12-2018

Increasing Engineering Retention with Mobile Technology: Using UX/UI for Us

James H. Cate Jr.
University of Tennessee-Knoxville, jcate6@vols.utk.edu

Andrey Karnauch
University of Tennessee-Knoxville, akarnauc@vols.utk.edu

Dakota Sanders
University of Tennessee-Knoxville, dsande30@vols.utk.edu

Matt Matto
University of Tennessee-Knoxville, hgd145@vols.utk.edu

Increasing Engineering Retention with Mobile Technology: Using UX/UI for Us

COSC402 / Chancellor’s Honors Program

6 December 2018

Customer / Faculty Mentor: Dr. Masood Parang

James Cate (jcate6)

Andrey Karnauch (akarnauc)

Dakota Sanders (dsande30)

Matt Matto (hgd145)

Table of Contents

Table of Contents **2**

Executive Summary **3**

Requirements **4**

Change Log **7**

Backend Design Process **8**

Research 9
Development Environment 9
Signup Forms 10
Database 12
Forms 12
Profiles 13
Goal Menu 15

Backend Summary **16**

Frontend Design Process **18**

Sign In 18
Sign Up / Verification 19
Form Collection 20
Profiles / Searching 21
Flavors of Home 22
Goals and Goal Management 23

Frontend Summary **23**

Lessons Learned **24**

General Lessons 24
Back End Lessons 25
Front End Lessons 25

Team Member Contributions **27**

James Cate 27
Andrey Karnauch 27
Matt Matto 28
Dakota Sanders 28

Executive Summary

The Engineering Mentor Program (EMP) at the University of Tennessee aims to increase underclassmen retention rates among the College of Engineering by matching freshmen with upperclassmen that share similar interests, goals, and majors. Currently, students join the program by filling out a Google Forms questionnaire. Results are then manually inputted into a matching algorithm that pairs students based on the results of these forms. The output is then manually parsed, administration makes note of matching pairs, and emails are individually constructed to contact the relevant students and inform them of their new mentor or mentee.

The goal of our application is to create a cross-platform mobile toolbox capable of executing all the aforementioned functions plus more. This starts with allowing interested students to sign up using a similar form to the one currently used, but housed in-app. Access to an active UTK netID is required to create an account. Once signed up, more information regarding student interest is collected, allowing students to be able to customize their user profile and view other prospective students (mentees can view potential mentors, and mentors can view potential mentees). A preference can be indicated by the student if a potential pair seems preferable to them (perhaps a friend from high school). Then, after a buffer period for sign-up, EMP administration can use our program to initiate automatic matching among all students left in the program's user pool. In addition, the app will allow students a scheduling and accountability tool in the form of a goals menu, which is further elaborated on below.

After much hard work this semester, nearly all of these goals were met and some extra features were added. Students can sign up using their UTK email address, fill out the questionnaire, create a profile, indicate their preference for mentor or mentee, and are matched by EMP. They can also create custom goals with specified timeframes that can be seen

between both members of the pair. These custom goals are a generic way of exchanging meeting times or other goals between the pair.

Challenges include configuration of the cloud computing framework used, following best practice guidelines, creating an intuitive user experience, implementing adequate testing to find bugs, and dealing with an extreme learning curve for all members of our team. Challenges are detailed in-depth throughout this document.

Requirements

1. Platform
    - 1.1. The application must be accessible on iOS devices.
    - 1.2. The application must be accessible on Android devices.
    - 1.3. The user interface must adjust to the size of the screen it is being accessed on.
    - 1.4. The application must distribute forms using Google Forms.
        - 1.4.1. The forms must store data using Google Sheets.
    - 1.5. The application must be run natively using Reactjs.
    - 1.6. The application must be architected using Amazon Web Services.
        - 1.6.1. All architecting must follow AWS best practices and white pages.
2. Functionality
    - 2.1. The application must allow the user to create an account.
        - 2.1.1. The application must require users to authenticate using their netID and password.
        - 2.1.2. The account creation must provide users with the option to upload a profile picture.
            - 2.1.2.1. The account creation must have a default profile photo for users.
        - 2.1.3. The account creation must feature a signup form.
            - 2.1.3.1. The signup form must be used to populate account information.
            - 2.1.3.2. The signup form must allow the user to specify whether they are registering as a mentor or mentee.
            - 2.1.3.3. The signup form must ask for a name, approximately three sentence bio, major, hobbies, courses taken or currently taking, gender, and other parameters already defined by EMP.
        - 2.1.4. The application must allow mentors and mentees to sign up in the middle of the semester.
    - 2.2. The application must transfer form data into a format interpretable by the matching algorithm.
        - 2.2.1. The application must store processed form data.

2.2.2. The application must sort all applicants into two groups: mentors and mentees.

2.2.3. The application must create two more groups, one including only female mentors, one including only female mentees.

2.3. The application must allow users to search for mentors/mentees and view their bios prior to pairing.

2.3.1. The application must allow users to filter mentor/mentees by their major, courses taken/taking, and employment information.

2.3.2. The application must allow users to specify their preferred mentor and mentee.

2.3.3. The application must try to match mentors and mentees who prefer each other.

2.4. The application must pair mentors and mentees together after a deadline at the start of the semester.

2.4.1. The application must allow accounts created in the middle of the semester to be paired with existing mentors.

2.4.2. The application must store mentor-mentee pairings.

2.4.3. The application must indicate who a user is paired with, sharing their profile.

2.5. The application must allow a mentor and mentee to fill out a contract.

2.5.1. The application must allow a mentor and mentee to indicate that they have read the contract and sign the contract.

2.5.2. This contract must be in accordance with the current EMP contract.

2.6. The application must have an admin portal for app management.

2.6.1. The admin portal must allow the distribution of push notifications.

2.6.2. The admin portal must allow the distribution of files.

2.6.3. The admin portal must allow the removal of users from the application.

2.6.4. The admin portal must allow manual override of a matching pair.

2.6.5. The admin portal must be able to edit events on a user’s calendar.

2.6.6. The admin portal must be able to edit a user’s goals.

2.6.7. The admin portal must allow system reset for the next school year.

2.6.7.1. This action must store the previous year’s data in a database.

2.6.7.2. The admin portal must allow for future maintainability of the app.

2.6.7.3. The admin portal must be documented well for future users.

2.7. The application must provide scheduling functionality between pairs.

2.7.1. The application must allow mentees to request meetings.

2.7.2. The application must allow mentors to schedule meetings.

2.7.3. The application must allow mentors/mentees to validate their attendance at scheduled events.

2.7.4. The application must allow mentors/mentees to cancel meetings.

2.8. The application must provide goal functionality between pairs.

2.8.1. The application must allow for required goals set by EMP or goals set by mentors that cannot be edited by mentees.

    - 2.8.2. The application must allow for non-required goals set by mentees which can be edited or deleted.
    - 2.8.3. The application must include account creation as a goal to complete.
    - 2.8.4. The application must include a successful pairing as a goal to complete.
    - 2.8.5. The application must create a goal for every scheduled meeting.
        - 2.8.5.1. The application must mark this goal as complete after the mentor and mentee validate their attendance.
    - 2.8.6. The application must keep record of goal completion for users and store it.
- 2.9. The application must provide the option for a mentee/mentor to contact EMP with any concerns.

