Team MSUFCU
Michigan State University
Achieve It Financial Education System
Project Plan
Spring 2019

MSUFCU Contacts
Samantha Amburgey
April Clobes
Austin Drouare
Collin Lochinski
Ben Maxim
Liam Petraska
Val Torrey

Team MSUFCU
Ben St. John
Michael Jajou
Rachel Hamilton
Benjamin Carroll

Contents

Executive Summary

With a member base numbering over 280,000 spanning the Michigan State University and Oakland University communities, MSUFCU is the largest university-based credit union in the world. MSUFCU offers a range of products and services including a variety of account types (savings, checking, IRA, etc.) as well as loan and insurance offerings. A leader in its market, MSUFCU recognizes the value of innovation and investing in its technology to provide superior products to members.

MSUFCU has an in-house software development team, numbering about 35 employees, that produces custom software. This provides the advantage of being able to build software to exactly address issues or wants of the company rather than trying to get vendor products to work with existing systems. MSUFCU seeks to apply these resources it has developed to provide more holistic financial services that enable members to maintain a healthy financial life.

Towards this goal, team MSUFCU is working with MSUFCU to produce the preliminary implementation of “Achieve It”; a financial education tool that enables children to learn about finances in an environment controlled by their guardian. This software creates a system in which users have access to information and experiences in a safe environment that is more engaging, rewarding, and meaningful than traditional methods of youth financial education. Such education includes but is not limited to understanding the value of saving, the value of money, accountability, loans and interest, good credit practices, and financial responsibility.

Functional Specifications

It cannot be assumed that the average consumer has had comprehensive personal finance education. Being aware of this, MSUFCU wishes to provide a highly accessible education experience platform targeted at children between the ages of 3 and 13. The goal is to provide educational material as well as low-risk, but otherwise realistic, financial experiences (e.g. account management, saving, loan management, working for money). “Achieve It” is a unified system of applications usable on smartphones, computers, and home assistants that aims to achieve this goal.

This solution can be broken up into three primary user demographics and their corresponding applications:

First, part of the solution is directed at parents. Parent accounts can monitor and interact with linked child accounts. Parent accounts have the capability of awarding money via MSUFCU account transfers as well as providing loans with optional interest. Tasks and learning modules can be managed and assigned by the parent with the option of a monetary award upon completion. The parent can approve that the youth has completed the tasks to satisfaction before the transfer is made. Parents have access to a web app and mobile app to oversee and manage the above features; a Google Assistant interaction is also available to the parent for simple status updates on individual child progress.

The second user type are children between the ages of 3 and 13 that have an existing MSUFCU youth account. Youth accounts cannot exist independently of a parent account. Moreover, youth accounts cannot be accessed without the linked parent account’s approval which can be given on a case by case basis or carte blanche. The youth also have a simple pin for logging into their account. Learning modules are provided by MSUFCU at age levels to expose children to content appropriate for their age. The value of saving is the primary financial concept that MSUFCU would like to teach. Youth can set savings goals and can designate part of their bank balance toward these goals. Youth can request loans from their parents through the app with an interest rate set by the parents. These features aim to show the results that come from saving and borrowing to youth.

Third, the MSUFCU staff wishes to view statistical data online measuring the usage of the various apps such as tasks completed and goals met. The data is meant to provide analytics on specific features and content to help MSUFCU refine the solution over time. The data collected for analytics is entirely anonymous and contains no personally identifiable information.

Design Specifications

Overview

The MSUFCU “Achieve It” system contains a number of applications on a range of platforms. Therefore, it is important to keep a consistent user interface across all platforms for frictionless transitions across a user’s devices. There will be a single web application for the MSUFCU administration to monitor statistics and manage high-level features and content of the application; the designs for this backend administration portal will be fairly bare-bones and focus on functionality over experience. There will be a web application, iOS, and Android application that will contain both “child” and “parent” sections of the application containing slightly different functionality with similar interfaces. Finally, there will be a Google Assistant application which will contain no front-end user interface. With the exemption of the Credit Union admin dashboard, the user experience will be designed to provide intuitive navigation and interaction with the app. It is incredibly important for the app design to be simplistic due to the large age range that will be interacting with it.

Child View vs. Parent View

This application is essentially built for two types of users. The parent user and the child user. They both serve very different roles, but they come together in providing financial education in a productive, fun, and new way. All designs of the application will remain similar for all users, with minor adjustments. Listed below will be specific design decisions made to differentiate controls and functionality between the parent and child. All designs shown in this document are the base designs that both the parent and child interfaces will inherit from.

