Trauma Ready Testing Guidelines

1. Identifying our end users

This application supports two types of users:

- **Paramedics, EMTs and other First Responders** will be using data provided by the mobile application in their everyday workflow
- **Hospital Administrators** will be using an authenticated web portal to create, read, update, and delete the data stored for use in the mobile application itself

2. What we would do with infinite time

Our project consists of essentially three smaller codebases integrated into a cohesive application: the cross-platform mobile frontend, web frontend, and server-side rendering/data-serving code.

(Address for each: Approach, Environment, Tooling, Correctness, Performance/Reliability)

- **Mobile Frontend** would be tested by both an automated suite and manual inspection on a comprehensive set of Android and iPhone devices/emulators. One disadvantage to using React Native is that we lose access to the dynamic layout systems provided by native XCode and Android Studio, so we'd want to check that our application looks and behaves as intended on all devices. Similarly, we would implement a feature that allows users to report bugs as they use the app. Because the possible environments for a paramedic to work in are so diverse, they will be likely to discover cases where our software breaks that we could never imagine.

- **Web Frontend** would be tested by being released to administrators to test authentication and the creation, reading, updating and deletion of the data stored for use for the mobile application. This would take the form of Acceptance testing, where our client determines if the Web Frontend is suitable to their needs.
    - Automatic testing for this would be performed in joint with testing done on the Server, essentially taking the form of System testing.

- **Server** would be tested extensively with Node supertest and mocha for Integration testing, along with extensive testing of the API endpoints through the applications Postman/Insomnia. This would allow us to not only test the correctness of our application's code, but also the performance and reliability, that a correct response is returned within a reasonable time frame, without crashing and in the expected format.

3. What we are actually going to test

Note: This portion should perhaps be more practical or in the form of a tutorial so Stotts/future devs know exactly how to set up and run our testing suite on their own machine. It's hard to write it like this now, so we can stay general and update it once we have actual testing software in place.

- **Mobile Frontend:**
  - Automated Snapshot Tests with Jest. These guarantee that our UI doesn't change itself unexpectedly
- Approach
  - Manual Testing
- Environment
  -
- Tooling
  -
- Correctness
  -
- Performance/Reliability
  -

Backend and Web Frontend

- Approach
  - Automatic Testing; Given that we, the developers of the application, are going to be performing the testing, the testing can be described as "White-box Testing," where the tester has access to and working knowledge of the source code.
  - Regression Testing: We will need to perform this kind of testing as well in order to ensure that fixing one section of buggy code does not hinder the functionality of a previously working part of the application
  - Acceptance Testing: We hope to be able to provide our client with access to the Web Frontend prior to the absolute completion of the project, so that they can provide feedback and we can make changes on this feedback.
- Environment
  - Internet Connection required to perform this kind of testing
  - The testing may be done either through running a localhost, or with the heroku application
  - A terminal command prompt would be used for testing with Supertest and Mocha, while the application interfaces for Postman and Insomnia would be used for testing with those applications
- Tooling

  - Postman/Insomnia: Both of these applications provide resources for testing our application's API; Postman provides many more resources for testing an API, while Insomnia's interface is more basic, yet intuitive. While Postman will almost certainly be the go-to for testing the backend, Insomnia may be also used at times to provide a different view of testing with the API.
  - Some Integration testing could also be performed using Node Supertest and Mocha to run from the terminal. Because the Server connects to the Web Portal, some of this testing would take the form of System testing.
- Correctness
  - Testing would be done to ensure that data the proper http responses are returned, and long with the correct data, whether in json or html format, depending on the needs of the portion of the application and of the test.
- Performance/Reliability
  - Testing would be done to ensure that the data returns the correct responses and and data, as demanded by the test.