

Managing Anxiety through Mobile Application Training Suites

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

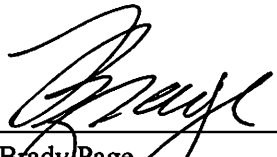
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Brady Page
Spring, 2020

Technical Project Team Members

Everett Adams
Jeffrey Gerken
Danielle Newman
William Ngu
Jacob Pacheco
Daniel Zarco

On my honor as a University Student, I have neither given nor received
unauthorized aid on this assignment as defined by the Honor Guidelines for
Thesis-Related Assignments

Signature  Date 5/5/2020
Brady Page

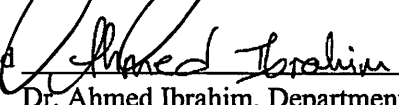
Approved  Date 4/23/2020
Dr. Ahmed Ibrahim, Department of Computer Science

Table of Contents

Abstract	3
List of Figures	4
1. Introduction	5
1.1 Problem Statement	5
1.2 Contributions	6
2. Related Work	7
3. System Design	9
3.1 System Requirements	9
3.2 Wireframes	11
3.3 Sample Code	13
3.4 Sample Tests	16
3.5 Code Coverage	19
3.6 Installation Instructions	21
4. Results	25
5. Conclusion	26
6. Future Work	27
7. References	29

Abstract

Our capstone team worked with MindTrails, an ongoing research initiative at UVA, in order to develop a mobile application to assist in the treatment of mental health patients. MindTrails utilizes cognitive bias modification, or CBM, in order to help their patients control their anxiety and develop healthy thinking habits. According to MindTrails, cognitive bias is the tendency to pay attention to, remember, and interpret things differently when processing information tied to emotional responses. Aside from helping their patients, the goal of MindTrails' research is to assess the efficacy of online interventions as a method of promoting positive thinking habits. The MindTrails team requested help creating a mobile application in order to increase user satisfaction and retention, as well as to provide training and interventions at critical moments that would not be possible with their current website.

Using React Native, our capstone team was able to design a mobile application that integrates many of the features that currently exist on the MindTrails website into a cross-platform mobile app. We utilized local notification scheduling systems to address concerns of user retention in addition to using local storage to allow for increased content availability. By addressing the issues of user retention and content availability within the MindTrails training suite, the research initiative is able to be more effective. The data collected through the suite can be more representative of the whole user base, helping future studies be better tailored to the needs of the MindTrails users. Additionally, the practical benefits of the training suite can be constantly available to users, increasing the overall benefit that MindTrails can offer.

List of Figures

Figure 1: Wireframes for word fill questions.....	12
Figure 2: Wireframes for yes/no question.....	12
Figure 3: Code that renders and updates the letters answers for word fill questions.....	13
Figure 4: State and styling initialization for buttons.....	14
Figure 5: Code to render the title, question, word for word fill questions.....	14
Figure 6: Code to handle answers from users and carry out corresponding logic.....	15
Figure 7: Code to return message in response to user answer.....	15
Figure 8: Code to render the question and options.....	16
Figure 9: Test to ensure that accounts will not be created when requirements are not met.....	17
Figure 10: Test to ensure that only eligible participants can go through the Calm Thinking program according to an initial questionnaire.....	17
Figure 11: Test to ensure that participants are alerted if their passwords do not match when creating an account.....	17
Figure 12: Test to ensure that a potential participant with a high enough initial questionnaire score is eligible for the Calm Thinking program.....	18
Figure 13: Test to ensure that checkboxes are initialized and updated as expected.....	18
Figure 14: Code coverage report for the components folder.....	20
Figure 15: Code coverage report for the components/FormlyComponents folder.....	20
Figure 16: Code coverage report for the components/Information folder.....	20
Figure 17: Project page of Expo app.....	24

1. Introduction

According to the World Health Organization, “the burden of depression and other mental health conditions is on the rise globally,” and “fewer than half of those affected in the world receive [effective] treatment” (WHO, 2018). MindTrails, a UVA research initiative, is aiming to increase the availability of treatment for those who suffer from these conditions. The sessions offered through MindTrails focus around a type of intervention called cognitive bias modification, or CBM. Cognitive bias is the tendency to interpret events differently when processing information tied to emotional responses. CBM trains users to better separate events and information from emotional responses that can skew interpretation in order to develop healthier ways of thinking. The American Psychological Association analyzed the effectiveness of short training sessions on anxiety and found that “after just eight 15-minute sessions... 72 percent of patients in the treatment group no longer met diagnostic criteria for social anxiety disorder, compared with 11 percent of patients in the control group” (Weir, 2011).

