

SP-26 PURPLE BUDGET APP

THE FINAL REPORT

Course: Senior Project Section 01 CS 4850

Semester: Spring 2024

Professor Perry

Date: 4/6/2024

Team Members:

Ann Nguyen

Cameron Lowry

Alexandra Clark

Number of lines in the project: about 2000 lines

Number of components in the project: about 45-50 components

Website: <https://sp-26-purple-budget-app.github.io/>

Github: <https://github.com/SP-26-PURPLE-BUDGET-APP>

Table of Contents

<i>I. Project Overview</i>	3
1. Objective:	3
2. Scope:	3
3. Constraints	3
<i>II. Project Artifact</i>	4
1. Project Schedule: Gantt Chart	4
2. Requirement	4
a. Functional Requirement	4
b. Non-Functional Requirement	5
<i>III. Design and Architecture</i>	6
1. System architecture Diagram	6
2. Detailed system Design	7
3. Screen Mockup	10
4. Tech Specification	10
<i>IV. Version Control</i>	11
<i>V. Narrative Discussion</i>	11
<i>VI. Challenges, Assumption, and Risk Assessments</i>	13
<i>VII. Test Plan and Test Report</i>	13
1. Testing Overview	13
2. Testing Objective	13
3. Testing Strategy	14
<i>VIII. Conclusion</i>	14

I. Project Overview

MoneyFlow is a simple Budget App aims to develop a user-friendly and efficient application to help individual to manage their finances effectively. With an increasing demand in digital solution for personal finance, this app strives to provide users with a convenient platform to track their incomes, expenses and savings monthly. Through intuitive interfaces and robust features, the budget app empowers users to make inform decisions, set goals, and achieve their financial goals. Whether it's budget planning, expenses tracking or analyzing spending patterns, this app endeavors to simplify the process and promote financial well-being for its users.

1. Objective:

- Provide users with a user-friendly tool for managing finances.
- Develop a platform allowing users to record and categorize their monthly income and expenses.
- Prioritize the security and privacy to ensure the confidentiality and integrity of user's financial data.
- Doing various testing to ensure that the apps meets its functional requirements and perform as expected.

2. Scope:

- User registration and authentication
- Welcome Screen provide a welcoming interface for users upon login.
- The Main Dashboard shows the line chart showing the relationship between income and expenses and the total amount of cash and bank account balance.
- Insight Page includes two simple pie charts categorizing expenses and income to provide user with insight into their financial habits.
- Transaction Page: all the history of user's transaction
- Account Overview: link user bank account, change password or help and support center.

3. Constraints

- Technical Challenges: limited expertise in certain technologies, compatibility issues between different platforms or integration complexities with external services.
- Time Constraints: deadlines and project timelines may impose limitation on the scope of work, necessitating prioritization of features to meet deliverables on time.
- Platform limitation: Limitation or restriction imposed by the chosen platforms may affect the design.

II. Project Artifact

1. Project Schedule: Gantt Chart

Here is the progress tracking document for the MoneyFlow project. It provides the overview of the project deliverables, tasks, completion status, assigned team members and milestones.

[illegible]

2. Requirement

a. Functional Requirement

- Login and Password Authentication:
 - Users should be able to create an account with our app. Each user should have a unique username and password.
 - The app must verify credentials provided during login to ensure security.
 - Error handling and logging: the app should provide informative error messages to users in case of authentication failures (incorrect password, account locked, etc.)
- Display Main Dashboard:
 - Upon successful authentication, the app should display the homepage/dashboard providing an overview of the user's financial status.
 - A line graph showing the relationship between income and expenses over a period.
 - A total balance that includes the cash and bank account amount.
- Transaction Screen:
 - A list of transactions that users have made or spent date by date.
 - Users should be able to add expenses or source of income through a button.
 - The App should be able to edit any expense currently on the balance sheet.
- Display Insight Page:

