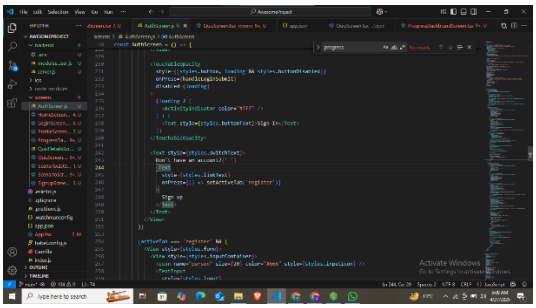
CYBER GUARD

AuthScreen:



Title: Understanding the AuthScreen Component in React Native

1. Introduction

The `AuthScreen` component is a React Native screen designed for user authentication, providing both login and registration functionalities for the SwiftRide mobile application. It interacts with a backend server to authenticate users and is written in JavaScript, the primary programming language for React Native development. This document explains the code structure, key features, functionality, and the role of JavaScript in building the `AuthScreen`.

2. Purpose of the AuthScreen

The `AuthScreen` serves as the primary interface for user authentication, allowing users to:

- Sign in with their email and password.

- Register a new account by providing their name, email, and password.

- Switch between login and registration forms using tabs.

- Persist user sessions using AsyncStorage and display feedback via toast notifications.

3. Programming Language: JavaScript

3.1. Overview

The `AuthScreen` component is written in **JavaScript**, a versatile, high-level programming language widely used for web and mobile application development. In the context of React

Native, JavaScript is used to define the component's logic, state management, and user interface rendering.

3.2. Role in React Native

- **React Native Framework**: React Native is a JavaScript-based framework that allows developers to build cross-platform mobile applications (iOS and Android) using a single codebase. JavaScript powers the logic and UI components of the `AuthScreen`.

- **Component-Based Architecture**: JavaScript, combined with React's declarative syntax, enables the creation of reusable UI components like `AuthScreen`. The code uses functional components and hooks (e.g., `useState`, `useNavigation`) introduced in modern JavaScript and React.

- Asynchronous Operations**: JavaScript's asynchronous capabilities (e.g., `async/await`, `Promises`) are used for API calls to the backend server and storage operations with `AsyncStorage`.

- Dynamic UI Updates: JavaScript manages the component's state (e.g., form inputs, loading status) to dynamically update the UI based on user interactions.

3.3. Key JavaScript Features Used

ES6+ Syntax: The code uses modern JavaScript features like arrow functions, destructuring, and template literals.

Modules: The code employs ES Modules (`import/export`) to organize dependencies and components.

-Error Handling: JavaScript's `try/catch` blocks handle errors during API requests and JSON parsing.

Event Handling: JavaScript manages user interactions (e.g., button presses, text input changes) through event handlers like `onPress` and `onChangeText`.

3.4. Advantages of JavaScript in AuthScreen

Cross-Platform Compatibility: JavaScript enables the `AuthScreen` to work on both iOS and Android without platform-specific code.

Rich Ecosystem: JavaScript's ecosystem provides libraries like `react-navigation`, `react-native-vector-icons`, and `react-native-toast-message`, which enhance the `AuthScreen`'s functionality.

Community Support: JavaScript's widespread use ensures access to extensive documentation and community resources for debugging and optimization.

4. Code Structure

The `AuthScreen` code is structured as follows:

Imports: JavaScript libraries and React Native components (`React`, `useState`, `react-native`, `react-navigation`, etc.).

State Management: Uses JavaScript's `useState` hook to manage form inputs, UI states, and loading indicators.

Components: A single functional component (`AuthScreen`) written in JavaScript, rendering the UI and handling logic.

Styles: A JavaScript `StyleSheet` object defining the visual appearance of UI elements.

5. Key Features

The `AuthScreen` includes the following features, all implemented using JavaScript:

Tab Navigation:

- Two tabs ("Sign In" and "Sign Up") allow users to switch between login and registration forms.
- Controlled by the `activeTab` state (`login` or `register`).