Mobile Application Designs

Login Screens

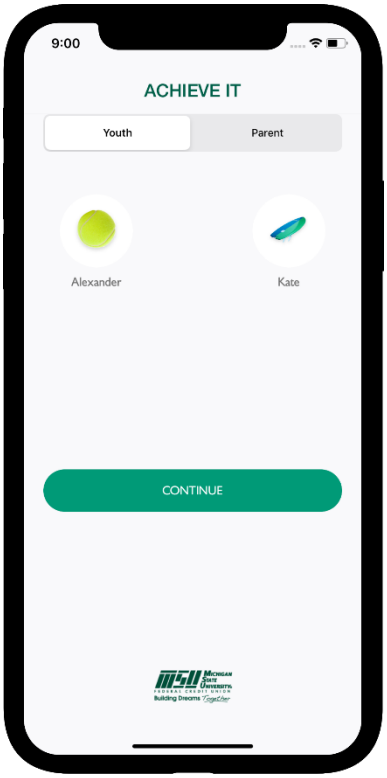

Figure 1: Youth login screen

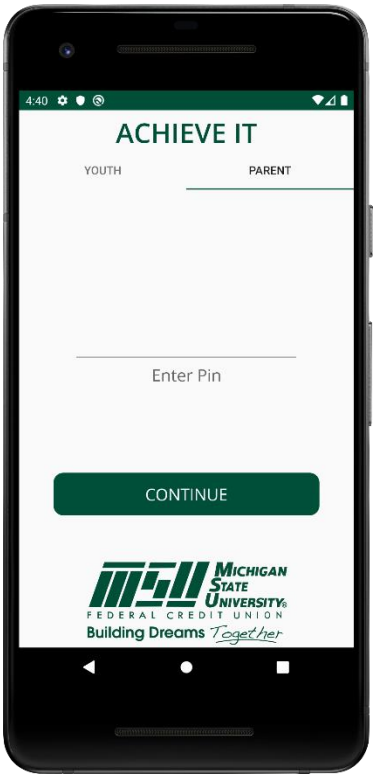

Figure 2: Parent login screen

Child vs. Parent:

On figure 1, the login page for the child, there will be a simple image associated with each child. We are going to add a very simple pin authentication for the child so that they can only access their own account. On the child account home screen, there will be a simplistic navigation panel to guide the child to their various options within the app.

On figure 2, the login page for the parent, they will have also had a pin authentication. The designs for the parent's home screen will be significantly different because they will have a dashboard that gives them access to manage each individual child's account, and the parent will also have a section to save tasks globally for all children.

The difference between the child login and the parent login is really what happens after the authentication process. After a child successfully logs in, they only have access to their data such as goals, tasks, loans, etc. The parent login acts as the admin for the children where the parent can add or monitor the children's data.

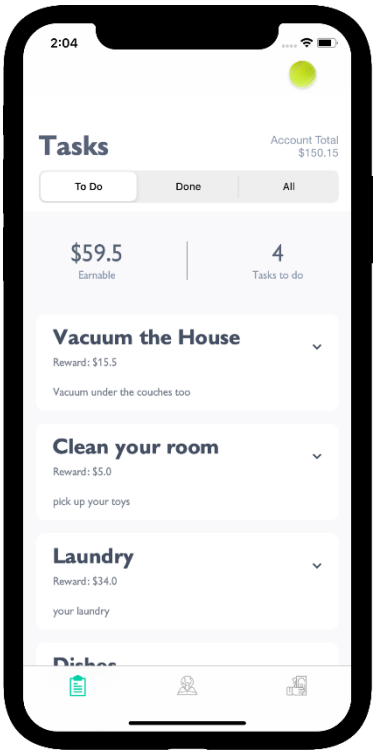

Figure 3: Youth tasks view

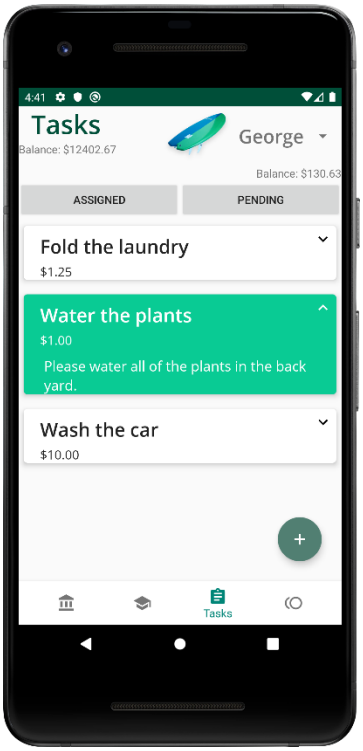

Figure 4: Parent tasks view

Child vs. Parent:

Figure 3, the design for the child view will only display simple task information and provide a way for the child to mark a task as complete. This child view will also provide a short summary of all earnable money through their current tasks.

Figure 4, the design for the parent view shows the parent the tasks currently assigned to a specific child as well as the tasks the child has marked as complete. Viewing of these two lists is determined by buttons near the top of the screen. Each of the task cards can expand to show more detail about the task. On the list of tasks marked as completed the cards will include a button for the parent to verify task completion and initiate the transfer of the reward to the child's account

The design of the task views for both parent and child are very similar. The key differences lie in their unique functionalities. The child view is fixed on one child account whereas the parent view can cycle through different child accounts through use of the drop-down selector in the top right. The parent has the capability of adding tasks and approving tasks, but the child can only mark assigned tasks as completed making them viewable to the parent for verification.