- Expenses categories breakdown: a pie chart showing the distribution of expenses across different categories (groceries, transportation, entertainment, etc.)
- Different sources of income (job, investment, or saving) that showing in a pie chart. Let the user know where their money coming from.
- User Account Management Page:
 - Users should have the ability to access and manage their account settings, including profile information, security setting, and linked bank account inside it.
 - The app also should allow users to update their account information such as email address.
 - Users also should be able to change the passwords and set the notification.
- Navigate to Linking bank account:
 - Users should be able to link their bank accounts securely to the app.
 - The app will use a plaid website to link bank account securely.
 - The process of linking bank accounts should be friendly, with clear instruction and guide the user through necessary steps.

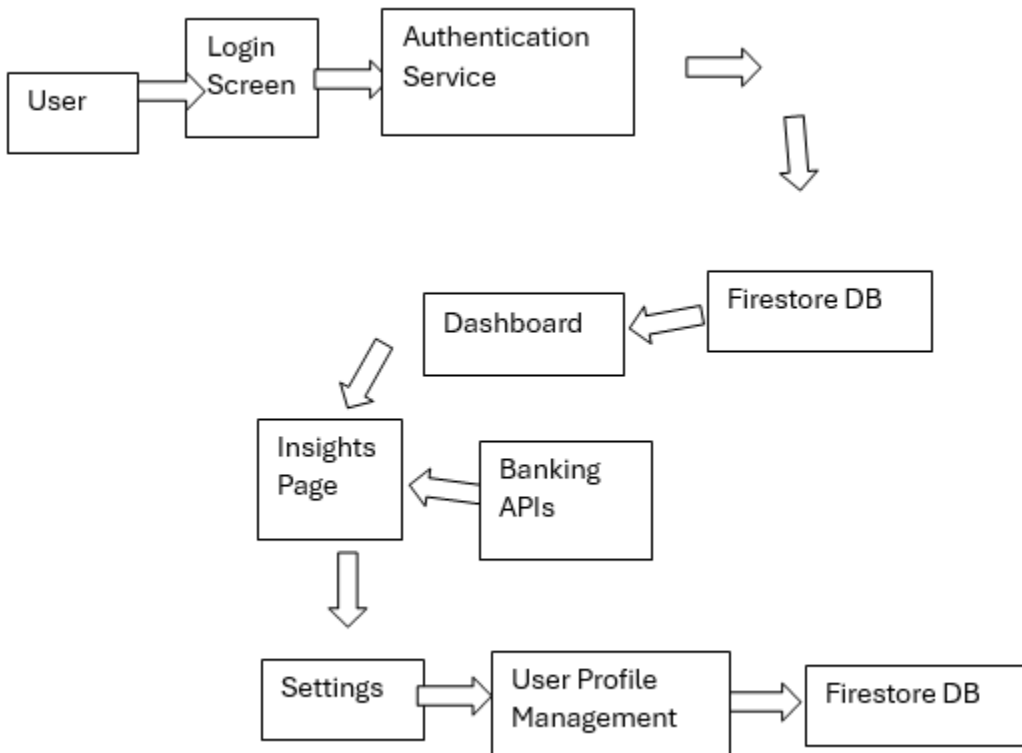
b. Non-Functional Requirement

- Security
 - Failed login must be tracked and restricted after a certain number of unsuccessful tries for security process.
 - All sensitive user data, including passwords, financial transactions, and personal information should be encrypted both in transit and at rest to prevent unauthorized access.
 - Implement strong authentication to ensure that only authorized users can access the app.
 - Keep a detailed record of everything users do in the app, then if something goes wrong, it can undo the action and find out what happened and who did it.
- Capacity
 - Performance: The app should be able to handle large volumes of simultaneous users and transactions without slowdowns or downtime.
 - Scalability: make sure the app's design can handle more users and data as it grows without slowing or breaking.
 - Load testing: test the app regularly to find any problems that could slow it down when lots of people are using it and fix them so the app can stay fast and reliable even during busy times.
- Usability
 - Consistency: The User interface should be consistent with Itself. This includes front sizes for titles, headers, and body texts. The app also should have a consistent color palette across all pages.

