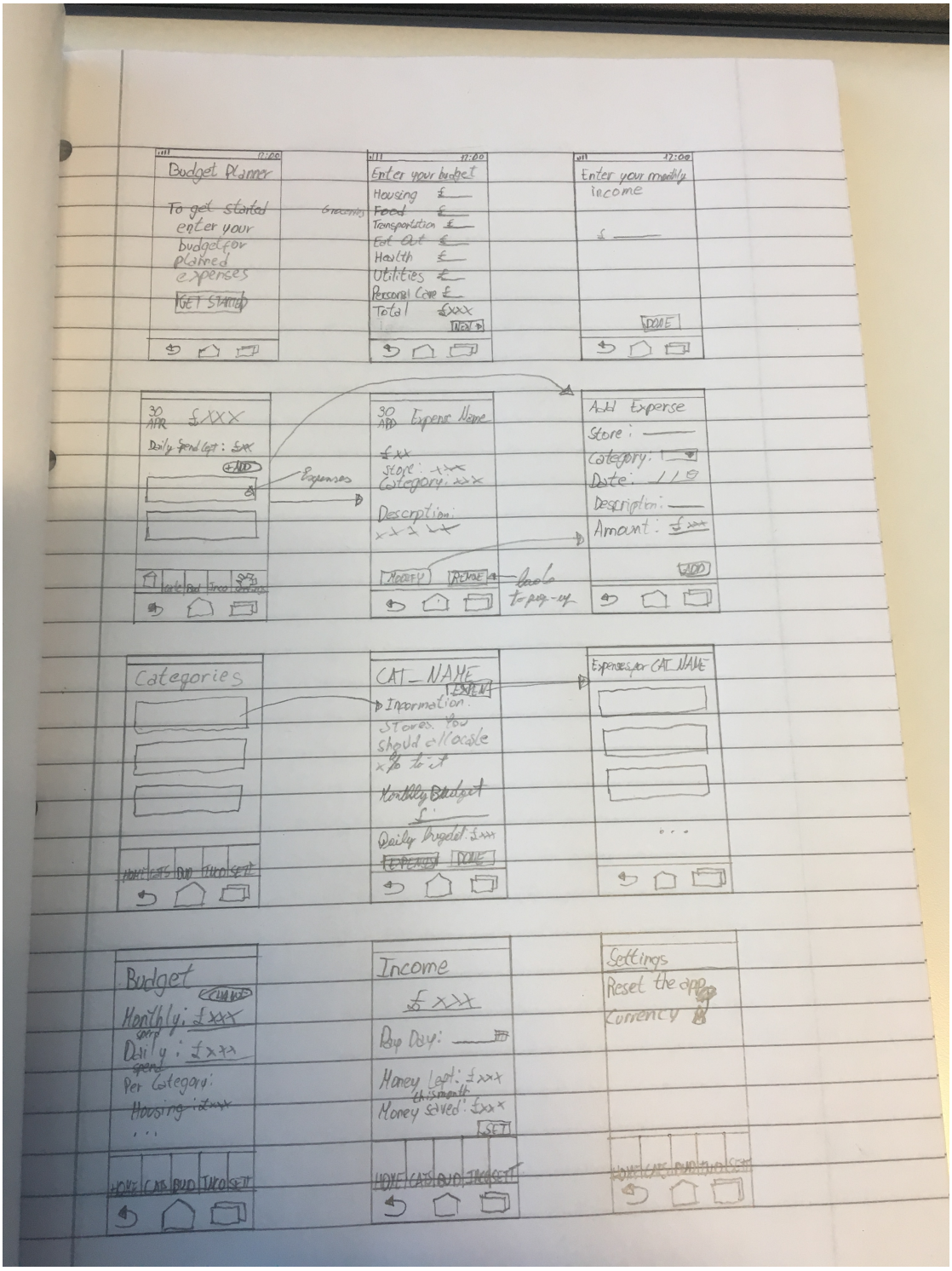
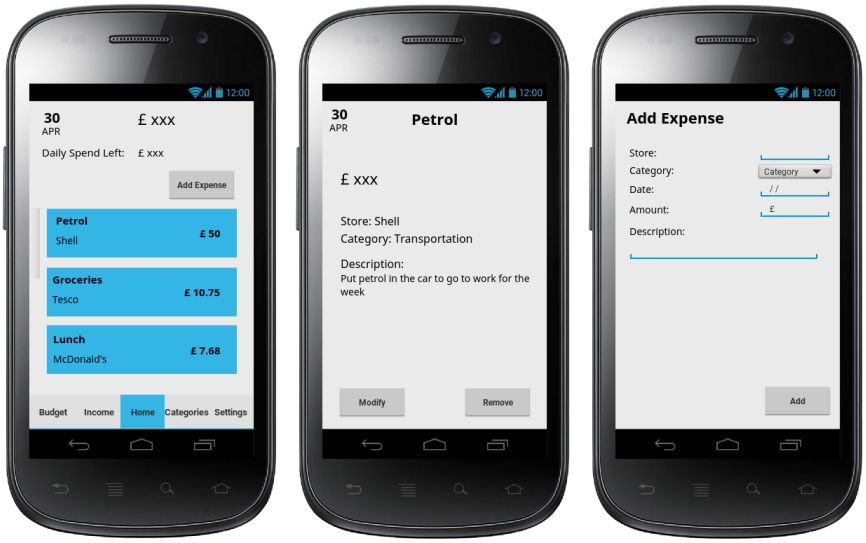