Savings Goals

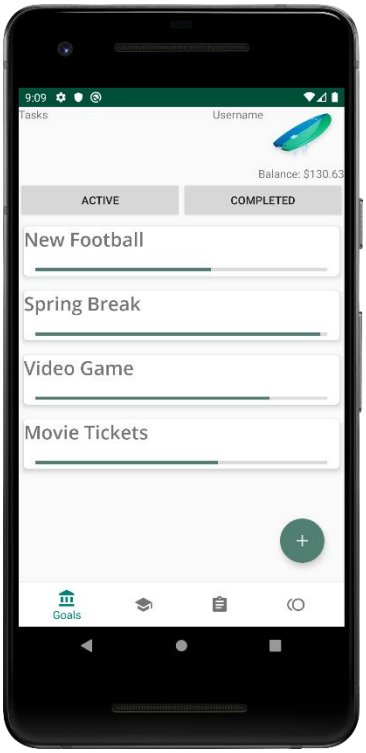

Figure 5: Youth goals view

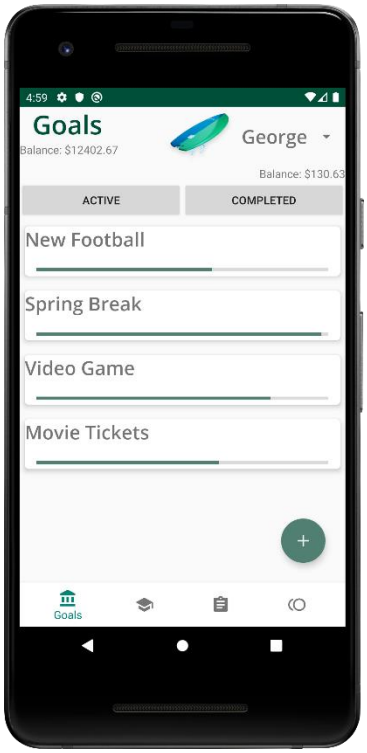

Figure 6: Parent goals view

Child vs. Parent:

Figure 5, the child view for the savings goals component will be a simple display of all current savings goals, which can be expanded for further detail, and a button to create a new savings goal. The expanded view for saving goals will show more details about a specific goal including a timeline of the savings and will include buttons to deposit money to the goal, edit, or delete it.

Figure 6, the parent view for the savings goals component shows lists of active and completed goals where the currently viewed list can be toggled by the control near the top. There are cards to represent the goals. These cards can be expanded for a summary of the goal. The parent can edit, delete, and add funds to goals through the cards. The parent can also make a goal for the child using the button in the bottom right. The parent view can stitch between specific children using the drop-down selector in the top right of the screen.

The parent view is very similar to the child view, except that the parent has much more control over the goals including the ability to edit and delete goals. The parent is also able to monitor the goals for all of the children under the parent account, where the child view only allows the logged in child to view their specific savings goal. The parent view will also give the user the ability to add funds to the savings goal which will help the child reach their goal faster without the money coming out of the child's account.

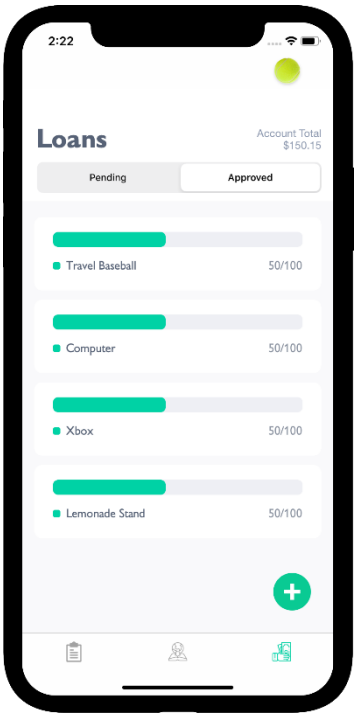

Figure 7: Youth loans view

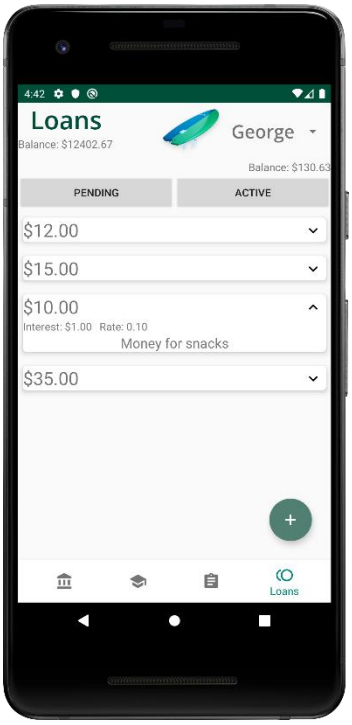

Figure 8: Parent loans view

Child vs. Parent:

Figure 7, the design for the child view loans shows the child lists of pending loans and approved loans. Viewing of these lists can be toggled using the control near the top of the screen. Each of the loans is represented as a card where a brief summary is viewable. On active loans the child will also have the option to make a payment on the loan. Loans administered to a child can have an interest rate that is applied once at the time the loan is dispensed. The child has the ability to make a request to the parent for a new loan. This request can be made by use of the button in the bottom right of the screen.

Figure 8, the design for the parent view shows the loan requests from a specific child awaiting the approval of the parent account and the current outstanding loans taken out by a specific child. The view of these lists can be toggled by the control near the top of the screen. The loans are represented by cards that can be expanded to show a summary. In the bottom right there is a button which allows the parent to create and administer a new loan to the specified child. The child is selected using the drop-down selector in the top right of the screen.

The design of the loans views for both parent and child are very similar. The key differences lie in their unique functionalities. The child view is fixed on one child account whereas the parent view can cycle through different child accounts through use of the drop-down selector in the top right. The parent has the capability of adding loans and approving loan requests whereas the child can make loan requests and work on loan repayment.