3. Views
- 3.1. The application must have a view for user account creation.
    - 3.1.1. This view must only appear during initial account setup.
- 3.2. The application must have a loading view.
    - 3.2.1. This view must appear upon any application loading time.
    - 3.2.2. This view must have a moving icon to indicate loading.
- 3.3. The application must have a dashboard view prior to matching.
    - 3.3.1. The dashboard must have the user's profile picture and name next to a default profile with a question mark to indicate that no match has occurred yet.
    - 3.3.2. The dashboard must have a HELP/CONTACT button for users to contact EMP, as described in 2.9.
    - 3.3.3. The dashboard must have a calendar icon that pulls up the calendar view.
    - 3.3.4. The dashboard must incorporate the goals view.
    - 3.3.5. The dashboard must have a notifications icon to view messages and files shared from their mentor or EMP.
    - 3.3.6. The dashboard must have a search icon to bring up the search view.
- 3.4. The application must have a dashboard view post-matching.
    - 3.4.1. This dashboard must provide all of the views listed in 3.3.
    - 3.4.2. This dashboard must include the user's matched mentor/mentee profile picture along with their name next to the user's profile picture and name.
    - 3.4.3. This dashboard must allow users to click on their partner's profile picture to bring up their profile view.
- 3.5. The application must provide a calendar view.
    - 3.5.1. This calendar view must provide all of the functionality described in 2.7.
- 3.6. The application must provide a goals view.
    - 3.6.1. This view must provide all of the functionality described in 2.8.
    - 3.6.2. This view must allow users to filter their goals.
- 3.7. The application must provide a notification and files view.
    - 3.7.1. This view must store all notifications and files shared with the user.
    - 3.7.2. This view must allow the user to see notifications/files sent in the past.
    - 3.7.3. This view must indicate when a notification has been viewed vs. when it has not.

3.8. The application must provide a search view.

3.8.1. This view must provide all of the functionality described in 2.3.

3.8.2. This view must allow a user to click on a search result to bring up another user's profile.

3.9. The application must provide a profile view.

3.9.1. This view must provide the indicated user's profile picture.

3.9.2. This view must provide the indicated user's bio.

3.9.3. This view must provide a "thumbs up" option for users to click on prior to matching, indicating a preferred match.

3.9.4. This view must provide all of the details filled out in 2.1.3.

Change Log

September 28th, 2018

- We realized that using Google Forms and Google Sheets for forms and data storage was far more difficult than just rebuilding EMP's current forms inside the application. As a result, Requirements 1.4 and 1.4.1 were altered as such:

  1.4. The application must recreate EMP's forms as an individual screen in the application.

  1.4.1. The forms must store data using AWS Dynamodb.
- During the same session, we also realized that going through UT's IT department to authenticate users using the Central Authentication Service (CAS) presents several challenges in terms of security. As a result, we decided to authenticate using a UT email paired with a confirmation code. Requirement 2.1.1 was altered as such:

  2.1.1. The application must require users to authenticate using their UT email and a confirmation code.

October 7th, 2018

- When creating the Sign Up screen, we noticed that EMP's application form for mentors and mentees should belong on a separate screen. If we put it on the Sign Up screen, users would be immediately deterred due to the length and time consuming aspect of simply signing up. As a result, we changed Requirement 2.1.3.3 as follows:

  2.1.3.3. The signup form must ask for a name, email, password, phone number, and whether they are signing up as a mentor or mentee.
- After receiving EMP's current materials, we changed Requirement 2.3.1 because EMP did not ask their members for courses taken and employment information. As a result, Requirement 2.3.1 was altered as follows:

  2.3.1. The application must allow users to filter mentor/mentees by their name and major.

November 16th, 2018

- Rather than splitting up calendar functionality and goal creation, our team decide to merge the two into a single Goal Creation view. This affected several of the requirements, with changes noted below.

    - Removal of Requirements 2.6.5, 3.3.3, 3.5,
    - Change Requirement 2.8.1 to “The application must allow for required goals set by EMP that cannot be edited by mentees.”
    - Change Requirement 3.6.1 to “This view must provide all of the functionality described in 2.7 and 2.8.”

Did Not Meet

- Requirement 2.1.2: The account creation must provide users with the option to upload a profile picture.
    - This requirement was consistently put on the backburner as it was never critical to any greater app functionality. Despite this the requirement for a default photo, 2.1.2.1, was still met and the app is styled so that the addition of user photos will have no negative effect on the system.
- Requirement 2.5: The application must allow a mentor and mentee to fill out a contract.
    - This requirement was partially implemented with the Terms and Conditions, but it would be useful to get contractual agreements in other forms as well by a more generalized feature that could collect digital signatures. We never found an adequate tool to assist in this, so we moved on in order to continue development on more critical features.
- Requirement 3.7: The application must provide a notification and files view.
    - This was the primary requirement we were sad not to meet, as it adds the most additional functionality and would have been very helpful to EMP. This would prevent EMP from needing to use Email to distribute files, making the organization almost entirely contained by the app. This feature was not implemented because notification services require API keys, which cost a significant amount of money per year, and an entire new set of backend services. We also did not find a pdf sharing tool that we were happy with. For the sake of time and quality of other features, we did not spend time obsessing over this difficult requirement.

Backend Design Process

This section will address the thought process behind all design decisions for the backend architecture. This includes planning for navigation, view hierarchy, Amazon resource setup, and linking the application to cloud resources. It stops just short of the final backend step, input validation and error passing, which will be addressed in the front end design portion as it is user-facing. The data flow diagrams in this section will abstract frontend to labeled squares.

Research

The first step of our design process began with doing extensive research into frameworks or methods that could be used to develop a cross-platform application. We knew that developing the application twice, once for iOS and once for Android phones, was unreasonable for us to complete in the one semester we had to develop the app. Instead, we found the React Native framework for Javascript. React Native is a framework developed by Facebook that allows developers to write code one time in Javascript and distribute it on both platforms, “natively”. Once we decided on this platform, we needed to research a backend. We understood that a cloud computing framework was needed, as it is easily the most economical way to develop and maintain an application as small as ours. There are several major providers for cloud computing services, including Amazon, Google, IBM, and Microsoft. Amazon Web Services, or AWS, was quickly decided upon, especially since several members of our team have experience using different tools inside of AWS, and it is especially cost-effective for smaller projects like ours. With a backend decided upon, we then moved on to setting up our development environment. Both the decisions to use React Native and AWS were made before the requirements document was created, and this is reflected in requirements [1.5] and [1.6].

Development Environment

Our development environment consisted of setting up both our frontend and our backend. Our frontend was set up using Expo, a free, open-source toolchain specifically made for React Native that allows for easy setup, development, and testing on both iOS and Android.

