

FLORIDA INTERNATIONAL UNIVERSITY



Capstone II Final Presentation Spring 2024

Stress and Anxiety Management Application

Team Member(s): Alexandr Motovilov, Lyn Quintana, Ian Rodriguez, Oghenetega Idogun, Rafael Ojeda

Product Owner(s): Liane Sangollo
Professor: Masoud Sadjadi

School of Computing and Information Sciences
Florida International University

Team Members

- Lyn Quintana, Frontend Designer
- Ian Rodriguez, Lead Developer
- Rafael Ojeda, Lead Developer
- Alexandr Motovilov
- Oghenetega Idogun

Product Owner: Liane Sangollo, HR Vision Consulting

Instructor/Faculty: Dr. Masoud Sadjadi, Florida International University

Understanding The Problem

- **Issue Identification:** Increasing prevalence of stress and anxiety among populations worldwide, impacting daily functioning.
- **Resource Gap:** A noticeable lack of accessible tools that effectively assist individuals in managing mental health challenges in their daily lives.
- **Project Goal:** To develop "HowRU.life," an interactive application designed to aid users in assessing and managing their mental health through scientifically-backed practices.
- **Target Audience:** Individuals across diverse age groups and backgrounds, aiming to provide a universally accessible mental health resource.

Agile Methodology and Project Coordination

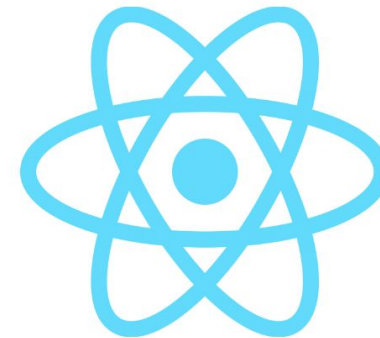
- **Development Strategy:** Adoption of agile methodologies to enhance flexibility and responsiveness to feedback throughout the development process.
- **Sprint Dynamics:** Strategic planning sessions to refine project goals, address emerging challenges, and adjust priorities based on iterative feedback.
- **Team Composition:**
 - **Product Owner:** Liane Sangollo - Guides the project's strategic direction and prioritization.
 - **Academic Supervisor:** Dr. Masoud Sadjadi - Offers academic oversight and technical advice.
 - **Project Team:** Includes Lyn Quintana, Ian Rodriguez, Oghenetega Idogun, Rafael Ojeda, Alexandr Motovilov - Tasked with the design, development, and testing phases.
- **Tools and Practices:** Utilization of GitHub for collaborative coding, Discord for team communication, and Figma for project management and task tracking.