- Error Handling: Notify the user what went wrong if the app crashes or an error is found. Create a bug reporting section to allow users to notify the developers if there is a bug.
- Other
 - The app must be confidential, integrated, and availability.
 - Design the app to be robust and resilient, capable of recovering from failure and error without any data loss or service disruption.
 - Structure the app codebase in a modular and well-organized manner, with a clear separation of components, easy to maintain and enhance in the future.
 - Ensure the app can integrate and communicate with other systems, services, or platforms.

III. Design and Architecture

1. System architecture Diagram



- a. User Interface (UI)
 - The user interface is front end component of the budget app that users interact with
 - It includes screens for welcoming, logging in, viewing financial data, transaction history, adding transaction, and categorizing the expenses and income.

- The UI provides a responsive and intuitive interface for users to manage their finances effectively.
- b. Backend Server
 - The backend will be the central processing unit of the budget app system.
 - It handles business logic, data processing, and communication with other component.
 - The backend server validate user inputs, executes transactions, and retrieves data from database.
- c. Database (FireStore)
 - Stores all the financial data and user information required by the budget app
 - It includes tables for storing user profile (name, password and email, etc.), transaction history, budget categories and other relevant data.
 - The database ensures data integrity, security, and efficient retrieval of information.
 - FireStore provides real time synchronization and offline support, allowing users to access their financial data from anywhere, even if they're offline.
- d. Authentication service (Firebase)
 - Handle user authentication and authorization processes.
 - It verifies user credentials during login, generates authentication tokens and manages user sessions.
 - The authentication ensures secure access to the budget app and protects user accounts from unauthorized access.

2. Detailed system Design

- a. Design approach.
 - The system is divided into two subsystems: budget management and user management subsystem. Each subsystem is responsible for handling the specific aspect of the application functionality.
 - The overall system focuses on a user-friendly interface, with intuitive navigation and clear visualization of financial data. The app design also emphasizes security to protect user database and profile.
 - The system is structured to allow scalability and flexibility. New features can be added easily, and an existing feature can be modified.
 - Overall, the app is designed to provide users with comprehensive tools for managing their finances effectively, with a focus on security, usability and scalability.
- b. Classification
 - Subsystem
 - Budget Management subsystem
 - User management subsystem
 - Modules
 - User authentication module
 - Income module

