11/29/2020
Author: Michael Thomason (https://github.com/michaelthinks/)
Team Members: Paul Kruskamp, Michael Thomason, Novann Rouse, Patrick Mean, Seth Moreland
Rev. 1

Developer Documentation: CCSE Events

Overview

CCSE Events is a cross platform mobile application built using React Native and the Expo platform. Expo allows for easy testing and building of React Native applications as well as additional modules and libraries for accomplishing different functions on mobile devices (ie. notifications). This project was built to support both the Android and iOS platforms but was developed primarily using Android and Windows. Testing on iOS was performed using the Expo iOS application available on the App Store (this application allows you to view Expo applications on a remote device connected to the same WiFi network as the development server). This application is also available or Android, but the Android Emulator included with Android Studio was primarily used for development. Any build instructions contained in this document were used on Windows specifically but should translate to macOS. Building and development on Linux is essentially the same as on Windows, but any differences will be noted throughout the documentation.

Application Data Sources

The CCSE Events applications retrieves event data from the CCSE Calendar RSS feed, located at:

https://calendar.kennesaw.edu/department/college_of_computing_and_software_engineering/calendar/xml

Any events that need to be shown in the application need to be posted to the CCSE Calendar and therefore show up in this feed. Any events outside of this feed will not appear in the CCSE Events app. If the format of this feed is ever altered or the feed is moved or removed, then the CCSE Events application will need to be updated to support the new format or to access the new data source.

This application stores all event data within the application itself using React Native's persistent AsyncStorage module. Data is loaded from AsyncStorage into easily accessible global variables for use around the application so that functions do not necessarily alter the data in AsyncStorage directly - they alter data in the global variables which are then saved to AsyncStorage. All application data is stored in JSON format.

Important Storage Items

- AsyncStorage items:
    - Key: events
        - Persistent storage for the primary list of CCSE Events
    - Key: likedEvents
        - Persistent storage for the list of events liked by the user
    - Key: settings
        - Persistent storage for user settings
- Global variables used within the application (the data in AsyncStorage is transferred to one of these variables for easy access throughout the application):
    - global.eventDataSource
        - Accesses all available events (events in AsyncStorage)
    - global.likedEvents
        - Accesses events liked by the user (likedEvents in AsyncStorage)
    - global.settings
        - Accesses application settings (settings in AsyncStorage)
- Style variables that access the stylesheet for the application, stored in styles/styles.js. This is not a global variable, it is just imported into each screen under this variable name:
    - globalStyles

External Documentation

- React Native: https://reactnative.dev/docs/getting-started
- Expo: https://docs.expo.io/
- Android Studio: https://developer.android.com/studio/intro
- Mozilla MDN (for generic Javascript documentation): https://developer.mozilla.org/en-US/

The Project Environment

- Primary Desktop Operating Systems used for this project: Windows 10, Ubuntu Linux 20.04
- Primary Mobile Operating Systems used for this project: Android 10 (API 29)
    - Testing was performed on iOS 13 using the Expo application available on the App Store
- Build Tools - node.js, npm, expo, Android Studio
    - Node.js (includes npm, the node package manager, that is used to install expo): https://nodejs.org/en/
    - Android Studio: https://developer.android.com/studio

Building The Project For Development

- Install node.js
    - The latest LTS release was used for this project - as of this writing it is 14.15.1 LTS
- Install Android Studio
    - Once Android Studio has been installed, go into AVD Manager (Android Virtual Device Manager) and create a new virtual device. This project was created using the following virtual device properties
        - Pixel 3a
        - API 29 (Android 10)
    - Once setup, run the newly created Android emulator
- Open a command prompt as an Administrator (or terminal in Linux/macOS) and navigate to the root project folder
    - Note - building WILL NOT COMPLETE IN WINDOWS unless the command prompt is opened as an Administrator
- Install expo-cli
    - npm install expo-cli
- Install expo - this command will create a few new directories in the root project folder (node_modules, .expo, etc.) - THIS COMMAND MUST BE RUN IN THE ROOT PROJECT FOLDER
    - npm install expo
    - *Linux note: You may need to use yarn instead of npm - ie. yarn install expo*
- Install dependencies - run the following commands to install the dependencies for the project - THESE COMMANDS MUST BE RUN IN THE ROOT PROJECT FOLDER
    - npm install react-native-webview
    - npm install react-native-xml2js
    - npm install @react-native-community/async-storage
    - npm install expo-calendar
    - npm install expo-permissions
    - npm install expo-notifications
    - npm install expo-constants
    - *Linux note: You may need to use yarn instead of npm to install the above modules*
- Run the project
    - npm start
        - *Linux note: You may need to use expo start instead of npm start*
    - Once you run the project, a web browser will open with access to the expo build tools
        - If you are using the Android Emulator for testing, make sure it is up and running
    - Select the link to Run in Android Device/Emulator in the bottom left corner
        - Expo will install the expo app within the emulator and run the CCSE Events application