Learning Modules

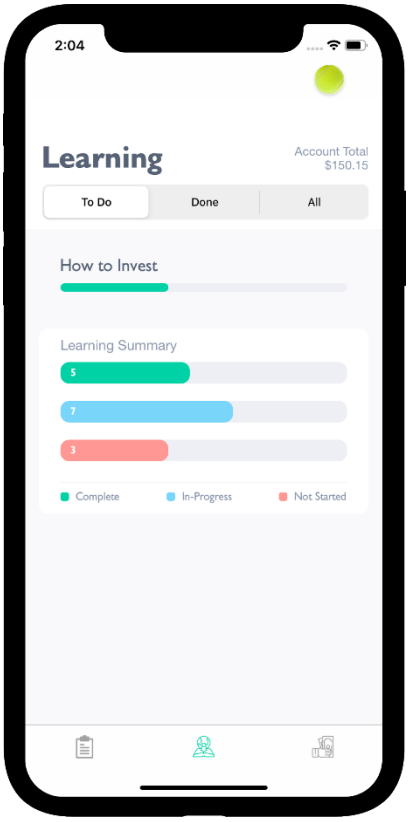

Figure 9: Youth learning view

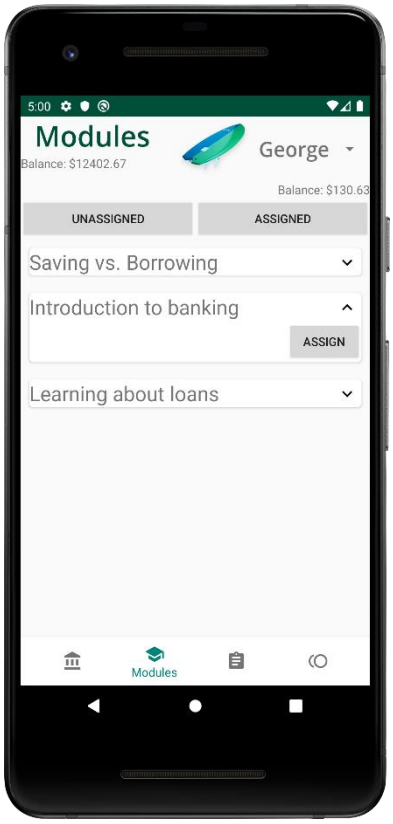

Figure 10: Parent learning view

Child vs. Parent:

Figure 9, the child view for the learning module component will have a simple display of all available learning modules, and an overview of the progress across all modules. These modules can be viewed as lists of cards that are toggleable from the control near the top of the screen. The cards show a summary of the module and an option to start the module if it has not been done before. Upon completion the module is moved to the list for completed modules.

Figure 10, the parent view for the learning module component will show the assigned and unassigned modules as cards. The viewing of these lists can be toggled by the control near the top of the screen. The parent can expend cards in the unassigned list to access the option to assign the module to the child currently selected in the drop-down selector found at the top right of the screen.

As with other components, the parent and child views for modules are visually similar but differ in capabilities. The parent can only view and assign modules where the child can view the modules and start them.

Web Application Designs

End-User Web Application

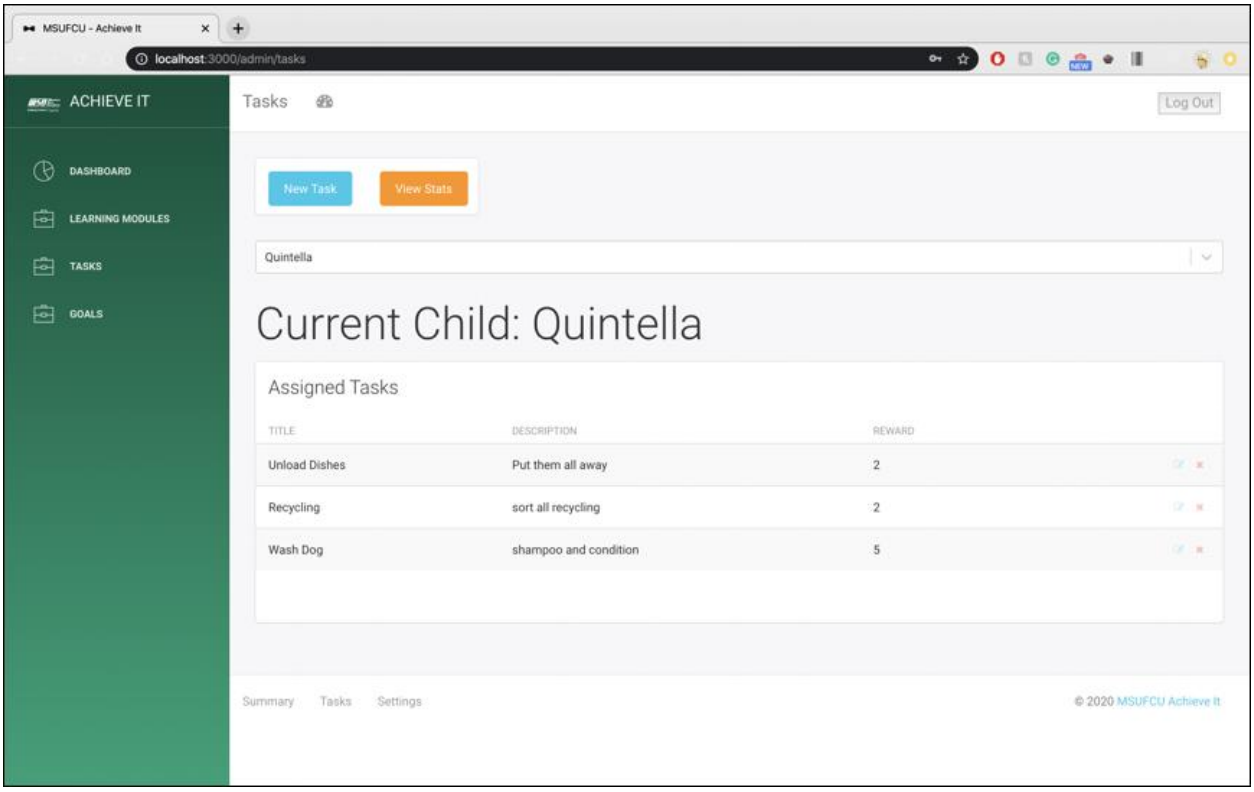

Figure 11: Example of web application page, Parent tasks view

Figure 11, the end-user web application designs will be very similar to their mobile application counterparts. There will only be cosmetic and minor interaction changes implemented on the web application to take into consideration the user's larger screen size, keyboard, and mouse/trackpad. Below is a sample design for the task management system within the web application. While there will be a "child" and "parent" view for this end of the application, they are still currently being implemented and the only difference will be in functionality, not necessarily design components.

