Development Plan
Software Engineering

Team 14, Reach
Aamina Hussain
David Moroniti
Anika Peer
Deep Raj
Alan Scott

Table 1: Revision History

Date	**Developer(s)**	**Change**
09/22/23	David	Added first draft for tech and coding standard
09/23/23	Deep	Added first draft for proof of concept plan and project schedule
09/24/23	Anika	Added draft for intro and sections 1 - 4
09/25/23	Anika	Made changes based on sync with project supervisors
11/12/23	Deep	Fixed formatting and modified POC plan
11/13/23	Anika	Changed all instances of Jira to Github (in accordance with our new tracking)
04/04/24	Anika	Added final changes to document
...	...	...

The following document outlines the development plan for REACH. As the scope of the project is further elaborated upon, this document will be revised in order to reflect the most updated plan for the project.

1 Team Meeting Plan

Team meetings will occur at least once a week, during the scheduled tutorial time Mondays from 2:30 PM - 4:30 PM ET. All members are expected to attend these meetings, they will be referred to as development (dev) team syncs. If a member has a timing conflict, they must notify the rest of the team as soon as possible, in order to allow for rescheduling. Any additional dev team syncs

will occur based on member schedule and are not mandatory unless otherwise specified.

On a biweekly basis, the team will meet with the project supervisors to discuss progress and gain clarity on any roadblocks, this is called the stakeholder sync. The timing for these meetings depends on the availability of the project supervisors and can be subject to change. Although attendance for the stakeholder sync is not mandatory, it is highly encouraged. As with the dev sync, team members should notify the team of any scheduling conflicts although rescheduling may not be possible due to the busy schedules of project supervisors.

Depending on team availability, meetings may occur on campus or virtually. Given the travel time for certain team members, virtual meetings are preferred.

AGENDAS:

Where agendas are concerned, all team members are expected to contribute to the agenda prior to the meeting, and should expect to lead the topics they contribute to the agenda. Any concerns, questions, updates or roadblocks should be jotted in point form on the agenda and can be further discussed in meeting. Furthermore, team members should include a time estimate on agenda items in order to accurately book time for the meeting. Finally, all agendas will be tracked as issues in the project repository and will be closed after meetings.

MEETING ROLES:

Chair: This individual will change every meeting.
They ensure that the meeting stays on track and that all agenda items are addressed. They are also responsible for ensuring that the meeting progresses at a decent pace and starts and finishes on time.

Scribe: This individual will change every meeting.
They are responsible for taking notes during the meeting and ensuring that all new action items have been recorded and have been assigned to a member. They will also cross off completed action items from previous meetings.

Scrum Master: Anika/David.
While this role will be elaborated upon in the "Team Member Roles" section, the scrum master is responsible for updating the Github in accordance with (applicable) new action items as well as any new information about current Github issues. They will also be responsible for ensuring that all action items are assigned to a member and have a due date.

2 Team Communication Plan

In order to facilitate communication between all team members, the following tools will be used:

- **Git Issues**
  These will be used to track agendas as well as meeting notes. Git issues provide a centralized location for all meeting information and allow for ease of maintenance as well as updates. Generally, Git issues should not be used as a primary form of communication. Comments under issues MAY be addressed, but team members should expect to discuss via chat or meeting instead. For further details on agendas please see the "Team Meeting Plan" section.

- **Teams**
  Teams will be used for all virtual meetings and as the primary method of (chat-based) communication. Meetings should be scheduled ahead of time on Teams in order to appear on all members' Teams calendars. Syncs are scheduled as recurring meetings on Teams, but can be cancelled or rescheduled as needed. Meetings should only be used for longer updates and discussions, and should not be used for quick updates or questions. The team will use the Teams chat for quicker updates and minor roadblocks or questions. For further details on meetings please see the "Team Meeting Plan" section.

- **Google Calendar**
  The team will use a shared Google calendar that will be updated any time a new deadline, meeting, or action is set. Updates to the calendar are the responsibility of all team members and shall be managed diligently.

3 Team Member Roles

- **Project Manager: David**
  The project manager will be responsible for any and all communication between the team and all project stakeholders. They will also be responsible for ensuring that all other roles are completing the work they are responsible for. The project manager sets and enforces the schedule for deadlines and due dates (with the caveat that members must reach consensus on the schedule). The project manager should be the first point of contact for team members when deadlines/expectations cannot be met.