1.1 Problem Statement

Currently, MindTrails offers their CBM training through a website platform. While the MindTrails team has had success with getting users to register, they face problems retaining users until the end of the study. They have tried to increase user retention by sending reminders through email or text messages and by offering gift card rewards for reaching session milestones. While these actions moderately increased retention, the gift card rewards do not keep enough participants in the study to warrant their cost as the program expands. Another key issue of the current MindTrails training suite revolves around the need for constant network access.

MindTrails wishes to offer interventions that can be available whenever users experience spikes in anxiety or stress. However, users are not always in contact with working Wi-Fi or cellular networks, cutting them off from receiving help from the website.

1.2 Contributions

Our capstone team worked with MindTrails in order to create a mobile application that would address the issues of user retention and content availability. The frontend of our mobile app was created using React Native and Expo to allow for compatibility with Android and iOS platforms. The backend of the app was created using Flask and a PostgreSQL database. Our app addresses concerns about user retention by utilizing native notification systems to alert users that new sessions are available or to remind them to take training after a certain amount of time has passed. In order to increase content availability, we utilized local storage to cache a subset of training sessions when connected to Wi-Fi or cellular networks. This allows users to continue using the app for a period of time while the app is disconnected.

2. Related Work

There are many services that offer functionalities needed by the MindTrails team. For example, many other UVA research studies use services like Qualtrics in order to collect user responses in an efficient manner. However, these data collection services would not be sufficient for MindTrails due to their inability to seamlessly integrate the anxiety reduction training developed by the MindTrails team. Additionally, MindTrails offers its services to anyone in the general public, including international users, who may not have access to UVA offered services. Services that are offered externally from UVA would not be a valid option for the MindTrails team either. Due to the nature of the study and the security protocols of collecting and storing participant data, a UVA owned and operated system would be necessary in order to comply with HIPAA practices and University rules. Lastly, the content and practices of the MindTrails training suite are dynamically changing as the study progresses. Changing the available content and adding new features can be done more efficiently using an in-house solution that the MindTrails team can edit directly. As a result, the services required by the MindTrails team are unique as a whole, even though some individual components of these services already exist within the research space.

The current iteration of the MindTrails website is insufficient due to issues with keeping users engaged within the study. Many users experience burn out within the study and abandon the training before its completion. This results in a considerable loss of data for the MindTrails team. One key aspect of this issue revolves around the structure of the MindTrails training suite. The suite is broken up into sessions that must be completed with specified breaks in between each session. Once a particular session is completed, users have to wait days, and sometimes

weeks, in order to complete the next stage of the program. As a result, the MindTrails team needs to capture their users' attention continuously throughout the program once these breaks are completed. The MindTrails team has used reminders sent out through email and text messages to notify users that they can start their next session, in addition to offering gift card rewards for completing consecutive sessions. The current user retention rates experienced by MindTrails highlight the limitations of these techniques.

MindTrails has tasked our capstone team with developing a mobile app in the hopes of raising user retention rates. The app aims to solve retention issues by increasing engagement and ease of access to MindTrails content through the use of a mobile interface. In the last few years, there had been a rise in the number of apps in the mental health space. Currently, an app exists for nearly every mental health condition; however, due to the individualistic nature of mental health conditions and treatments, no app has emerged as a household name. MindTrails stands out in the current mental health app market due to their university sponsored background. Combining this with a free-to-use and cross-platform app, MindTrails can expand their studies and beneficial training to a wider audience.

3. System Design

The purpose of our system is to allow users to take training and questionnaires provided by the MindTrails team. MindTrails is currently researching cognitive bias modification through their website and wants to expand into the mobile market. Through a mobile application, users can be given a more accessible and simplified user interface. To address attrition rate concerns, our application will utilize local notification systems to remind users of available training. Our application's frontend is built using React Native, a Facebook open source framework. This framework enables us to create a native application that utilizes both iOS and Android native components with minimal changes to our main code base. The backend of our application is written in Flask, a Python framework. Flask was chosen by the MindTrails team due to its ease of use, minimal learning curve, and prior experience within the team. Our system's code is available under the MIT license.

3.1 System Requirements

Gathering requirements was important for the overall success of our project. The requirements that we gathered from the MindTrails team determined the work done within our capstone project. Quality requirements are essential to ensure that the problems that originally created the desire for a mobile MindTrails app are resolved in our solution.