Figure 5.2: Introduction Screen wireframes

Figure 5.3: Home Screen and expenses-related wireframes



This last activity also allows the user to modify an expense

Figure 5.4: Category-related views wireframes

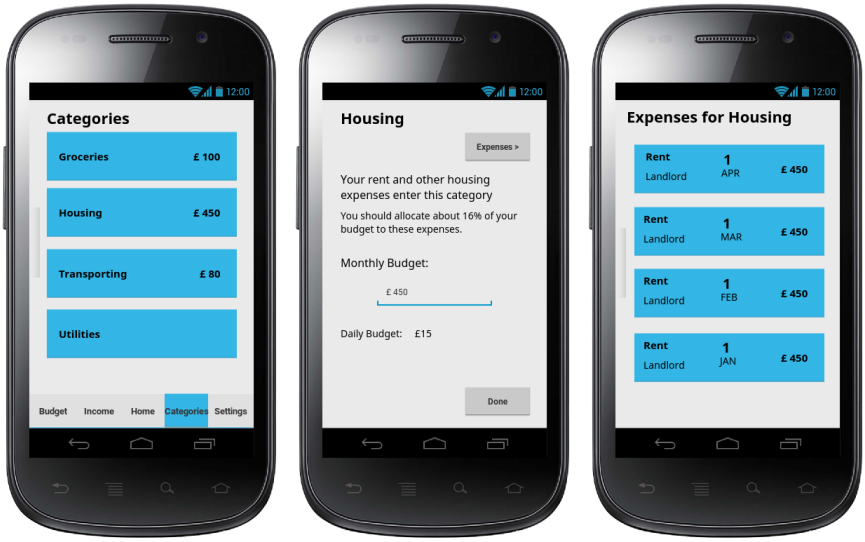
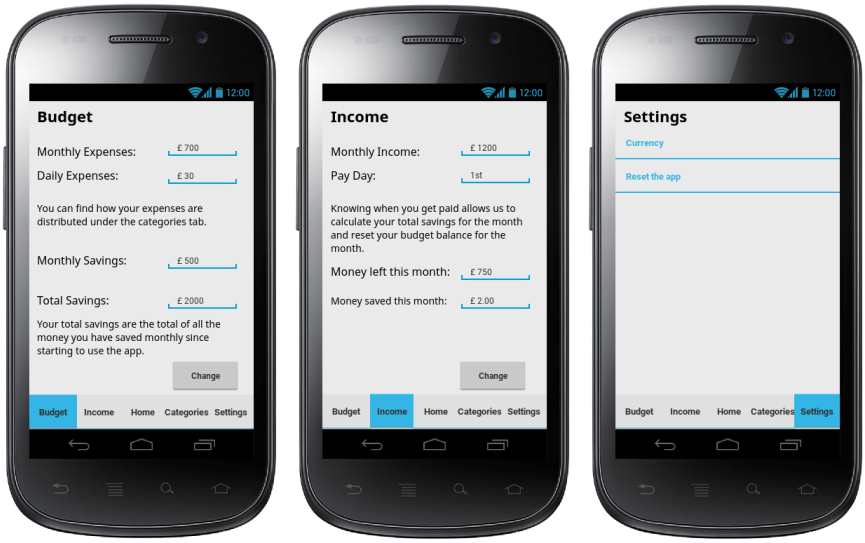


Figure 5.5: Budget, Income and Settings views wireframes



- In App Inventor the elements have fixed positions that are automatically allocated in by the system and are not customizable, therefore I had to place different horizontal and vertical arrangements and play with the height and width of the elements to get them positioned where I wanted them to be.
- App Inventor doesn't allow certain elements to be clickable or to have certain types of event handlers (which I only realised when trying to implement the programming into the User Interface), therefore I had to spend extra time into customizing the appereance of certain elements such as buttons or lists to get an appearance similar to the one I had originally designed. This wasn't possible for every case, unfortunately, and some elements of the interface do look quite different when compared to the wireframes.

Here is the final user interface of the application, as seen in the design screen of App Inventor:

