Pandemic Processing

Software Design

(Version 1.2)

February 18th, 2019

Project: Epidemiological Modeling Portal

Team Members:

Anthony Schroeder

Joseph Eppinger

Tanner Massahos

Sponsor:
Dr. Joseph Mihaljevic

Mentor:
Dr. Eck Doerry

Table of Contents

1 INTRODUCTION

Infectious disease has been a problem throughout all human history. For example: The Plague of Justinian in 541 lasted over 200 years and resulted in an estimated 25-50 million deaths; the Black Death during 1347-1350 killed ~60% of the European population. Compare this to a modern epidemic: The West African Ebola Outbreak resulted in the deaths of 11,310 individuals. It is clear that there has been a dramatic change in the number of casualties due to infectious disease when compared to plagues of the past, and one reason for this is the development of the science of epidemiology.

Simply put, epidemiology is the study of infectious disease and how it spreads within a defined population. An important tool used by epidemiologists to manage infectious disease outbreaks is modeling. Models are essentially predictive networks of mathematical formulas developed by epidemiologists that show how disease can spread through a community. Given a set of initial conditions, a model can generate predictions of key outcome variables, such as how the fraction of the population that is infected changes overtime, speed of disease spread, and mortality rate. These models ultimately are an attempt at predicting how a disease will affect a population given what epidemiologists know about it. When a model closely represents the real world public health officials can plan properly and respond accordingly.

These models can differ in style such as deterministic mathematical models or complex spatially-explicit stochastic models. Models are programmatically created to guarantee reliability and maintainability of data. Epidemiological modelling allows:

1. Scientists to create models of infectious disease given a certain set of assumptions and data.
2. Based on these models, scientists can predict important parameters dealing with the infectious disease such as speed of spread and potential population impacts.

Epidemiologists also apply various methodologies and base their models on varying assumptions, meaning there could be a handful of differing models for one infectious disease all of which could generate different optimal vaccination times and predictive preventative measures.

Many of these scientists are creating these models in parallel, and must discuss them in order to determine which is the most accurate. This deliberation component

is key in that it ensures the optimal model is chosen for a given situation. The best model could be any one of the models proposed, or even a new hybrid created from the discussion.

This is where the major problem arises, according to the project client Dr. Joseph Mihaljevic, an epidemiologist and assistant professor at NAU SICCS. There is a lack of an efficient system where epidemiologist from all around the world can share, review, test drive, then critique and discuss one another's models in order to receive an optimized model that generates the best predictions as quickly as possible. Streamlining this collaborative discussion and refinement process is key to improve rapid and efficient responses to developing disease outbreaks.

It is not that this current workflow is at its core broken, it is simply that it could be further optimized. In the world of epidemiology, there is a lack of an efficient system where epidemiologists from all around the world can critique and discuss one another's models to arrive at an optimized model that generates the best predictive measures. Some specific problems with this workflow include:

- Epidemiologists must wait until full publication to view models of other epidemiologists they are not personally acquainted with.
- Deliberation is difficult when no public space for it exists. Currently, epidemiologists can only communicate through emails and GitHub comments.
- Communities of epidemiologists are isolated clusters, as in they only tend to communicate with those that they already know well, which reduces the extent to which deliberation can produce new ideas.
- Current discussion and models are not extremely public or easily accessible. An epidemiologist must publish or actively send their data to authorities before action can be taken.
- Difficult to communicate results with managers and public health officials.

The modelling of infectious disease is a rather well-defined process. Each time an epidemiologist wants to model the outbreak of an infectious disease, they step through the following process:

1. **Collect data via data sets or researched data:**
   The epidemiologist gathers data about the population and how the infection spreads from a mathematical standpoint.
2. **Model data using C, R, or Python:**
   Using this data, the epidemiologist crafts a model using a series of mathematical equations which models dynamics of interest such as the rate of mortality and speed of spread. This model is then implemented as a

modular computer program so that it can be rerun quickly with varying parameters.