In the first few months of our capstone project, we worked with the MindTrails team in order to establish a list of requirements for the application. Although the requirements were subject to change at the time we collected them, they provided us with a solid base to begin

working on the system. The final requirements are broken into three sections: minimum, desired, and optional.

Minimum Requirements: Functionality required by the end of the Fall 2019 semester

- As a User, I should be able to login using a username and password
- As a User, I should be able to log out
- As a User, I should be able to take trainings
- As a User, I should be able to take questionnaires
- As a User, I should be able to view and edit account information (which includes name, email address, and phone number)
- As a User, I should be able to leave a training or questionnaire and resume it at a later time

Desired Requirements: Functionality required by the end of the Spring 2020 semester

- As a User, my user data (which includes training and questionnaire answers) should be consistent across Android and iOS platforms
- As a User, my training and questionnaire progress should be saved across sessions
- As an Admin, I should be able to see a dashboard displaying the number of users and their current location in the study
- As an Admin, I should be able to search through users based on email or name
- As an Admin, I should be able to view user data (which includes training, questionnaire, and phone sensor data)

- As an Admin, I should be able to export user data (which includes training, questionnaire, and phone sensor data)
- As an Admin, I should be able to view the time users spent on individuals questionnaires and trainings

Optional Requirements: Functionality that is wanted, but not required

- As a User, I should be able to use wearable technology within the app
- As a User, I should be able to login using fingerprint or face ID (on applicable devices)
- As a User, I should be able to manually sync progress between web and mobile platforms
- As a User, I should be able to opt into the collection of my phone sensor data

3.2 Wireframes

Wireframes were also important for our capstone team as they provided a reusable, flexible, and easy to understand goal to base our frontend design on. The wireframes enabled our capstone team and the MindTrails team to come to an agreement on how various UI elements should be laid out and what actions they should be able to perform. This greatly increased our ability to communicate our ideas to the MindTrails team and allowed us to move forward with the project quickly. The following page displays some examples of the wireframes used within our capstone project (Figures 1 and 2).