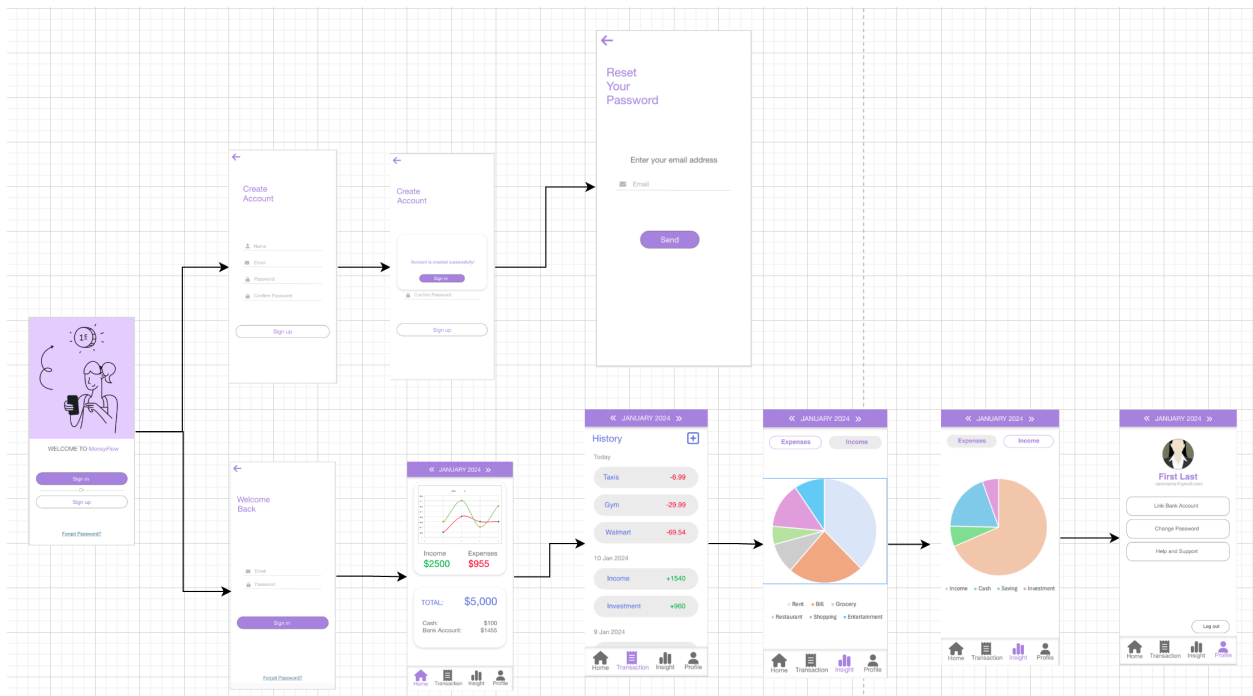
- Expenses module
- Pie chart generation module
- Line chart generation module
- Transaction History module
- Class
 - User class
 - Manager class
 - Transaction class
 - Budget class
 - Chart class
 - Expenses Category class
 - Income Category class
- Function/ method
 - calculateTotalIncome()
 - calculateExpenses()
 - categorizeExpenses()
 - categorizeIncome
 - addExpenses()
 - addIncome()
 - generatePieChart()
 - generateLineChart()
 - fetchTransactionHistory()
- Package
 - Budget_Planning
 - User_authentication
 - Chart_Generation
 - Transaction_History

c. Responsibilities

- Budget Management Subsystem:
 - Assist users in setting up budgets for different expense categories.
 - Provide tools for monitoring budget adherence and analyzing spending patterns.
 - Generate reports and visualizations to help users gain insights into their financial habits and make informed decisions.
 - Handle interactions with external financial APIs or services.
- User Management Subsystem:
 - Enable users to register accounts securely within the application.
 - Authenticate users during login to ensure access control and security.

- Manage user profiles, allowing users to update personal information and preferences.
 - Implement access control mechanisms to restrict unauthorized access to sensitive functionalities and data.
 - Handle password management tasks such as reset and recovery.
- Income tracking module:
 - Record the various income for user.
 - Categorized the income sources by using the pie chart for simplicity
 - Allow users to input income or integrate with external sources like bank accounts.
 - Calculate total income over time.
 - Expenses tracking module:
 - Categorize different types of expenses.
 - Enable user to input expenses manually.
 - Tracking spending habit and pattern over time and visualize their spending pattern by using the pie chart
 - User class
 - Representing user data: some user essential information: username, password, email.
 - Have access control to certain part of the application.
 - Provide functionalities for password reset, recovery, and ensuring the password is secure.
 - Allow users to update their information.
 - Transaction class
 - Representing individual financial transaction: amount, date, category, etc.
 - Categorization and tagging provide methods to create new transactions, updating existing transactions.

3. Screen Mockup



4. Tech Specification

- **Programming Language and Framework:** JavaScript with React Native was chosen because of its cross-platform mobile application development.
- **User Interface:** A mobile app interface using React Native.
- **Data Management:** Firestore is used for the database.
- **Communication:** The application includes API integration for backend communication and banking data synchronization
- **Debugging:** Uses react native's debugging tools and libraries to identify and fix the issue during development.

IV. Version Control

We used Git as our version control system, with the repository hosted on GitHub. This setup allowed for a way to manage the project's development lifecycle.