Form Inputs:

- Login Form: Collects email, password, and a "Remember me" checkbox.
- Registration Form: Collects full name, email, password, confirm password, and terms agreement checkbox.
- Inputs are styled with icons and support password visibility toggling via JavaScript state.

- Backend Integration:

- Login: Sends a POST request to `http://192.168.x.102:5000/api/auth/login` using JavaScript's `fetch` API.

- Registration: Sends a POST request to `http://192.168.x.102:5000/api/auth/signup`.

- JavaScript handles timeouts and error responses.

- Session Management:

- JavaScript's `AsyncStorage` API stores authentication tokens and user data for persistent sessions.

- User Feedback:

- JavaScript powers toast notifications using `react-native-toast-message`.

- Displays a loading indicator during API requests.

- Navigation:

- JavaScript's `react-navigation` library navigates to screens like `Home`, `ForgotPassword`, `Terms`, and `Privacy`.

-Error Handling:

- JavaScript validates form inputs and handles network errors, timeouts, and invalid responses.

6. Detailed Functionality

6.1. State Management

JavaScript's `useState` hook manages:

- `activeTab`: Tracks the current form (`login` or `register`).

- Form fields: `loginEmail`, `loginPassword`, `name`, `registerEmail`, `registerPassword`, `confirmPassword`.

- UI states: `showLoginPassword`, `showRegisterPassword`, `showConfirmPassword` for password visibility.

- `rememberMe` and `terms`: Checkbox states.

- `loading`: Controls the loading indicator.

6.2. Login Functionality (`handleLoginSubmit`)

- JavaScript's `async/await` sends a POST request to the login endpoint.

- Stores token and user data in `AsyncStorage`.

- Navigates to the `Home` screen on success using JavaScript navigation logic.

- Displays toast messages for success or failure.

- Uses `AbortController` for a 5-second timeout.

6.3. Registration Functionality (`handleRegisterSubmit`)

- JavaScript validates password matching and terms agreement.

- Sends a POST request to the signup endpoint.

- Stores token and user data in `AsyncStorage`.

- Switches to the login tab and clears the form on success.

- Handles errors and timeouts using JavaScript's `try/catch`.

6.4. UI Component

-Tabs: JavaScript renders `TouchableOpacity` buttons with dynamic styles.

- Inputs: `TextInput` components with JavaScript event handlers for text changes and password toggling.

- Checkboxes: `CheckBox` components with JavaScript state updates.

- Buttons: `TouchableOpacity` for form submission, disabled during loading via JavaScript.

- Links: JavaScript handles navigation for forgot password, terms, privacy, and tab switching.

6.5. Styling

JavaScript's `StyleSheet.create` defines styles for:

- Container and card layout with shadows and rounded corners.

- Header with a light blue background (`#ADD8E6`).

- Tabs with an active state indicator.

- Inputs with icons, borders, and padding.

- Buttons with a consistent color scheme.

7. Dependencies

The component relies on JavaScript-based packages:

- `react`, `react-native`: Core framework for building the UI and logic.

- `@react-navigation/native`: For navigation.

- `react-native-vector-icons`: For icons.

- `react-native-toast-message`: For notifications.

- `@react-native-community/checkbox`: For checkboxes.

- `@react-native-async-storage/async-storage`: For storage.

8. Potential Improvements

- Input Validation: Add JavaScript-based validation for email and password formats.

- Error Messages: Enhance JavaScript error handling for specific backend responses.

- Secure Backend URL: Replace the hardcoded URL with a configurable JavaScript variable.

- Accessibility: Use JavaScript to add accessibility labels for screen readers.

- Localization: Add JavaScript logic for multi-language support in the UI.

- Rate Limiting: Implement JavaScript retry logic for API requests.

9. Conclusion

The `AuthScreen` component is a robust React Native implementation of an authentication interface, powered by JavaScript. JavaScript's flexibility, asynchronous capabilities, and rich ecosystem enable the component to deliver a user-friendly experience with tabbed navigation, form handling, backend integration, and error management. This makes it a critical part of the SwiftRide application's authentication flow.