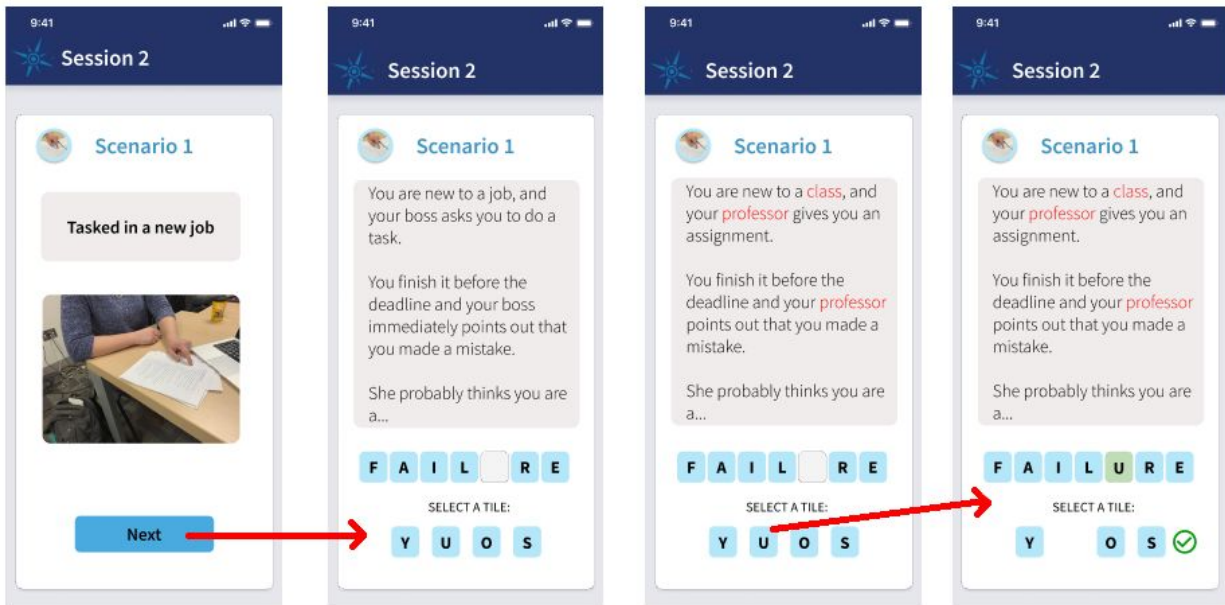


Figure 1: Wireframes for word fill questions

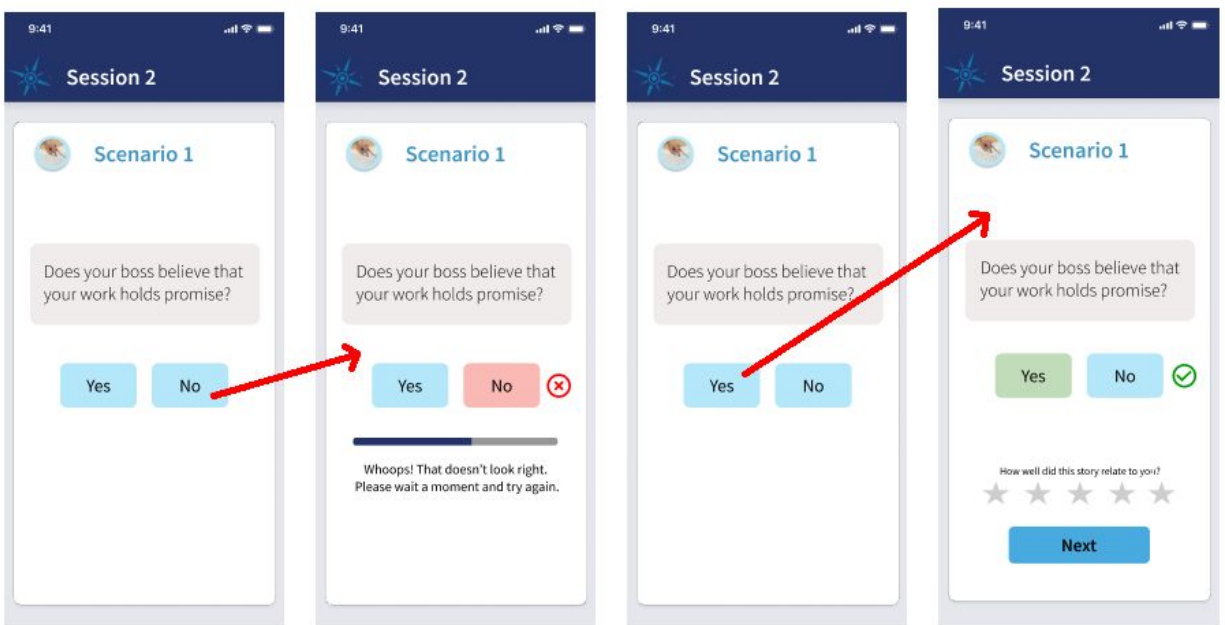


Figure 2: Wireframes for yes/no question

3.3 Sample Code

The following code snippets (Figures 3, 4, and 5) are taken from the word fill component, which can be found in `src/frontend/components/FormlyComponets/WordFill.jsx`.

```
render() {
  const { config, model, onChange } = this.props; // destruct props
  const { buttonStyles, word } = this.state; // destruct props
  const to = config.templateOptions; // template options from json

  // map each letter option to a rendered button
  const choices = to.options.map((option, i) => (
    <Button
      onPress={() => {
        if (option !== to.answer) { // if letter is incorrect
          const currStyles = buttonStyles;
          currStyles[option].disabled = true; // change button style to disabled
          this.setState({ buttonStyles: currStyles }); // set button styles in state
        } else { // if answer is correct
          this.setState({ word: to.filledWord });
          setTimeout(() => {
            model.questionIndex += 1; // go to next question after 2 seconds
            onChange();
          }, 2000);
        }
      }}
      buttonStyles={{ color: buttonStyles[option].color }}
      key={i}
      title={option}
      style={styles.choice}
      type={buttonStyles[option].type}
      disabled={buttonStyles[option].disabled}
    />
  ));
}
```

Figure 3: Code that renders and updates the letters answers for word fill questions

```

getInitialState() {
  const { config } = this.props;
  const buttons = {};

  // each button has its type/disabled status stored in buttons
  config.templateOptions.options.forEach((element) => {
    buttons[element] = {
      type: 'solid',
      disabled: false,
    };
  });

  return {
    buttonStyles: buttons, // object that maps a letter to its buttons styles
    word: config.templateOptions.word, // unfilled word from form json
  };
}

