Extra Credit Programming Assignment - GitHub Issue Resolution

Rohit Reddy Musukudabbidi
Web Technologies - 01 (Fall 2024)

December 1, 2024

1. Introduction

The **CSRankings React Native** project aimed to create a robust, mobile-optimized version of the *CSRankings* website, known for ranking computer science institutions based on faculty research contributions. The existing web version, while functional, lacked usability and responsiveness on mobile devices. This necessitated a transition to a React Native application to enhance user experience and make the rankings more accessible to a global audience using Android and iOS platforms.

This project was guided by a clear objective: to replicate the core functionality and dynamic features of the *CSRankings* website, such as filtering institutions by research areas, visualizing publication data, and providing detailed faculty profiles. The mobile application had to be intuitive, visually appealing, and efficient while maintaining the integrity of the data and rankings.

React Native was chosen for its ability to build cross-platform applications using a single codebase, reducing development time and ensuring consistency across platforms. The project focused on creating interactive features, such as dynamic filters and charts, while ensuring smooth navigation and a responsive user interface. The process involved designing custom components, integrating data from JSON files, and leveraging libraries like `react-navigation` and `react-native-chart-kit` for navigation and visualization.

2. Project Objective

The primary objectives of the **CSRankings React Native** project were as follows:

1. **Improve Mobile Usability:**
   Develop a React Native application optimized for mobile platforms to address the usability challenges of the original *CSRankings* website on smaller screens.

2. **Seamless Cross-Platform Performance:**
   Ensure smooth and consistent performance across both Android and iOS devices.

3. **Feature Replication:**
   Recreate all key features of the *CSRankings* website, including rankings, filters, detailed institution pages, and research area visualizations.

4. **Enhanced UI/UX:**
   Design a clean, modern, and intuitive user interface that maintains consistency with the original website while improving user engagement.

5. **Interactive Functionalities:**
   Implement advanced features such as dynamic filters for research areas and real-time, interactive charts for publication data visualization.

3. Requirements Analysis

Based on the project brief, the application was designed to replicate the core functionality of the *CSRankings* website while enhancing its usability for mobile devices. The key requirements identified for the application were as follows:

1. **Homepage Listing Institutions:**
   A primary screen displaying institutions ranked based on publication data, similar to the original website, but optimized for smaller screens and touch navigation.

2. **Dynamic Filters:**
   Interactive filters allowing users to narrow down rankings based on research areas such as Artificial Intelligence, Systems, Theory, and Interdisciplinary Areas.

3. **Institution Detail Pages:**
   Dedicated pages for each institution providing detailed information, including faculty names, their research areas, and links to their academic profiles.

4. **Charts for Data Visualization:**
   Interactive bar charts to dynamically visualize publication data for different research areas, enhancing user engagement and comprehension.

5. **Cross-Platform Compatibility:**
   Seamless functionality and consistent UI/UX across both Android and iOS platforms to cater to a wide user base.

6. **Modern Design and Performance Optimization:**
   Ensure the application delivers a smooth experience, with minimal load times and responsive interactions tailored for mobile devices.

4. Technology Stack

The project utilized the following technologies:

- **React Native:** For cross-platform mobile development.
- **Expo:** To simplify the development, testing, and deployment process.
- **React Navigation:** For navigation and routing.
- **React Native Chart Kit:** For data visualization through charts.
- **Jest:** For writing and executing test cases.
- **GitHub:** For version control and collaboration.
- **VS Code:** As the primary code editor.

5. Development Process

This section provides a detailed step-by-step explanation of the development process, starting from understanding the CSRankings concept and data structure to implementing the features and functionalities of the application.

5.1 Understanding CSRankings and Data Structure

The first step involved thoroughly understanding the *CSRankings* website, its purpose, and how it operates:

Concept of CSRankings

CSRankings is an interactive platform that ranks computer science institutions based on research publications in selective conferences. It is highly data-driven and focuses on categorizing institutions by research areas and their contributions.

Key Components of CSRankings

- **Institution Rankings:** Institutions are ranked based on the number of publications by their faculty.
- **Research Areas:** Publications are grouped into categories such as Artificial Intelligence, Systems, Theory, and Interdisciplinary Areas.
- **Faculty Details:** Each institution lists its faculty members, their research areas, and links to their academic profiles.

Understanding Data Sources

- `universities.json`: Provided the list of institutions.
- `rank.json`: Contained publication data linked to institutions.
- `subarea.json`: Defined the research areas and subcategories.
- `faculty.json`: Detailed the faculty members and their publication records.

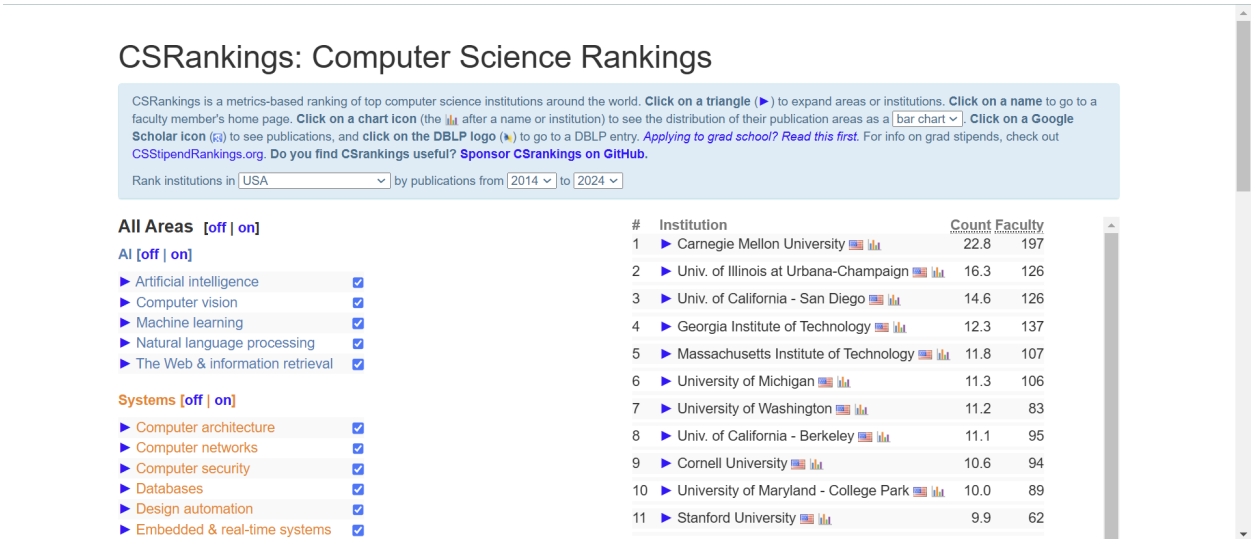

Figure 1: CSRankings top institutions overview.