Instructions to Create the Word Document

1. Open Microsoft Word:

- Create a new blank document.

2. Set Up Formatting:

- Use Heading 1 for the title and Heading 2 for section titles (e.g., "Programming Language: JavaScript").
- Use Normal style for body text.
- Add bullet points for lists (e.g., JavaScript features, dependencies).
- Use Bold for emphasis (e.g., `useState`, `AsyncStorage`, JavaScript).

3. Copy and Paste Content:

- Copy the updated content above into the Word document.
- Adjust formatting (e.g., font size: 12pt for body, 16pt for Heading 1).

4. Add Visuals (Optional):

- Include screenshots of the `AuthScreen` UI (login and registration forms).
- Create a table to summarize state variables:

State Variable	Purpose
`activeTab`	Tracks login or register tab

| `loginEmail` | Stores login email input |

| `loginPassword` | Stores login password input |

| `rememberMe` | Tracks "Remember me" checkbox |

| `name` | Stores registration name input |

| `registerEmail` | Stores registration email input |

| `registerPassword` | Stores registration password input |

| `confirmPassword` | Stores confirm password input |

| `terms` | Tracks terms agreement checkbox |

| `loading` | Controls loading indicator |

5. Highlight JavaScript Section:

- Use a different font color (e.g., blue) or bold text for the "Programming Language: JavaScript" section to emphasize it.

- Optionally, add a sidebar or callout box in Word to summarize JavaScript's role.

6. Save the Document:

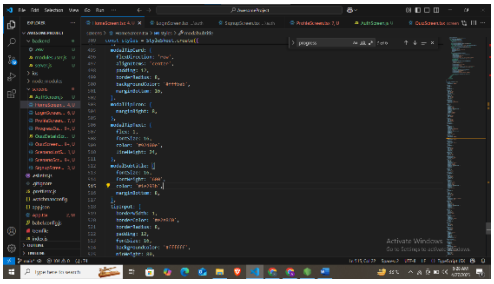
- Save as `AuthScreen_Explanation_With_Language.docx`.

7. Review and Finalize:

- Proofread for clarity, especially in the JavaScript section.

- Ensure technical terms are explained for non-technical readers if needed.

HomeScreen Component:



What does it do?

The HomeScreen has these main features:

- **Welcome Message**: Greets users with "Welcome back, Cyber Defender!" and a subtitle.
- **Stats Cards**: Shows three stats (e.g., 3 scenarios completed, 75% success rate, 2 badges earned). These are hardcoded (fixed) for now.
- **Navigation Cards**: Three buttons let users go to:
    - **Scenarios**: Start interactive cybersecurity lessons.
    - **Progress**: Check their learning progress and achievements.
    - **Profile**: Manage their account settings.
- **Daily Security Tip**: Shows a new tip each day (e.g., "Always verify the sender's email address..."). Tips come from a Firebase database.
- **Tip Submission**: Users can add their own security tips through a pop-up (modal) window.
- **Animations**: The screen fades in smoothly when it loads.
- **Cool Design**: Uses gradients (color blends) and shadows for a modern look.

How does it work?

Here's a simple breakdown of how the code works:

1. **JavaScript and React Native**: The code uses JavaScript to control the app. React Native helps create the buttons, text, and layouts you see on the screen.
2. **Firebase for Tips**: The app connects to Firebase (an online database) to get and save security tips. It picks a random tip daily based on the date.
3. **User ID**: A fake user ID (1234567890) is used to track who submits tips. In a real app, this would come from a login system.

4. **Pop-up Modal**: When you tap the daily tip, a pop-up lets you see the tip, add a new one, view all tips, or close it.
5. **Navigation**: Tapping cards takes you to other app screens (like Scenarios or Profile) using React Navigation.
6. **Error Messages**: If something goes wrong (e.g., no internet), the app shows an alert like "Failed to load tips."

