##### **Release Management Plan**

This document describes how the team intends to manage the project's software release such that anyone familiar with the project can read to understand how the stable release will be distinguished from the current development version as well as how the project's deployment dependencies (technology, libraries, resources, etc.) are managed. This document serves the purpose of clarifying the packaging and delivery of the elements of a product.

In your repository Wiki, describe your plan for the following:

* **Release Environments**. List all the places you expect the app to be deployed (e.g., app and server, app on multiple platforms, web or mobile app, etc.) and your ideal production environment for each (note, include in your planning the *minimal* version of the software you can use, i.e., deploying only to the latest version of iOS or Android limits your app to only those users that have upgraded recently). How do you plan to separate the production environment(s) from the test environment(s) (i.e., there should always be a current stable version of your app available to demo)?
  + We will deploy Cornelius on Google Cloud Platform. Cornelius is a mobile app and we will deploy to the latest version of iOS and Android.
  + We have a master branch that contains a current stable version of our app. This branch contains the latest stable version of the code and all the features that pass the testing stage will be merged to master. We also have another development branch that contains the ongoing code. The newest features will be merged to this branch first and we will conduct test and code refactoring in this branch.
  + In the future, the test version of the app will be deployed in QA and the production version of the app will be deployed to another domain.
* **Release Packaging**. List the current and expected project dependencies. How do you plan to make them known and easily gettable for future maintainers (e.g., Docker container, maven or package-info file or other dependency manager, VM, etc.)?
  + Current dependencies
* Frontend (package.json):
  + react-navigation
  + expo
  + expo-font
  + expo-status-bar
  + native-base
  + react
  + react-dom
  + react-native
  + react-native-gesture-handler
  + react-native-reanimated
  + react-native-safe-area-context
  + react-native-screens
  + react-native-elements
  + react-native-web
* Backend (requirements.txt):
  + Django==3.1.1
  + djangorestframework==3.10.3
  + psycopg2==2.8.4
  + whitenoise==4.1.
  + gunicorn==19.9.0
  + firebase-admin==3.2.0
  + django-admin==2.0.0
  + Google Speech API
  + For dependency manager, we use npm node package manager for the front end and pip package manager for the back end
* **Release Build Management**. What is your pipeline (i.e., process and tools) used for building a development versus a production release? How are these integrated with your GIT workflow (i.e., the branches you merge and pull from and the tags you use)?
  + We will use one master branch for our production app which will contain the most recent version of the app with end-to-end features that have been tested and considered complete by our team. The development branch will pull from master and add new features to the stable version. Feature branches will be merged into the development branch, and when that version is stable we will merge development into master.
  + For the frontend, we will use Expo and Expo Client to build and test our app in development and share it with our client. The process we will use for beta testing is described in the link below.
    - <https://medium.com/@jamischarles/easily-share-a-react-native-app-during-development-d40ac27af6ef>
  + We will use a different Expo workflow to get the frontend app built and deployed to the app store for iOS and Android. This workflow is described in the link below and is more complex than the one above.
    - <https://instabug.com/blog/react-native-app-ios-android/>
  + We will use built-in python commands to build our Django server locally using the runserver command. In production, we will have to use Google Cloud’s console to build and deploy our backend Django APIs and database services.
* **Release Deployment.** How do you plan to deploy to each release environment? What parts of the process can be automated and what cannot?
  + We plan on deploying the app to Google Cloud platform, and Google provides platforms and services to support serverless CI/CD pipelines. Google Build is a platform that allows the automation of these processes and can create builds very quickly using Google’s global network. It can also help automate deployments with Google Kubernetes Engine, Google App Engine, Cloud Functions, and Firebase. We can set up triggers to monitor our Git repo so that we can trigger building, testing, and deploying from a pull request.
* **Defect Tracking**. How do you plan to track project bugs from open to close? We are not suggesting learning true bug-tracking software, but you should at least make bugs into actual Gitlab issues, [**using this template**](https://www2.cs.duke.edu/courses/fall20/compsci408/assign/templates/ISSUE_TEMPLATE.md), that are fixed in separate commits with separate tests (ideally made *before* the bug is fixed to better verify the bug and the fix).
  + While developing a feature, if a teammate finds a bug, they would create a Gitlab issue and ideally assign it (or @mention) to the person that worked on the feature that is producing the bug. They would include comments in the issue in order to make it clear what produced the bug and suggestions on how to go about fixing it, if known.
* **Software Configuration/Change Management (SCM)**. How will the team collectively determine if a feature is ready to be released or a change (e.g., a bug or security fix or a feature update) should (or should not) be added to the current production release? What is the overall process for managing which features are in which releases, tracking which changes have been applied to which releases, verifying a change works in the current production release, and determining when the current release should be replaced by a new release?
  + We will be using our Gitlab issues board to manage which features are in which releases using the Milestones and Label functionality. At the end of each Sprint, we will review features we aimed to implement and make sure that we only release those that are thoroughly reviewed and tested.
  + A feature is ready to be released or changed if tests accounts for multiple user stories and edge cases and is reviewed by at least one other team member and test coverage for that feature is 80% or above.
  + If a previously released feature is producing a bug, then the team should make that bug or security fix a top priority and roll back to the previous release and make sure that the above mentioned criteria is fulfilled before releasing the fix. Once the bug is fixed, we would release the newest production that originally contained the bug.