3. **Develop the best model through discussion:**
   Next, they discuss with other epidemiologists, critiquing and modifying the model eliminating of any potential incorrect assumptions. Ideally, this discussion leads to refinements to produce a model that should most closely approximate the real-world spread of the disease.
4. **Build predictions based on the model:**
   The epidemiologist then predicts the best possible course of action for minimizing the damage done by the disease, and proposes preventative measures to the authorities who can act accordingly.

Epidemic Observation Network (EON) will be a secure, fast, and user friendly web application where epidemiologists from all around the globe can share and discuss their models. EON will allow epidemiologists to:

- **Share models with the community**
  - *Method:* EON will be a place where epidemiologists can post their models' code for others to view and discuss.
  - *Purpose:* This will allow for quick critiques and harbor discussion. These critiques and discussions will lead to an optimized model which will generate the best predictive measures.
- **Decide how their models appear to viewers**
  - *Method:* EON will allow epidemiologists to decide how public their models are. For example, a scientist might want to keep their code private, but the final graphical output is fine for viewing.
  - *Purpose:* Some scientists do not want to share their models or data due to publication purposes, so the user will have the option to make a post public, private, or limited sharing.
- **Interact with and provide feedback on other user's models**
  - *Method:* EON will provide a discussion forum for epidemiological models. Additionally, users will be able to modify and re-upload each other's code.
  - *Purpose:* This builds a sense of community and allows users to modify assumptions and visualize how it changes the output.
- **Discuss future models without a fully developed model existing**
  - *Method:* EON's forum will not just be for models which already exist. Users can create discussions without any working prototype developed.
  - *Purpose:* This allows users to simply post an in-progress model for discussion, meaning no code is required upon upload.

- **Edit and share the code used for models in a GitHub type of fashion**
    - *Method:* EON will support version history and cloning of the model code.
    - *Purpose:* This will allow users to view and edit the code used to generate a model and create new models based on old ones for further critiquing and discussion.

By streamlining these key steps in the workflow EON will provide an invaluable missing resource for epidemic management. Specifically, it will streamline comparing and discussing models to come to a faster consensus. EON will allow epidemiologists to quickly and efficiently generate models, compare and discuss models, and arrive at an optimal model so they can create preventative measures to combat disease outbreaks.

This software design document acts as a blueprint for the implementation of the EON web app and is a live document. First the implementation and architectural plans for EON will be outlined. Following that will be module and interface descriptions which will delve into how each module of the architecture fits within a larger context. This will allow us to capture any design flaws early on, saving us time in the future.

2 IMPLEMENTATION OVERVIEW

Proper implementation of the web application is critical for the future growth and maintenance of its platform. To ensure the proper selection of components for this and the functional requirements stated above, the team performed research comparing the different available tools. Below are short descriptions of each element that was chosen with its function, and responsibility. Figure 2.1 below is a diagram to get a better idea of how these components will operate together to form EON.

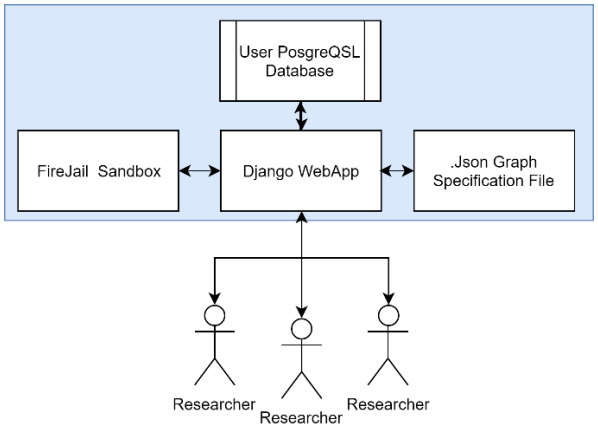

Figure 2.1 Diagram of Architecture

Here are the following components being used to build EON:

Django Web Framework

Django is an excellent framework for developing the EON web application, as it provides an easy to use object-oriented environment in python allowing for many different components to be implemented together. Its implementation provides for scalability and offers many helpful tools for development with plenty of online resources for debugging and troubleshooting.

