Builds & Versioning (Cycling Route Marker - Mobile App)

Basics

Android .apk binaries can be downloaded and installed on any Android device that you have access to (the device password is usually sufficient). This document is for the team to have a unified procedure for versioning and builds in a way that doesn't confuse the rest of the team.

The commands and steps for checking if the build is ready and using EAS builds through Windows Powershell (you may need to enable signed script execution to do this, or your PC will block it for security reasons) are below. **Internet access is required.**

The bolded text in the Instructions section are shell commands, not including the preceding LT symbol (represents cmd line opener). The purple text is code in the JS files, and italics are file names. These have been tested in Windows Powershell with the appropriate permissions; if you do not have script execution permission for signed scripts, you might need to enable it.

Instructions

1. > **npm install -g eas-cli**
    a. You do not need to repeat this after you can see it was installed for the user globally using **npm ls -g**, but you may need to add **@latest** to the end of that command if versioning becomes an issue in the distant future.
    b. This command can be run anywhere if you have admin access.
2. > **eas whoami**
    a. If this does not return your expected Expo account name, you may need to log in
    b. This is the first instruction that may need script execution. If it fails, figure out how to modify your Windows settings using an administrator powershell.

The following commands should all be run in the same folder as the app's *package.json*

3. > **npm install**
4. > **npx expo-doctor@latest**
5. Fix any errors that expo-doctor finds, and then start over from step 1 if changes were made.
6. Versioning and profile selection (see Versioning & Profile section)

    a. Preview build step 1: in App.js, comment out import 'expo-dev-client';
7. Make sure all code content has been committed to .git and pushed to Github (see below for rules)
8. > **eas build --profile development --platform android**
    a. Preview build step 2: use **--profile preview** instead for preview builds

Step 7 will likely have a several-minute wait time; iOS builds take longer due to requiring Apple silicon (so we wait for Apple servers as well as Expo servers). If the connection to your console does not time out (or is not exited so we can keep using the shell, not all of us will stop working while it is building), it will return a QR code or notify you about errors in the build process. Any build issues will also be visible on the expo.dev dashboard for this project. When making a preview build, not commenting out import 'expo-dev-client' in *App.js* will cause the app to still have the dev server requirement to run.

Production builds capable of being sent to the App Store have more stringent requirements and are not implemented at the moment; please do not use this profile until further notice. Unexpected and unknown behavior will likely occur.

Versioning & Profile

The reason for the development and preview build profiles are simple: one is for debugging functions with the dev server and a native app, and one is for testing feature functionality in practical uses. If you do not have the dev server, you cannot start an app using Expo Go. While we can remove the dev server from the development profile, it is intended to include the expo-dev-client wrapper that Expo Go provides for each Expo SDK.

There are differences between builds and Expo Go, however. Because Expo Go does not always behave as the native build would, we should use development builds to ensure our preview builds do not have any outstanding issues that would block testing of the target features.

LONG STORY SHORT: Make sure your feature works in Expo Go before trying to build a native version. Make sure your feature at least visually functions (logs and debug state) in a development build before building and testing the preview build.

Versioning is intended to make sure that multiple people with the ability to create builds do not interfere with one another as they work. While we can tell who calls for a build, it is not always obvious which build requires which version for the dev server, or will have what features available to test. In tasks that multiple developers are working on, **communication and pre-warning (as well as checking GitHub for commits and expo.dev for the latest builds) is very important**.

Note that the versioning number isn't especially useful for tracking task completion. Its most important function is to make sure we can always find specific builds on expo.dev because I've likely already demonstrated what the problem is (gaze upon my despair with Task80 and expo-sensors).

Main/Dev Build Versioning will observe the following rules:

- Versions **1.X.Y** will encompass the progress made with this project during Capstone 2023-2024
- Version **1.0.Y** began during Sprint 4 and will end during Sprint 5
- **X** will be the Sprint # minus 5 during 2024
    - Example: Sprint 6 = **1.1.Y**
- **Y** will be incremented with each user story or task added to dev that modifies code functionality. Modifications to the packages and build files do not count.
- If a build fails, add **-rev#** to the project name for the next commit-and-build attempt, with the number starting at 1 and incrementing with each revision commit (because further edits to the versioning number is not possible)
- The version of this product provided to the sponsor at the end of the Capstone will be incremented to 2.0.0

Build Versioning on task branches will extend the above rules in the following ways:

- Add the Task# of the branch (keep the X & Y from where it branched off) with the format **-t#** to the project name field in *app.json*, resetting (removing) the revision tags for the first build of that specific task# at the same time
    - Theoretical Example: **GPXImport-t46-rev3**, version **1.1.4**
    - The example shows task 146's 4th (the 0th won't have a **-rev#**) build attempt
    - Add a letter after the task # if extreme modifications are made to the task goals
    - This should be removed before/during pull requests to dev or main
- If a task starts having too many revisions and builds, it may be a sign that it encompasses too many goals for a single task. If that seems to be the case, try splitting the task and moving to a new task # so that we don't sort through twenty revisions for a single task.
    - This is left to developer discretion, but should be kept in mind.
    - Expo Go testing should pass muster before native builds are attempted; do not use preview builds for debugging!

The version tag will be in *app.json*, with the important Y-increment number being either 1 or 2 greater than the current dev version. Modify before the pull request goes through if other pull requests have incremented the tag on dev before your pull request.