Figure 5.6: Introduction Screen

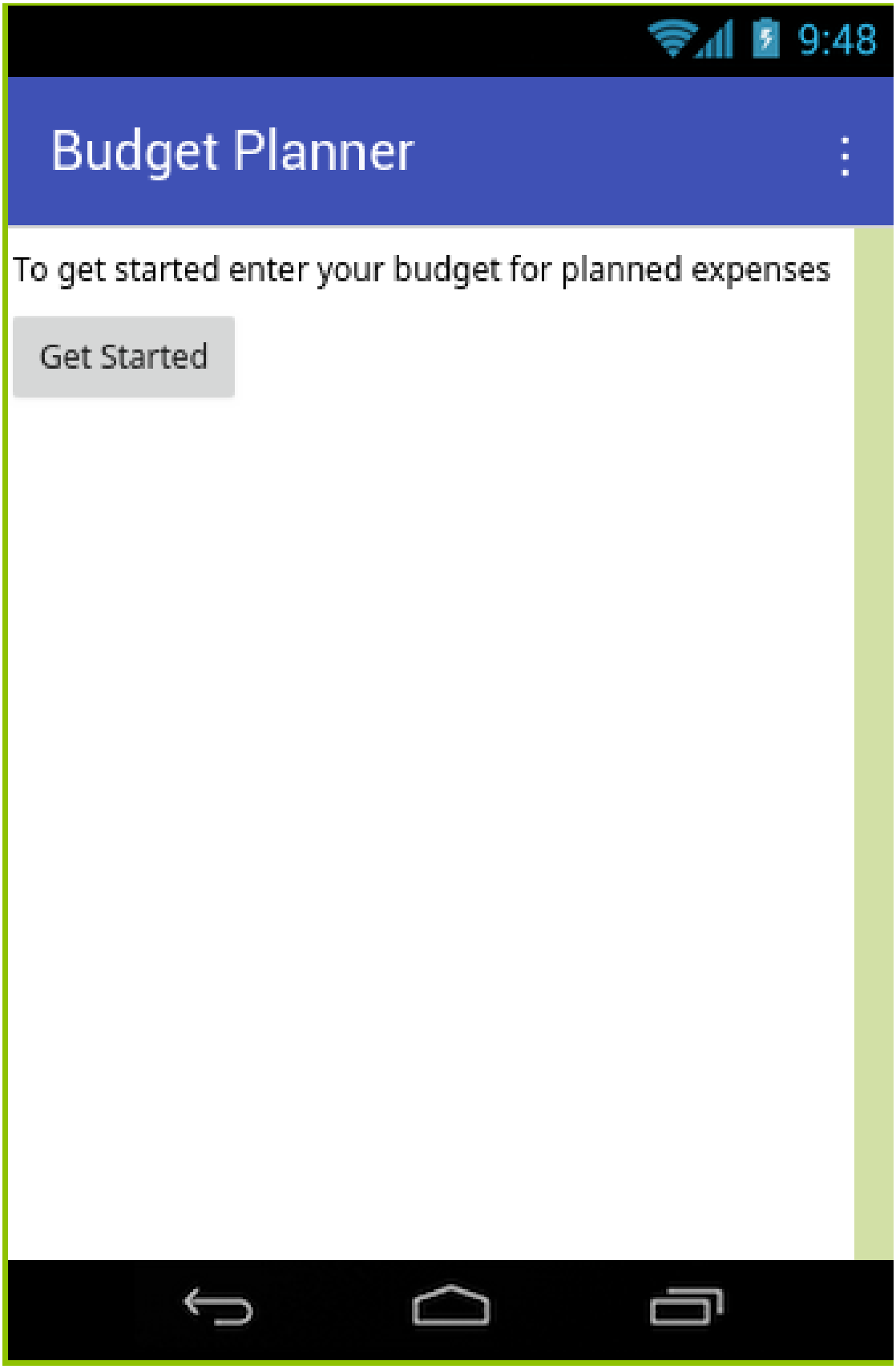


Figure 5.7: Introduction Budget Input

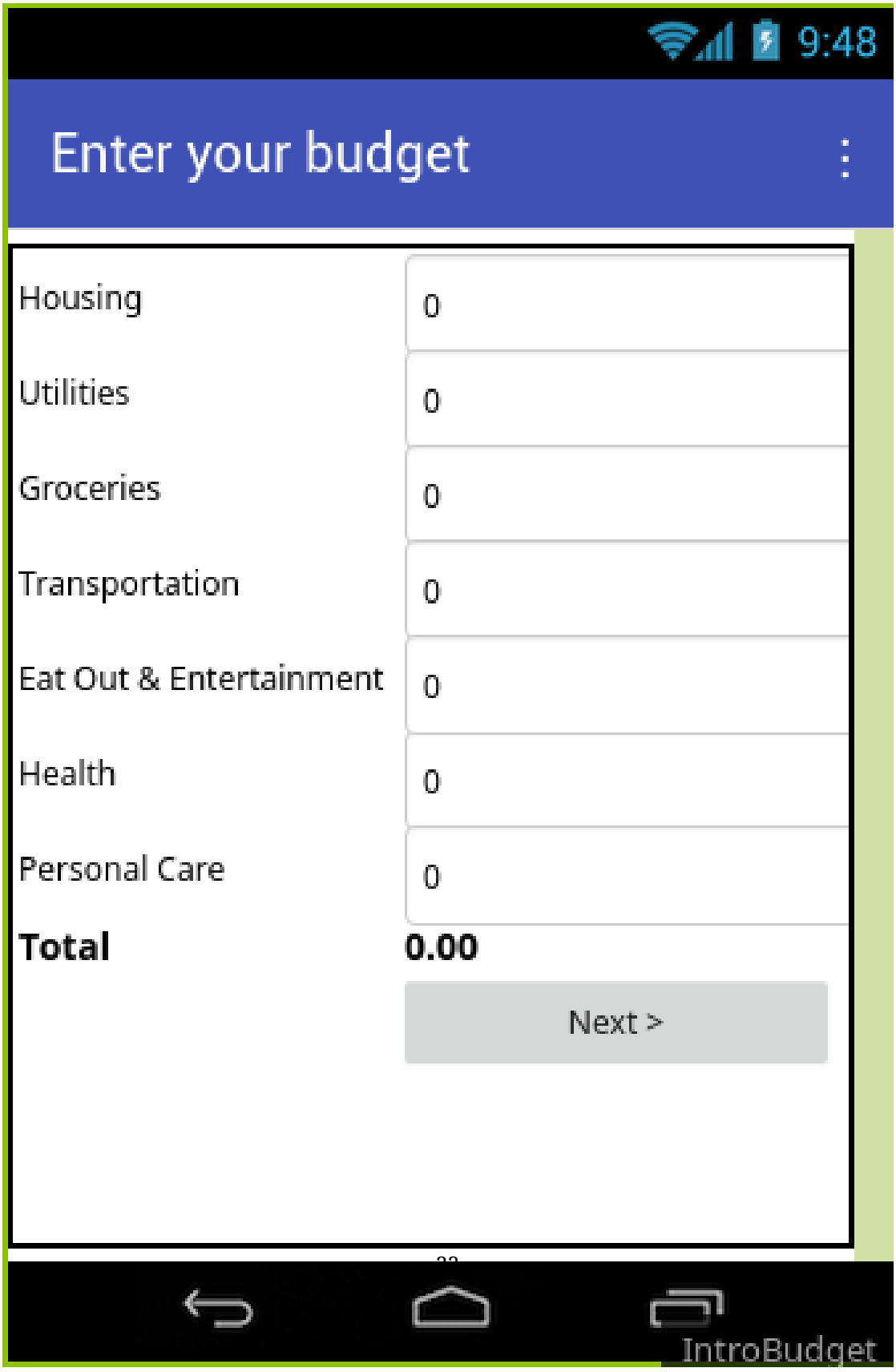