What do you need to run it?

To use this code in your app, you need:

1. **Node.js**: A tool to run JavaScript outside a browser. Download it from nodejs.org.
2. **React Native Setup**: Follow the React Native guide to set up your computer for building apps.
3. **Firebase Account**:
    - Create a Firebase project at firebase.google.com.
    - Set up a Realtime Database and note the URL (e.g., https://cyber-80650-default-rtdb.firebaseio.com/).
    - Add some security tips in the SecurityTips section of your database.
4. **Dependencies**: Install these libraries using npm or yarn:

npm install react-native-safe-area-context lucide-react-native react-native-linear-gradient

These add safe area support, icons, and gradients to the app.

5. **React Navigation**: Set up React Navigation to handle moving between screens (e.g., Scenarios, Progress, Profile).

How to add it to your app?

1. **Copy the Code**: Save the HomeScreen code in a file called HomeScreen.js in your React Native project.
2. **Set Up Navigation**:
    - Make sure your app has a navigation setup with screens for Scenarios, Progress, Profile, and Tips.
    - Example: Use a StackNavigator to link these screens.

3. **Connect to Firebase**:
    - Replace the Firebase URL in the code (https://cyber-80650-default-rtdb.firebaseio.com/SecurityTips.json) with your own Firebase database URL.
    - Ensure your Firebase database allows reading and writing (update security rules if needed).
4. **Add AuthContext**:
    - The code uses a fake AuthContext with a userId. In a real app, replace it with your authentication system (e.g., from the AuthScreen login).
    - Example: Update AuthContext to get the userId from your logged-in user.
5. **Run the App**:
    - Start your React Native app with:

npx react-native run-android

or

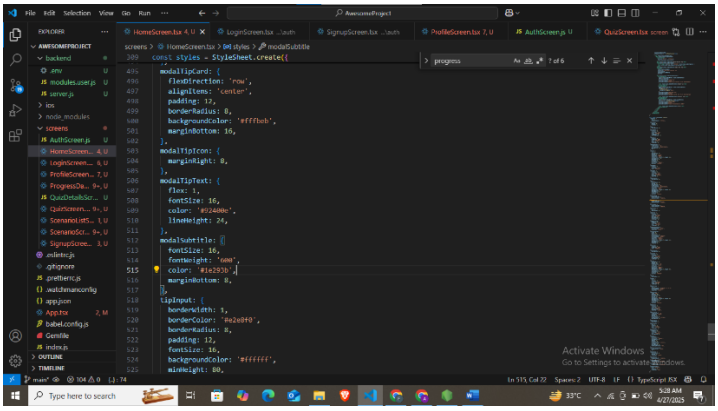
npx react-native run-ios

    - Check the screen to see the dashboard!

Things to know

- **Hardcoded Stats**: The stats (3 scenarios, 75%, 2 badges) are fake. To show real stats, connect to a backend or database.
- **Accessibility**: The buttons have labels for screen readers (e.g., "Start Scenario"), making the app usable for people with disabilities.
- **Errors**: If Firebase fails to load tips, the app shows an error alert and uses a default tip.
- **"use client"**: This line is not needed for React Native. You can delete it without any issues.

Login Screen:



This code is for a **Login Screen** in a mobile app called **CyberGuard**, which helps users learn cybersecurity through games.

Main Features

1. User Login

- Users can enter their **email** and **password** to log in.
- If the login details are correct, they are taken to the main app.
- If there's an error (wrong password, empty fields, etc.), an error message appears.

2. Forgot Password (Reset Password)

- If users forget their password, they can click **"Forgot Password?"**
- A **pop-up (modal)** appears where they can:
    - Enter their email.
    - Set a new password.
    - Confirm the new password.
- The password must be strong (at least 8 characters, uppercase letters, and numbers).
- If successful, the password is updated in the database.

3. Password Strength Checker

- When setting a new password, the app checks:
    - Length (8+ characters)
    - Uppercase letters
    - Numbers

- A **strength meter** shows if the password is **Weak, Fair, or Strong**.