Expo allows a developer to live stream a Javascript bundle to a mobile device from a computer by scanning a QR code that is generated through a command line interface within the application source folder. We also utilized some specialized packages found in Node.js that are helpful for setting up a React Native Application. Once these steps were made, we had a sample frontend we could tweak and modify as much as we pleased. After this step, we created our AWS account and initialized our backend. We utilized AWS Amplify to create most of our backend infrastructure. Amplify is a command line interface that allows developers to easily create mobile-friendly systems that sync with the frontend easily. One can specify what resources to create, disable, delete, or modify. Resources can easily be put offline, swapped for new resources, or run in parallel, all in compliance with AWS's guidelines. This means Amplify also assists us in achieving our goal [1.6.1] of designing our architecture to match AWS's best practices. GitHub was used for source control, safely merging new features into an existing codebase, and documenting workflow. Much of the rest of the backend would be developed later on, in accordance with the discovery of the problem that each respective piece aims to solve.

Signup Forms

The major development work began with the implementation of our signup forms. After receiving the Google Forms signup form from EMP, we knew what questions needed to be asked during the signup process; however, we needed to first initialize a user authentication system, as specified in requirement [2.1]. Thus, the first major development to our backend was the addition of AWS Cognito. Cognito is a service that provides easily-scalable user authentication through the use of user pools. We chose to implement an email verification system. In other words, the user begins the signup process by entering an email address, password, name, phone number, as well as an indication as to whether they intend to be a mentor or a mentee. As a way to enforce requirement [2.1.1], we enforce that the email must be a valid UTK domain, and we use Cognito to send a verification email to the user. This email contains a random personal identification number that must be entered into the app before the profile or any functionality within the application is usable. We also ensure good security practices by enforcing an uppercase character, a lowercase character, and a number in a password that is at least eight characters long. Cognito then handles storing this verification information inside of a user pool. This process is visualized below in Figure 1. It is important to note the security and privacy concerns introduced at this step. Storing a user's personal information such as contact information and passwords requires us to be ethical and store them in an encrypted, secure manner. Thankfully, this is Cognito's specialty, and all of the personal information is securely stored in the cloud outside of the context of our normal databases. This information is only accessed during account verification and sign-in.

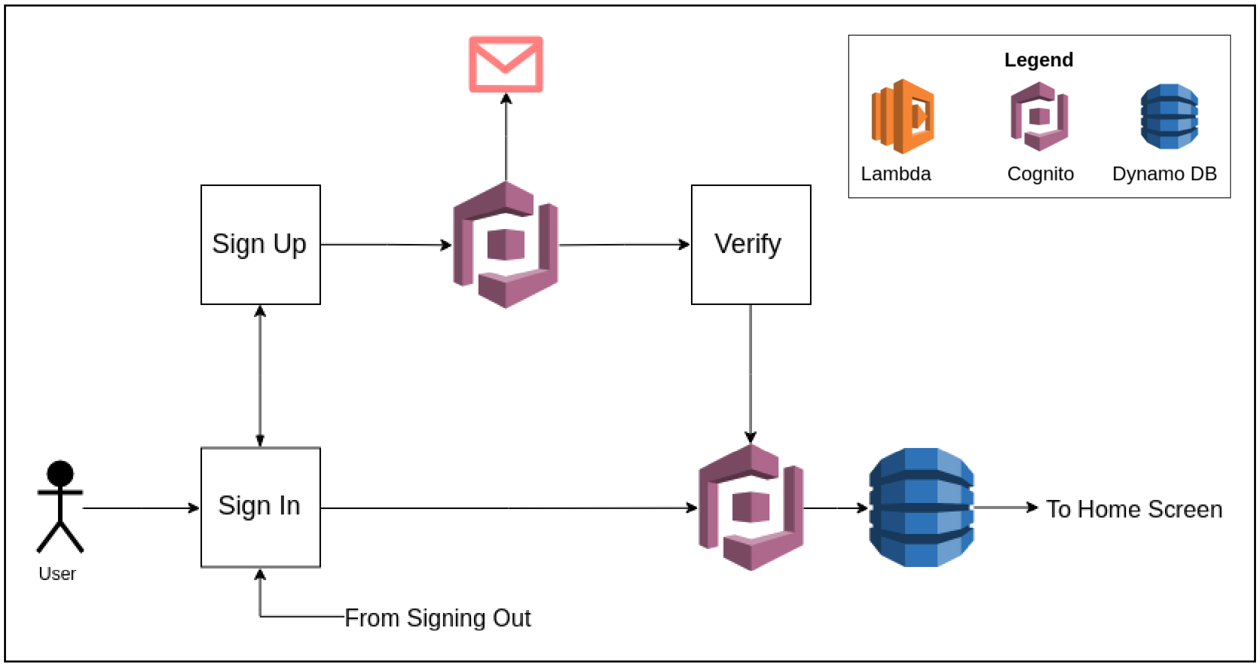


Figure 1: User Authentication Diagram

Database

The next step of development involved setting up a query-able database system capable of storing information gained from users interacting with the signup form. AWS provides two different database systems, RDS and DynamoDB. RDS is a more traditional relational database model that requires consistent information fields. DynamoDB is a noSQL database that is much more flexible in the types of information that can be stored. Since we were planning on using our database to store several different types of data, including JSON strings, we decided to use DynamoDB as our storage implementation. By selecting this type of storage, we satisfy requirements [2.2.1], [2.3.1], and [2.4.2].

Forms

With the storage system in place, we developed two separate signup forms, one for mentors and one for mentees. After filling out the questionnaire, users are required to read and accept a terms and conditions page, which meets requirement [2.5]. This is another important ethical checkpoint, as users' profiles will display some personal information such as their interests, major, minor(s), and goals. Therefore, we must get approval from users before we share this information and ensure that this information is only shared to relevant parties of consent. Here, we also enact a code of conduct for using the app and attempt to define what acceptable relations between a mentor and mentees are. We realized at this point that we needed to add some code that would take this information and send it to DynamoDB, so we set up AWS Lambda functions to handle this. A Lambda function is a way to specify code that should be executed upon activation of certain triggers specified within AWS. In our case, we made a function that is activated upon the tap of the accept button in the terms and conditions page. This function handles formatting and sending data to our database, described in Figure 2. The most beneficial part of using Lambda is that it eliminates the need for a costly, always-available server in favor of code that can be executed only when it is needed. This is both the most economical solution as well as the solution with the least maintenance.