Figure 5.8: Introduction Income Input

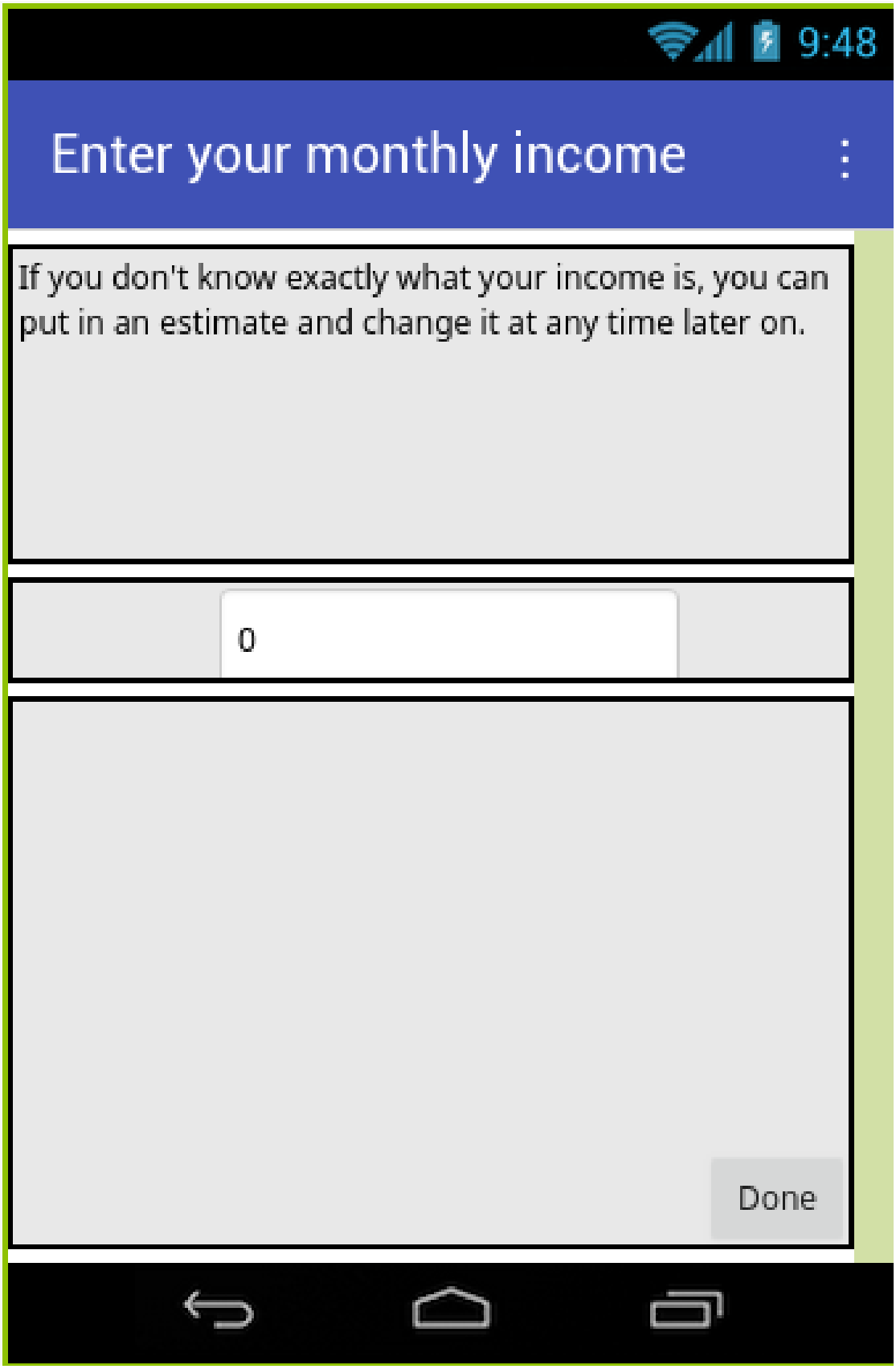


Figure 5.9: Home Screen

Figure 5.10: Expense View

Figure 5.11: Add Expense Screen

Figure 5.12: Categories List View