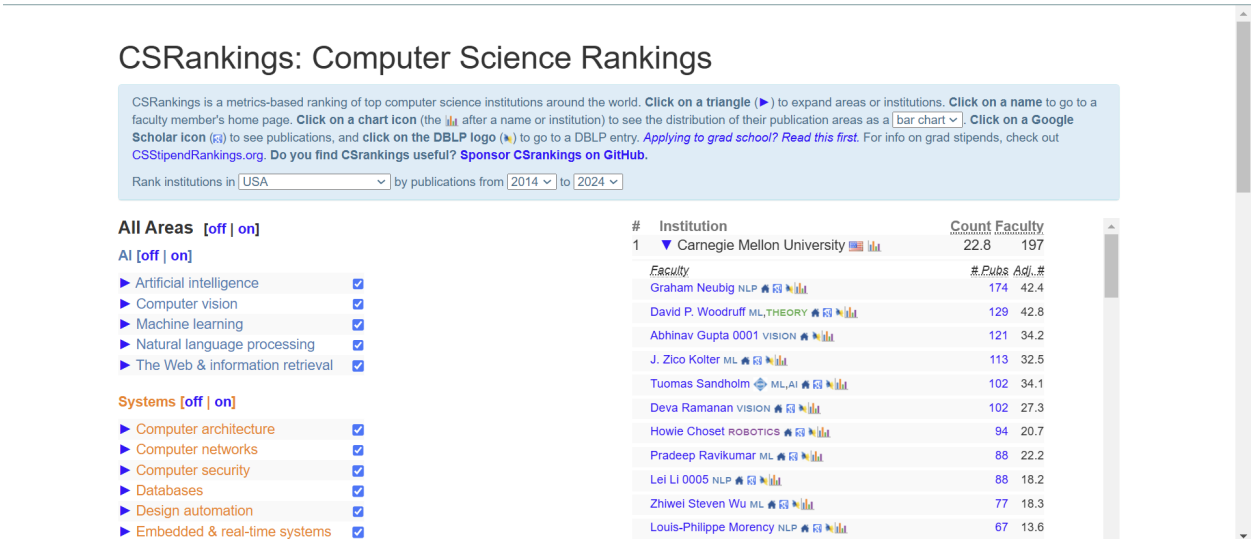

Figure 2: CSRankings platform highlighting Carnegie Mellon University and faculty details.

5.2 Setting Up the Project Environment

Choosing the Framework

- **React Native:** Selected for its ability to create cross-platform mobile applications using a single codebase.
- **Expo:** Used for easier setup and debugging during development.

Project Initialization

- Project initialized using the following command:

```
npx create-expo-app CSRanking
```

- Installed the required dependencies:
  - `react-native-chart-kit` for charts.
  - `react-navigation` for multi-screen navigation.
  - `@react-native-picker/picker` for dropdowns.
  - `react-native-vector-icons` for UI icons.

Project Folder Structure

- Organized the project with directories for screens, components, and data:
  - **Components:** `HomeScreen`, `InstitutionDetails`, `ChartScreen`, `AboutScreen`, `Navigator`.
  - **Data:** Static JSON files simulating backend data.
  - **_tests_:** Jest test files.

5.3 Data Integration

Creating Data

- Created JSON files (`universities.json`, `rank.json`, `subarea.json`, `faculty.json`) for the project.
- Used `useEffect` hooks to simulate asynchronous data loading.

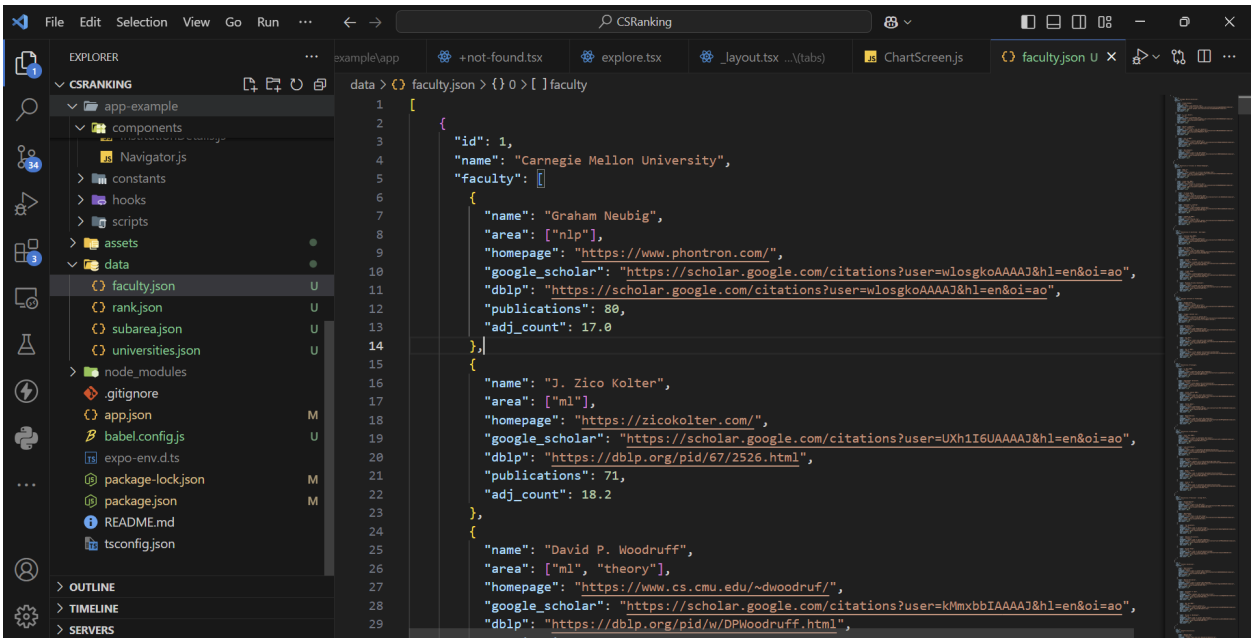

Figure 3: faculty.json.