- Branch Strategy: We used a streamlined branch strategy to keep our development process organized and manageable. This approach allowed us to focus on developing features without the overhead of managing multiple branches.
- Commit Protocol: The commit process was kept straightforward, allowing us to maintain a steady pace of development.

V. Narrative Discussion

Over this Spring Semester we have been designing and implementing our Senior project. For this project we decided we would build a budgeting app by the name of MoneyFlow. This app is designed to help users manage their income by tacking daily expenses, categorizing them and providing insight. This will allow users to make better financial decisions and avoid overspending. The app was designed using JavaScript through React Native and integrates several APIs for backend communication with a database and with banks.

As with all projects MoneyFlow began simply as an idea for a simple budgeting app that would allow users to keep better track of their spending. This came with our first project goals which included expense tracking, visualization of expenses and privacy. To accomplish these goals, we understood that there would be a variety of different constraints we would have to deal with. One of the biggest constraints we had to deal with was budget limitations as we had no funding for this project. Budget limitations affected a large part of our project as we had to use free resources in order to build MoneyFlow. Many of our design choices such as using react native for our implementation and Firestore database would stem from this limitation. Another large limitation we would face would be the time constraint that all senior projects face. This means we would only have one semester to design and implement this app, meaning some features would have to be not fully realized or placed on the back burner. Other smaller constraints that we had to deal with were that the app would have to run on both IOS and Android and that the app would need to be connected to the internet for some of its features to work. During this planning phase we also had to figure out who our target audience was. After some discussion, we decided to target people between the ages 18-35 with stable internet access and a cellphone. This allowed us to make some assumptions about who these people are such as that they had a moderate proficiency with their device and that their financial literacy could vary

heavily. This second assumption would lead us to provide clear explanations in the app to help those with low financial literacy better understand the app.

Based on our goals, assumptions and restraints we decided on some fundamental requirements that we would have to meet. The first of these was a login screen with username and password functionality. This would both protect the user's information with a password and allow for each individual user to have their own account. In the design we wanted to make it quick and easy for users to see their data, so we also decided to add a Homepage that would provide an overview of the user's information. In addition to this we had the idea of a page that would show more information such as the relationship between income and expenses. Also, we had the idea of a page that would show more information like the relationship between income and expenses. For MoneyFlow it was also required that a user would be able to easily input and edit their expenses and income. For this we designed a balance sheet that would allow the user to see their total budget and input new expenses as they are made. We also had some non-functional requirements that we be shooting for but did not know if we would be able to include. We split these non-functional requirements into four distinct categories, Security, Capacity, Usability and Others. The security category held all the ideas we wanted to add to make the app more secure for users but were not entirely necessary for the app to function. Capacity spanned issues such as performance and scalability across the app when it was experiencing heavy traffic. Usability was a category we put ideas into that would help users more easily navigate the app and understand their financial information. Finally, the other category held ideas that did not necessarily fit in with the other categories such as improvements to code readability and app communication.

To implement these ideas, we decided to use several different programs and APIs. The first program we decided on using was React Native. As we were discussing different programming languages and programs to help us build MoneyFlow the idea of React Native was thrown out. React Native had a lot of things going for it such as being free and providing the framework for app integration for both IOS and Android. One of our group members also had prior experience using this program so we decided to use this to build our app. When first using React we made a few mistakes as we were learning how to use the app better. However, these were overcome quickly and helped us get to implementing our design. Later into development as we were creating the database we had to agree upon which database system to use. We knew that building our own database would simply take too long and more effort for less results than using a premade cloud database. After some discussion within our group and with other project groups we decided it would be best to use Firebase. Firebase has access to Firestore which is a NoSQL cloud database built by google that is used for web, mobile and server development. Using Firebase and Firestore we were able to design a database to hold all the user's information. This also came with its own data transfer security, taking a lot of work off our shoulders.