Figure 5.13: Individual Category View

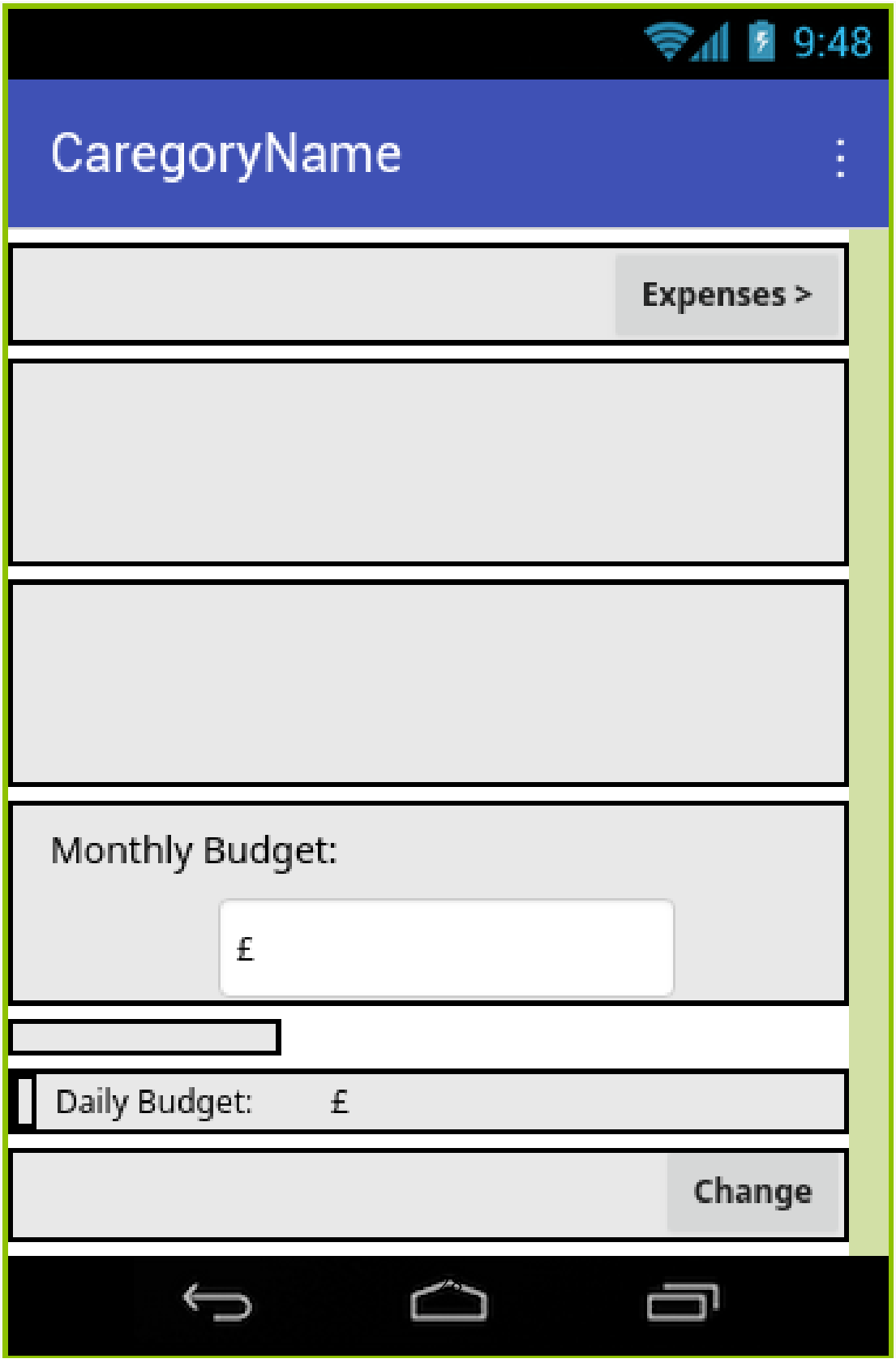


Figure 5.14: Expenses for a Category View

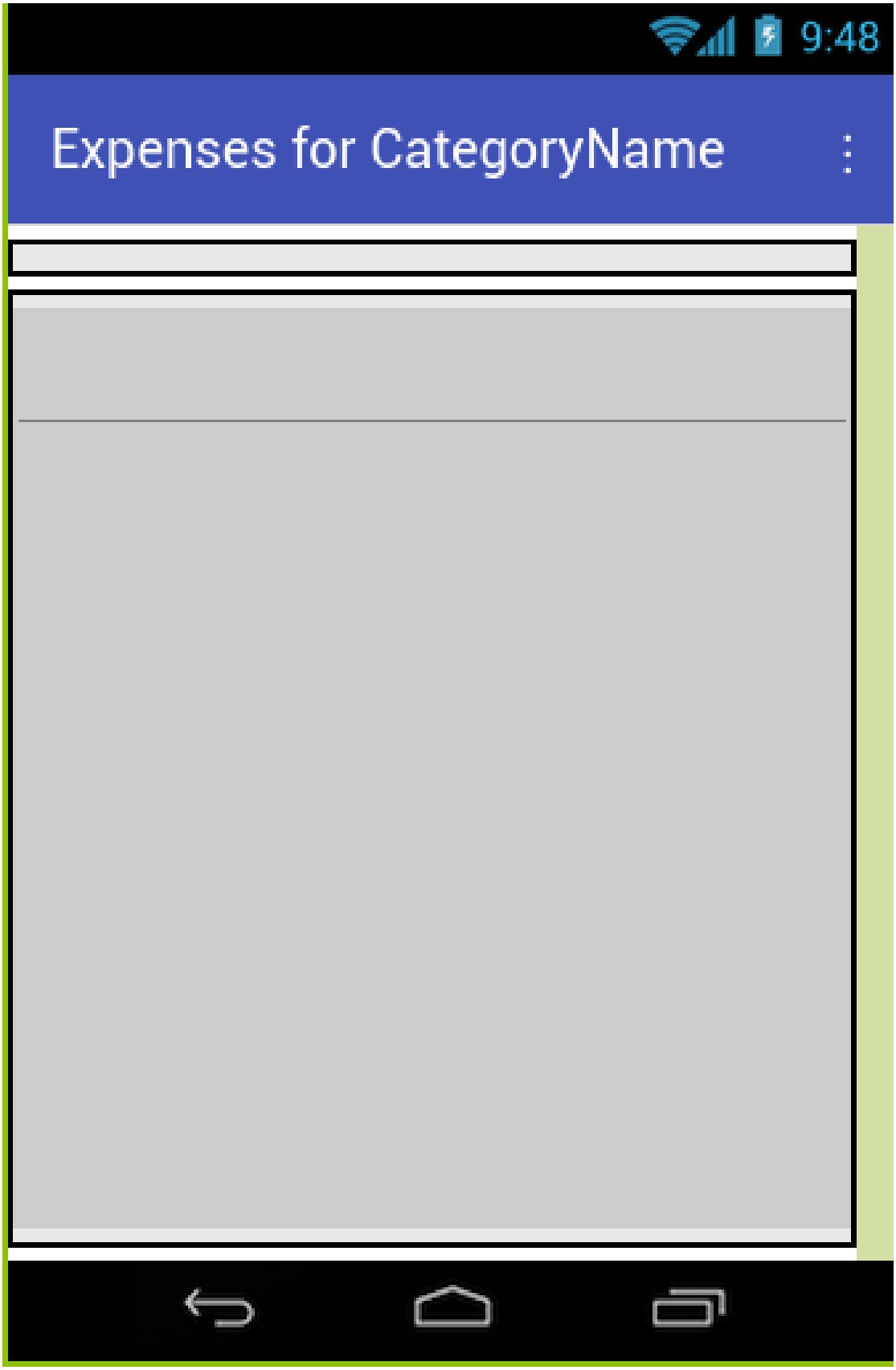