The Django web framework will not only provide the structure and serve views for the front-end; it will communicate with the back-end components and manage the different modules providing an interactive user experience for the end user. This includes:

- Providing a web structure for user coherency
- Providing consistent visual appearance
- Providing an interface for discussion through forums
- Initiating and managing user code through Firejail sandbox
- Displaying graphs illustrating user models

PostgreSQL Database

PostgreSQL is an easy to use database allowing for table creation, standard queries, and ease of scalability. The PostgreSQL database will oversee the storage of most data, including forum activity, user activity, group activity, and the creation and removal of user models. Some of these tables will point to flat folder locations were both the user's model code, and the model specification file exists (JSON files).

Firejail Sandbox

Firejail is a secure environment that allows server hosts to run foreign code with little to no threat to security. The Django framework will use Firejail system commands to run user code securely, limiting access to the machine, runtime duration, network access, system commands, and more. The Firejail sandbox has an advantage over virtual machines because running code through a sandbox doesn't require the booting and managing of virtual environments.

JSON Dictionary files

JSON files are an easy way to store complex pieces of information without creating and managing tables. By avoiding tables, JSON files provide a more natural way to

access large or complex data formats, which would otherwise require several database calls and complex join commands to obtain the equivalent data. This file type allows us to implement quick access to larger sets of data, without creating a complex management code for database requests. These (JSON files) will be responsible for providing all relevant information associated with a given model, including the needed labels for displaying its graphs.

By having a designated folder to keep a model's information the complex requirements of SQL queries can be reduced when implementing EON's modular design. Researchers will not be required to produce or maintain these files as the application will automatically manage these to maintain user's graphs and model information.

Now that the components and the roles that they provide within the application have been explained, the interactions of these components can be adequately explained.

3 ARCHITECTURAL OVERVIEW

The best way to ensure future support, and expandability was to implement the components based on a modular approach, allowing for the addition of new modeling languages, and future Django applications to expand the web application's capability. Based on this approach, the application will have several components control function and appearance. Figure 3.1 is a more detailed architectural overview depicting a high-level view of the different components of the EON web application and the flow of information between the elements.

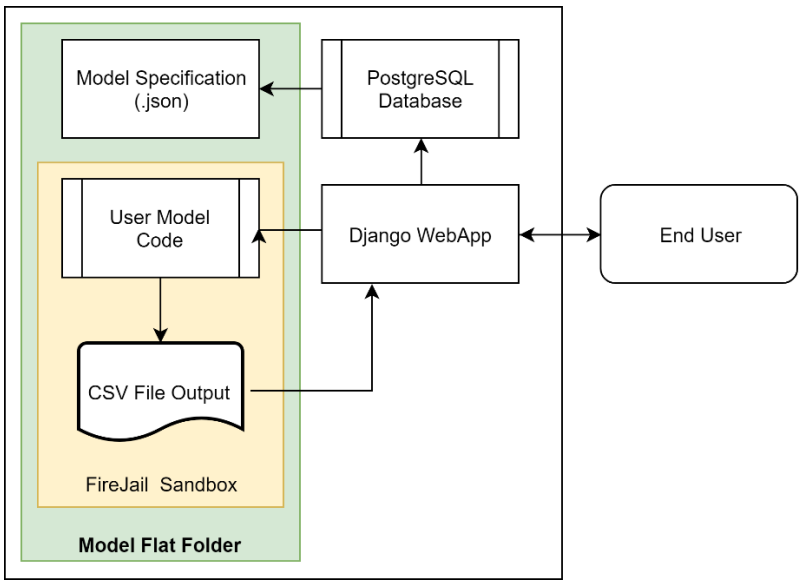

Figure 3.1 High Level Architectural Overview

The Figure above describes how the components will interact with each other. In a general sense, the user will send requests to the Django web application for all pages on the website. The PostgreSQL database will provide information to construct general user experience, and reference model specification (JSON) files. These JSON files will provide support for Django by holding relevant structure and descriptions of the graphs, and specifications about user code. When executing user simulation models, Django will run user code through the Firejail sandbox and process the produced CSV file into a graph.