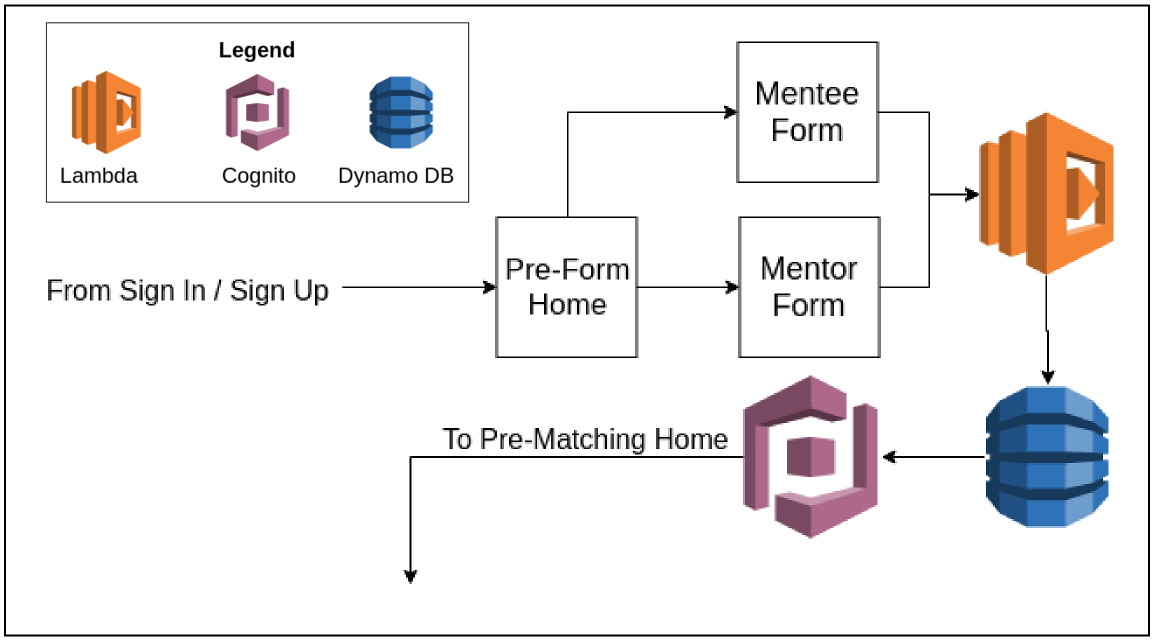


Figure 2: Form Diagram

Profiles

Requirement [2.3] requires that we allow users to create a profile that can be browsed by other users. To do this, we automatically populate a profile page for each mentor and mentee based on the answers given in their signup form. The information displayed consists of personal details like the user’s name and major, as well as any professional details, such as the user’s dream job or life goals, as well as interests, such as hiking, traveling, and doing research. Once the user is registered, they are able to view their own profile and the profiles of some other users. Mentees are allowed to view available mentor profiles, and mentors are allowed to view available mentee profiles. This process is described in *Figure 3*.

The primary reason for including browsable profiles is to assist in the matching protocol. Our matching protocol consists of two methods of matching. The first method relies heavily on a feature we identify as “preferring,” which is where a user is allowed to select someone out of their browsable profiles as their preferred partner. This designation automatically pairs partners if and only if both partners prefer each other. If only one partner prefers the other, then that decision weighs more heavily in our matching protocol, but does not guarantee that *that* user will be paired with their preference. The second method of matching assigns weights to similarities between a given user and all available partners. The potential partner with the highest total weight is then assigned as the user’s mentor or mentee.

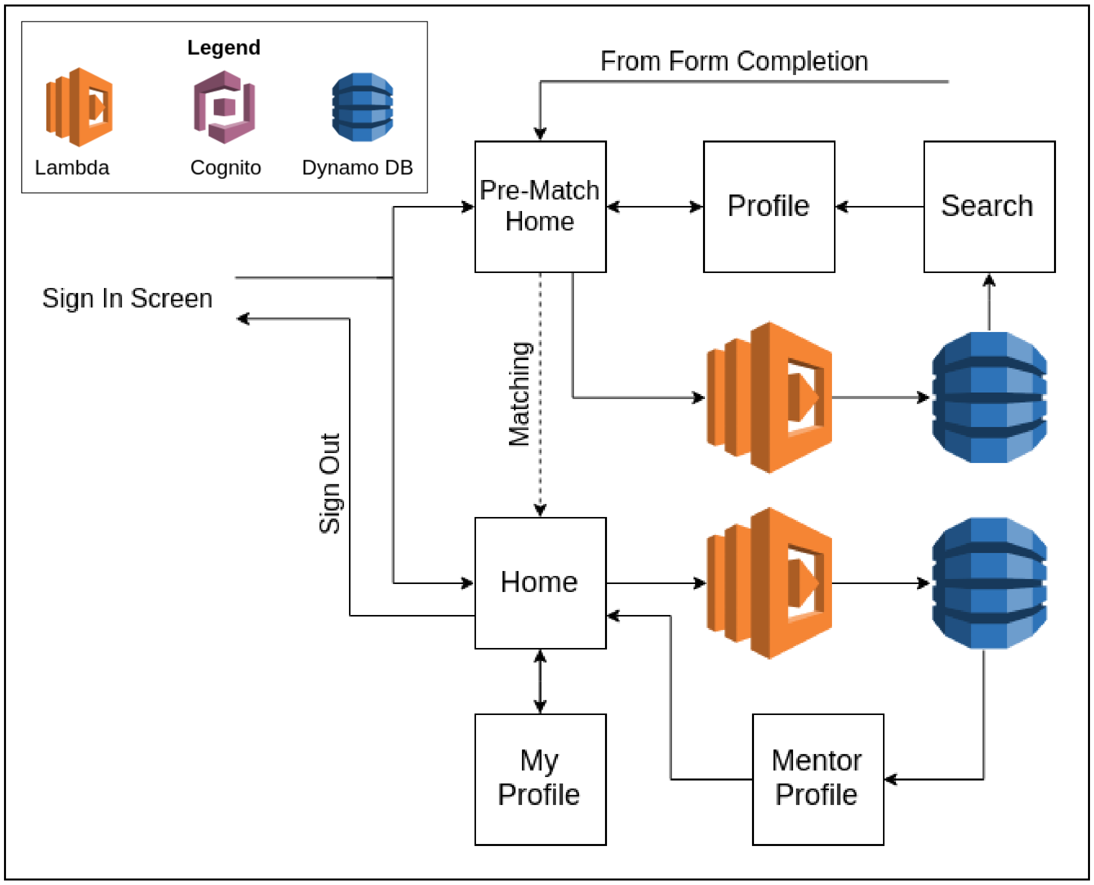


Figure 3: Profile Creation Diagram

Creating profiles allowed us to confirm that our form feature was working properly, as the profiles are automatically generated by the form data. During the process of implementing profiles, we discovered some issues in our form input validation that required correction. Because of the testing done during this process, we could validate much of our previous work. Additional testing was performed on the “preferring” feature. For example, we tested behavior for multiple preferences, over-writing preferences, removing preferences, and post-matched behavior ensuring that no unexpected states were reached and the feature was air-tight.

Goal Menu

After a user populates their profile with form data, the home screen which they see upon signing in will display a menu with three sections. Each section is labelled and corresponds with one of the following: an in-progress goal, a missed goal, or a complete goal. These goals are shared between a mentee and mentor pair. Goals can have a due date or be more abstract reminders of things to be done or thoughts had. This allows for mentees to populate the list with assignments, test, quizzes, etc., and allows the mentor to hold them accountable, check on progress or results, and offer help. This feature can also be used to schedule meetings between a pair, as the meeting’s goal will appear on both users’ home screens. Similarly, EMP can add goals to a user or set of users which could be used to remind them about upcoming organizational meetings, documents that should be reviewed, or to send encouraging messages around exam weeks. Goals are stored as an array of JSON objects in the Dynamo table.

Backend Summary

Our design process more or less followed an incremental flow. We began by devoting a significant amount of time to researching options. By devoting time to this process, we ensured we did not waste time developing on frameworks that would not yield results that propelled us towards our final product. Once we decided on React Native and AWS for our frontend and backend, respectively, we began the process of setting up our development environment. While this section of development was time-consuming, having a properly configured development environment shared among all group members meant that adding features or fixing bugs was dependant on our knowledge of the code, not problems with our framework.