    - Other options for running the project are available in the bottom left corner including running on iOS (macOS only), running in a web browser, or you can scan the QR code using the expo application that is downloadable from the Apple Store and Play Store to test the app on your own device
        - Please see the expo docs for more information: https://docs.expo.io/

Troubleshooting Tips when using expo and the Android Emulator

- Expo is not perfect when it comes to running and testing the app - the expo application within the emulator will sometimes freeze or fail to load the app
- Common troubleshooting procedures include:
    - Opening the Recents menu in the Android Emulator and closing the expo and CCSE Events applications and re-running the app via the expo build web page
    - Fully uninstalling the Expo application from the emulator and selecting Run on Android Emulator again in the expo web page to allow it to re-install a fresh copy of the expo app and the CCSE Events app in the emulator
    - Uninstalling the expo application from the emulator, shutting down expo, and starting it all back up again
        - ie. closing the command prompt/terminal that expo is running in, opening it open again, navigating to the project root directory, and running npm start to restart expo
        - Restarting the emulator is normally not necessary - just uninstall the expo app from the emulator

Building the Project For Production

- The expo platform provides cloud build tools for easily building the app for deployment.
- To build the app for Android, open a command line (as an Administrator) or terminal and navigate to the root project folder.
    a. Run the following command: expo build:android -t apk
    b. You will be prompted to enter your expo credentials or create a new expo account in order to access the build server.
    c. Expo will then bundle the application up, send it to their build servers, and build the apk for deployment on Android devices. You will be given a URL in the command line/terminal that you can go to in order to view the build process.
    d. Building takes 10-15 minutes, depending on what kind of load the build server is experiencing. Once it is finished, you will be able to download the completed Android apk file.
- Building was ***not*** tested for the iOS platform since it requires macOS and XCode for deployment to an iOS device or the App Store.
    a. iOS testing was performed during testing and development using the expo app available on the App Store.

- You can find more information on building the project for deployment to both Android and iOS at the following link:
    a. https://docs.expo.io/distribution/building-standalone-apps/

Project Documentation

App.js

App.js is the main starting point for the application. It handles loading the navigation drawer (which, despite its name, handles navigation and routing application wide, not just in the drawer itself) and startup procedures for the application including calling the functions that check and retrieve the event data (located in AppFunctions.js). It is important to note that the name of the screens in Drawer.Navigator are the names used to access the screens throughout the application. For instance, if you need to navigate from one screen to another, you will use the name it is given in its respective Drawer.Screen component to get to it.

Scripts/AppFunctions.js

AppFunctions.js contains the majority of the functions that handle data retrieval, data formatting, data storage, and notifications. It is imported into various screens throughout the app and is normally accessible via the AppFunctions variable.

- retrieveAndSaveEventData
    - Parameters: URL for the data source (the CCSE Event RSS Feed)
    - Asynchronous function
    - Description: Main function used to retrieve and save the event data from the data source. Also handles updating the data. This function is called when the application is initially loaded, every time the application is loaded. If event data already exists, it checks to see if it has been updated and updates the data accordingly. The data retrieved from the CCSE feed is in RSS format, and this function uses the xml2js module to convert it to a JSON object. That raw JSON object is passed to formatEventData to format it into a more readable and usable JSON object with any unnecessary elements removed. It then stores the data in AsyncStorage as well as the eventsDataSource global variable.
- storeDataItem
    - Parameters: AsyncStorage key, the item to store
    - Asynchronous function
    - Description: Stores the data passed (the item to store) under the given storage key in AsyncStorage for persistent storage. Data stored in AsyncStorage will persist after the application has been shut down.
- loadDataItem
    - Parameters: AsyncStorage key
    - Asynchronous function