As the user does simple operations, such as create forums, posts, and groups, Django will initiate different functions using the various components recording and retrieving relevant data. Here is a quick sample of some use cases, and the appropriate functional reactions:

A user wishes to create a new model:

1. After the user fills out a short form describing the model such as the model's name, description, and more. They then upload the code and an appropriate makefile to the cite.

    I. Before executing any database queries, the input parameters will be sanitized with the SanitizeInput() function, and check inputs for validity and screen for malicious intent.

    II. Django will call ModelEntry(), and an entry in the PostgreSQL model table will be made to store the code's saved location, and the appropriate privacy settings filled out by the user.

    III. Depending on the options provided within the form of additional entries to the PostgreSQL Forum table will be included, adding a forum for the model via CreateModelForum().

2. The user will then be prompted to fill out at least one graph form describing the different graphs they wish to display. They enter general information about the functional parameters such as names, the modifiable ranges, the default values displayed when viewing the graph, and the selected CSV columns to the graph.

    I. This form will be stored as a JSON file through StoreModelForm(), and the file location will be logged within the PostgreSQL table.

II. RunUserCode() will call on the SQL database requesting the program's location, and run a system command using the Firejail framework like so.

III. The sample CSV generated from the user's code will be stored with the JSON file, recording the program's run for reference.

A user wishes to create a forum:

1. The user fills out a short form describing the forum such as the forum's name, description, and more.

    I. Django will call FormEntry(), and an entry in the PostgreSQL forum table will log the user, privileges, forum name, and more.

A user wishes to post on a model:

1. The user navigates to a model view and selects either the post/comment link.

    I. Django will provide a dialog box, that the user can fill out. The user is free to fill in the dialog box and submit. An entry in the PostgreSQL post table will be added referencing the primary form key to the model's forum, along with the comment.

In addition to the functional descriptions above, the flat file implementation provides a simple solution to keep relevant model information in one location. Having a flat file location also makes it easy to implement the Firejail sandbox and keeping user's code organized.

In addition, JSON files will help leverage the flat file structure as they are excellent at recording dynamic structures, allowing for many different graph types to be included. JSON files will also provide the ability to have a modular design for future language support, allowing for many different structures to maintain user data.

Now that the primary responsibilities and roles of each component have been described, the next section can be explained. This section describes module, and interface design, and further describes the interactivity between the different parts and internal functionality.

4 MODULE & INTERFACE DESCRIPTIONS

The module components and interfaces are extremely important for a functioning web application. As such, each component described here contains a more extensive diagram that further expands on the operations of each component.

4.1 PostgreSQL Database

The figure below is the planned implementation of EON's PostgreSQL database, that will track: users, teams, models, and forum activities. This will store most of the activity on the EON web application and will be responsible for supplying information for the Django framework. The ERD for the current database can be seen below in Figure 4.1.