4. Smooth Animations

- The screen has **fade-in and slide-up animations** when it loads for a better user experience.

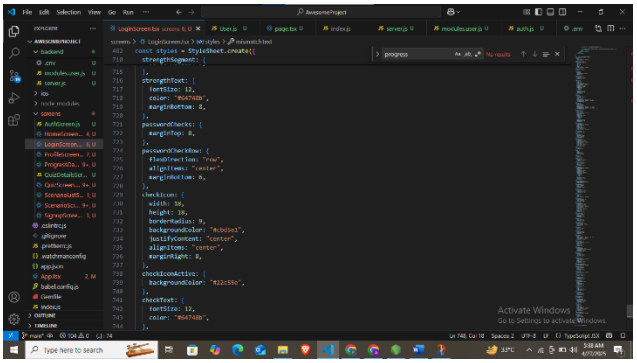
5. Other Features

- **"Continue with Google"** button (though not fully implemented yet).
- **"Sign Up"** link for new users to create an account.
- **Eye icon** to show/hide password.

How It Works

- The app connects to a **Firebase database** to check login details.
- If the email and password match, the user logs in.
- If not, an error is shown.

Profile Screen:



This code is for a **Profile Screen** in the **CyberGuard** app, where users can manage their account settings, preferences, and security.

Main Features

1. User Profile Section

- Displays the user's **name, email, and join date**.
- Shows a **profile avatar** with the user's initials.

2. Account Management

- **Edit Profile**: Change username (checks if it's already taken).
- **Change Email**: Update email (validates format and checks for duplicates).
- **Change Password**:
    - Requires **strong password** (8+ characters, uppercase, numbers).
    - Shows a **password strength meter** (Weak, Fair, Strong).

3. Preferences

- **Notifications**: Toggle on/off.
- **Dark Mode**: Toggle on/off (UI changes not fully implemented).
- **Biometric Login**: Enable/disable fingerprint or face recognition.

4. Support & Help

- **Help Center**: Opens a help section (placeholder).
- **Report a Bug**: Opens a bug report form (placeholder).
- **Privacy Policy**: Displays privacy terms (placeholder).

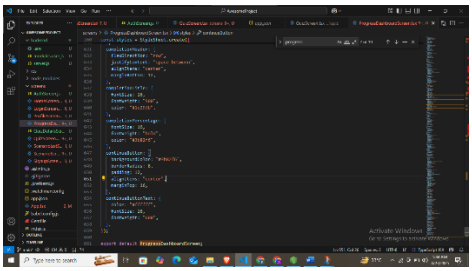
5. Logout

- Confirms before logging out and redirects to the **Login Screen**.

How It Works

- **Modals** pop up for editing profile, email, or password.
- **Firebase Database** checks and updates user data.
- **Password Strength Checker** ensures security.
- **Error Handling** shows alerts if something goes wrong.

Progress Dashboard:



This code is for a **Progress Dashboard Screen** in the **CyberGuard** app, where users can track their cybersecurity learning progress, achievements, and quiz performance.

Main Features

1. User Progress Overview

- **Level System**: Shows the user's current level based on XP earned from quizzes.
- **XP Progress Bar**: Visualizes how close the user is to the next level.
- **Quick Stats**: Displays:
    - **Scenarios Completed** (e.g., "3/6")
    - **Badges Earned** (e.g., "2/4")
    - **Quiz Average Score** (e.g., "85%")

2. Badges & Achievements

- Users can earn **badges** by completing quizzes with high scores.
- Example badges:
    - **Phishing Expert** (100% score in phishing quizzes)
    - **Password Pro** (80%+ score in password security quizzes)
- Locked badges appear grayed out until earned.

3. Recent Activity

- Shows a list of **recent quiz attempts** with:
    - Quiz name
    - Date taken
    - Correct & incorrect answers
    - Final score

- Users can tap on a quiz to see **detailed results**.