With the research done and the application set up, the rest of the development process entailed adding new features. By following a logical flow of requirements in our application, most new features inadvertently verified the work of previous features. The process of populating our database verified our user signup process worked, and the rest of the features verified the previous work in a similar way. We managed to add a signup form, a functional database for users and their information, and browsable user profiles generated by dynamic form data to our application by the end of the semester, while successfully matching mentors and mentees together. This whole process is visualized in *Figure 4*. These features mean that a majority of the requirements specified in our requirements section were completed, and any that were not fulfilled are very simple to add because of the rigid structure of our application. Future development work will continue to build upon the work done over the course of this semester, and should not require any substantial refactoring for new features to be added. Once we finish building our application, we will move on to publishing and distributing the application on the

Apple App Store for iOS devices and the Google Play Store for Android devices, as well as shopping it to other departments across campus.

Our customer, Dr. Parang is in negotiation with our Solutions Architect, James Cate, and Lead Developer, Andrey, about hiring them as software engineers to add some functionality and extensively bug and load test the application before Summer orientation of 2019. The functionality in question pertains to screening mentor applications and providing the ability for a mentor to easily address multiple mentees at once. Neither of these features were in our requirements, but it has been decided that they are critical to the app's success in the future. Another critical step to success is documentation of administrative function on AWS, which is complete for all presently implemented features. Lambda functions for populating users goals, sending organizational emails, and tweaks to the matching will also be made.

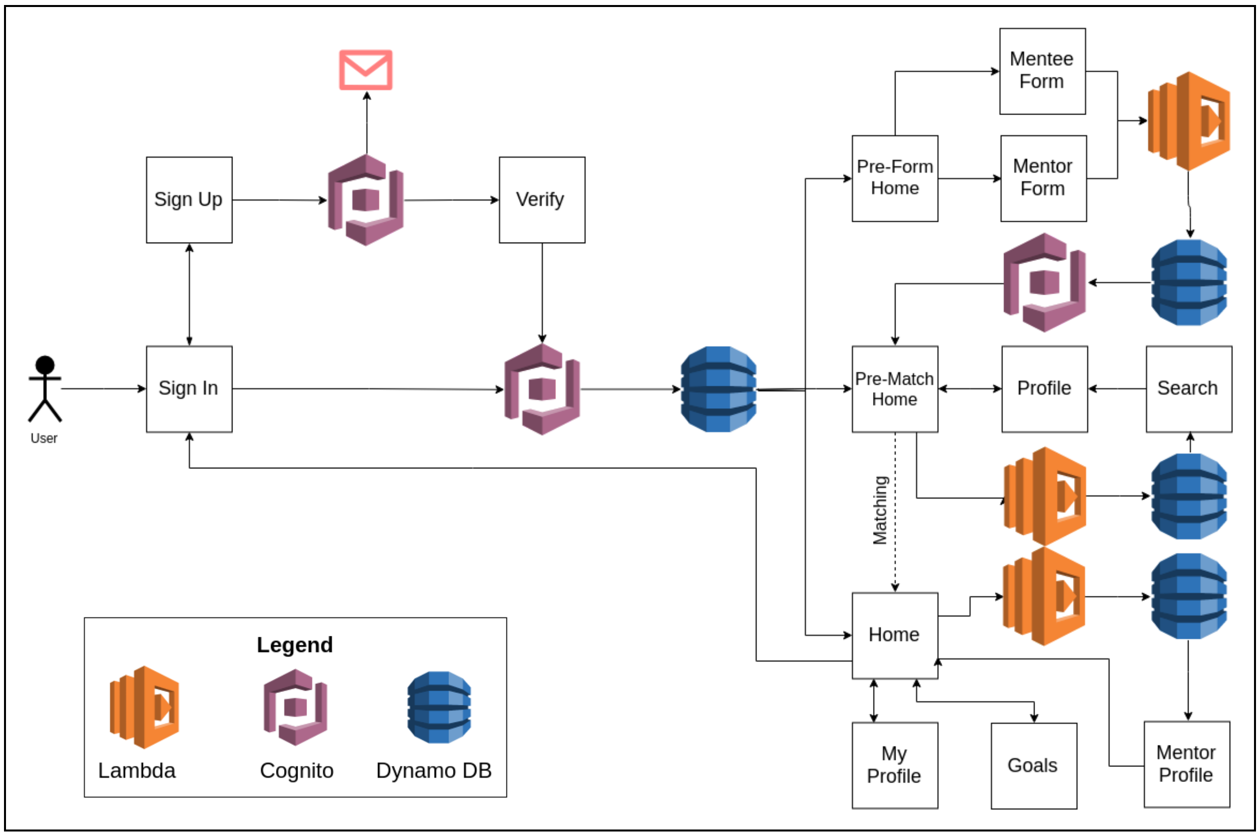


Figure 4: Application Architecture Diagram

Frontend Design Process

This section will address the thought process behind all design decisions for the frontend views. This includes the evolution of screens as data is collected, the flow between screens, user interface guidelines, style and branding guidelines, and how the user is taught how to use the app through game theory and modern user experience strategies. Error messages that accompany input validation will also be mentioned. It is worth noting that screenshots in this section come from both iPhone and Android to highlight the app's cross-platform functionality.

First, all components, such as action buttons, radio buttons, text fields, modal selectors, multiple choice selectors, chips, and transitions follow Google's Material Design guidelines. Material design is a standard for mobile applications that attempts to establish some high-quality regularity across mobile apps. This helps the app to feel familiar, intuitive, playful, and simple. These styling rules contribute greatly to the functionality of the app.

Second, all color choices and purely aesthetic decisions were made in accordance to UT's brand guidelines. Simple decisions such as never using black contribute to the feel of the app and make it immediately familiar with the UTK brand we all know and love. This is critical for incoming freshman, as it sets a precedent for what they can expect to see across numerous digital services for their time at this university.

Sign In

Upon opening the application a user will see *Figure 5*, the sign-in screen. From this screen they can log in with an existing account verified through AWS Cognito. If sign-in is attempted with an email not mapped to a user, the user is prompted to create an account. If there is an email password mismatch, then they are requested to re-enter their password.

According to modern practices, email is not case sensitive. With correct validation a user can navigate to their home screen with the “Sign In” button.

Alternatively, a user can navigate to the sign up screen with the “Create a New Account” button. The final bit of functionality for this simple screen is a “Contact Us” button which launches the user’s email client with the subject line pre-filled to EMP’s organizational email. This allows users to contact EMP about any errors in the app, trouble signing up, questions about the organization, or even request account deletion.