Figure 5.15: Budget View

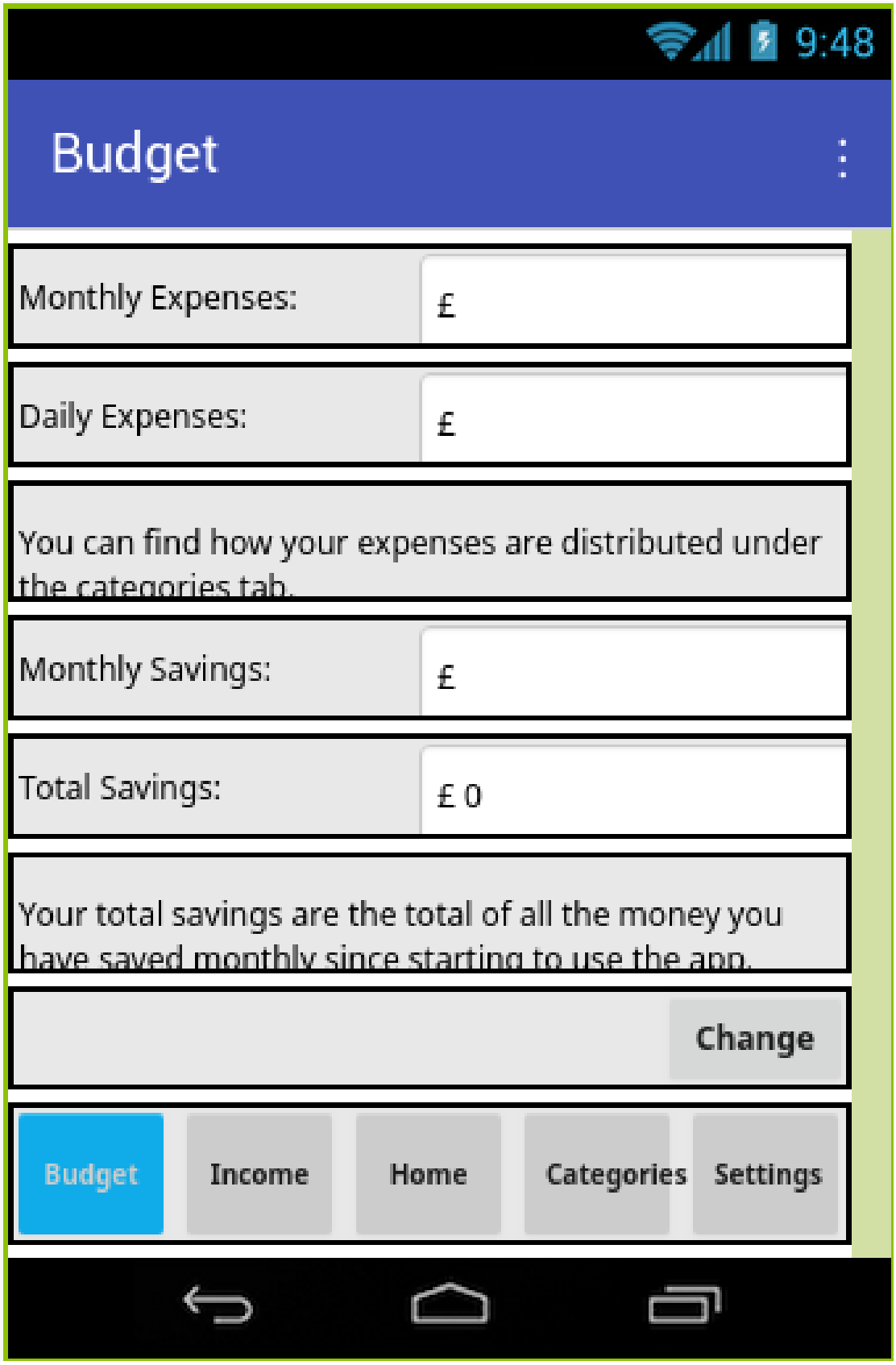


Figure 5.16: Income View

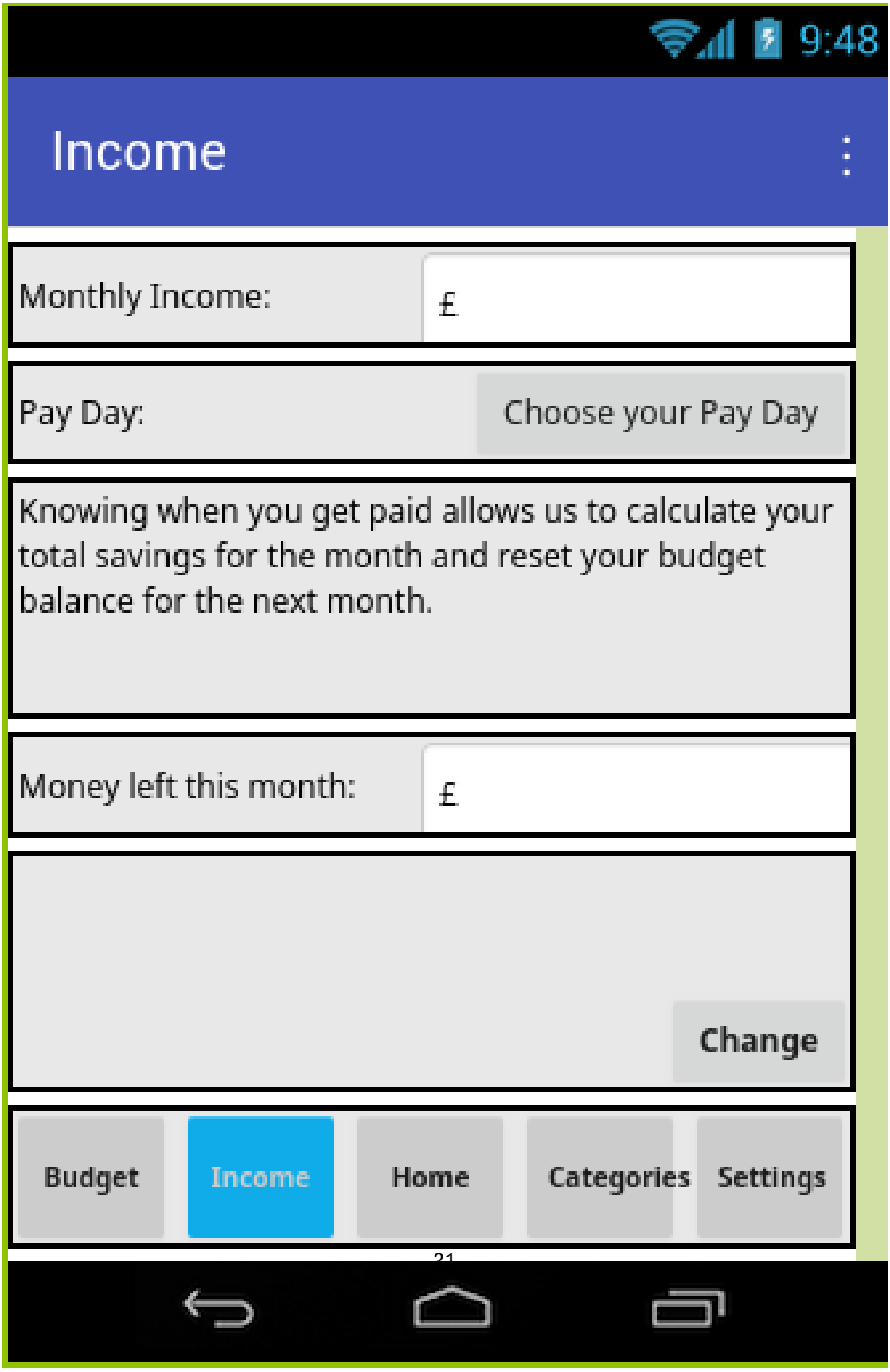
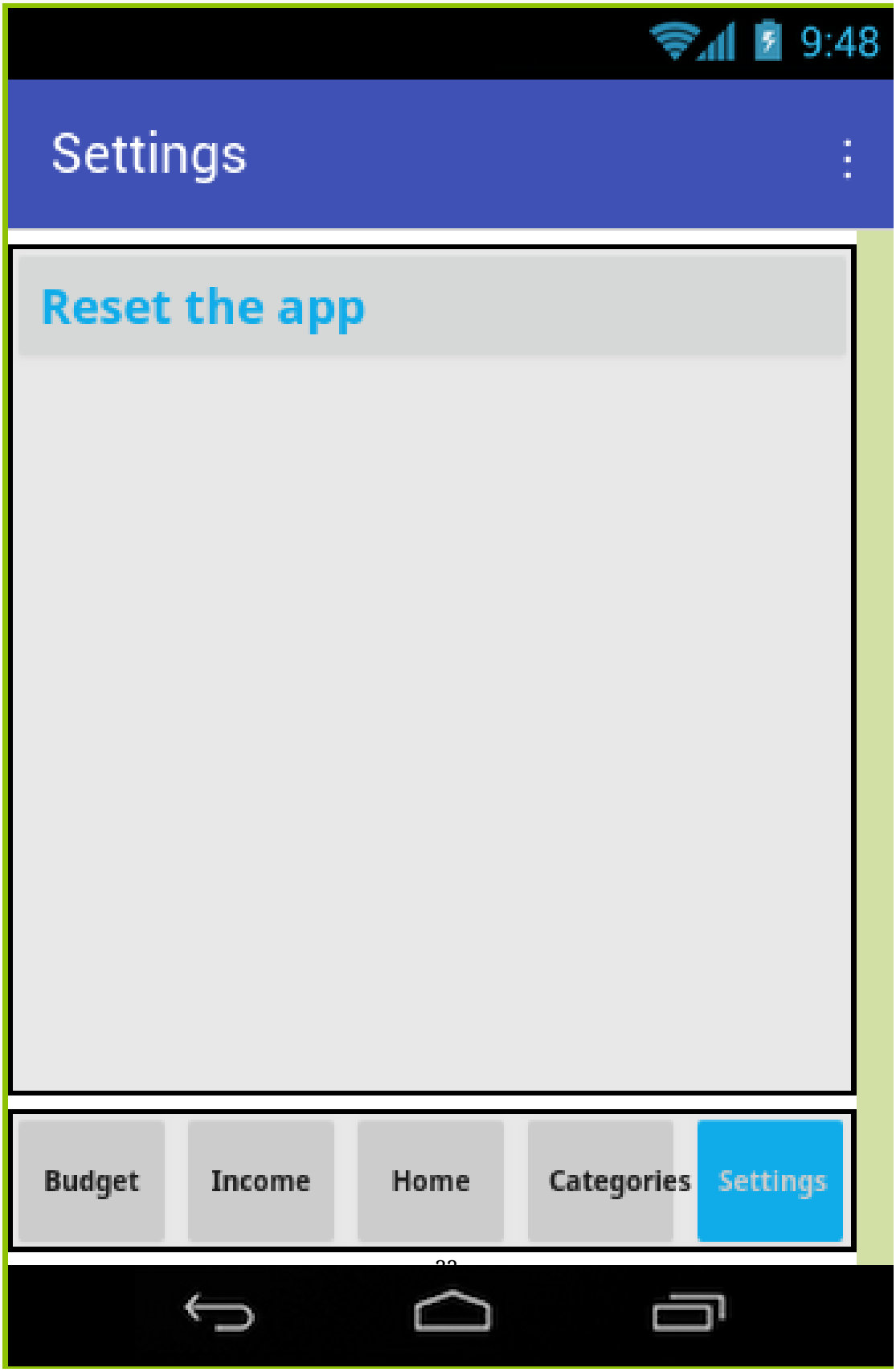