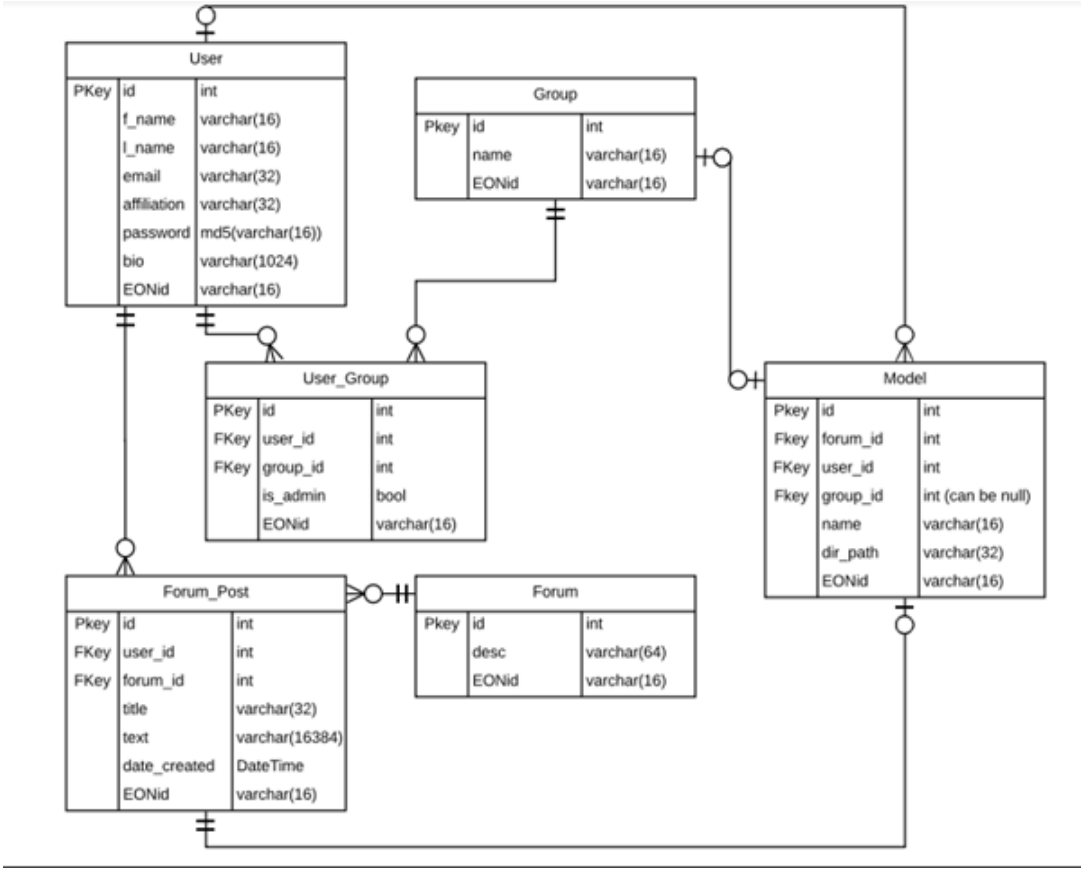

Figure 4.1 ERD of PostgreSQL Database

4.1.1 User Table

The user table is a standard SQL table that holds each individual researcher's information and allows functionality to manage their accounts. Through different functions, this table will direct SQL joins and queries to create, modify and delete user information found within their groups, models, forums, and posts. Here is a

quick description of some of the table fields to get a better understanding of what will be stored here:

- Affiliation: Users may provide labels to their association with organizations.
- Bio (Biography): Users may provide a longer description of their experience, hobbies, and general interests.
- Last Day Active: For active user tracking.
- EONid: Is a general indicator to distinguish between duplicate users with the same names. This gives a way to identify groups, individuals, models, and forums.

4.1.2 Group Table

The Group table exists so as users create groups to work on projects they will be added here. The group table will give individuals in a group privileges to view and modify group items, such as models and forum posts. Here is a quick description of some of the table's fields:

- Team Name: A way to personalize a group, and an easy to distinguish different groups.
- Description: Gives groups a way to state their aspiration and purpose of the group.
- Team Lead: FK, that identifies the individual responsible for the group.
- EONid: Is a general indicator to distinguish between duplicate model entries with the same titles and/or names. This gives a way to identify groups, individuals, models, and forums.

4.1.3 Model Table

The model table provides a reference to the models uploaded and shared on the EON platform. Not only does this table promote methods to describe and run model simulations, it also contains the viewing and editing privileges of users and groups. Only allowing authorized users to use methods to either view code, modify parameters, delete, and edit the values. Here is a quick description of some of the table's fields to give a better understanding of its functionality:

- Description: Gives users the ability gives a description of their models, and that way others can view and understand the intent of the
- Dir_path: Each graph generated for a model, will be saved to a JSON file (described in the section below). will allow them to fill out a forum, that will create a dictionary description how they would like their graph to be displayed.

- EONid: As described above, is a general indicator to distinguish between duplicate model entries with the same titles and/or names. This gives a way to identify groups, individuals, models, and forums.

4.1.4 Forum Table

The forum table provides structure of the general forum, and comment section for models. This table will provide functionality to generate discussion forums, post comments about model, post instances of ran models, reply to posts, and other user activities. This table will also include a respective EONid: which as described above, is a general indicator to distinguish between duplicate model entries with the same titles and/or names. This gives a way to identify groups, individuals, models, and forums. Forums are essential to the success of this project once released.