Sign Up / Verification

From the point of view of a new user, the next screen seen is the account creation screen, *Figure 6*. Here we collect all data needed to create a new user in our Cognito user pool. An example of how our text fields adhere to Material Design and give dynamic error message can be seen in *Figure 7*. The particular message seen in *Figure 7* shows how the app attempts to coach the user through sign-up, making the normally dull process as painless as possible. This styling is consistent for all text fields throughout the app, including those already seen on the sign in screen. Additionally, the radio button styling is also highlighted in *Figure 8* alongside an example of subtle but important user experience. The reminder to double check the form may seem

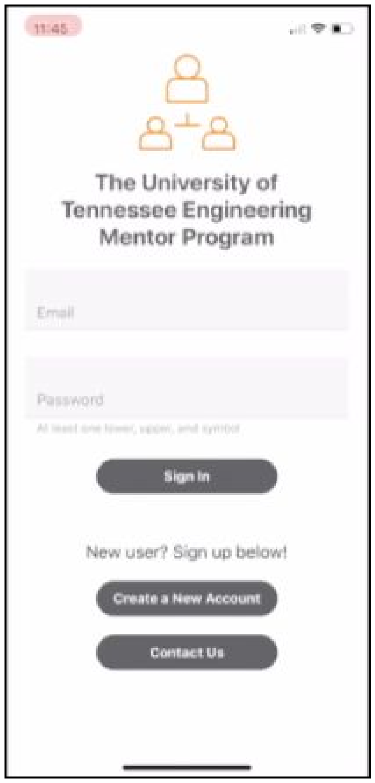

11:45
The University of Tennessee Engineering Mentor Program
Email
Password
At least one lower, upper, and symbol
Sign In
New user? Sign up below!
Create a New Account
Contact Us

Figure 5. The Sign in screen (iPhone)

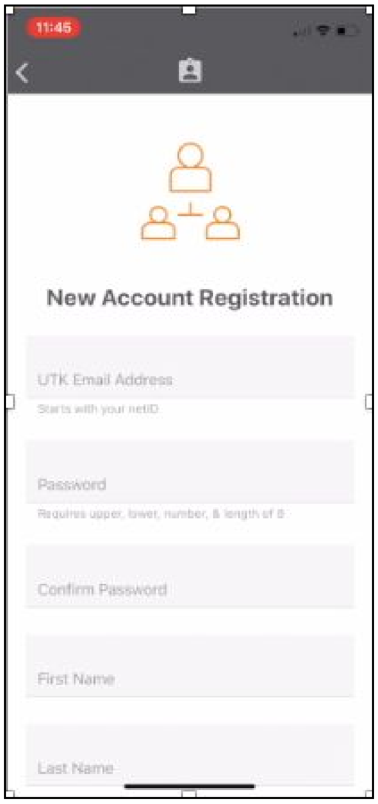


Figure 6. The Sign up screen (iPhone)

trivial but can save all parties headache in the future. Another example of this simple user experience tailoring is if a field is left blank, instead of of scolding the user we gently remind them “Oops! You forgot this one.” This enforces the playful tone of positive reinforcement found throughout the app’s UX. These may not seem worth highlighting, but we feel they are some of our most important work.

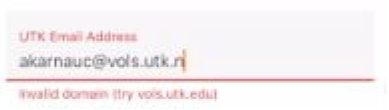

UTK Email Address
akarnauc@vols.utk.n
Invalid domain (try vols.utk.edu)

Figure 7. Text Field Dynamic Error Message with Coaching

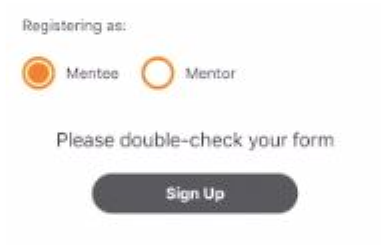

Registering as:
Mentee Mentor
Please double-check your form
Sign Up

Figure 8. Radio Buttons and Reminders

Form Collection

Once a user has created their account and signed in for the first time they will be taken to a home screen. At this point their only option is to sign out, contact EMP, or fill out the questionnaire so EMP can learn about its candidates. In *Figure 9* one can see an example of the form that has been submitted with some filled fields, and others empty. Once again this

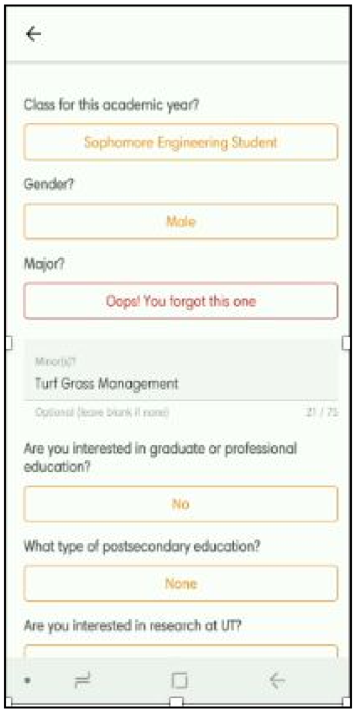

Class for this academic year?
Sophomore Engineering Student
Gender?
Male
Major?
Oops! You forgot this one
Minor(s)?
Turf Grass Management
Optional (leave blank if none) 21 / 75
Are you interested in graduate or professional education?
No
What type of postsecondary education?
None
Are you interested in research at UT?

Figure 9. Form Screen (Android)

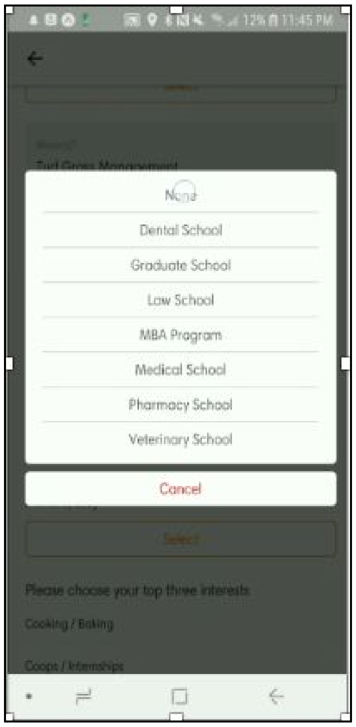

Turf Grass Management
None
Dental School
Graduate School
Law School
MBA Program
Medical School
Pharmacy School
Veterinary School
Cancel
Select
Please choose your top three interests
Cooking / Baking
Coops / Internships

Figure 10. Modal Selector (Android)

highlights are gentle, intuitive directing. *Figure 10* shows the modal selector itself. The “Terms & Conditions” button acts as a gatekeeper to validate form input before the user can agree and begin using the full app.

Profiles / Searching

For a user, form completion and profile creation happen at the same time. Profiles are dynamically generated based on form data so that other users can find a partner that is a match. In *Figure 11* the search screen is shown. Users can parse the pool of users by either major or name in attempt to find a friend or someone with similar goals in mind. *Figure 12* shows a profile. Here all colors come from UTK’s secondary brand colors to reinforce the app’s theme once more. The most notable feature is the green prefer button which can be used to weigh a user heavily for automated matching or potentially subvert the process entirely by matching on the spot.