```

Figure 4: State and styling initialization for buttons

```

return (
  <View style={styles.container}>
    <Card>
      title={<Text style={{ textAlign: 'center', fontWeight: 'bold', fontSize: 32, marginBottom: '5%' }}>
        {to.title}
      </Text>
      containerStyle={styles.buttonContainer}
      borderRadius={5}
    >
      <Text style={{ alignSelf: 'center', textAlign: 'center', fontSize: 20, width: '90%' }}>
        {to.body}
      </Text>
      <Text style={{ textAlign: 'center', fontSize: 24, marginVertical: '10%' }}>
        {word}
      </Text>
      <View style={styles.choices}>
        {choices}
      </View>
    </Card>
  </View>
);

```

Figure 5: Code to render the title, question, word for word fill questions

The following code snippets (Figures 6, 7, and 8) are taken from the yes / no component, which can be found in src/frontend/components/FormlyComponets/YesNo.jsx.

```
getInitialState() {
  return {
    incorrect: false, // user answered incorrectly
    correct: false, // user answered correctly
  };
},

// Displays red or green message to user depending on their answer
handleAnswer(correctAnswer) {
  const { config, model, onChange } = this.props;
  const to = config.templateOptions;
  if (to.yesNoAnswer === correctAnswer) { // if answer is correct
    this.setState({ correct: true });
    setTimeout(() => {
      model.questionIndex += 1; // move to next question
      onChange();
    }, 5000);
  } else { // if it is not correct
    this.setState({ incorrect: true });
    setTimeout(() => {
      this.setState({ incorrect: false }); // reset incorrect in state so red message disappears
    }, 5000);
  }
},
```

Figure 6: Code to handle answers from users and carry out corresponding logic

```
// renders proper red or green message depending on correctness of user response
renderReply() {
  const { incorrect } = this.state;
  return incorrect
    ? (
      <Text style={{ fontSize: 20, color: 'red', paddingHorizontal: '8%', textAlign: 'center' }}>
        That response is incorrect. In a moment, you will have a chance to respond again.
      </Text>
    ) : (
      <Text style={{ fontSize: 20, color: 'green', paddingHorizontal: '8%', textAlign: 'center' }}>
        Great job!
      </Text>
    );
},
```

Figure 7: Code to return message in response to user answer

```

render() {
  const { config } = this.props; // destruct props
  const { incorrect, correct } = this.state; // destruct state
  const to = config.templateOptions; // template options from json
  return (
    <View style={styles.container}>
      <Card
        title={<Text style={{ textAlign: 'center', fontWeight: 'bold', fontSize: 32, marginBottom: '5%' }}>
          {to.title}
        </Text>
        containerStyle={styles.buttonContainer}
        borderRadius={5}
      >
        <Text style={{ fontSize: 20, paddingBottom: '5%', textAlign: 'center' }}>{to.yesNoBody}</Text>
        {!incorrect && !correct ? ( // if user has not responded yet, incorrect and correct should both be false
          <View style={styles.choices}>
            <Button
              onPress={() => this.handleAnswer('Yes')}
              style={styles.choice}
              title="Yes"
            />
            <Button
              onPress={() => this.handleAnswer('No')}
              style={styles.choice}
              title="No"
            />
          </View>
        ) : this.renderReply()}
      </Card>
    </View>
  );
}

```

Figure 8: Code to render the question and options

3.4 Sample Tests

Tests are essential in software development as they help assure the developers and customers alike that code is working as desired and that desired requirements are being met. Although no amount of testing can ensure perfect code, well-written tests help to signal developers that a feature has been sufficiently completed. The following pages display some examples of the tests used within our capstone project (Figures 9, 10, 11, 12, and 13).


```

it('Create Account fail: conditions not met', async () => {
  const login = renderer.create(<CreateAccount navigation={navigation} />).getInstance();
  login.state = {
    passwordsMatch: false, // non-matching password
    legalAge: false, // not 18
    notRobot: false, // robot
  };
  await login.register(); // try to create new account
  expect(login.state.failedRegistration).toEqual(true); // should fail
});

```

Figure 9: Test to ensure that accounts will not be created when requirements are not met

```

it('Eligibility Questionnaire not eligible', async () => {
  const eli = renderer.create(<EligibilityQuestionnaire navigation={navigation} />).getInstance();
  eli.state = {
    eligibleAnswers: {
      // answers that will lead to ineligibility
      0: 0,
      1: 0,
      2: 0,
      3: 0,
      4: 0,
      5: 0,
      6: 0,
    },
    over18: true,
  };
  const response = await eli.checkEligible(); // pass answers to eligibility endpoint
  expect(response.eligible).toEqual(false); // should not be eligible
});