Internal Web Application

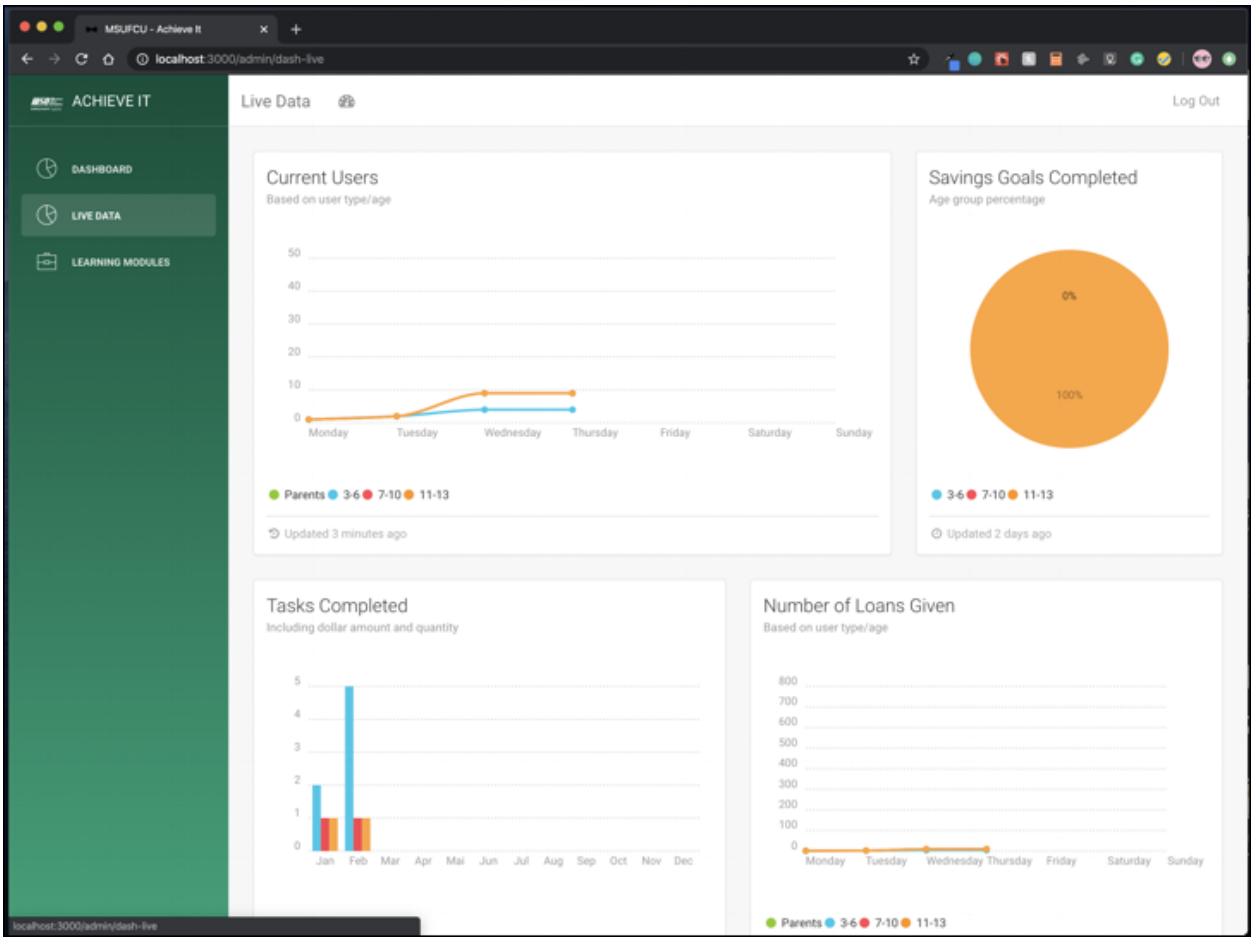

Figure 12: Internal web dashboard statistics page

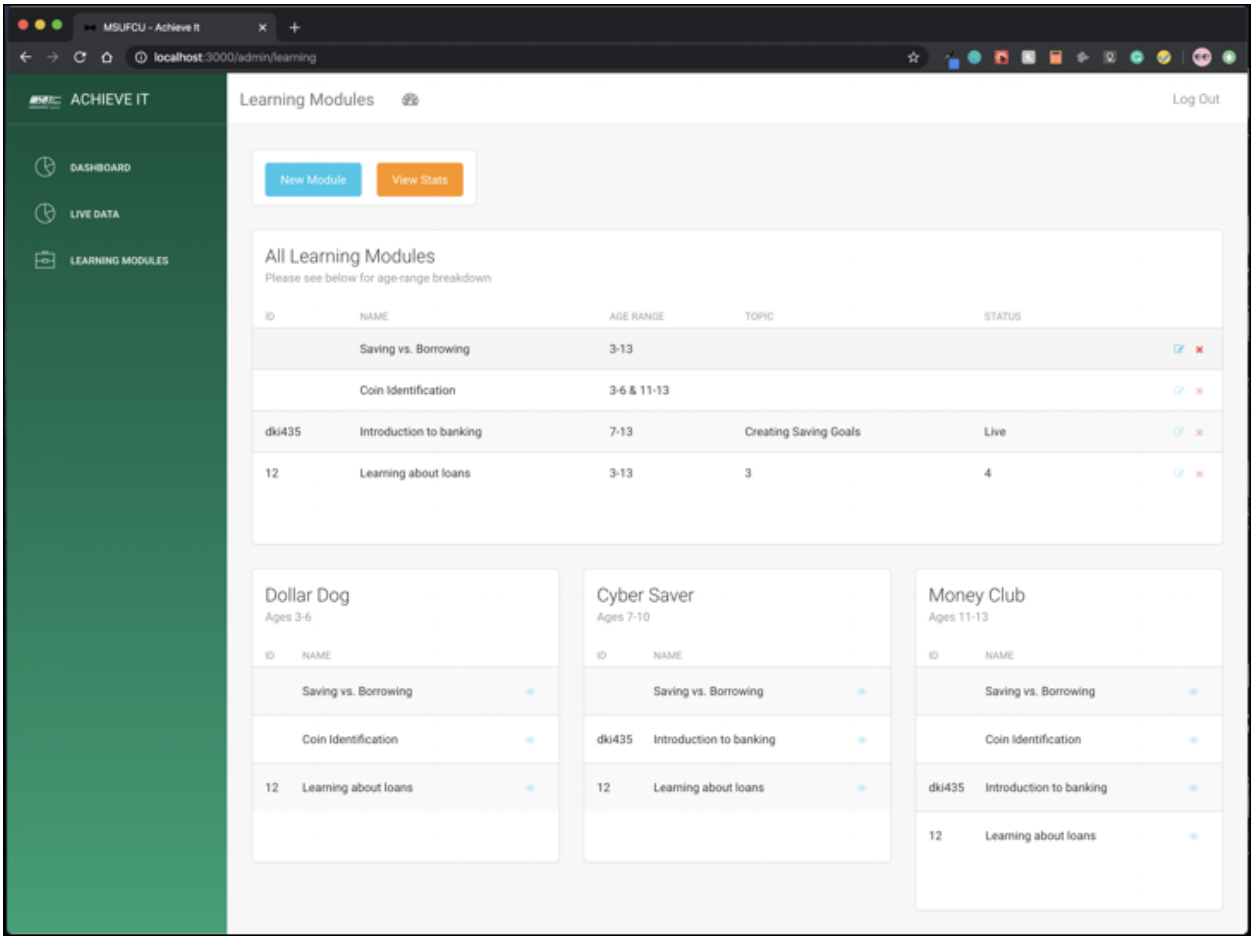

Figure 13: Internal web dashboard learning module creation page

The internal web application will be the only user interface for MSUFCU administration to monitor and interact with the "Achieve It" system. Based on instruction and advice from MSUFCU faculty, the designs for this end of the system will come from a premade template for data analytics.

Figure12 is the data analytics page for the MSUFCU internal application. This shows a number of data points about application usage. We are currently at a stage where we are waiting for MSUFCU to request any more data points they would like to monitor, otherwise this page is complete.

Figure13 is the learning module CMS hub. This is the central page where the internal admin will be able to create new learning modules for the entire system, add and edit the actual learning content with a rich text editor, and assign the modules to specific age groups. This is a very intuitive and simple view that is designed for maximum efficiency in creating new content for the system.