    - Description: loads the data under the given AsyncStorage key from AsyncStorage.
- checkLikedEvents
    - Parameters: None
    - Asynchronous function
    - Description: Called each time application is initially loaded. Checks to see if the likedEvents object exists in AsyncStorage. If it does not, it creates it.
- calendarPermissions
    - **Experimental Function**
    - Parameters: None
    - Asynchronous function
    - Description: Called when the application initially loads to check for the appropriate calendar permissions. This function is not actively used and is an experimental function that remains for future development.
- formatEventData
    - Parameters: raw JSON event data
    - Description: formatEventData accepts the raw JSON data from retrieveAndSaveEventData and creates an easily digestible JSON data object. It removes any unnecessary data fields created during the conversion from RSS to JSON and renames the object keys to more friendly names. It returns the newly formatted JSON data object.
- OpenURL
    - Parameters: the URL to open
    - Description: Opens the given URL in the devices external web browser. Does not support deep links such as slack://, twitter://, etc.
- getPermissions
    - Parameters: None
    - Asynchronous Function
    - Description: Only applicable for iOS platforms. Called when the application initially loads. Checks to make sure the application has notifications permissions.
- sendNotifications
    - Parameters: notification title, notification body
    - Description: Sends an immediate notification to the user with the given title and body.
- scheduleNotification
    - Parameters: notification title, notification body, notification time in milliseconds since UNIX epoch
    - Asynchronous function
    - Description: Schedules a notification with the given title and body to occur 30 minutes before the given time. Time must be passed in milliseconds since the UNIX epoch.
- loadSettingsState
    - Parameters: None

    - Description: Called when application initially loads. Loads settings data from AsyncStorage into a global variable.
- cancelNotification
    - Parameters: notification ID
    - Description: Cancels the notification at the given notification ID
- cancelAllNotifications
    - Parameters: none
    - Description: Cancels all scheduled and current notifications.

Screens/Home.js

Home.js is the first screen the user sees when starting the app. It contains buttons for quick navigation to CCSE Events.

Screens/EventList.js

EventList.js renders the list of all CCSE Events available using a React FlatList. It also contains most of the functions pertaining to rendering dynamic UI items, liking events, updating screens, and other UX related functions.

UI Functions

- renderEventItem
    - This JSX function is called by the FlatList for each event that needs to be rendered. The FlatList hands the event to renderEventItem (the item) which then renders it on screen as a list item.

Functions

- goToEvent
    - Parameters: the event object
    - Description: Opens the Event Details screen for the given event
- componentDidMount
    - Description: this is a builtin React Native function - it assigns a listener to the screen so that it updates properly each time the user visits it.
- likeEvent
    - Parameters: event name
    - Description: Triggers when the user clicks the like event button. It adds the event to the likedEvent variable and it's respective AsyncStorage data source. It also handles disliking/deleting events from likeEvents.
- updateEvents
    - Parameters: boolean describing whether or not an event was liked
    - Asynchronous Function

    - Description: called after the user likes an event or clicks the refresh button. Calls retrieveAndSaveEventData to load any new event data and refreshes the screen. If the user selected the refresh button, it also displays a toast notification letting the user know that a refresh is occurring.
- checkLiked
    - Parameters: event name
    - Description: Used to determine whether to show the liked or not liked icon in the events list. Checks global.likedEvents to see if the given event has been liked and returns the JSX for a liked event icon if so (a solid heart) or a non-liked event icon if not (empty heart).
- formatDate
    - Parameters: date
    - Description: formats a date given in Javascript format to a user readable string for display in the UI. Returns the formatted date ready to display.
- openCal
    - **Experimental Function**
    - Parameters: event name, date in Javascript format
    - Asynchronous Function
    - Description: Experimental function for opening the devices calendar with a new event at a specific date and time. This function did not work properly, but it was left for further development. This feature may require the use of native code (Kotlin/Java/Swift) in order to properly implement, as React Native/Expo does not seem to have any readily available and usable solutions.