Once we decided on the programs we would use, we began to design the UI and implement a test version. One of the first things that we did was build a mock-up of the UI that would later be added into MoneyFlow. This mockup was made using various images pasted together to make a “fake image” of what our UI would look like. At first this design included only a few pictures and certain icons did not quite fit the rest of the aesthetic. Over about a week and a half we went through several iterations of this UI with icons being changed and distinct color schemes applied. During this time, we also discussed which font would be best to use in this app. We wanted one that would be easy to read and look sleek and professional. Once the UI was designed, we added functionality by implementing it inside of React Native. Using React Native we added buttons and swipe features to allow for easy navigation of the app. Finally, we implemented the database using Firebase. This took some time to get working as none of us has ever used firebase or designed a database. After creating each of the core features, we went through testing each section. For testing we tested the app by running through it ourselves as we do not have the funding to hire a team of testers to thoroughly test every part of the application.

VI. Challenges, Assumption, and Risk Assessments

- During the project, one significant obstacle we faced was integrating the banking data synchronization. We encountered difficulties in accessing and securely managing user’s financial data from different banking situations.
- The potential risk we identified early in the project was the dependency on the third-party PLAID for linking/ connecting user’s bank account to our app.

VII. Test Plan and Test Report

1. Testing Overview

- Test plan is a comprehensive document outlining the approach, objective and scope of testing activities.
- Ensure that the software meets quality standards and functional requirements before deployment to the user.

2. Testing Objective

- Verify the functionality, reliability and performance of the budget app across various scenarios.
- Other objectives include validating user experience, ensuring data integrity and security.

3. Testing Strategy

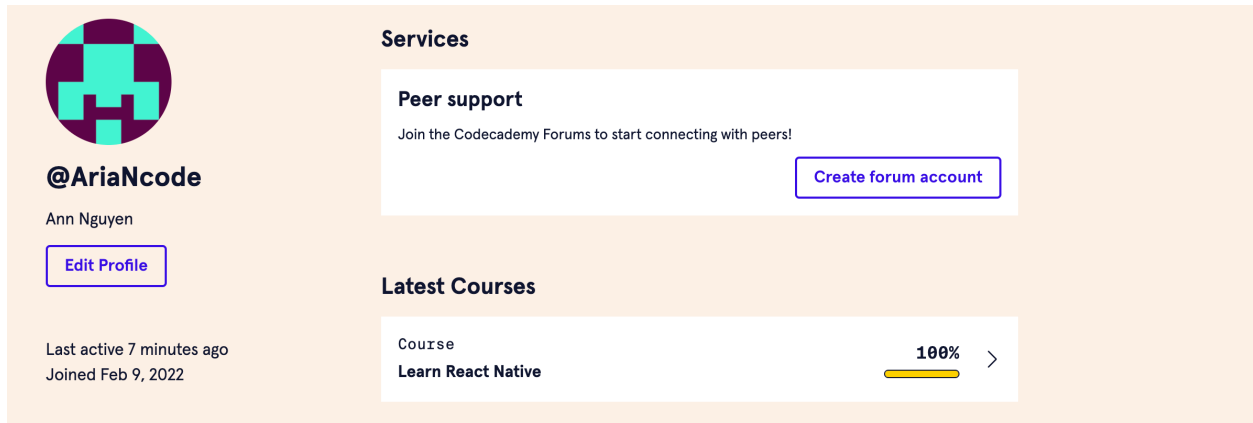
- Local Testing: All new features and changes were tested locally by the coder to ensure that they worked correctly before being committed to the development branch.
- Unit Testing: Tested the individual components to make sure there were no issues.
- Integration Testing: Ensured that different parts of the app worked together without errors.
- UI Testing: Test user interface to ensure it functions correctly.
- End-to-End Testing: test entire application flow from start to finish.
- Performance Testing: Test the app's performance under various conditions.
- Security Testing: Test for security vulnerabilities to ensure user data is protected.