Application Use Cases and Functionalities

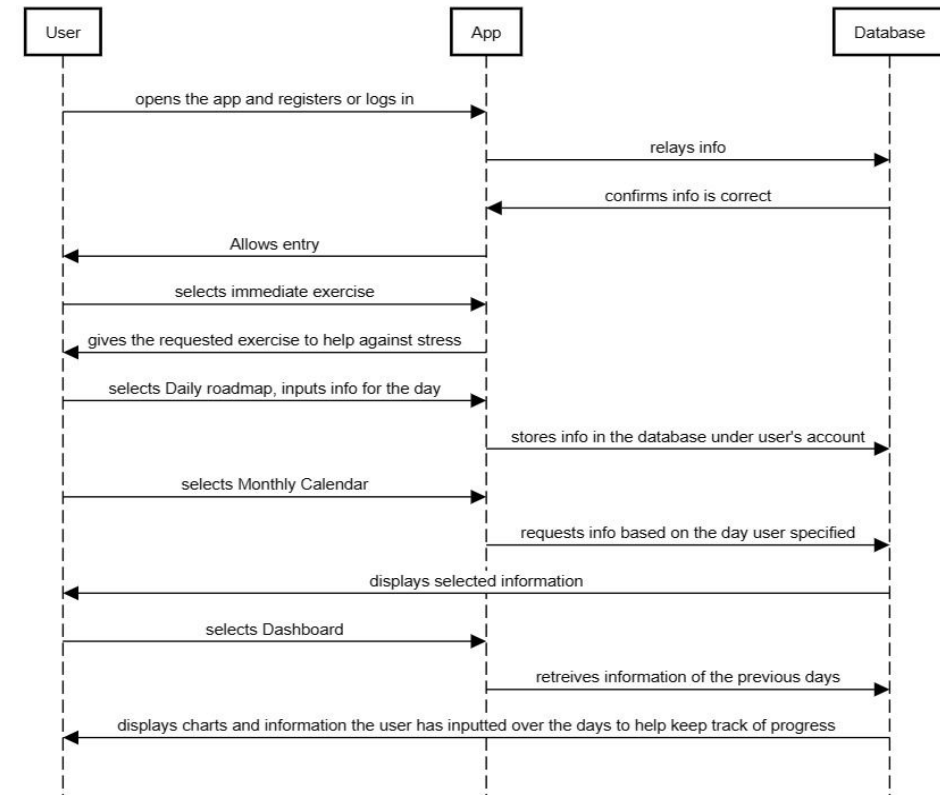
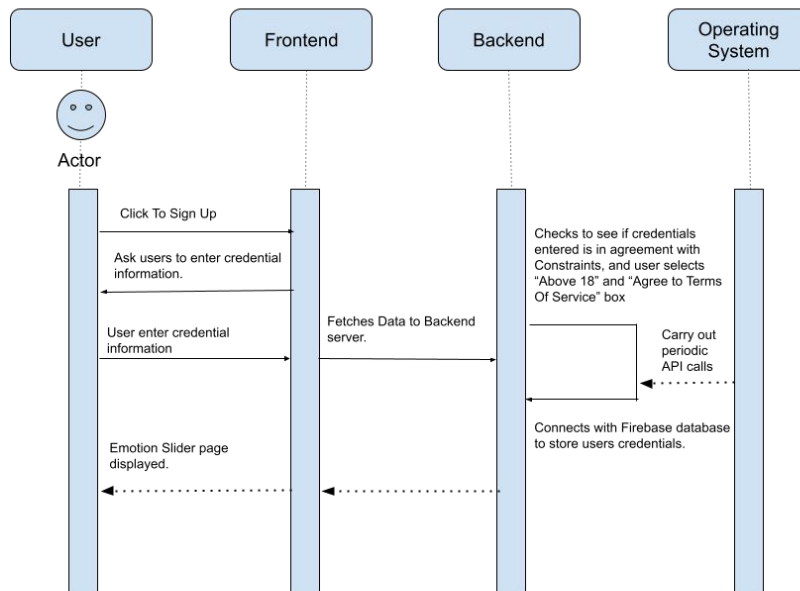
- Users can assess their current emotional state and receive personalized suggestions for stress and anxiety management.
- The application is designed to be accessible via both desktop and mobile platforms, ensuring usability across various devices.
- Streamlining the user interface by removing non-essential elements to enhance ease of use.
- Development of a feature that enables users to access historical data on demand, facilitating personal progress tracking.
- Implementation of data visualization tools to help users identify patterns in their emotional responses over time.
- Ensuring a fluid user experience with intuitive navigation between various application screens.

Application Requirements

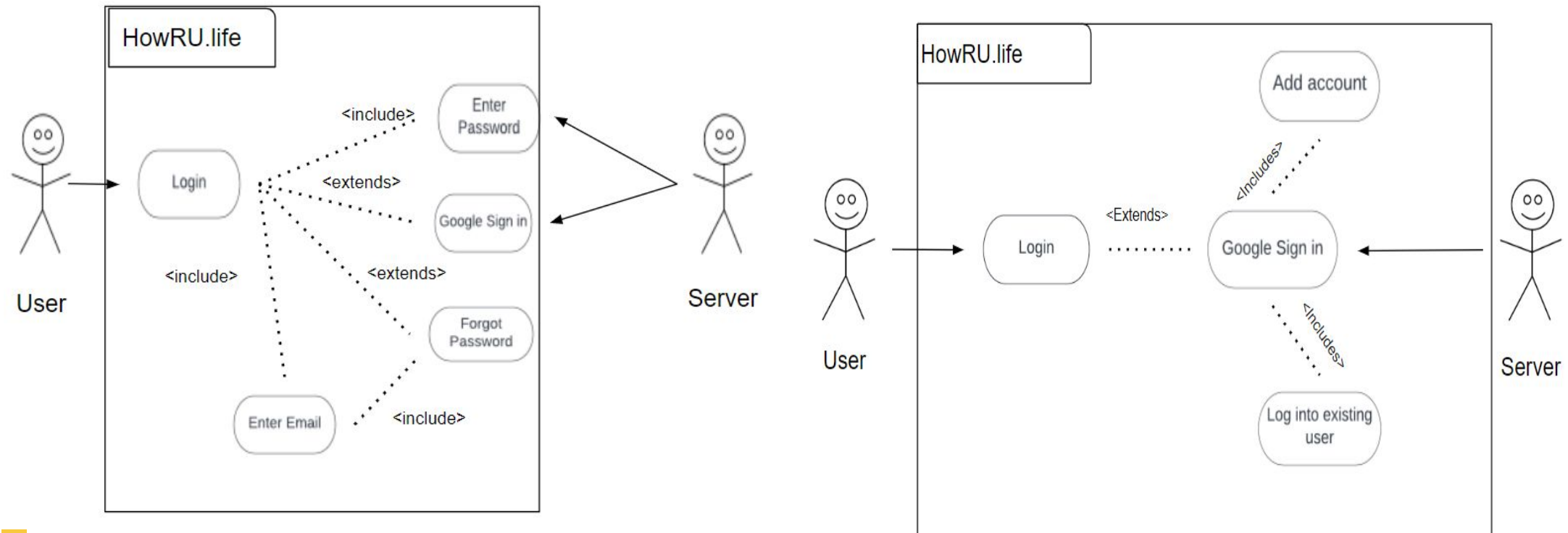
- Node.js v20
- Express.js backend framework
- Firebase for database and authentication
- React v18 for frontend
- HTML5, JavaScript, CSS for web technologies
- Modern web browser (e.g Mozilla Firefox, Google Chrome)