```

Figure 10: Test to ensure that only eligible participants can go through the Calm Thinking program according to an initial questionnaire

```

it('Create Account passwords do not match', () => {
  const tree = renderer.create(<CreateAccount navigation={navigation} />).getInstance();
  tree.state.passwordsMatch = false; // passwords do not match
  const newTree = tree.render(); // re-render, error text should appear
  expect(newTree).toMatchSnapshot();
});

```

Figure 11: Test to ensure that participants are alerted if their passwords do not match when creating an account

```

it('Eligibility Questionnaire eligible', async () => {
  const eli = renderer.create(<EligibilityQuestionnaire navigation={navigation} />).getInstance();
  eli.state = {
    eligibleAnswers: {
      // answers that should lead to eligibility
      0: 40,
      1: 40,
      2: 40,
      3: 40,
      4: 40,
      5: 40,
      6: 40,
    },
    over18: true,
  };
  const response = await eli.checkEligible(); // pass answers to eligibility endpoint
  expect(response.eligible).toEqual(true); // should be eligible
});

```

Figure 12: Test to ensure that a potential participant with a high enough initial questionnaire score is eligible for the Calm Thinking program

```

it('Regular Checkbox onPress', () => {
  // grabbing the first checkbox
  const regularCheck = foundChildren[0];
  const checkOnPressFunc = regularCheck.props.onPress;
  let isSelected = regularCheck.props.checked;
  const { state } = root.instance;
  // before calling on press it should be unchecked
  expect(isSelected).toBeFalsy();
  // making sure no answers are saved in the state right now
  expect(state.model['Regular Checkbox']).toBeFalsy();
  act(() => {
    checkOnPressFunc();
  });
  // expect answers to be saved in the state
  expect(state.model['Regular Checkbox']).toBeTruthy();
  // after calling on press it should be checked
  isSelected = regularCheck.props.checked;
  expect(isSelected).toBeTruthy();
});

```

Figure 13: Test to ensure that checkboxes are initialized and updated as expected

3.5 Code Coverage

In order to test our application, we utilized Jest. Jest is a JavaScript testing framework that provides us with an easy to understand and automatically generated code coverage report. Some examples of code coverage report pages generated by our Jest setup are shown on the following page (Figures 14, 15, and 16). In order to set up automated code coverage reports using Jest, follow the directions below.

- From the MindTrails/src/frontend directory of the project, run the “npm install” command to ensure that all dependencies needed for Jest are currently installed.
- Run the “npm test” command in order to run all tests using Jest
- Move to the MindTrails/src/frontend/__tests__/lcov-report directory of the project.
- Open up index.html in a browser. The resulting page will show the code coverage of various files organized by folder. Folders can be opened to look at more specific results at a file-by-file basis. Information such as the number of statements, branches, and functions covered will be shown for each file.

All files components

94.01% Statements 157/167 88.89% Branches 32/36 96.15% Functions 50/52 94.01% Lines 157/167

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements	Branches	Functions	Lines
CreateAccount.jsx	<div><div></div></div>	94.87% 37/39	93.33% 14/15	100% 18/18	94.87% 37/39
EligibilityQuestionnaire.jsx	<div><div></div></div>	97.37% 37/38	100% 2/2	100% 8/8	97.37% 37/38
FormWrapper.jsx	<div><div></div></div>	87.23% 41/47	81.82% 9/11	80% 8/10	87.23% 41/47
Home.jsx	<div><div></div></div>	100% 6/6	100% 0/0	100% 3/3	100% 6/6
Login.jsx	<div><div></div></div>	95.45% 21/22	100% 4/4	100% 7/7	95.45% 21/22
Progress.jsx	<div><div></div></div>	100% 15/15	75% 3/4	100% 6/6	100% 15/15

Figure 14: Code coverage report for the components folder

All files components/FormlyComponents

91.33% Statements 158/173 81.15% Branches 99/122 90.74% Functions 49/54 91.57% Lines 152/166

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements	Branches	Functions	Lines
Multiselect.jsx	<div><div></div></div>	89.47% 85/95	80.61% 79/98	89.66% 26/29	89.77% 79/88
Panel.jsx	<div><div></div></div>	100% 16/16	70% 7/10	100% 5/5	100% 16/16
SelectPanel.jsx	<div><div></div></div>	88.89% 16/18	75% 3/4	100% 6/6	88.89% 16/18
WordFill.jsx	<div><div></div></div>	100% 22/22	100% 2/2	100% 6/6	100% 22/22
YesNo.jsx	<div><div></div></div>	86.36% 19/22	100% 8/8	75% 6/8	86.36% 19/22