Screens/EventDetails.js

EventsDetails renders the details of a specific event and is called by EventsList or LikedEvents. It is passed an event to render, which is accessed via this.props.routs.params. It looks up the event in the events list and renders it accordingly. EventDetails renders the event description in a WebView as the CCSE data source provides a description with embedded HTML tags. This greatly reduces the need to parse and format the description data as we can just let the WebView handle it. There are 2 functions in EventDetails.

- updateEvents
    - Parameters: None
    - Description: Forces the screen to update in case there is new information.
- openLinkFromWebView
    - Parameters: a url to a web site
    - Description: This function is called when a user clicks a link in the event description - it opens the link in the devices web browser.

Screens/LikedEvents.js

LikedEvents is built very similarly to EventDetails (almost identical, but LikedEvents does not contain any additional functions). It renders the events in global.likedEvents using a FlatList. The JSX function renderEventElement is called for each event item by the FlatList component.

UI Functions

- renderEventItem
    - This JSX function is called by the FlatList for each event that needs to be rendered. The FlatList hands the event to renderEventItem (the item) which then renders it on screen as a list item.

Functions

- goToEvent
    - Parameters: the event object
    - Description: Opens the Event Details screen for the given event.

Screens/Settings.js

Settings contains 2 radio buttons for enabling/disabling notifications. It is important to note that this screen works directly with global.settings to update settings; however, a state variable is needed for the switches to properly display whether or not they are enabled or disabled. If you attempt to reference their state directly from the global.settings variable, they will not update correctly, thus the state variable is used as a middle man for the switches. Each switch calls its corresponding function when the user selects it - the functions then either turn on or off the notifications pertaining to the switch.

- notificationsForLikedEvents
    - Parameters: None
    - Description: Checks to see if liked event notifications are enabled or disabled in global.settings and reverses it. If the notifications are disabled, it cancels all scheduled events. If they are enabled, it iterates through global.likedEvents and creates a new scheduled event for each liked event.
- notificationsForNewEvents
    - Parameters: None
    - Description: Checks to see if new event notifications are enabled or disabled in global.settings and reverses it. No further actions are necessary as new events are called as needed and not scheduled. Any function that wants to send a notification simply checks global.settings.newEvents and, if it is set to false, does not send a notification.

Screens/Maps.js

Maps displays 2 buttons and allows the user to view maps of the Kennesaw and Marietta campuses. These maps are scrollable vertically and horizontally. Selecting a map button sets the screen state variables to the proper values and then the page refreshes and renders the corresponding map. A close button is displayed above the map and, when the user selects it, it resets the state variables to their default values and the current map disappears.

Screens/CcseHomepage.js

CcseHomepage displays a WebView containing the KSU CCSE Homepage. An icon available in the upper right corner allows the user to open the web page in the devices web browser.

Screens/Twitter.js

Twitter displays a WebView containing the KSU CCSE Twitter page. An icon available in the upper right corner allows the user to open the web page in the devices web browser.

Screens/Facebook.js

Facebook displays a WebView containing the KSU CCSE Facebook page. An icon available in the upper right corner allows the user to open the web page in the devices web browser.

styles/styles.js

Styles contains all of the JSX styles (very similar to CSS) for the app. The app is rendered mainly using Flexbox styling techniques. Styles are imported within each screen and are accessed via the globalStyles variable. It is important to note that some screens (notably Settings and Maps) did not properly display the header text and icons for some reason. Therefore, they have their own set of styles that are prefixed with “adjusted” to correct for this issue.

Assets

The Assets folder contains all images used within the app (icons, maps, backgrounds, etc.).

TestData

The TestData folder contains JSON files with test data that you can view if you need an idea of how the data is formatted after it is imported and stored.

- EventDataPreFormatSample.json - contains the raw JSON data before it is formatted by formatData.
- EventDataSample.json - contains formatted JSON data - this is how data is stored in global.eventsDataSource and AsyncStorage.
- LikedEventsDataSample.json - contains formatted JSON similar to what is stored in global.likedEvents and AsyncStorage (it is identical to normal event data except that it contains only events that have been liked).
- RawRssDataSample.json - contains the raw JSON data right after it has been converted by xml2js. Only the data is “channel” is sent to format data, the rest is stripped before formatting.

App.json

App.json is the expo configuration file. It allows you to specify details like the app icon, the splash screen, package name, tablet support, etc. See the expo documentation for more information on app.json.