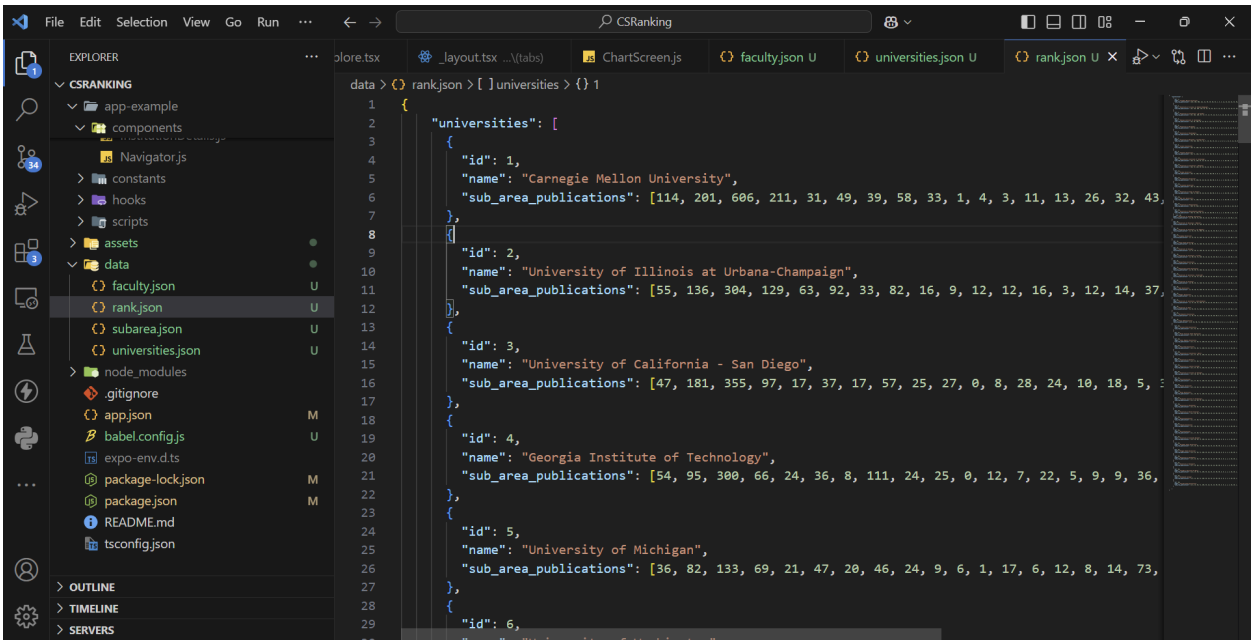

Figure 4: rank.json.

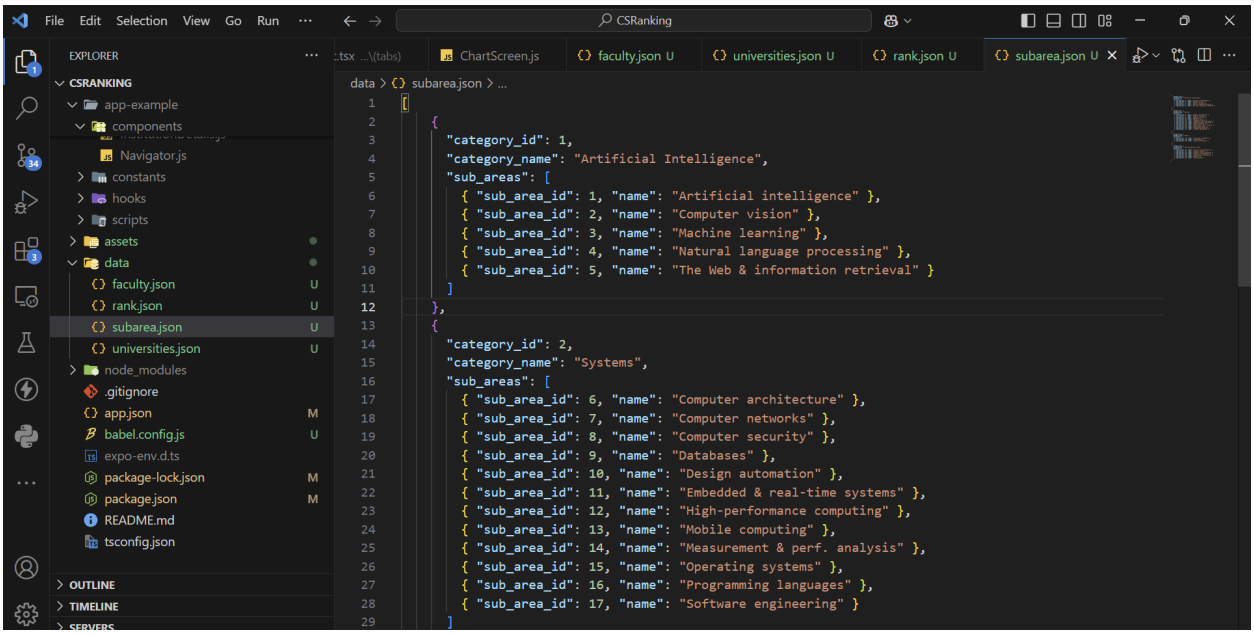

Figure 5: subarea.json.