Figure 15: Code coverage report for the components/FormlyComponents folder

All files components/Information

100% Statements 33/33 100% Branches 12/12 100% Functions 21/21 100% Lines 33/33

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements	Branches	Functions	Lines
About.jsx	<div><div></div></div>	100% 6/6	100% 0/0	100% 4/4	100% 6/6
ContactUs.jsx	<div><div></div></div>	100% 5/5	100% 0/0	100% 3/3	100% 5/5
Eligible.jsx	<div><div></div></div>	100% 7/7	100% 0/0	100% 5/5	100% 7/7
Ineligible.jsx	<div><div></div></div>	100% 8/8	100% 0/0	100% 6/6	100% 8/8
Level.jsx	<div><div></div></div>	100% 5/5	100% 12/12	100% 2/2	100% 5/5
ResearchSupport.jsx	<div><div></div></div>	100% 2/2	100% 0/0	100% 1/1	100% 2/2

Figure 16: Code coverage report for the components/Information folder

3.6 Installation Instructions

Since we are building a mobile app, the app would ideally be installed from the App Store or Google Play Store. Therefore, these installation instructions are for starting a development tunnel for Expo that would serve the React Native frontend.

Assumptions

- Node 10 LTS or greater is installed
- Amazon Web Services Account
- AWS CLI installed
(<https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html>)
- AWS Elastic Beanstalk CLI installed (<https://github.com/aws/aws-elastic-beanstalk-cli-setup>)

Installing the Expo Simulator

- Download the Expo app on your Android or iOS device. It's available on the Google Play Store and on the iOS App Store
- Create a free Expo Account
- Note: Android or iOS device must be connected to the same WiFi as your Expo tunnel

Starting Expo

- On the machine you will be developing on, use npm to install the Expo CLI command line utility: `npm install -g expo-cli`
- From unzipped code source folder named MindTrails-[version] run `cd src/frontend`

- Run npm install to install project dependencies
- Inside the frontend folder there is a file named package.json that contains a list of scripts/commands available for starting an expo tunnel and running tests.
- To start the expo tunnel, run expo start
- A window should open up in your browser displaying information about the expo tunnel
- Take note of the localhost port that expo opens for you (e.g. http://localhost:19002)
 - Once the Expo Developer page is open, several options will be displayed for how to run the app locally

Local Simulator Options

- Easiest - Tunnel, LAN, or Local Connection
 - Choosing any of these three options will result in a URL and QR Code being generated within the Expo Developer page in the bottom left corner of the screen.
 - On Android, open the Expo App, go to Projects and scan the QR Code. For iOS, open up the native camera app on your phone and scan the QR code.
- Android device/emulator
 - On the Expo Developer page, click Run on Android device/emulator
 - This will attempt to connect to a Android device nearby or running locally as an emulator
 - Note: An Android emulator must already be installed in order to use this option
- iOS Simulator
 - On the Expo Developer page, click Run on iOS Simulator

- This will attempt to connect to a locally running iOS emulator
- Note: An iOS emulator must already be installed in order to use this option

Backend Flask Installation

- Clone this repository: `git clone https://github.com/uva-cp-1920/MindTrails.git`
- In your terminal, navigate to `Mindtrails/src/backend`
- To double-check you are in the correct directory, make sure the file "application.py" is contained in your current directory
- Initialize a new AWS Elastic Bean repository with the command `eb init -p python-3.6 flask-tutorial --region us-east-2`
- Create a new Elastic Bean environment and add the Flask app to it with the command: `eb create mindtrails-flask-env`
- It may take several minutes for the AWS environment to be fully set up, the console will output when the process has finished
- Once your environment has been successfully created, you can access the backend by running the command: `eb open` inside the backend directory

Demo Overview

- This section includes instructions for the customer/white team to access and test the app

How to access the App:

- Download the Expo app on the Google Play Store or Apple Store
- Open the Expo app and login in using these credentials:
 - Username: MindTrails
 - Password: MindTrails2020!
- Ensure you are in the Profile section in the expo navbar (see Figure 17)