4. Completion Status

- Tracks how many **scenarios** the user has completed (e.g., "50%").
- Includes a **"Continue Learning"** button to go back to scenarios.

5. Error & Loading Handling

- Shows a **loading spinner** while fetching data.
- Displays an **error message** if data fails to load, with a **Retry button**.

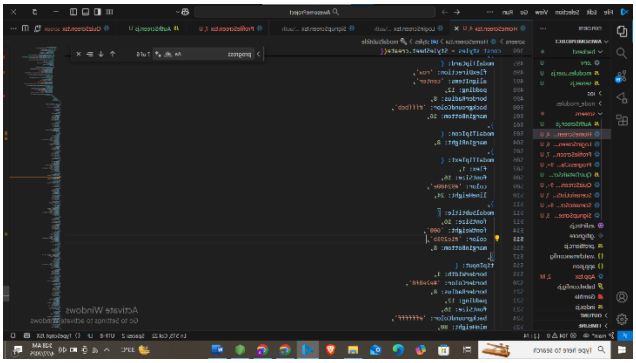
How It Works

- **Fetches quiz data** from Firebase when the screen loads.
- **Calculates XP, levels, badges, and stats** based on quiz performance.
- **Updates in real-time** when new quizzes are completed.
- **Badges are automatically awarded** when users meet the criteria.

UI Design

- **Clean, card-based layout** with progress bars and icons.
- **Horizontal scroll for badges** to save space.
- **Highlighted activity** if coming from a specific quiz.

Quiz Details:



This code creates a **Quiz Results Screen** for a mobile app (built with **React Native**). It displays the user's quiz score, correct/wrong answers, and a detailed breakdown of each question.

What the Screen Shows:

1. **Quiz Summary Section**
    - **Quiz Name** (e.g., "Science Quiz")
    - **Score (%)** – The percentage of correct answers.
    - **Correct Answers** – Number of right answers (in **green**).
    - **Wrong Answers** – Number of wrong answers (in **red**).
    - **Completion Date** – When the quiz was taken.
    - **Share Button** – Lets users share their results as a text message.
2. **Answer Details Section**
    - Each question is shown in a **card** (box with rounded corners).
    - **Green border** = Correct answer.
    - **Red border** = Wrong answer.
    - For each question, it displays:
        - The question text.
        - The correct answer (in **green**).
        - The user’s answer (marked as **"Right" or "Wrong"**).

        - An explanation (if provided).
    - If no answers are available, it shows: *"No answer details. Try completing a quiz!"*

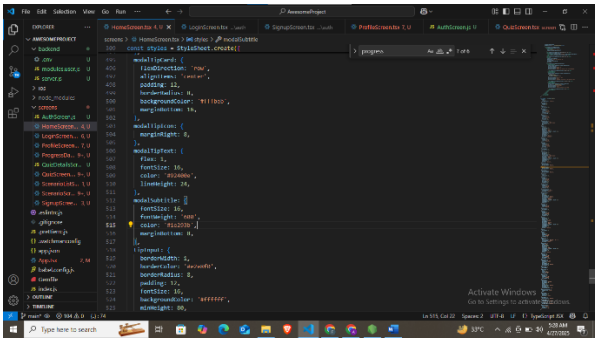
How It Works

1. **Data Comes from Previous Screen**
    - The quiz results (name, score, answers, etc.) are passed to this screen.
2. **Share Function**
    - When the user clicks **"Share Results"**, it creates a message like:

"I scored 85% on the Math Quiz! Correct: 17, Incorrect: 3. Completed on 26 April 2024."

    - If sharing fails, an error message appears.
3. **Answer List**
    - Loops through each question and shows:
        - The question number and text.
        - The correct answer.
        - The user's answer (with **"Right"** in green or **"Wrong"** in red).
        - An extra explanation (if available).
4. **Styling (Design Choices)**
    - Clean, card-based layout with shadows.
    - Correct answers = **Green text and border**.
    - Wrong answers = **Red text and border**.
    - Share button = **Blue background with white text**.