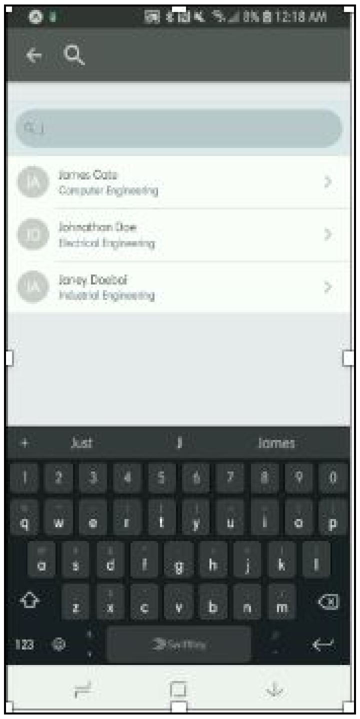
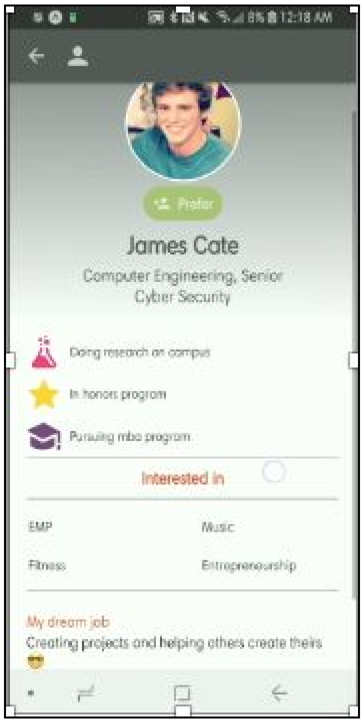

James Cate
Computer Engineering
Johnathon Doe
Electrical Engineering
Janey Doebol
Industrial Engineering

Figure 11. Search Screen (Android)

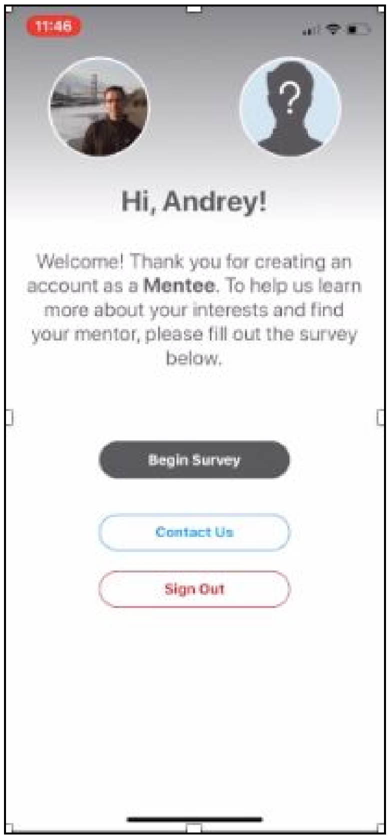


James Cate
Computer Engineering, Senior
Cyber Security
Doing research on campus
In honors program
Pursuing mba program
Interested in
EMP
Music
Fitness
Entrepreneurship
My dream job
Creating projects and helping others create theirs

Figure 12. Dynamic Profile Screen (Android)

Flavors of Home

Here all of the different home screens are shown as the user navigates through the data flow diagram. The home screen is a portal by which a user can access all functionality they have unlocked through using the app and contributing to EMP. The home screen shows a great example of how one can discreetly funnel users toward a desired function, preventing unwanted states. Notice the difference between main and secondary function buttons in *Figure 13.* In *Figure 14*, Pre-Match home, one can view their own profile, or browse other profiles. In *Figure 15*, Final Home, one can only view themself and their mentor. The goals are addressed shortly.



Welcome! Thank you for creating an account as a **Mentee**. To help us learn more about your interests and find your mentor, please fill out the survey below.

Figure 13. Pre-From Home (iPhone)

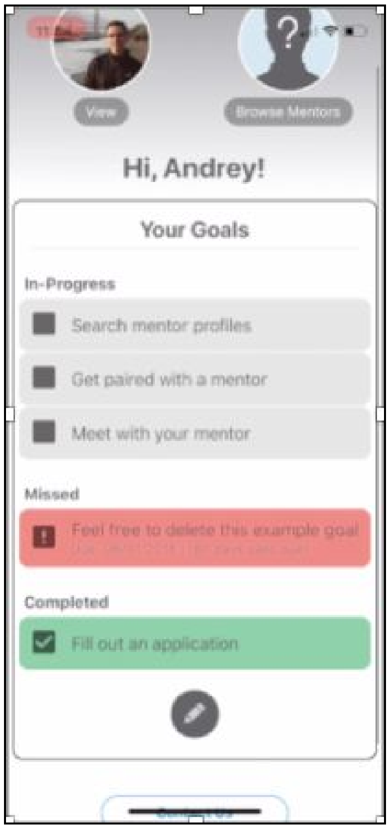


Figure 14. Pre-Match Home (iPhone)

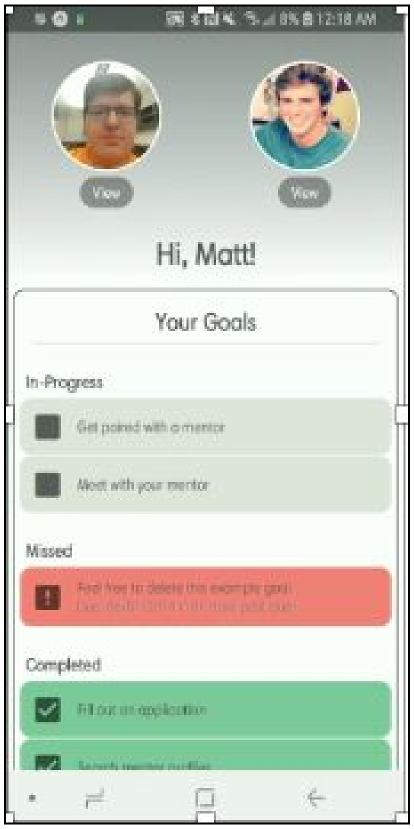
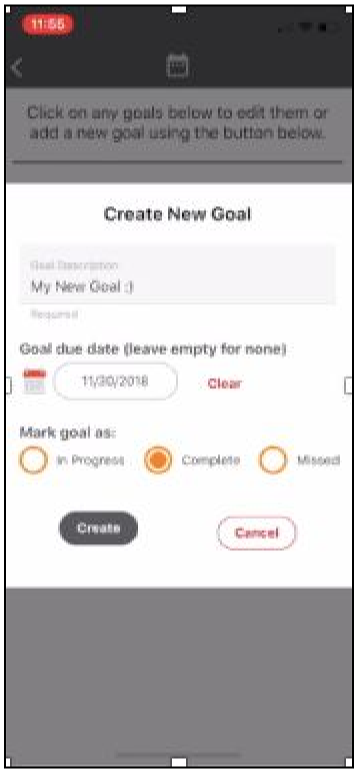


Figure 15. Search Screen (Android)

Goals and Goal Management

The final piece of front end is the goals, which are contained on the home screen, and the goal management screen, seen in *Figure 16*. Goals are first seen in *Figure 14* after form completion, and they come pre-populated with goals designed to teach the user about goal functions and how to use the manager. For example, the user starts with the completed goal of filling out the form. This shows them that green means complete and introduces them to the concept of checking off goals. Above this they will notice a missed goal stating "Feel free to delete this goal". This urges the user to experiment with the goal manager, and instructs them that not only is creating and editing goals possible, but also deleting. This goal also has a deadline, teaching the user that they can make time dependent goals. Above this is another set of goals labeled in progress, the first of which is "Search Mentor Profiles". This functions to direct them toward the newly available search screen, encouraging exploration. The idea of guiding, not forcing, users toward desired actions is key to any app with intuitive, pleasant user experience. In *Figure 16* we see the goal management screen with a goal selected to edit. Here, deletion, titling, and deadline setting can all be seen.