4.1.5 User Group Table

A bridge table which contains the many to many relationships between users and groups.

4.2 Flat file and JSON file representation

As explained earlier the flat file implementation provides a simple solution to keep relevant model information in one location and allows easy to implementation of the Firejail sandbox. Below is a representation of the "Model Flat Folder", and the "Graph instance" folder (Figure 4.2), these two folders hold data about different models that have been uploaded for later usage.

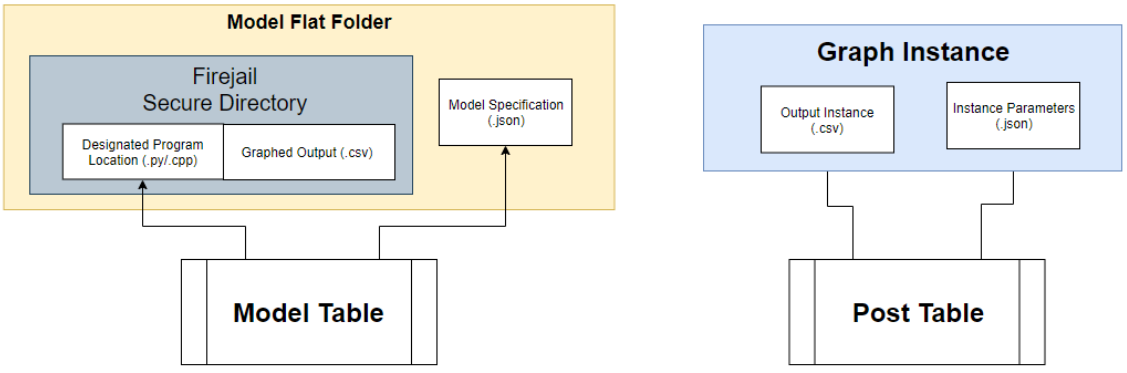

Figure 4.2 Model Flat Folder and Graph Instance

4.2.1 The Model Flat Folder

The model flat folder contains both the user's uploaded code and a corresponding JSON file containing most information about an individual model. The PostgreSQL table, the model will contain the file location of each model's "Model Flat Folder", as shown in the figure above.

4.2.2 Model Specifications (JSON file)

The JSON model specification files contain the title and characteristic values of an individual graph. The Django framework will use the PostgreSQL table to acquire the model flat folder location, and perform either right or read requests on the files representation of the graph such as Title, X-axis, Y-axis, Parameter_Name_0, etc. This folder will contain a referenceable instance of the model to generate the default graph for a model.

4.2.3 Graphed Output (.csv)

Contains the generated output of the model and is likewise referenced to generate the default graph for a model.

Note: If multiple users are requesting "Runs" of an individual model, the model flat folder will be copied, and the different copies of the program will be run separately.

4.2.4 Referenced Graph Instance (Folder)

A referenced graph instance is stored in a folder and is responsible for containing any relevant information about an individual run of a model, and this folder will be accountable for forum posts discussing different examples of a given model. This graph instance folder contains the output CSV file of the generated graph, and a JSON file containing runtime information.

4.2.5 Instance Parameters (JSON file)

The instance parameters is a file which contains the parameters passed to the model's executable that achieved the instance of the output graph. These include but are not limited to: Parameter_0, ..., Parameter_N.

4.2.6 Output Instance (.csv)

Output instance is the generated output of an individual's results from running a given model. This is stored to keep a record of different graphs produced by the model and allow for discussion about anomalies, and the results that are produced. And is likewise referenced to generate the referenced graph of a model.

The next part describes the current plan to implement this structure together and plans to finish the remaining documentation for the EON project.

5 IMPLEMENTATION PLAN

The project has been segmented into several milestones to indicate the progress and the section of the project that is currently be worked on. These milestones being, “Planning implementation”, “Front and back end implementation”, “Testing and requirements verification”, “Finalize design and prepare for delivery”, and “Final presentation and completion of project”. Small tasks and general topics will be delegated between group members, to ensure even distribution of work and responsibility. A visual representation of the implementation plan can be seen in Figure 5.1 depicted on the next page.