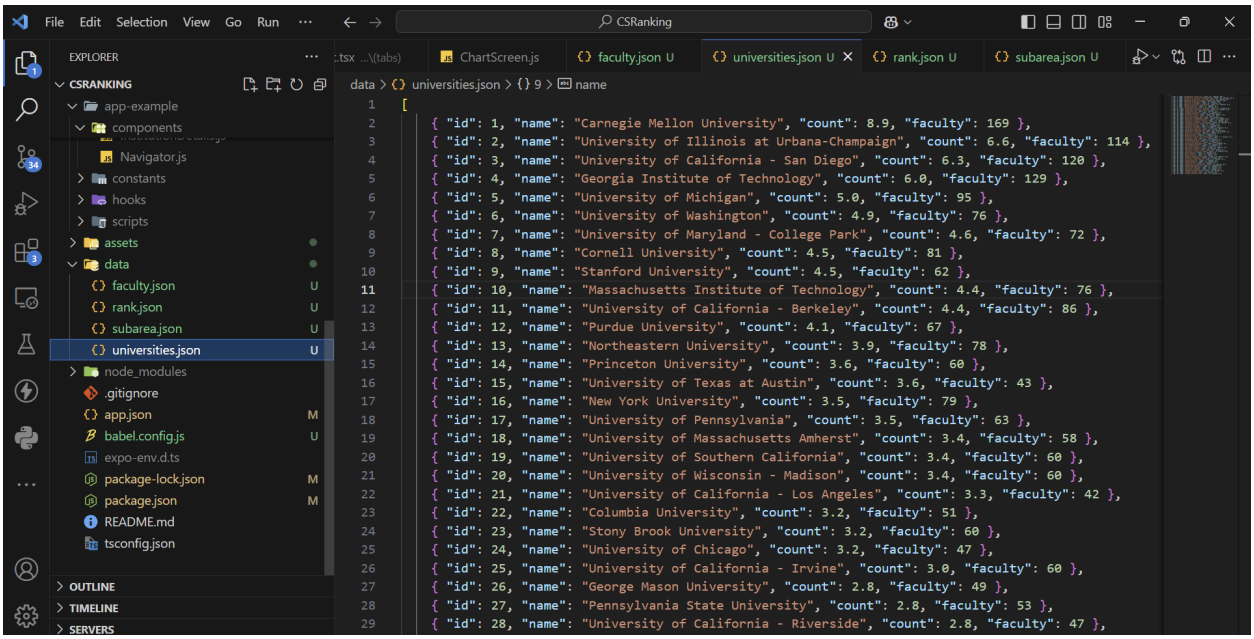

Figure 6: universities.json.

Linking Data

- Merged `universities.json` with `rank.json` to associate publication counts with institutions.
- Linked `faculty.json` to each institution to provide faculty details.

```
useEffect(() => {
    setTimeout(() => {
        const mergedData = data.map((university) => {
            const rankInfo = rankData.universities.find((rank) =>
               rank.name === university.name);
            return {
                ...university,
                sub_area_publications: rankInfo ? rankInfo.
                   sub_area_publications : [],
            };
        });
        setInstitutions(mergedData);
        setLoading(false);
    }, 1000);
}, []);
```

Listing 1: Dynamic Data Integration with `useEffect`

5.4 Dynamic Data Handling

- Created utility functions to filter and sort data dynamically based on user interactions.
- Used React state (`useState`) to manage filters, search input, and selected institution details.

5.5 Frontend Development

5.5.1 Homepage Development

- **Institution List:**
  - Displayed a ranked list of institutions dynamically.
  - Used `FlatList` for efficient rendering of large datasets.
- **Search Bar:**
  - Added a `TextInput` component for users to search institutions by name.
  - Dynamically filtered the institution list based on search input.

```
const filteredData = getFilteredData().filter((item) =>
    item.name.toLowerCase().includes(search.toLowerCase())
);
```

Listing 2: Filtering Data by Search Term

- **Filters:**

- Created a filter modal to toggle between research areas (AI, Systems, Theory, etc.).
- Updated the institution ranking based on selected filters.
- Used state to track active filters.

```
const getFilteredData = () => {
    if (filters.allAreas) return institutions;

    const { selectedSubAreas } = filters;
    if (selectedSubAreas.length === 0) return institutions;

    return institutions
      .map((inst) => {
        const totalPublications = selectedSubAreas.reduce((sum,
           subAreaId) => {
          return sum + (inst.sub_area_publications[subAreaId - 1] ||
              0);
        }, 0);
        return { ...inst, totalPublications };
      })
      .sort((a, b) => b.totalPublications - a.totalPublications);
};
```

Listing 3: Filtering Data by Sub-Areas

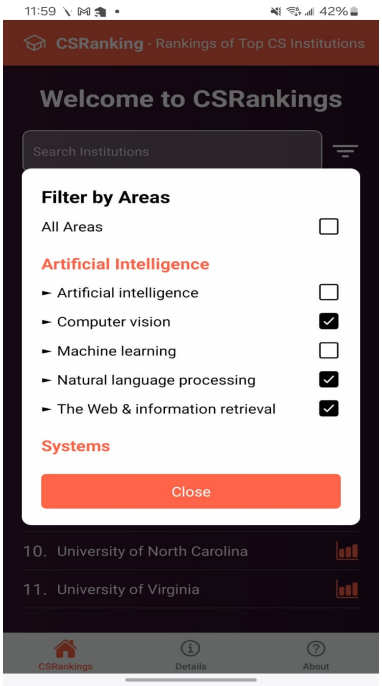

Figure 7: Filters.

5.5.2 Institution Details Page

- Displayed detailed information for the selected institution:
  - Faculty names, research areas, and publication statistics.
  - External links to Google Scholar, DBLP, and personal homepages.
- Integrated `react-native-vector-icons` for navigation buttons and styling.

```
useEffect(() => {
    if (institution) {
        const institutionDetails = facultyData.find(
            (inst) => inst.name === institution.name
        );
        setFacultyList(institutionDetails ? institutionDetails.
           faculty : []);
    }
}, [institution]);
```

Listing 4: Fetching Faculty Data based on Institution

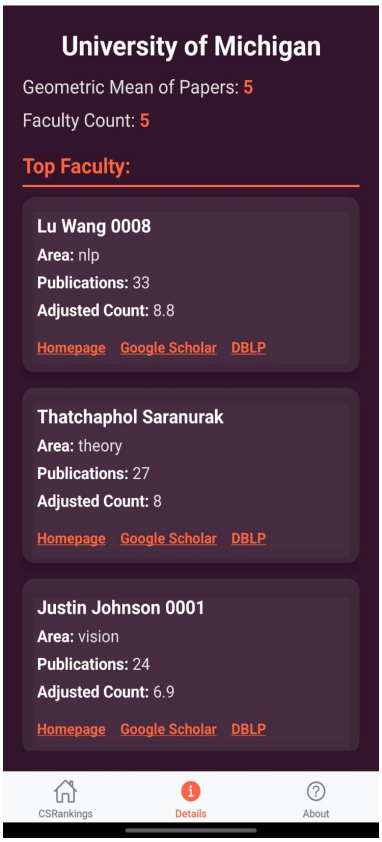

Figure 8: Institution Page.