- **Lead QA for Documentation: Alan**
  The lead QA for documentation will be responsible for ensuring that all documentation is up to date and accurate. They will also be responsible for ensuring that any updates to documentation are communicated in team meetings. If a member's documentation does not meet expectations, the lead QA for documentation will give feedback (in meeting or on the group chat) and request that the member updates their documentation.

- **Scrum Master: Anika/David**
  The scrum master will be responsible for updating the Github in accordance with (applicable) new action items as well as any new information

about current Github issues. They will be responsible for making sure that there are dates and completion times on action items through Github issues. They will collaborate with the lead QA for Git to ensure that there is linkage between Github issues and git commits. If a member is not completing assigned issues on time and according to predetermined standards, the scrum master will first communicate with the project manager (who is also responsible for deadlines). The scrum master and the project manager will give feedback to the member and work in tandem with the team to formulate a plan to get the member on track.

- **Lead QA for Git: Deep**
  The lead QA for Git will be responsible for enforcing the git-related sections of the workflow plan. They will be held accountable for ensuring that the code base is as organized as possible and that team members adhere to the workflow plan. If a member's git usage does not meet expectations, the lead QA for Git will give feedback (in meeting or on the group chat) and request that the member follows the workflow plan.

- **Front-end SMEs: Aamina and Anika**
  Front-end SMEs will be the members of the team with the most knowledge and experience in hosting web applications and front-end work. These members will set the "predetermined standards" for completion of the front-end work. While all members will work on front-end development, the front-end SMEs will be responsible for ensuring that the front-end is completed according to the SRS and design plans. These SMEs will also be the first-point of contact for any questions or advice about front-end development for the project.

- **Back-end SMEs: Alan and David**
  Back-end SMEs will be the members of the team with the most knowledge and experience in the back-end work. These members will set the "predetermined standards" for completion of the back-end work. While all members will work on back-end development, the back-end SMEs will be responsible for ensuring that the back-end is completed according to the SRS and design plans. These SMEs will also be the first-point of contact for any questions or advice about back-end development for the project.

- **Testing SMEs: Deep**
  Testing SMEs will be the members of the team with the most knowledge and experience in testing. These members will set the "predetermined standards" for completion of all components of testing. While all members will work on writing tests for their own code, the testing SMEs will be responsible for ensuring that the testing is completed according to the SRS and Test Plan. These SMEs will also be the first-point of contact for any questions or advice about testing for the project.

- **Scribe: Any**
  For more information on the scribe role, please see the "Team Meeting

Plan" section.

- **Developer: All**
  The developer role is the default role for all team members. While members may be experts in a particular domain, all members are expected to be doing development work throughout the project.

4 Workflow Plan

The workflow process begins with the creation of Github issues. Through the Github issue integration, Github issues will be linked to git commits. The team will work as a cohesive unit in order to create Github issues (which are also referred to as tickets) and assign those to the appropriate team members. From there, the project manager will take the lead on deciding priorities and deadlines for the tickets. The scrum master will be responsible for dragging tickets into a sprint in a manner that is consistent with those deadlines.

For each sprint, tickets will be visible on a issue-tracking (Kanban) board with 5 columns:

- **To Do:** Tickets that have not yet been started in any capacity but are assigned to a sprint.
- **In Analysis:** Tickets for which the assignee/assigned developer has begun brainstorming an implementation.
- **In Progress:** Tickets that the assignee/assigned developer has begun working on.
- **In Test:** Tickets that the assignee/assigned developer is testing in order to ascertain readiness for merge.
- **In Review:** Tickets that the assignee/assigned developer has completed and is ready to be merged into the master branch. These tickets are in review because they are expected to be reviewed by other team members prior to merging into the main branch. At this point the developer should have a pull request in place and should be ready to merge.
- **Done:** Tickets that have been merged into the master branch.

The following steps describe the process for a developer wanting to work on a ticket that they have been assigned in this sprint:

1. The developer will move the ticket into the "In Progress" column of sprint board.
2. The developer will pull any new changes from the master branch.
3. The developer will copy the assigned branch name from the ticket and create a new branch with that name.

4. The developer will make changes to the code base in accordance with the ticket.
5. The developer will commit their changes to their branch with a detailed commit message.