Phishing Quiz App Explanation:



This code creates a mobile app that tests users' ability to identify phishing emails through an interactive quiz.

Core Features:

1. **Email Display**
    - Shows realistic email examples with:
        - Subject line
        - Sender address
        - Email content body
2. **Quiz Mechanics**
    - For each email, users must:
        - Select either "Phishing" or "Legitimate"
        - Explain their reasoning in a text box
    - Progress tracker shows current question (e.g., "2 of 4")
3. **Answer Verification**
    - Immediate feedback after submission:
        - Correct answers highlighted in green
        - Incorrect answers highlighted in red
        - Shows the correct explanation
        - Displays the user's submitted explanation
4. **Scoring System**
    - Calculates percentage score at the end

- Provides performance feedback:
    - 80%+ = Excellent
    - 60-79% = Good
    - Below 60% = Needs improvement

5. **Data Management**
    - Saves results to Firebase database including:
        - Score
        - Each answer
        - Explanations
        - Timestamp
    - Tracks correct/incorrect counts
6. **Navigation Options**
    - "Retake Quiz" - Resets to first question
    - "View Progress" - Shows historical results

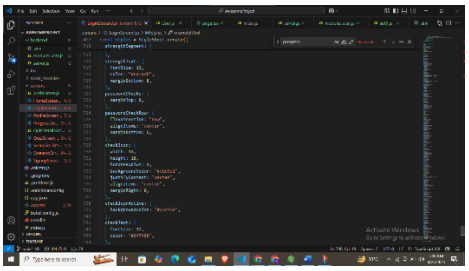
Technical Implementation:

- Built with React Native for cross-platform mobile
- Uses Firebase Realtime Database for storage
- Implements smooth animations between questions
- Features responsive design with:
    - Color-coded buttons (blue/green/red)
    - Clean card-based layout
    - Gradient backgrounds

User Flow:

1. View email → 2. Select classification → 3. Submit explanation → 4. Receive feedback → [Repeat] → 5. View final score → 6. Choose next action

Scenario List Screen Explanation:



This code creates a simple **list of cybersecurity scenarios** that users can select to learn about different threats (like phishing emails or fake websites).

What the Screen Does

1. **Displays a Title** ("Scenarios") at the top.
2. **Shows a Scrollable List** of cybersecurity scenarios, including:
    - **Suspicious Email** (Phishing scenario)
    - **Fake Website** (Website spoofing scenario)
3. Each scenario is a **clickable button** that takes users to a detailed view.

How It Works

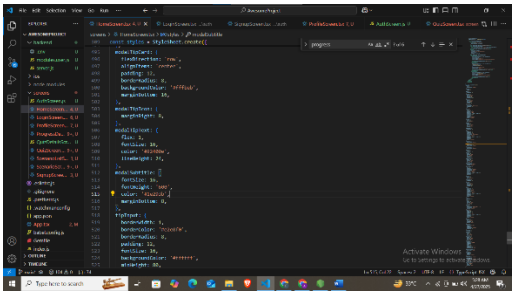
- **FlatList** renders the list efficiently.
- Each **TouchableOpacity** (button) navigates to a **"Scenario" screen** when pressed, passing along:
    - id (unique identifier)
    - title (scenario name)
    - scenario (description)
- **Accessibility features** are included for screen readers.

Styling (Appearance)

- **Background**: Light gray (#f8fafc)

- **Title**: Bold, dark text (#1e293b)
- **Scenario Items**:
    - White background (#ffffff)
    - Light gray border (#e2e8f0)
    - Rounded corners (borderRadius: 8)
    - Dark text (#334155)

Scenario Screen Explanation:



This screen presents an **interactive cybersecurity training scenario** where users learn to identify phishing emails through guided decision-making.