5.5.3 Chart Integration

- Added interactive bar charts using `react-native-chart-kit`.
- Dynamically displayed publication data for selected research areas.
- Enabled horizontal scrolling for charts with long labels.
- Included a legend to decode chart abbreviations.

```
<ScrollView horizontal>
    <BarChart
        data={{
            labels,
            datasets: [{ data: chartData }],
        }}
        width={Math.max(labels.length * 80, Dimensions.get('window')
           .width - 20)}
        height={400}
        yAxisLabel=""
        chartConfig={{
            backgroundColor: '#1cc910',
            backgroundGradientFrom: '#eff3ff',
            backgroundGradientTo: '#efefef',
            decimalPlaces: 0,
            color: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`,
            labelColor: (opacity = 1) => `rgba(0, 0, 0, ${opacity})
               `,
            barPercentage: 0.8,
            useShadowColorFromDataset: false,
        }}
        style={styles.chartWithPadding}
        verticalLabelRotation={0}
        fromZero
        showValuesOnTopOfBars={true}
        onDataPointClick={(data) => handleBarPress(data.index)}
    />
</ScrollView>
```

Listing 5: BarChart Integration within ScrollView

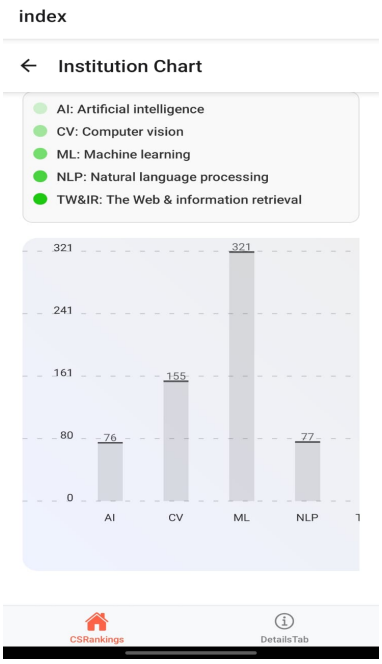

Figure 9: Chart Screen Page.

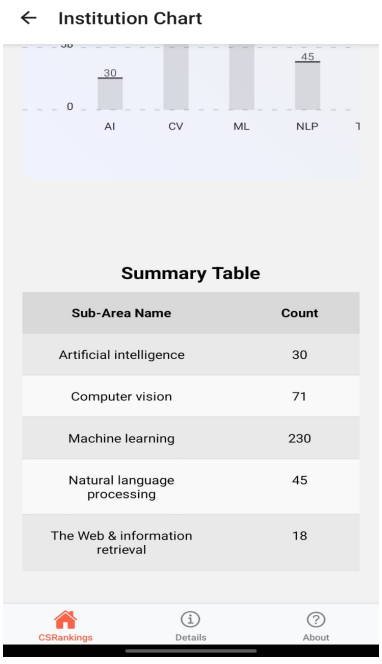

Figure 10: Chart Screen Page.

5.5.4 About Page

- Provided an overview of CSRankings, including its purpose and methodology.
- Highlighted mentions of CSRankings by renowned institutions and individuals.
- Included clickable links to GitHub, DBLP, FAQs, and licensing details.
- Integrated a clickable YouTube thumbnail linking to an introductory video.
- Styled with responsive design for mobile devices using `ScrollView` and consistent fonts and colors.

```
<View style={styles.container}>
  <ScrollView contentContainerStyle={styles.contentContainer}>
    <Text style={styles.title}>About CSRankings</Text>
    <Text style={styles.description}>
      This ranking is designed to identify institutions and faculty actively engaged in research across a number of
      areas of computer science, based on the number of publications by faculty that have appeared at the most
      selective conferences in each area of computer science (see the FAQ for more details).
    </Text>

    <Text style={styles.description}>
      We gratefully acknowledge the generous support of our sponsors, including Stony Brook University. Sponsor CSrankings.
    </Text>

    <TouchableOpacity onPress={() => Linking.openURL('https://www.youtube.com/watch?v=hOSl3xPmHiQ&t=232s')}>
      <Image
        style={styles.videoImage}
        source={{
          uri: 'https://img.youtube.com/vi/hOSl3xPmHiQ/0.jpg',
        }}
      />
      <Text style={styles.videoText}>Click the image above to watch the introduction video on YouTube.</Text>
    </TouchableOpacity>

  </ScrollView>
</View>
```

Listing 6: About CSRankings View

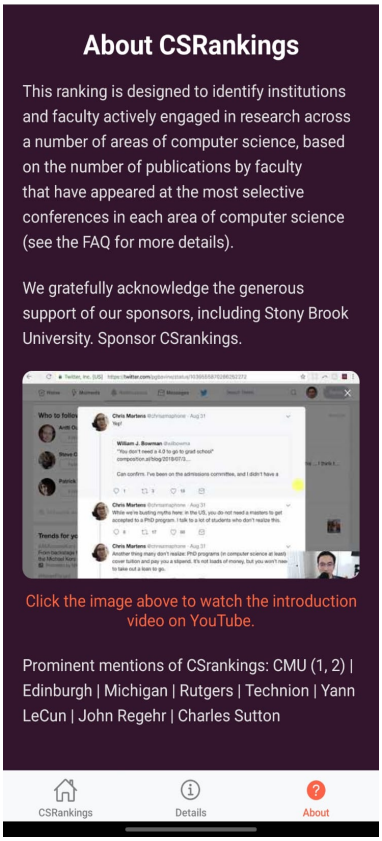

Figure 11: About Page.

5.5.5 Navigator Component

The Navigator component is the central navigation structure of the CSRankings app. It combines a Bottom Tab Navigator for main sections and a Stack Navigator for deeper navigation, enabling seamless transitions between screens.

```
<Tab.Navigator
  screenOptions={({ route }) => ({
    headerShown: false,
    tabBarStyle: { backgroundColor: '#f8f9fa' },
    tabBarIcon: ({ focused, color, size }) => {
      let iconName;
      if (route.name === 'HomeTab') iconName = focused ? 'home' : 'home-outline';
      else if (route.name === 'DetailsTab') iconName = focused ? 'information-circle' : 'information-circle-outline';
      else if (route.name === 'AboutTab') iconName = focused ? 'help-circle' : 'help-circle-outline';
      return <Ionicons name={iconName} size={size} color={color} />;
    },
    tabBarActiveTintColor: 'tomato',
```

```
    tabBarInactiveTintColor: 'gray',
  })}
>
  <Tab.Screen name="HomeTab" component={HomeStack} options={{ title:
      'CSRankings' }} />
  <Tab.Screen name="DetailsTab" component={InstitutionDetails}
     options={{ title: 'Details' }} />
  <Tab.Screen name="AboutTab" component={AboutScreen} options={{
     title: 'About' }} />
</Tab.Navigator>
```

Listing 7: Navigator Component Implementation

Component Details

HomeStack:

- Implements a Stack Navigator to manage navigation between:
  - **Home Screen:** Displays the list of institutions and rankings.
  - **Chart Screen:** Visualizes publication data with interactive bar charts.
- **Key Configurations:**
  - `headerShown: false`: Ensures no header is displayed for the Home screen for a cleaner UI.
  - `Dynamic Titles`: Displays a descriptive title for the Chart screen.

