Team JSHD - Meeting 10/03/2021

- We have installed Docker and tried to get our project containerized.
- Stew containerized our project and pushed the changes to github.
- We have assigned roles to the subtasks in milestones 1 and 2 (see below)
- We decided to wait until Milestone 2 before assigning roles for milestone 3.
- We have fleshed out the details of Milestones (see below)
- We decided to include 75%+ coverage for unit tests in Milestone 3.
- We reorganized Milestones 2 - 3 - 4 into Milestone 2 and 3.
    - We didn't think there was enough work in Milestone 2 for a separate item.
- James helped setup Heroku.
- Henry will finish the readme files.

Milestone / A1 Discussion:

- Milestone 1: Set up our environment, dependencies and tech stack (done by: Oct 4th)
    - A github repo is created with our team members for the project (Henry)
    - We have a node project that is runnable and has a simple test suite (Daniel)
        - Dependencies should be ready to install in one command
        - Running the project takes only one command after dependencies are built
        - Running test suites takes only one command after dependencies are built
        - Project functionality and test coverage is not required for this milestone
    - The project is containerized in Docker (Stew)
    - Project is hosted on Heroku, deployed with our docker image (James)
    - A readme is provided in top level directory of the repo with the following: (Daniel/Henry)
        - Instructions on installing and running the project
        - A summary of our tech stack and the relevant decision making
        - Milestones of the project with member ownership of each step
        - Directions to our collection of meeting notes

- Milestone 2: Setting up DB and implementing basic functionality + unit tests (done by: A2 deadline)
    - Create db/schemas and populate our downloaded guideline definitions (James)
        - Download a zip file of guidelines in json format from https://build.fhir.org/ig/HL7/cqf-recommendations/definitions.json.zip
        - Create a new database called "guidelinesdb"
        - Creates schemas
        - Create two collections for "plandefinitions" and "activitydefinitions"
        - Insert all of the plan/activity definitions from the zip file into the db
        - Verify that the db works by reading a few of the elements.
    - We should be able to read a list of all guidelines and create a unit test to verify it's functionality. (Henry)

    - We should be able to retrieve a specific guideline and create a unit test to verify it's functionality. (Daniel)
        - Creating different router endpoints depending on which guideline type is requested
    - We should be able to add a new guideline to the db and create a unit test to verify it's functionality. (Stew)
    - We should additionally verify the above functionality using manual testing with Postman (Henry/Daniel/Stew)

We will assign Milestone 3 responsibilities during Milestone 2.

- Milestone 3: Additional functionality + unit tests, integration tests, documentation (done by: A3 deadline)
    - We should be able to update a guideline's json and create a unit test to verify it's functionality.
    - We should be able to delete guidelines and create a unit test to verify it's functionality.
    - We should additionally verify the above functionality using manual testing with Postman
    - Our test coverage at the end is >= 75%
    - We have written integration tests to verify the following functionality:
        - adding a new guideline to DB
        - reading a list of all available guidelines (?)
            - updating guideline's json
        - retrieve the guideline and verify contents
        - delete the guideline.
        - Should verify both activity and plan definition.
    - We have either a website or readme in our github repo (undecided), which documents the following:
        - What our project is about.
        - What the various API calls do, how to use them and an example.

- Possibly: ci/cd, alt goal: updating guidelines

Tech stack summary:

- Frontend: N/A
    - Will not be required as our project is just an API for other apps
- Backend: Node.js
    - Node.js with Express.js framework
        - Every team member has prior experience
        - Easy integration with mongodb

        - Flask
            - Not everyone used it
- Database: MongoDB
    - MongoDB
        - Mongoose integration with Node.js
        - Team has familiarity with mongo
    - We will only need to save ActivityDefinitions and PlanDefinitions into a database
    - id properties aren't unique across both activity and plan definitions, so we will create separate db schemas for each
    - Our database will only store JSON objects, so it's better not to use a relational database
- Hosting service: Heroku
    - Heroku
        - Free
        - Compared to Firebase, Heroku is more flexible and easier to integrate with other services
    - Firebase
        - Has its own DB and authentication
        - Limited flexibility. Ex. using our choice of DB.
- Container service: Docker
    - Vagrant
        - Advantage over docker: can run in non-linux environments because it functions as a VM
    - Docker
        - Team has prior experience with docker
        - More popular choice
    - App will be run in a linux environment, so we're choosing docker because we're more comfortable with it
- Testing: Jest
    - Jest is simple to use
    - Mocha does not have its own assertion libraries
        - Chai is used for assertion
        - Sinon is used for spies/stubs
    - Jasmine isn't able to run tests without a third party
    - Decided to go with Jest because the scope of the project isn't large enough to justify using a more heavyweight option like mocha

Decision to end meeting