Figure 5.17: Settings View



Section 6

Data Storage Design

After designing the User Interface for the application, the next thing that I needed to figure out was how to store all the data for the application.

I knew that since the application was designed with the intent of only being used in one particular device at one time that the persistent data was to be stored locally.

After watching a video on YouTube from Jones (n.d.), I learned that the only way to store the data locally was through the usage of an App Inventor element called TinyDB.

The way TinyDB stores data is through the usage of tag-value pairs, meaning that the data stored works in a very similar way as the one used for programming normal variables, where the tag would be the equivalent for the name of a variable and the value would hold the value that we want to store.

Since TinyDB wouldn't store a full database system, I had to figure out a way to store my data that would still be able to hold indexes so that all the different expenses and categories could be differenciated.

The way I tackled this particular issue was through the usage of a code in the way my tags would be named, and through the usage of the split and join text functions to retrieve indexes and the different values.

I structured my tags in the following way for data that needed to be indexed:

- The first three letters would tell if the data was related to an expense (Exp) or category (Cat)
- The following section of the tag would hold the particular name of the data being held (Name, Date, Amount, Budget, etc.)
- The last section of the data being held would be a numerical value that would correspond to an incremental index

Implementing this method to tag my data, if I wanted to store the name of my first category, which is the Housing category, it would be stored with the following tag: CatName1. Since all the categories are established during the first launch of the application and no categories can be deleted or added to the system, there was no need to keep track of the last index for that data.

However, considering that expenses can be added indefinitely to the system, there was a need to keep track of the last index used for the expenses. In order to do that, I added another tag-value pair to the TinyDB element of the system that would hold the last index used when creating an expense. That value is stored under the tag ExpensesIndex.

Other data that I held in the TinyDB, along with their tags, are the following:

- The total expenses budget for the month: Budget
- The monthly income: Income
- The day of the month that the user gets paid: PayDay
- the total amount of savings made: Savings
- The total amount of expenses for the month: TotalExpenses
- The daily spend allowance: DailySpend
- The amount of expenses made during the day: DailyExpenses

Section 7

Other Design Decisions

In order to not only facilitate the programming of the application but also to make calculations and the amount of processing easier, the following decisions had to be made:

- Assume that all months are 30 days long for the calculation of the daily spend allowance
- Store the pay day as a number instead of a date and compare it to the current day through formatting the current date and time instant to only display the day in a single number

Part C

Implementation

Section 8

Implementation Process

When I started adding back-end processes to my user interface, I organized myself in order to get the easier and most repetitive tasks out of the way in order to be able to allocate more time for more important and complicated tasks.

The way I ended up doing my implementation was in the following way:

1. Program the elements related to the UI and display of screens (links between screens through buttons)
2. Storage of the data
3. Storage of data after modification
4. Retrieval of data to populate the UI elements for the screens that didn't require the passing of parameters
5. Retrieval of data to populate the UI elements of screens that require the passing of parameters

I started with the programming of the buttons that created the links between screens since those were the easiest to program, since all they required was a button click handler and a control function to open another screen of a particular name.