Click on any goals below to edit them or add a new goal using the button below.

Create New Goal

Goal Description
My New Goal :)
Required

Goal due date (leave empty for none)
11/30/2018

Mark goal as:
In Progress [✓ Complete] Missed

Figure 16. Goal Management Screen (iPhone)

Frontend Summary

When designing an app to help people it is important to make the app feel as helpful as possible. If a user has any reason to turn away or be rubbed wrong, they may just leave and uninstall. Because of this it was crucial to work extensively on user interface and user experience. By adhering to Material Design our app feels familiar, professional, and simple. Many sources mentioned that this app did not look "student-created" and that is simply because its component design was forged over a decade of application development trial and error that has culminated in the modern design standards. The branding and style selection clearly marks it as a University of Tennessee associated product, and it should be a happy welcome to incoming freshman. By silently gatekeeping, thoughtfully directing, encouraging exploration, gently reminding, and using a friendly tone, we make the app's user experience, intuitive, playful, comfortable, and easy. By focusing with such intensity on the user, we hope to assist EMP with increasing the retention rate of the discipline we love so much.

Lessons Learned

Many problems were encountered on both the frontend and backend development during the course of this project. This led to quite a bit learned on the development of a multi-platform mobile application, implementation of a backend database, and styling of each view in the app.

General Lessons

We learned a few lessons about the general process of building a mobile application. One of the first and most obvious lessons was that React Native was a very good choice. It was almost surprising sometimes how similar the app looked on both iOS and Android. There were a few times where we had to implement styling differently, but most of the application works exactly the same on both iOS and Android.

Another big lesson we learned was dealing with merge conflicts with such a wide code base. Pushing commits directly into our main development branch would run the risk of overwriting past commits. Our solution was a more professional approach by making a new pull request for each commit. This would allow each commit to be viewed and manually confirmed to not overwrite anything. After that was done, the pull request would be merged and closed.

The final and most important lesson that we learned revolved around third party applications and libraries. With mobile application development, specifically via React Native, we did not realize how much of a reliance there was on external libraries. This lead to several learning experiences, such as learning how to sift through a library's GitHub issues to fix a bug, how to avoid libraries with potential security holes, and how to stitch together multiple libraries to create a cohesive application.

Back End Lessons

The most experience that our team had in terms of back end came from James and Dakota who worked with Amazon Web Services during their internships. However, they never dealt with Cognito or DynamoDB, two of the main services we ended up using. Furthermore, the back end portion of the application did not directly communicate with AWS. The actual data transfer went through a third party library called Amplify, which James and Andrey devoted a lot of time to learning. Some of the biggest back end lessons that came out of struggling with AWS and React Native were learning how to deal with unfamiliar environments effectively, handling error response data from API calls, and designing a thorough database.

Front End Lessons

The main problems encountered with front end development were mostly based on the learning curve involved with styling the views. When styling, it was initially difficult to find relevant help as there are so many other Javascript frameworks with similar structure. This led to finding help, but it would have slightly different syntax or details that, obviously, would not apply to React Native. Sometimes, the front end developer would have to interpret the found solution and appropriate it to work with React Native components. Another issue that came up frequently was dealing with styling in general. The way the screens are set up, the first section of the file is the javascript that runs the actions and modules on the screen, the second section is the HTML which formats the screen, then the last section is a CSS stylesheet that the HTML pulls its colors and other fine details from. In the beginning, it was sometimes hard to understand which section was the correct section to add an addition to. Until the frontend developers were more comfortable and experienced with the project, it would sometimes take a while to do something as simple as center a button on the screen. Eventually basic styling tasks became second nature and basic styling could be done very easily.

Additional lessons came when working with navigating across screens in the app and passing necessary data between screens in order to minimize the number of GETs to our database. This, along with properly setting state for individual screens, took time to get accustomed to, as there were often race conditions.

When styling components of an app, there are infinite choices that can be made, and by referencing other application UI and material design standards, we learned volumes on how to create an engaging, intuitive experience that is both pleasant and familiar to the user. Becoming aware with these resources and component specifications was invaluable. Also, the hex value for UT’s smokey gray is #58595B and this is forever burned in our memory.

Team Member Contributions

James Cate

Solutions Architect, Lead Tester, Developer - Computer Science

James Cate was involved in design, frontend, and backend development. In terms of frontend development, he created the form screens, individual profile screen, and worked extensively with Andrey on various components of the home screen. In terms of backend, he did all input validation for all relevant screens and worked extensively with AWS. For input validation, he tested every screen of the app over multiple weeks trying to break them, adding error messages to all text fields and forcing that forms were satisfactorily completed. He found numerous holes throughout the app and patched them. In terms of Amazon Web Services, James worked directly with Andrey to set up all of the appropriate services (Cognito, DynamoDB, and Lambda) for the application through the Amplify library. All compliance with Google's Material design guidelines and UT Brand guidelines were enforced by James. In addition, James added all frontend portions, ethical concerns, and UI/UX focused points to this document and edited other sections extensively to adapt what was a final report into a more refined thesis.

Andrey Karnauch

Lead Developer, Designer, Backend Architect - Computer Science

Andrey Karnauch was heavily involved with the entire development process of the application. He got the ball rolling in terms of software development by designing both the frontend and backend of the Sign Up and Sign In screens over fall break. He was also the main architect of the Goal and Search screens. Furthermore, he managed all of the navigation within the application by creating the appropriate stack that redirected users based on their current and desired screen. Andrey also had his hand in every other view within the application. In terms of Amazon Web Services, Andrey worked directly with James to set up all of the appropriate services (Cognito, DynamoDB, and Lambda) for the application through the Amplify library. Lastly, he provided input and worked on all presentations and reports.

Matt Matto

Designer, Presenter, Cross Platform Tester - Computer Science

Matt was mostly assigned on working on the front end styling of the application. He styled the Terms and Conditions page. He also put the Terms and Conditions text into a .json file that could be read in by the program. One final thing he did was clear up a few warnings that would pop up when running the application. When preparing presentations, he created the wireframe used in a couple of the group’s presentations and assisted in creating the slides. Being the only person on the team with an Android device, he was the sole person responsible for testing the application on Android.

Dakota Sanders

Presenter, Trainer, Tester - Computer Science

Dakota Sanders was instrumental in handling all administrative details for the group. Any presentations, reports, or communication with the customer were handled by Dakota. Additionally, Dakota was responsible for some features within the application, namely the contact functionality and some quality-of-life features pertaining to handling user input. Dakota also ensured any new features were thoroughly tested for bugs and modified features to handle those bugs. Dakota also designed and formatted the final demo powerpoint to dynamically display the content of the application without having a mobile device open running the application. Dakota also created in-depth documentation to provide to the Engineering Mentor Program to ensure a smooth transition at the end of the semester.

Resources

"React Native Quickstart." Getting Started, facebook.github.io/react-native/docs/getting-started/.

"Color Palettes." Brand Guidelines, The University of Tennessee, brand.utk.edu/colors/palettes/.

"Design." Material Design, Google, material.io/design/.

AWS-Amplify and React, Amazon, aws-amplify.github.io/docs/js/react.