Tab.Navigator:

- Implements a Bottom Tab Navigator for the main sections:
  - **HomeTab:** Navigates to the Home Stack.
  - **DetailsTab:** Directly displays the Institution Details screen.
  - **AboutTab:** Displays the About screen with project information.
- **Dynamic Icon Handling:**
  - Uses `react-native-vector-icons/Ionicons` for tab icons.
  - Icons change based on focus (e.g., `home` vs `home-outline`).
- **Styling:**
  - **Active Tab:** Highlighted with a tomato color.
  - **Inactive Tab:** Shown in gray.
  - Background styled with light gray.

Key Functionalities:

- **Responsive Navigation:** Works seamlessly across Android and iOS.
- **Modular Design:**
  - Separates the navigation logic for clarity and reusability.
  - Ensures easy updates and maintenance.

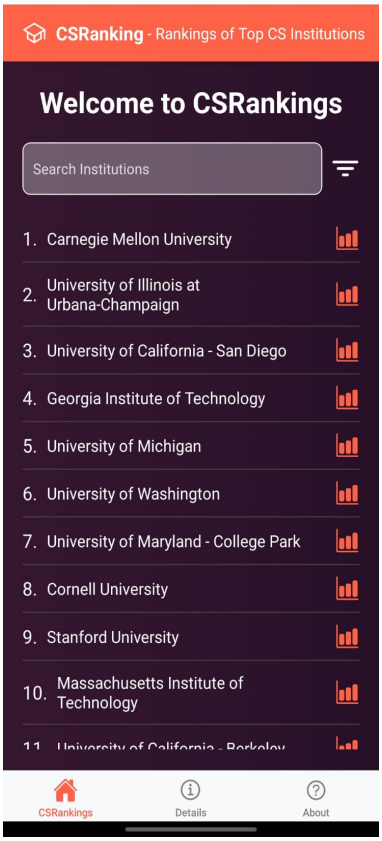

Figure 12: Navigator.

5.5.6 Styling and Theming

- Chose a consistent color scheme based on HEX code `#33152d`.
- Designed responsive layouts for various screen sizes.
- Used `StyleSheet` to modularize and reuse styles across components.

5.6 Feature Highlights

Interactive Features

- Dynamic search and filtering on the homepage.
- Interactive bar charts with data point click functionality.

Navigation

- Used `react-navigation` to implement stack navigation for multi-screen workflows.

Accessibility Enhancements

- Ensured touch targets were appropriately sized.
- Added labels and alt-text for accessibility compliance.

Responsive Design

- Tested on both Android and iOS to ensure consistent UI/UX.

6. Final Testing and Debugging

Testing and debugging were critical steps in ensuring that the `CSRankings` app was reliable, user-friendly, and performed well across devices. The following methods were employed for manual testing, automated testing, and resolving bugs during the development process.

6.1 Manual Testing

Platforms Tested:

- The app was tested on Android simulators, iOS simulators, and a variety of physical devices with different screen sizes.
- Devices used for testing included:
  - **Android:** Pixel 5, Samsung Galaxy S10.
  - **iOS:** iPhone 12, iPhone SE.

Features Tested:

- **Navigation:**
  - Ensured smooth transitions between tabs (Home, Details, About).
  - Verified the Stack Navigator functionality for navigating from Home to Institution Details and Chart screens.

- **Filters:**
  - Tested *"All Areas"* toggle to confirm it resets all sub-area selections.
  - Verified that selected sub-areas dynamically filtered institutions.
- **Chart Interactivity:**
  - Checked the rendering of bar charts for institutions with varying numbers of publications.
  - Confirmed that tapping on a chart bar displayed the correct publication details in an alert.
- **Institution Detail Page:**
  - Verified the accurate display of faculty information, including publication counts and external links.
  - Tested external links (e.g., Google Scholar, DBLP) to ensure they opened in a web browser.
- **UI Responsiveness:**
  - Ensured proper layout and spacing for different screen resolutions.
  - Checked the responsiveness of the search bar and tab navigation.

Edge Cases Tested:

- Institutions with no faculty data.
- Long institution names or sub-area titles.
- Empty search results.
- Rapid toggling of filters to ensure the app didn't crash.

Performance Checks:

- Verified that the app maintained a smooth scrolling experience even with a large dataset.
- Ensured minimal lag while navigating between screens and applying filters.

6.2 Automated Testing

To ensure the robustness of the application, automated tests were implemented using `Jest` and `@testing-library/react-native`. The focus was on validating the functionality of core components, navigation, and edge case handling. Below is an overview of the testing framework and the specific tests conducted.

Unit Testing Framework

- **Jest:** Used as the primary testing framework for its compatibility with React Native.
- **@testing-library/react-native:** Enabled testing of React Native components and their interactions.