When the ticket is complete and the developer wishes to merge into the master branch, the following steps should be taken:

1. The will move the ticket into the "In Test" column of sprint board.
2. The developer will run the automated testing and linting pipeline in the CI environment. This should also spin up docker containers from the images in order to do any environment testing before merging to main.
3. Provided tests are passed, linting is successful, and the spun up environment does not error out, the developer will create a pull request.
4. The developer will move the ticket into the "In Review" column of the sprint board.
5. The team will review the pull request, provide feedback, and request changes if necessary.
6. Once the pull request is approved by the rest of the dev team, the developer will merge into the master branch.
7. The developer will make sure to delete the branch they were working on after merge.
8. The assigned developer/scrum master will move the ticket into the "Done" column of the sprint board.

In the case of merge conflicts prior to merge, developers are expected to rebase, and pull in changes from the master branch. If a developer is unable to resolve a merge conflict, they should reach out to the team for assistance. From there, the developers who were working on the same file will work together to resolve the merge conflict. A pull request cannot be merged if conflicts exist.

Additionally, the team will be tagging everything from the initial commits with rev0 after all documentation has been created. The rev tag will increment by one for each revision of the project documentation.

5 Proof of Concept Demonstration Plan

The success of our project hinges on addressing two main risks. Firstly, none of the members of the team have succesfully deployed a web app in a cloud environment prior to this project. Secondly, we will have to depend on the clinicaltrials.gov or a similar repository's API to retrieve trials.

This risk would not be possible to eliminate wholly as a large portion of our goals depends on cloud deployment and the overall application interface. Where mitigation is concerned, we will likely need to devise a more in-depth plan on working with cloud services and deployment. This will likely involve a lot of research and trial and error, but we will have to ensure that we are able to deploy our application to the cloud.

All of our trials being retrieved from clinicaltrials.gov or a similar repository is also a risk since we will have to research and work with that api to know what is possible using it. Relying on external APIs is always a risk because we are not in control of the data source, data format, or data availability. This would also be a hard requirement to mitigate because it would not be feasible for us to collect clinical trials individually, so we must use a preexisting repository of trials and its corresponding API.

Therefore, to demonstrate that we can overcome these risks, our proof of concept demonstration will have to; be deployed to the cloud, be able to be accessed from a device that is not running the source code, and the application must be able to retrieve and display clinical trails from a data source given some information on the participant.

6 Technology

As a result of our system being a pretty complex full-stack application, it will be built using several different tools and technologies, across both the frontend and the backend.

Frontend

- Programming language - Typescript
- Linter - ESLint
- Unit testing framework - Jest
- Frontend web framework - React

Backend

- Programming language - Python
- Linter - Flake8, Mypy, autoflake
- Unit testing framework - pytest
- Backend web framework(s) - Django, FastAPI

In addition to the technologies mentioned above, we will be using a few tools that will help us develop our app more efficiently and effectively. First, we will containerize our app by using docker, which will make it easier to test and deploy our application (also use of docker desktop for running/testing our

application locally). Furthermore, for our continuous integration pipeline, we will be using github actions to run automated tests and linting for both the frontend and backend services/code, and to build our docker images. Finally, we plan to use firebase as our cloud environment, which is where our application will actually run.

There are also a few tools/technologies that still need to be decided upon. We will need to have some database to store client information. It is likely that this will be a relational database, and will be one of Postgres, MySQL, or SQLite. Additionally, it is possible that we will need to use some existing library/tool for parsing trial eligibility criteria, however this will become more clear in the following weeks (i.e., an advanced parsing library, or existing NLP designed for healthcare systems).

7 Coding Standard

For all of our python code/services, we will be following the Pep8 coding standard, and we will be enforcing this with the help of the flake8 linter. We will also be using mypy, which will enforce strict typing (which is usually not present/mandatory in python), meaning all of our function parameters, return types, class variables, etc.. will need to have its type defined. Finally, we will have docstrings present in each module, to give a brief description of what the main purpose of the module is.

For the frontend code we will not be following any formal coding standard, but we will be following best practices when coding in typescript and when using React. Some of these best practices include:

- use functional components
- don't use inline-styles
- maintain a proper import structure
- code checked/fixed by ESLint

This is not an exhaustive list, but it highlights some of the best practices that we will follow when using typescript/React.
(credit to - https://www.makeuseof.com/must-follow-react-practices/).

8 Project Scheduling

- September 25 - Problem Statement, Proof of Concept Plan, Development Plan
- October 4 - Requirements Document
- October 20 - Hazard Analysis

- November 3 - Verification and Validation Plan
- November 13-24 - Proof of Concept Demo
- January 17 - Design Documentation
- February 5-16 - Revision 0 Presentation
- March 6 - Verification and Validation Report
- March 18-29 - Final Demonstration
- April 4 - Final Documentation
- April 9 - Capstone Expo