APPLICATION ARCHITECTURE OVERVIEW



DEPLOYMENT STRATEGY & INFRASTRUCTURE



Core Class Design Overview

- **Class Diagram Highlights:**
 - **User Class:** Manages user profiles including authentication data and session management.
 - **Survey Class:** Handles dynamic creation and management of surveys used for assessing user stress levels.
 - **Analytics Class:** Aggregates and analyzes user data to provide insights into stress patterns and app usage.
 - **Exercise Class:** Represents different types of exercises and activities recommended to the user.
- **Relationships:**
 - The User class interacts with the Survey and Analytics classes to fetch and store data relevant to user sessions.
 - Exercise class is used by the Analytics class to recommend personalized activities based on user responses and historical data.

PROJECT CODE OVERVIEW

File Structure Organization Rules

The Stress and Anxiety Management web app is quite large. Therefore a structure must be kept in order to maintain readability and to easily find everything. The **public folder** is to be able to easily access a document from anywhere in the application. Things like audio/images and other files that would be posted on the web app.

- **server folder**
 - The server folder is meant for anything that will be communicating with the database/APIs ect. The information for the database is stored in a .env file that your team would create and reference throughout the folder. This was not added to the project due to security.
 - All API endpoints are created in the main.js
- **src folder**
 - Assets/ is for the images/audio *Components When creating pages in the components folder here are some things to keep in mind:
 - Each page has its own folder within the components/
 - Specific components for a page should be in a folder within the page's folder.
 - Components that are in many pages i.e. navigation menu should be within the GeneralComponents folder.
- **Styling folder**
 - Used for any global commonly used styling throughout the whole web app.



Directory Quick Look:

```
howru/  
├── public/  
├── server/  
│   └── src/  
│       ├── firebase.js  
│       └── main.js  
└── src/  
    ├── Assets/  
    ├── Components/  
    └── Styling/
```

Startup Commands:

```
# Start client  
~ cd howru/  
~ npm install  
~ npm start  
  
# Start server  
~ cd server/  
~ npm install  
~ npm run dev
```

License: BSD-Clause 3

Github Link: <https://github.com/IRODR1502020/HOWRU-Spring-2024>

KEY SYSTEM COMPONENTS:

AccountNavbar Class

The `AccountNavbar` is a React component designed to provide a navigational interface for a user account area within a web application. It utilizes React Router for in-app navigation and conditional rendering to adapt its layout and available options based on the user's authentication state.

This component manages user sessions using `sessionStorage` to store and retrieve user-related data (like 'user_email' and 'name'). Depending on whether the user is logged in or not, the navbar displays different sets of options—links to the user's dashboard and logout for authenticated users, and sign-in or account creation options for guests.

The `handleLogout` function is a crucial part of the navbar, handling user logout by clearing the user session and redirecting the user to the login page. The styling of the navbar and its elements is specified using inline styles, focusing on adaptability and aesthetic integration with the rest of the application.



1. Import necessary libraries and components.
2. Define AccountNavbar component:
 - Initialize navigation hook.
 - Define styles for the logo, button, and navbar.
 - `handleLogout` function:
 - * Retrieve the user's email from `sessionStorage`.
 - * Invoke the `logoutUser` method from `AuthService` with the `userEmail`.
 - * Clear the `sessionStorage`.
 - * Navigate to the login page.
 - Render the component:
 - * Navbar with specified style, containing a logo and a menu toggle for smaller screens.
 - * Navigation items which include:
 - Home icon that navigates to the root when clicked.
 - Conditional rendering for navigation links:
 - If a user is logged in (`sessionStorage` contains 'name'), display links to 'Dashboard' and a personalized greeting with a 'Logout' button.
 - If no user is logged in, display links to 'Sign in' and 'Create account'.
 - * Each navigation item has styles applied conditionally based on the state of the user session.
3. Export the AccountNavbar component.

KEY SYSTEM COMPONENTS:

AuthService Class

The `AuthService` class in the provided JavaScript file encapsulates methods related to user authentication and data handling within a web application. The methods `loginUser`, `registerUser`, `logoutUser`, and `resetUserPassword` perform HTTP POST requests to corresponding endpoints, sending user data as JSON and handling the server's response. These methods make use of modern JavaScript `async/await` syntax for asynchronous operations, with error handling implemented using `catch` blocks.