Tests Implemented

1. **HomeScreen Component**
    - **Purpose:** To validate the homepage functionality and UI elements.
    - **Tests Conducted:**
        - Verified the rendering of the welcome message and search bar.
        - Ensured the filter modal opened upon clicking the filter icon.
        - Tested search functionality to filter institutions based on user input.

```
import React from "react";
import { render, fireEvent } from "@testing-library/react-
   native";
import HomeScreen from "../HomeScreen";

describe("HomeScreen", () => {
  it("renders the welcome message", () => {
    const { getByText } = render(<HomeScreen navigation={{}}
       />);
    expect(getByText("Welcome to CSRankings")).toBeTruthy();
  });

  it("renders the search bar", () => {
    const { getByPlaceholderText } = render(<HomeScreen
       navigation={{}} />);
    expect(getByPlaceholderText("Search Institutions")).
       toBeTruthy();
  });

  it("opens filters modal when filter icon is pressed", () => {
    const { getByTestId, getByText } = render(<HomeScreen
       navigation={{}} />);
    const filterButton = getByTestId("filter-icon");
    fireEvent.press(filterButton);
    expect(getByText("Filter by Areas")).toBeTruthy();
  });
});
```

Listing 8: HomeScreen Tests

2. **ChartScreen Component**

   - **Purpose:** To verify the proper rendering of charts and the functionality of chart interactions.
   - **Tests Conducted:**
     - Ensured the chart title and legend rendered correctly.
     - Validated the click functionality on chart bars to display an alert with accurate details.

```
import React from "react";
import { render, fireEvent } from "@testing-library/react-
   native";
import ChartScreen from "../ChartScreen";

const mockRoute = {
  params: {
    data: [5, 10, 15, 20],
    name: "Test Institution",
    categories: [
      {
        category_id: 1,
        category_name: "Category 1",
        sub_areas: [
          { sub_area_id: 1, name: "Area 1" },
          { sub_area_id: 2, name: "Area 2" },
          { sub_area_id: 3, name: "Area 3" },
          { sub_area_id: 4, name: "Area 4" },
        ],
      },
    ],
  },
};

describe("ChartScreen", () => {
  it("renders the chart title", () => {
    const { getByText } = render(<ChartScreen route={mockRoute}
        />);
    expect(getByText("Research Areas for Test Institution")).
       toBeTruthy();
  });

  it("renders the legend correctly", () => {
    const { getByText } = render(<ChartScreen route={mockRoute}
        />);
    expect(getByText("Area 1")).toBeTruthy();
    expect(getByText("Area 2")).toBeTruthy();
  });
```

```
35
36    it("handles bar clicks correctly", () => {
37      const { getByText } = render(<ChartScreen route={mockRoute}
            />);
38      const bar = getByText("Area 1");
39      fireEvent.press(bar);
40      expect(getByText("Bar Clicked")).toBeTruthy();
41    });
42  });
```

Listing 9: ChartScreen Tests

3. **AboutScreen Component**

   - **Purpose:** To validate the content rendering on the About screen, including links and titles.
   - **Tests Conducted:**
     - Verified the correct rendering of the *"About CSRankings"* title.
     - Ensured all external links and text content were visible.

```
1  import React from "react";
2  import { render } from "@testing-library/react-native";
3  import AboutScreen from "../AboutScreen";
4
5  describe("AboutScreen", () => {
6    it("renders the about title", () => {
7      const { getByText } = render(<AboutScreen />);
8      expect(getByText("About CSRankings")).toBeTruthy();
9    });
10
11   it("renders the video link", () => {
12     const { getByText } = render(<AboutScreen />);
13     expect(getByText("Click the image above to watch the
          introduction video on YouTube.")).toBeTruthy();
14   });
15
16   it("renders external links correctly", () => {
17     const { getByText } = render(<AboutScreen />);
18     expect(getByText("https://github.com/emeryberger/CSRankings
          ")).toBeTruthy();
19   });
20 });
```

Listing 10: AboutScreen Tests

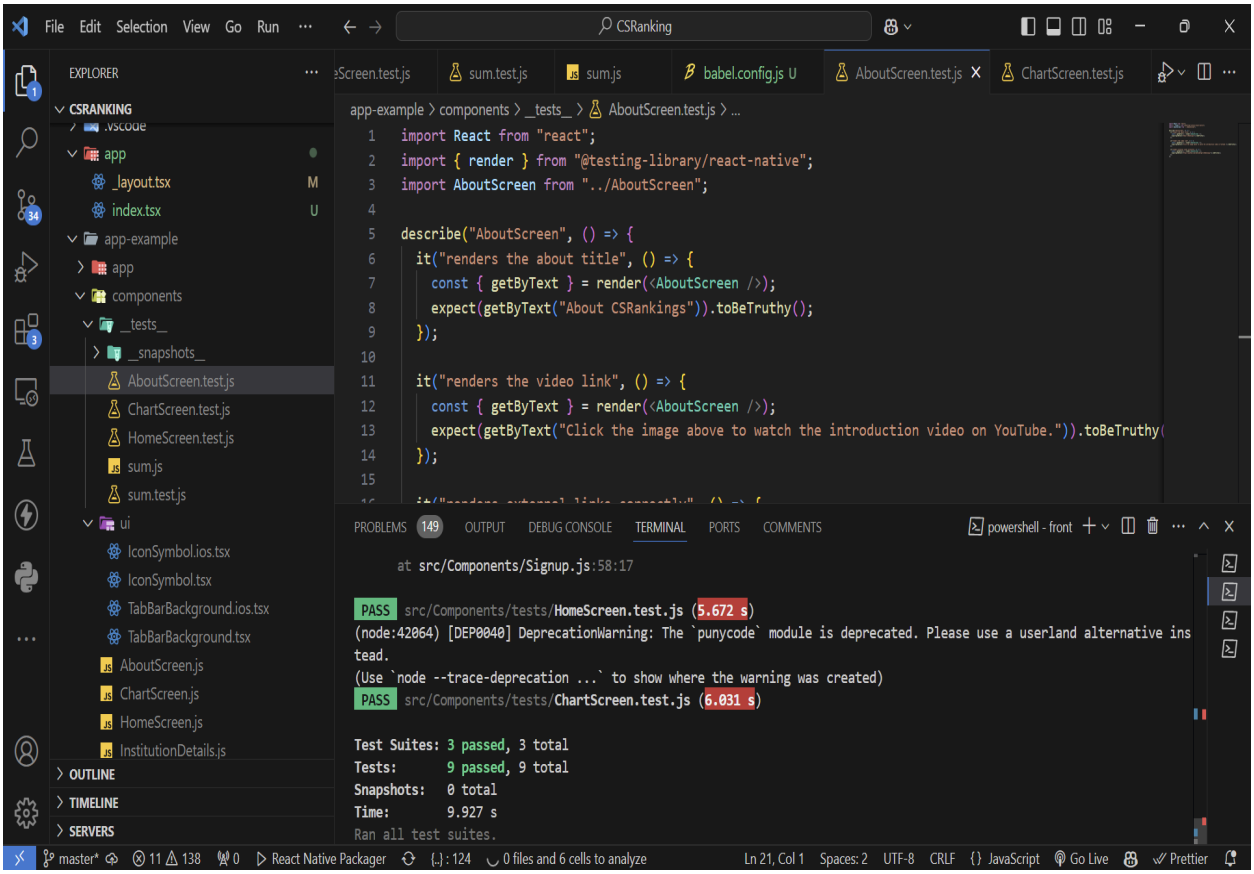

Figure 13: Test Cases.