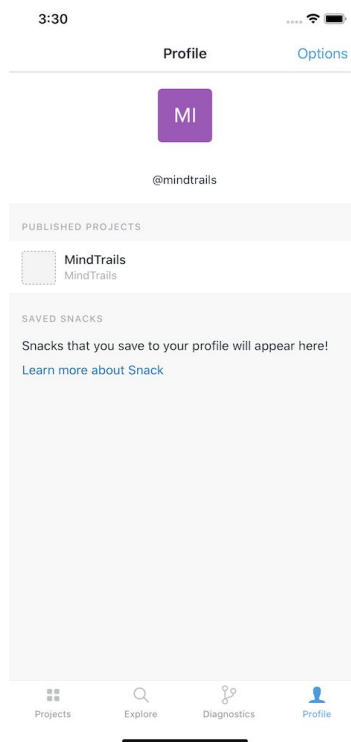


Figure 17: Profile page of Expo app

- Click on MindTrails under Published Projects to get the most recent build

4. Results

From our finished capstone project, there is still work that needs to be done in order to have a MindTrails app that can be released to the public. Due to MindTrails' position as a psychology research project that is open to the public, the application would need to be enhanced to better deal with sensitive mental health information collected from participants. Special precautions would have to be in place such as storing certain data on UVA servers and making sure that data stored externally has no personally identifiable information.

Retention rates were the main measure that our team concentrated on. This was due to the importance of collecting complete sets of data from each participant for the Calm Thinking program. Although our capstone team and MindTrails both believe that the finished capstone project will help address the concern of high user attrition rates, it would take a substantial amount of time to specifically measure these rates. Due to the structure of the program, an individual participant would need at least 80 days to complete the entire study in the best-case scenario. Our capstone team would have had to finish the project within a single semester in order to release our product with enough time to collect comparable data on participant attrition rates. However, there is information related to the benefits that our application can bring to the MindTrails user base. MindTrails states that a large majority of its users (66.3%) use a mobile interface at some point during the study. There are also high activity/focus users (57.7%) that primarily complete the training through the mobile website interface. Our application was developed in order to directly increase retention and accessibility for this crowd (and also those that use both mobile and website platforms). As a result, the accuracy and effectiveness of the MindTrails research study can benefit from the increased ease of access for their mobile users.

5. Conclusion

As a result of our capstone project, a solid base is in place for the MindTrails team to continue to develop this app. Although our final product was a proof of concept, we included many aspects of dynamic formatting and UI styling to allow the MindTrails team to continue building the app without having to dive too much into the actual code of the system. The mobile functionality of our application allows the MindTrails team to gather more information about their users than what could be done through their website alone, allowing them to better adapt their study to fit the needs of their users. Additionally, the app allows users greater accessibility than the current website. This should increase retention rates among users, leading to more effective treatment results for the users, and collecting more relevant data for the MindTrails team. Overall, the designs of our application allow the MindTrails team to expand their study into the mobile marketplace, helping them disseminate the benefits of their study among a wider audience that did not previously have access to university-backed mental health services.

6. Future Work

One of the original goals of our capstone project was to enhance the current system's ability to collect user data. A requirement to utilize Sensus, an end-to-end system for mobile sensing developed at UVA, was put in place to accomplish this goal. Due to time constraints within our capstone course and the complexity of integrating Sensus, this requirement was eventually dropped by the MindTrails team. However, Sensus integration still remains as a long-term goal for MindTrails as it would allow for the collection of new types of data and triggers that could be programmed to target users for new training sessions automatically. Another long-term goal of the MindTrails team involves integrating wearable technology, such as the Apple Watch, into our application. Similar to Sensus, this goal would help to increase visibility of the app towards the user and help MindTrails collect more data to revise their study.

Our application has a backend that is currently isolated from the rest of the MindTrails database. This choice was made during the beginning of our project due to the issues surrounding our team accessing the website's backend that holds sensitive medical information about its users. The MindTrails team stated that having a unified backend between the mobile app and the main website is a major priority, but that it would not be feasible during our capstone project. Having two separate databases for mobile and website users makes the system less convenient for those switching between platforms and makes data cleaning and analysis more difficult. A unified backend would resolve these issues and would make expanding the MindTrails system to incorporate other changes much easier.

During our capstone project, MindTrails was also working with a capstone team in the Systems Engineering department in order to design a new look and feel for their current website

and our upcoming application. It was difficult for us to include the Systems team's work into our own since they were being done at the same time. We were already having concerns about time and trying to work with another capstone team would have increased this concern. The new designs and gamification elements made by the Systems team still remain as a major goal for the MindTrails team.

7. References

Weir, Kirsten (2011). Behavior change in 15-minute session?, 42(10), 42.

<https://www.apa.org/monitor/2011/11/behavior-change>

WHO. (2018). World Health Organization. Depression.

<https://www.who.int/en/news-room/fact-sheets/detail/depression>