Technical Specifications

System Architecture

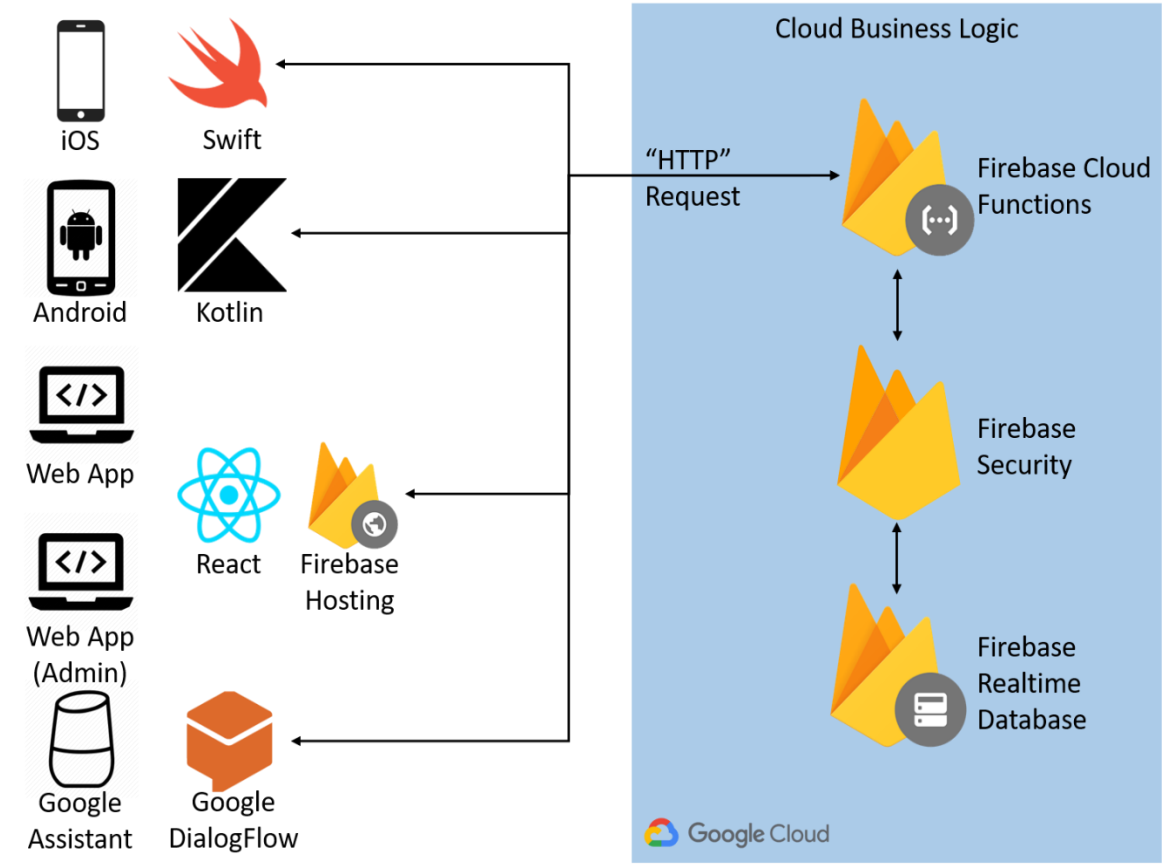

Figure 14: System architecture diagram

"Achieve It" consists of native applications for iOS and Android, two web applications built on top of the ReactJS framework, and Google Voice Assistant interactions with DialogFlow. One of the web applications will allow the MSUFCU team to administer the "Achieve It" application. Firebase Hosting provides hosting for web applications. Cloud Functions for Firebase allows the integration of Firebase Features and is shared by all devices. Data and application security are handled by Firebase Security. Data is stored in Cloud Firestore.

System Components

Cloud Platform

Firebase

Firebase is a web and mobile application development platform built on top of Google Cloud. Our system will be leveraging a number of Firebase services to create real-time data-driven applications.

Firebase Cloud Functions

Cloud Functions is an event-driven serverless compute platform that is hosted and managed on Google Cloud. All user-end components will directly interact with Firebase Cloud Functions to securely receive, send, and process data. Google Cloud handles the operational infrastructure of the application.

Firebase Security

Firebase Security Rules are node-based and managed by a single JSON object. Security Rules provide access control and data validation. Firebase Authentication working with Security Rules allows the team to build user-based and role-based access systems, keeping the users' data secure. All Firebase Security Rules are fully customizable for the system's specific needs.

Cloud Firestore

Cloud Firestore is a flexible, scalable NoSQL cloud database for mobile, web, and server development. Firestore keeps data in-sync across client apps and offers offline support for mobile and web. Firestore integrates with other Firebase and Google Cloud Platform products, such as Cloud Functions.

Firebase Hosting

Firebase Hosting provides infrastructure, features, and tooling tailored to deploying and managing websites and apps. Files are deployed from local directories to the Hosting server using Firebase CLI. Cloud Functions can be used to serve dynamic content and host microservices on sites. All content is served over an SSL connection from the closest edge server on Firebase's global CDN.

DialogFlow

DialogFlow is a human to computer interaction development platform that is specifically designed for natural language conversations across a number of platforms. DialogFlow will be used to interact with the Google Assistant component of the "Achieve It" system.

Web Applications

React

React is a JavaScript library for building user interfaces. React will update and render components when application data changes. Components are encapsulated and manage their own state. Data is easily passed through the application because component logic is written in JavaScript instead of templates.

Mobile Applications

Swift

Swift is a powerful and intuitive programming language for iOS. Specifically, our team will be using the most stable version of Swift: currently Swift 5.1.13

iOS SDK

The iOS SDK contains a set that grants developers access to various functions and services of iOS devices. The SDK also contains an iPhone simulator for testing applications. Developers are able to write iOS apps using Swift by combining the iOS SDK with Xcode.

Firebase SDK

Cloud Firestore supports SDKs for Android, iOS, and Web. Realtime updates and offline data persistence are supported by mobile and web SDKs. An Admin SDK is used to initialize access to Cloud Firestore and other services from a single SDK. The Firebase Admin SDKs support Cloud Firestore access in Java, Python, Node.js, and Go.

Kotlin

Kotlin is a programming language that is fully supported in Android Studio and compatible with the Android build system.

Android SDK

The Android SDK and API allow the development of software for devices running Android. The target build is to cover users across Android 9.0(Pie) to Android 4.4(KitKat) which should cover all MSUFCU Android members.

Risks

Content of Appropriate Complexity for Age Groups

- **Difficulty:** Medium
- **Description:** The targeted age range for the child side of the application is from ages 3-13. The challenge here is that a 3-year-old is very different from a 13-year-old. Therefore, there must be different forms of content and layout structures for each of these age groups. Moreover, in terms of learning modules, it will be difficult to define what is complex enough for the targeted age group but at the same time not so complex to the point where it becomes impossible. We are not really sure what would define a user-friendly interface for a 3-year-old all the way through a 13-year-old.
- **Mitigation:** Adopt MSUFCU's own distinctions between ages and follow their design and content decisions

Modularization of Learning Activities

- **Difficulty:** Medium
- **Description:** To keep the system easily extendable a generic format for learning activities needs to be established. That is to say, some sort of standard is needed to allow for the easy addition of new modules without needing to alter the source of the "Achieve It" system.
- **Mitigation:** From looking at other systems that host semi-complex dynamic content the popular solution seems to be restricting the format of the content.