7 Challenges and Assumptions

7.1 Challenges

- **Understanding the Website's Structure:** Without the ability to ask questions, reverse-engineering the hierarchy and functionality of the original website was challenging. I relied on closely analyzing the website's behavior and data presentation to make accurate assumptions.

- **SDK Compatibility Issues:** Midway through the project, Expo SDK 51 faced compatibility issues with several dependencies. This required updating the project to SDK version 52. I resolved this by running the installation with the `@latest` flag, which ensured all dependencies were updated to versions compatible with the latest Expo SDK.

- **Dynamic State Management:** Managing dynamic states for filters and charts presented challenges in ensuring real-time updates while maintaining performance. React's `useState` and `useEffect` hooks were extensively optimized to handle the complex dependencies.

- **Chart Interactivity:** Implementing interactive charts required additional attention to handle tap events accurately, especially for small screens. Adjusting bar spacing and scaling dynamically improved usability.
- **Data Integration:** Merging multiple static JSON files to simulate backend behavior was tricky. I ensured that the linked data between institutions, faculty, and publication areas was robust and error-free.
- **UI Responsiveness:** Designing the UI to maintain consistent layouts across various screen sizes and orientations required careful styling adjustments using Flexbox and percentage-based dimensions.

7.2 Assumptions

- **Hardcoded Data:** As backend integration was outside the scope of this project, I assumed that all research areas, institutions, and publication data could be hardcoded into JSON files.
- **Core Functionality:** The focus was primarily on replicating the main features of the website, such as rankings, filters, and charts, rather than implementing advanced backend features or additional enhancements.

8 Platform Compatibility Testing

8.1 Android Compatibility

- Verified the application on Android simulators and physical devices.
- Tested navigation, filters, and charts on various Android screen sizes.
- Ensured responsiveness and performance remained optimal, even on older devices.

8.2 iOS Compatibility

- Tested the app on iOS simulators and devices such as iPhone 11 and iPhone 14.
- Verified smooth transitions between screens and chart interactivity.
- Checked compatibility with iOS-specific behaviors like swipe gestures and safe areas.

8.3 Cross-Platform Testing

- Ensured the app adhered to platform-specific design guidelines while maintaining consistent functionality.
- Adjusted navigation and modal styles to align with Android's Material Design and iOS's Human Interface Guidelines.
- Validated the use of React Native's platform-specific APIs to ensure seamless behavior on both platforms.

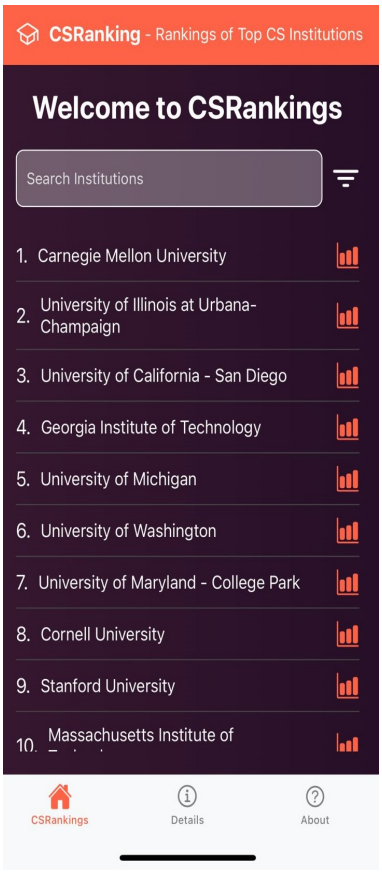

Figure 14: CSRanking app on IOS.

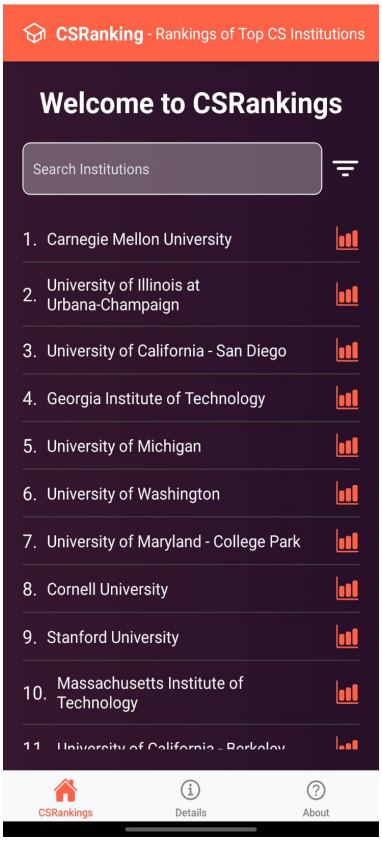

Figure 15: CSRankingAPP on Android.

9 Conclusion and Future Enhancements

9.1 Conclusion

The `CSRankings` app successfully replicates the core functionality of the website in a mobile-friendly manner. The project demonstrates a robust understanding of React Native, state management, and dynamic UI/UX development.

9.2 Future Enhancements

- Adding real-time backend integration to fetch live data.
- Including additional interactive visualizations (e.g., pie charts).
- Enhancing filters with multi-select and search options.
- Improving accessibility features, such as voice commands and larger text support.

10 Disclosure of AI Tool Usage

LLMs like ChatGPT were instrumental in developing the `CSRankings` Mobile App by assisting with:

- **Styling:** Suggested color schemes, typography, and responsive UI layouts to enhance cross-platform usability.
- **Logic Development:** Provided optimized solutions for managing dynamic filters, charts, and data integration using React hooks.
- **Debugging:** Helped resolve layout issues, SDK compatibility challenges, and state management bugs.
- **Report Writing:** Structured and articulated the project report, including challenges, solutions, and technical implementations.
- **Code Quality:** Guided best practices for React Native, modularization, and writing unit tests with Jest.

11 GitHub Repository

The project can be accessed at the following link: `https://github.com/rmusukudabbidi/CSRanking`