The method `addFeelingCheckIn` is noted for containing an unhandled promise, which could lead to issues not caught by the error handling logic if the response handling throws an error. Each method is designed to be reusable and encapsulates functionality to keep the calling code clean and focused on the application logic rather than the details of the HTTP communication.



1. Define `AuthService` class with methods to handle user authentication and data posting:
 - `loginUser(userEmailAndPass)`:
 - * Send POST request to login endpoint with user credentials.
 - * Convert response to JSON.
 - * Handle any errors that may occur.
 - `registerUser(userData)`:
 - * Send POST request to register endpoint with user data.
 - * Convert response to JSON.
 - * Handle any errors that may occur.
 - `logoutUser(userEmail)`:
 - * Send POST request to logout endpoint with user email.
 - * Convert response to JSON.
 - * Handle any errors that may occur.
 - `resetUserPassword(userEmail)`:
 - * Send POST request to password reset endpoint with user email.
 - * Convert response to JSON.
 - * NOTE: Error handling is not explicitly done here.
 - `addFeelingCheckIn(userCheckIn)`:
 - * Send POST request to feeling check-in endpoint with user data.
 - * Convert response to text format.
 - * Handle errors explicitly and throw if any, noting potential unhandled promise.
2. Create an instance of `AuthService`.
3. Export the `AuthService` instance for use elsewhere in the application.

KEY SYSTEM COMPONENTS:

FeelingsCheckIn Class

The `FeelingsCheckIn` component is structured as a multi-page form within a React application, managing various states like `feeling`, `control`, and stressors across different life domains (home, work, school, social). It uses local state management via `useState` for storing user inputs and navigational state. The component conditionally renders different sections of the form based on the current page state, allowing the user to input their feelings, control level, and identify primary stressors.

Functions like `updateFeeling` and `updateControl` update the respective states and move the user to the next section of the form. Each section allows the user to specify more detailed stressors relevant to their selected main stressor. Finally, the `uploadFeelingCheckIn` function collects all user inputs as a token and uploads it using a service instance, while handling form submission events.

The pseudocode captures the logical flow and structure of the component, focusing on key state management and navigation aspects crucial for understanding the component's functionality without getting bogged down in implementation details.



1. Import necessary libraries and components.
2. Define `FeelingsCheckIn` component with initial states for feelings, control, and various stressors.
3. Define page display logic based on current state.
4. Define functions:
 - `updateFeeling()`: Update feeling state based on user input and move to next page.
 - `updateControl()`: Update control state based on user input and move to next page.
 - `uploadFeelingCheckIn()`: Collect all state data and upload via `AuthService`, preventing default form action.
 - Define `onClick` handlers for each stressor to update the respective stressor state and navigate to the corresponding details page or the thank you page.
5. Render the component layout using a form that includes:
 - Sliders for inputting feeling and control levels.
 - Radio button groups for selecting the main stressor and specific stressors in different life areas.
 - Conditional display of different sections based on current page state.
 - Thank you message display upon completion.
6. Buttons for restarting the check-in or submitting the form.
7. Export the `FeelingsCheckIn` component.

VISUAL DEMO OF EXPECTED USE CASE SCENARIO

Android Demo

Web Demo

FORWARD LOOK

- **Innovations & Expansion:**
 - Implement machine learning for personalized recommendations.
 - Introduce complex algorithms for deeper psychological insights.
 - Broaden wellness activities and community support features.
- **Strategic Goals:**
 - Extend global reach with multilingual support.
 - Forge partnerships to amplify accessibility and impact.
- **Outlook:**
 - HowRU.life is evolving into a more expansive resource for global mental health management.



PROJECT RECAP

- **Overview:**
 - The "HowRU.life" application was developed to help individuals manage their stress and anxiety through a user-friendly digital platform.
 - The application provides tools for emotional state assessment and offers personalized activities based on the user's emotional input.
- **Key Features:**
 - User authentication for personalized experience.
 - Dynamic emotional assessment forms.
 - Personalized activity recommendations.
 - Data visualization of emotional trends.
- **Technical Achievements:**
 - Utilized React Native and Firebase to create a scalable and robust application architecture.
 - Implemented algorithms for emotion analysis, activity recommendation, and data visualization to enhance user interaction.

Summary

The expansion to HowRU.Life to include the web app expands the ability for it to be viewed by all devices and users who may not have an Android which addresses the main concern of the product owner from the beginning of the semester. Ian and Rafael also modified the database structure for it to be scalable for a large audience. Also, the addition of the RESTful API will be very useful in the future to expand the application to devices like Apple and also makes it easier to grow the already developed product.

Thank you
- HowRU.Life team