Apr 23rd, 9:00 AM

React Native Photo & Video Streaming/Processing API Integration

Anamaria Oharciuc
Rowan University

William Carr
Rowan University

Sushanth Ambati
Rowan University

Ryan Blaisdell
Rowan University

Kyle Reed
Rowan University

See next page for additional authors

Student Name
Anamaria Oharciuc, William Carr, Sushanth Ambati, Ryan Blaisdell, Kyle Reed, and Jack F. Myers

RowanUniversity
Department of Computer Science

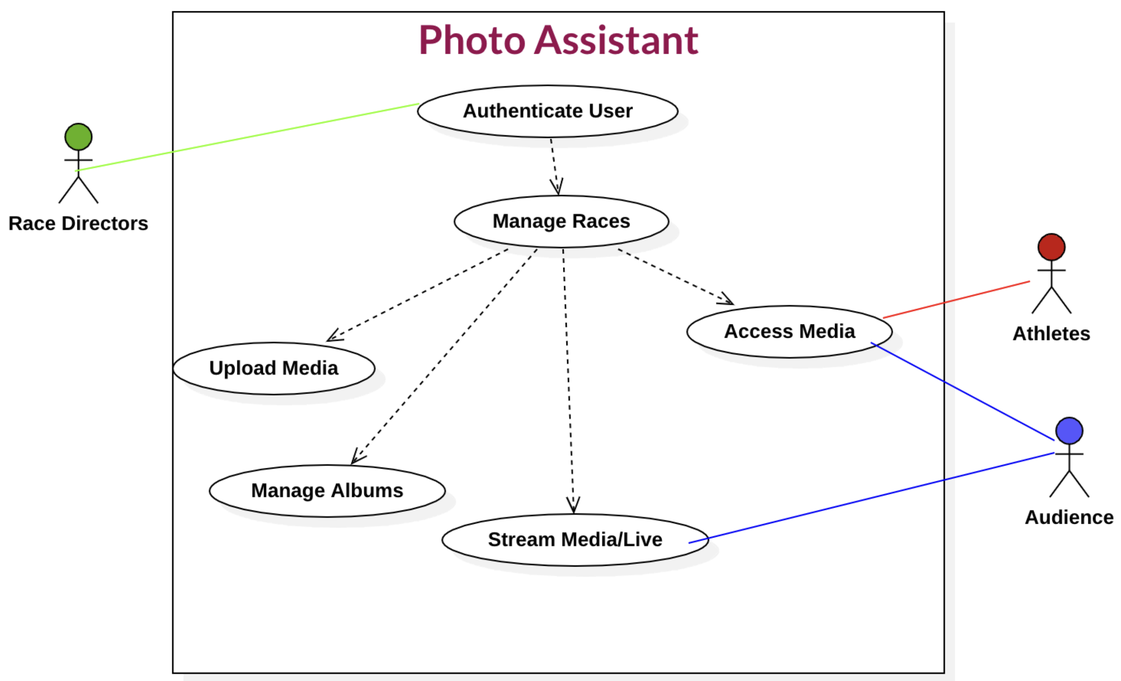
React Native Photo & Video Streaming/Processing API Integration

Sushanth Ambati, Ryan Blaisdell, William Carr, Anamaria Oharciuc, Kyle Reed
Faculty Advisor: Jack Myers

Client Proposal

RunSignup is a software company specializing in event management technology. Event organizers, known as Race Directors, utilize their platform to create and manage events, organize media albums, stream their events to the public, and more. In order to allow Race Directors to complete these actions in real-time at their events, RunSignup asked our team to develop a mobile app.

With the app, Race Directors would be able to upload and stream photos to the event's photo albums as soon as they are taken, and they could livestream the event to YouTube directly from their mobile device. Even if the device lost connection, the app would be expected to cache the photos until Internet connection is regained.



Project Description

- The application streamlines the media upload process by allowing Race Directors to access RunSignup's photo platforms on their mobile devices.
- We built the application using a variety of technologies we had no experience with, including React Native, OAuth 2.0, and the RunSignup and YouTube APIs.
- The RunSignup API was utilized along OAuth 2.0 to access RunSignup's photo platform, allowing us to manage event photo albums and upload photos to them from a mobile device. Photos can be uploaded from the user's camera roll or taken directly within the app using their camera.
- The YouTube API was integrated into the application for its livestreaming features. Race Directors can schedule livestreams for their events within the app and then stream directly from their device to their YouTube channel. The livestream is recorded in real time and automatically uploaded to the user's YouTube account upon completion. OAuth was used to facilitate the YouTube login process.
- The application is capable of caching photos in the upload queue if a device is offline; when the device is back online, the photos automatically upload themselves to their respective albums.

Image Processing in Different Scenarios

- The Race Directors would need to be able to send images to their event's albums in two different ways – uploading from their camera roll or being able to "stream" photos to the albums in real time.
- We needed to make use of different libraries and components within the React Native framework in order to develop both methods of image processing. This tasked the team with researching technologies that were very fresh to us.
- Our team was introduced to RunSignup's API to facilitate the upload of photos taken on the device to RunSignup's servers.

Network Failure Handling

- Our application was required to handle the process of uploading and streaming photos even if disruptions to the connection occurred.
- The first step in ensuring that an upload went smoothly was to check the fetch call's HTTP response code. A code in the 200-299 range meant a successful upload, while anything 300 and up could be considered a failure to call the API methods.
- In this case, we have the react native community package, netinfo, running in the background to notify the user upon losing connection.
- The package can only determine so much, as it was built as a quick multi-platform tool, which is why we decided to supplement it with our own periodic pings to a website on the Internet. The pings could be used to periodically reevaluate the internet connection status and determine if the app could resume its upload.
- While offline, a queue of photos would be managed by the app to keep track of which photos would need to be uploaded to the RunSignup API when the connection comes back.

OAuth 2.0 & YouTube Livestreaming

- YouTube presented many new hurdles for the project.
- First, in the Google Cloud API service we needed to register our app to have the ability to connect to the YouTube API.
- Next, OAuth 2.0 is required by Google in order to use certain functions we needed from the YouTube API, but our app had to move from being an Expo Go app into an Expo development build in order to do so.
- Once our build was set up, we needed a redirect URI scheme for the app to register with the Google Cloud API.
- After the registration process, we could finally access the YouTube API. This gave us access to the list of LiveBroadcasts and the ability to start a live stream with the use of api.video's React Native Livestream package.