VIII. Conclusion

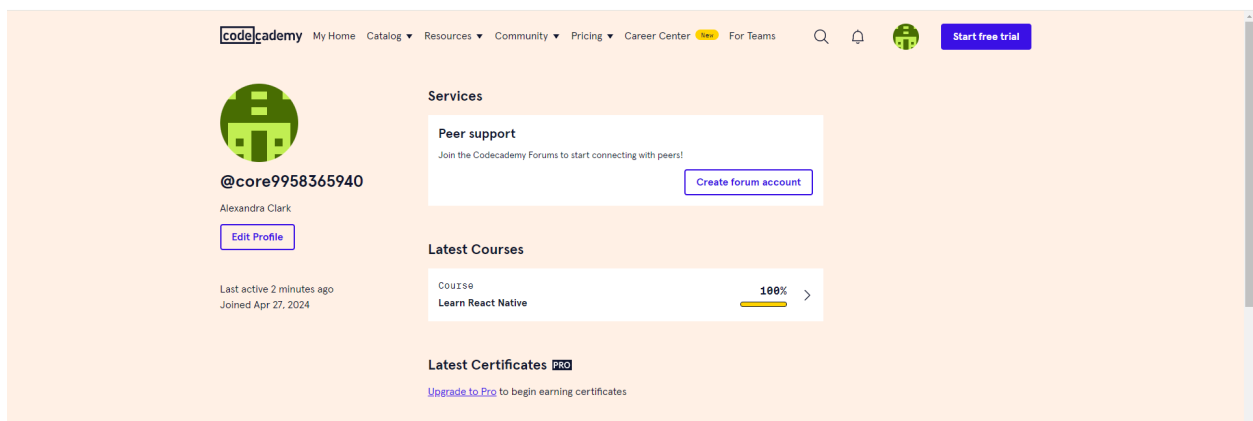
In conclusion, the development of the MoneyFlow app during our senior project has been a rewarding experience. We successfully created a functional and secure budgeting tool that addresses the financial management needs of young adults. Throughout this project, we improved our understanding of project management and user-centered design and our technical skills in React Native and API integration. This project has given us practical experience and insights into the complexities of building real-world applications.

Appendix

Appendix A: TRAINING



This screenshot shows a user profile on the Codecademy platform. The profile is for a user named Ann Nguyen, with the handle @AriaNcode. The profile picture is a circular avatar with a purple and teal pixelated design. Below the name, it says "Last active 7 minutes ago" and "Joined Feb 9, 2022". There is a button labeled "Edit Profile". To the right of the profile, under the heading "Services", there is a section for "Peer support" with the text "Join the Codecademy Forums to start connecting with peers!" and a button "Create forum account". Below this, under the heading "Latest Courses", there is a course titled "Learn React Native" with a progress bar showing "100%" and a right arrow.



This screenshot shows a user profile on the Codecademy platform. The profile is for a user named Alexandra Clark, with the handle @core9958365940. The profile picture is a circular avatar with a green and yellow pixelated design. Below the name, it says "Last active 2 minutes ago" and "Joined Apr 27, 2024". There is a button labeled "Edit Profile". At the top of the page, there is a navigation bar with links: "codecademy", "My Home", "Catalog", "Resources", "Community", "Pricing", "Career Center", "For Teams", a search icon, a bell icon, a profile icon, and a "Start free trial" button. To the right of the profile, under the heading "Services", there is a section for "Peer support" with the text "Join the Codecademy Forums to start connecting with peers!" and a button "Create forum account". Below this, under the heading "Latest Courses", there is a course titled "Learn React Native" with a progress bar showing "100%" and a right arrow. At the bottom, under the heading "Latest Certificates", there is a "PRO" badge and a link "Upgrade to Pro to begin earning certificates".

Appendix B: PROJECT PLAN

Appendix C: SOFTWARE REQUIREMENTS SPECIFICATION

Appendix D: SOFTWARE DESIGN DOCUMENT