Configuring Authentication into Google Assistant

- **Difficulty:** Low
- **Description:** The Google Assistant portion of the application is said to not have a graphical user interface. While the features of the assistant app are pretty minimal, it does require authentication in order to read and deliver the user data associated with the account. The question is how do we implement authentication without a user interface. Can authentication be implemented via voice or will a whole other application needs to be created just so users can log in and have access to the google assistant? At the same time, this needs to be done while maintaining the security of the account information.
- **Mitigation:** See if there is already a system for achieving this. Perhaps use a web or mobile app to authenticate for voice.

Testing plan

Testing for each platform will be handled in parallel to development by each member in a “test what you write” sort of policy. By nature of the project and our implementation with Firebase this system is UI heavy with minimal business logic so testing will be done via usage, for the most part. Voice assistant testing is easier to test through use. For field testing, the team at MSUFCU has generously offered to have their children test the app for feedback.

Schedule

Week 1: 1/5 - 1/11

- Initial meeting with the group (Establish weekly meetings)
- First meeting with clients (Establish weekly meetings)
- Assign roles

Week 2: 1/12 - 1/18

- First triage meeting (Weekly)
- Status update (All)
- Present status update (Ben S & Michael)

Week 3: 1/19 - 1/25

- Hello world (All)
- Establish architecture (Ben S & Rachel)
- The initial batch of UI designs (Ben S)
- Create the base layer for Firebase Cloud Functions and deploy to Google Cloud (Ben S)
- Project plan (All)

Week 4: 1/26 - 2/1

- Finish Wireframing/basic designs for the mobile and web application (Ben S)
- Get the web application up and running and connected to Firebase (Ben S & Rachel)
- Create most screens in Android app with unlimited navigation “white box” (Ben C)
- Get login screen implemented and perform basic authentication for iOS (Michael)
- Present project plan (All) unknown if this week or next

Week 5: 2/2 - 2/8

- Present project plan (All) unknown if this week or last
- Have all completed designs for the web and mobile apps, and export all assets for developers (Ben S)
- Finish up any lingering screens in Android (Ben C)
- Implement login functionality for all users on Android (Ben C)

- Implement login functionality for all users for Web App (Rachel)
- Finish login functionality and implement the Tab bar controller and task view for child side for iOS (Michael)
- Begin outlining basic code and conversations for Google Assistant in DialogFlow (Ben S)
-

Week 6: 2/9 - 2/15

- Status report (All)
- Present status report (undecided)
- Work with web and mobile developers to ensure Cloud Functions are providing accurate and helpful data upon request for all possible requests (Ben S)
- Continue adding functions to the web dashboard (Ben S)
- Add task and learning module functionality on Android parent (Ben C)
- Begin Add Child functionality for Parent View in Web App (Rachel)
- Finish and complete layout for loans view (Michael)
- Alpha Presentation (All)

Week 7: 2/16 - 2/22

- Present Alpha (All) unknown if this week or next
- Begin connecting the pre-made HTML and CSS in the MSUFCU data analytics template with real-time data from Google Firestore (Ben S)
- Start coding out Firebase Cloud Functions for one of the major Google Assistant intents (Ben S)
- Continue adding, debugging, and improving all Cloud Functions (Ben S)
- Add learning loan functionality on Android parent(Ben C)
- Finish Add Child for Web App (Rachel)
- Add learning module functionality for iOS & Implement Loan Functionality (Michael)

Week 8: 2/23 - 2/29

- Present Alpha (All) unknown if this week or last
- Adding in a WYSIWYG Editor for images and other rich text in the learning modules (Ben S)
- Add goal functionality on Android parent and extend to child (Ben C)
- Begin implementing Add Task and Add Goal in Parent View for Web App (Rachel)
- Start implementing Parent View Authentication (Michael)

Week 9: 3/1 - 3/7

- Spring Break
- Enjoy a much needed time off

Week 10: 3/8 - 3/14

- Status update (All)
- Present status update (undecided)
- Start to finish up with cloud functions and all debugging for web app (Ben S)
- Start testing mobile applications on the 4 major features and compile a list of all outstanding bugs (Ben S)
- Polish off the statistics end of the MSUFCU analytics dashboard (Ben)
- Finish functionality not shared by parent and child(Ben C)
- Finish Parent View for Web App (Rachel)
- Finish up Parent View such as Tasks, Loans, & Learning Modules (Michael)

Week 11: 3/15 - 3/21

- Continued development work on Google Cloud Assistant and DialogFlow (Ben S)
- Polish UI and ensure consistency across mobile platforms(Ben C)
- Work with Rachel on the client facing web application (Ben S)
- Implement Child View for Web App (Rachel)
- Create settings page to switch between accounts and log out

Week 12: 3/22 - 3/28

- Status update (All)
- Present status update (undecided)
- Continued development work on Google Cloud Assistant and DialogFlow (Ben S)
- Finalize Google Assistant and begin running all tests (Ben S)
- Cleanup, debugging, and user testing for Android (Ben C)
- Cleanup, debugging, and user testing for Web Apps (Rachel)
- Finalize, debug, and test iOS App (Michael)

Week 13: 3/29 - 4/4

- Present Beta (All) unknown if this week or next
-

Week 14: 4/5 - 4/11

- Present Beta (All) unknown if this week or last

Week 15: 4/12 - 4/18

- Status update (All) x2
- Present status update (undecided) x2
- Project video

Week 16: 4/19 - 4/25

- Project video

- Design day setup

Week 17: 4/26 - 5/2

- Design day
- Project videos