Then I implemented the storage of data, starting with the introduction page data, that way I had my database already designed and applied for the most part across the system. In order to do that I only needed to use some button click event handlers, the store value function for TinyDB and use the values entered in the TextBox elements of the form to be stored.

After that, I took care of storing the data after it had been modified, taking in particular care if said data would influence other data, such as for example, setting another budget for expenses, which would then influence the savings that could be made as well as the daily budget.

Afterwards, I started populating the UI elements of screens, and I started with the ones that weren't index specific, such as the home screen the and screen containing the list of all categories. For that I had to use for loops to iterate through all the tags of the TinyDB until I found the ones containing the names variables for the type of data that I was looking for (I could've also used other variables such as amount or the description), isolate the index of the data thorough string spliting and then retrieve the other values by joining the name of the variable and the index together to form the tag of the piece of that that I was looking for.

The last element that I tackled was the implementation of screens that required data to be sent to them thorough the previous screen, such as the screens that display the information of an expense, of a category, or the one that shows the expenses that belong to a certain category. For those, along with using a list to contain all the elements to populate the ListView of the UI for the previous page, I also had to populate a list with the indexes of the elements that were being populated. When the user would press an element in the ListView, I would then use a control block to open the new screen with a start value, and the start value would be the element of the indexes' list with an index correspondant to the one of the ListView element that was pressed.

In the screen that was being populated with a data of a certain index, I would then store the index in a global variable, and use it to retrieve the values that I was looking for by joining the name of the variable to the index to create the appropriate tag.

Part D

Testing

Section 9

Testing Process

Most of the application was tested as features were being developed. As soon as one feature would be implemented, such as the links between the different screens, I would connect either the emulator or my physical device to the development environment and test the feature to make sure of the following points:

- The application would behave in a predictable manner
- The UI transitions and data displayed were correct
- No errors would be thrown
- The calculations were correct

Whenever there would be errors, I would analyse them and fix them before moving to the next feature.

Part E

Deployment

Section 10

Deployment Process

During development, the application would be deployed to both the emulator and the physical device used for testing by the means of the aiStarter application that is to be used with App Inventor. When testing features, and especially during the transition between screens, I noticed that the transitions would take a long time to process, and I was afraid that the problem would persist when the application would be installed from an .apk file.

When I deemed the application ready to be deployed, I decided to store it in a public via link folder in my Google Drive, since I didn't have the intent of making it available to the general public and only wanted it to be available for presenting and for evaluation.

The application can be found in the following folder: Google Drive Folder

Also, after getting the application installed on my device through the .apk file, I noticed that when using it the problem of transitions taking a long time had disappeared, which for me was a relief.

Part F

Conclusion

Section 11

Errors Incurred

Throughout the development of this application, I came over the following problems and errors:

- Limited data storage possibilities
- Limited UI possibilities
  - Label items weren't clickable
  - Impossibility to dinamically populate screens
- Incorrect display of data
  - Impossibility to format the display of data inside ListView to get it to span throughout the whole element's horizontal space
- Issues with passing of parameters
  - Parameters not being registered correctly blocking the population of data

Whilst some problems had an easy fix, like the solution for storing the data or customizing button elements to resemble the layout with label elements, some other issues couldn't be fixed on time for the deadlines or are too complicated to be resolved with App Inventor, such as the impossibility to dinamically populate screens, or the impossibility to format the data inside a ListView element so that it occupies all the available horizontal space for that index.

Section 12

Possible Further Developments

There are many features that were left out of the application that can possibly be implemented for a next increment of the application:

- Remind users to enter their expenses through notifications
- Tell users how much they have saved through notifications
- Set an initial automatic budget based on income
- Allow the modification of entered expenses
- Alert users when daily budget is about to be reached
- Allow separation of expenses that count towards daily expenses or not
- Allow users to enter their income in the same way they get paid
- Allow the users to search for specific expenses by date or throughout a range of dates
- Calculate the daily spend allowance based on the number of days of the month
- Allow users to change the currency used in the application

Besides the implementation of these features, there is also a list of things that could be done to improve the application without adding new features:

- Re-design the application to give it a more modern look
- Implement the application in Android Studio
- Improve the data storage solution and structure
- Implement the dynamic population of screens with UI elements

- Implement data input verification

Due to time constraints, there were also some errors and small details that were left in the application. Those would also need fixing if the application were to have an increment.

Section 13

Learning Outcomes

Throughout the development of the application and the project, I learned the following things, both related to mobile applications development or as transferable skills:

- Have a better understanding of the limitations of mobile applciations development
- Have a better grasp of the visual design sacrifices that need to be made in order to provide a better user experience
- Learn and apply project management techniques
- Prioritise tasks and adapt goals to meet deadlines

I also noted some mistakes that I made throughout the development of the project that I will try to avoid next time I am faced with such an imposing project:

- Start working earlier on
- Give myself more time to complete tasks

Overall, I enjoyed doing this project and think that I learned a lot whilst doing it, especially considering it was my first time using App Inventor to build an Android application and that even though I had studied project management techniques I had never applied them to a project that I actually had to do and work on.

Part G

Appendices

Section 14

Learning Resources

In order to produce the application, I used the following resources to learn all the information that I needed:

- Sprint Planning: Scrum (n.d.)
- UK Household Average Budget: NimbleFins (n.d.)
- Video on YouTube about TinyDB: Jones (n.d.)
- App Inventor Documentation: of Technology (n.d.)

Section 15

Application and Code Download

As said in the deployment section, the app can be found in a public via link Google Drive folder. The same applies to the App Inventor project file.

References

(n.d.).

Jones, B. (n.d.), ‘Saving into tinydb, retrieve and show items in a listview’.
URL: *https://www.youtube.com/watch?v=opqVmpDv6LQ*

NimbleFins (n.d.), ‘Average uk household budget’.
URL: *https://www.nimblefins.co.uk/average-uk-household-budget*

of Technology, M. I. (n.d.), ‘The mit app inventor library: Documentation and support’.
URL: *http://appinventor.mit.edu/explore/library.html*

Scrum (n.d.), ‘What is sprint planning?’.
URL: *https://www.scrum.org/resources/what-is-sprint-planning*