Key Features

1. Scenario Flow

- **Introduction**: Presents a suspicious email scenario
- **Decision Points**: Users choose how to respond
- **Feedback**: Explains correct/incorrect choices
- **Completion**: Shows final score and summary

2. Interactive Elements

- **Multiple-choice questions** about email legitimacy
- **Text explanations** required for each decision
- **Progress tracker** (Step X of Y)
- **"Need Help?"** button for guidance

3. Data Handling

- Tracks all user choices
- Calculates performance score
- Saves results to Firebase database
- Prepares users for a follow-up quiz

User Experience

1. **Reads** a realistic email scenario
2. **Chooses** how to respond (multiple options)
3. **Receives immediate feedback** on their choice
4. **Learns** key phishing indicators
5. **Completes** the scenario with a score
6. **Option** to take a quiz or retry

Technical Implementation

- **React Native** mobile app components
- **Animated transitions** between steps
- **Firebase integration** for data storage
- **Accessibility support** for all interactive elements
- **Responsive design** with:
    - Color-coded feedback (green/red)
    - Progress bar
    - Gradient backgrounds

Visual Design (No Icons Version)

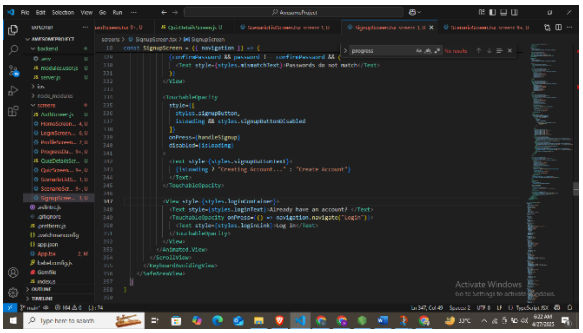
- **Text labels** replace all icon indicators
- **Color coding** maintains visual feedback:
    - Blue = Interactive elements
    - Green = Correct choices
    - Red = Incorrect choices
- **Cards and shadows** create depth

Educational Value

Teaches users to:

- Recognize phishing email characteristics
- Respond appropriately to suspicious messages
- Understand why certain approaches are safer

Signup Screen Explanation:



This code creates a **signup screen** for a mobile app (React Native) where users can create a new account. It includes:

1. What It Does

- Lets users enter a **username, email, password, and confirm password**.
- Checks if the **password is strong enough** (at least 8 characters, uppercase, lowercase, number, etc.).
- Shows **real-time feedback** on password strength.
- Stores user data in **Firebase Realtime Database**.
- Has **animations** for a smooth user experience.
- Shows **errors** if something goes wrong.

2. Main Features

A. Form Inputs

- **Username** (must be unique)
- **Email** (must be valid and unique)
- **Password** (hidden by default, can be shown/hidden with an eye icon)
- **Confirm Password** (must match the password)

B. Password Strength Checker

- Shows a **color-coded strength meter** (weak → strong).
- Displays **checkmarks** for:
    - At least 8 characters
    - Uppercase letter

- Lowercase letter
- Number
- Special character (like !, @, #)

C. Error Handling

- Shows errors if:
  - Fields are empty
  - Passwords don’t match
  - Username/email already exists
  - Password is too weak

D. Animations

- The logo and form **fade in and slide up** when the screen loads.

E. Database (Firebase)

- Checks if **username/email is already taken**.
- Saves new user data in **Firebase Realtime Database**.

3. How It Works

1. **User Types Info**
   - When the user types a password, the app **checks its strength** and updates the meter.
2. **User Clicks "Create Account"**
   - The app checks if:
     - All fields are filled.
     - Passwords match.
     - Password is strong enough.
   - If everything is OK, it **saves the data to Firebase**.
   - If there’s an error, it shows a **red error message**.
3. **Success!**

- If signup works, the user is sent to the **Login screen**.

4. Security Notes

Warning: This code saves passwords in **plain text** (not secure).
For real apps, always:

- Use **password hashing** (like bcrypt).
- Add **email verification**.
- Use **Firebase Authentication** (better than raw database).

5. How to Improve

- Add **MongoDB support** (for users who prefer it over Firebase).
- Add **Google/Facebook login**.
- Send a **welcome email** after signup.