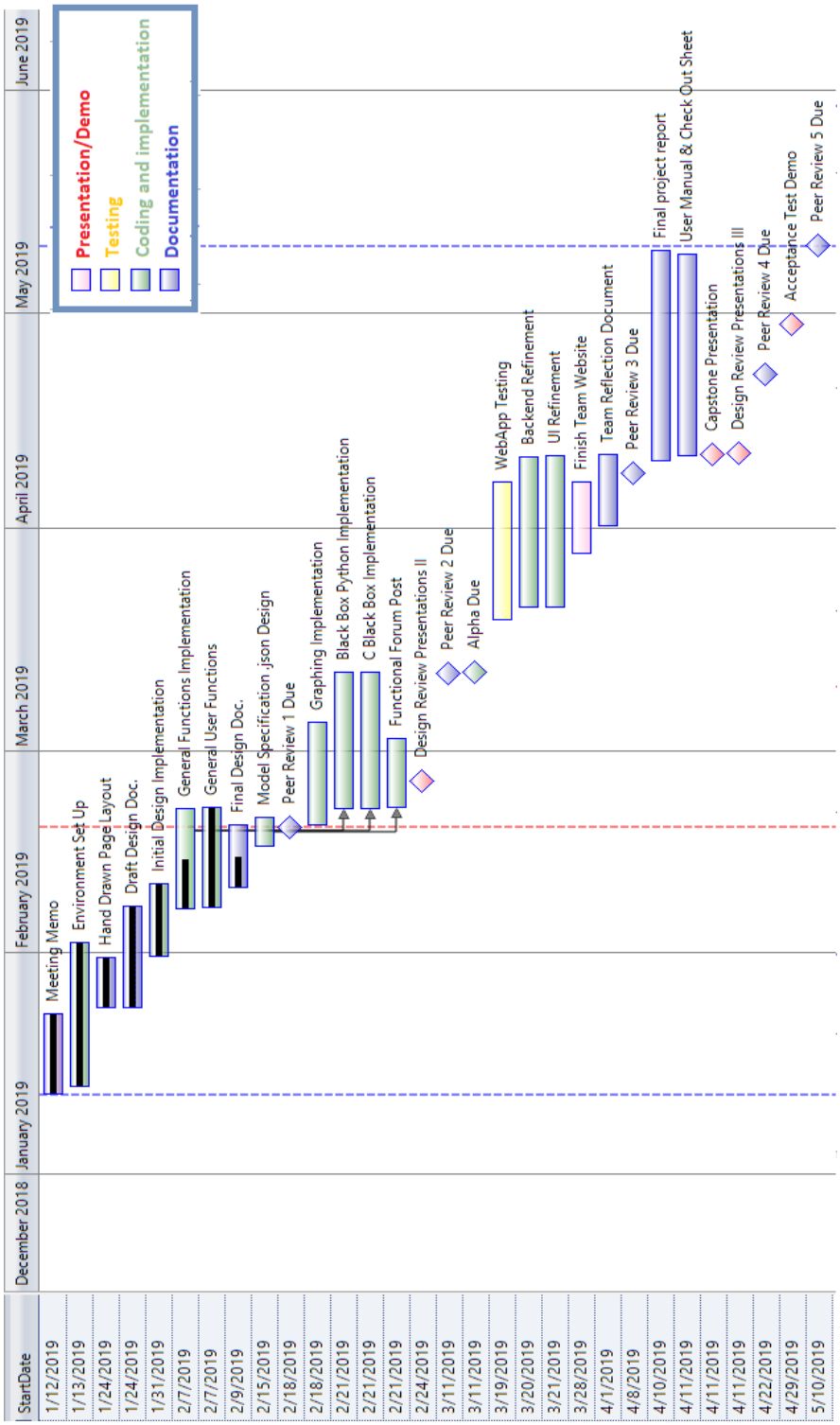

Figure 5.1 Current Gannt Chart as of 2/18/19

Planning implementation: *Dec. 2018 – Feb. 2019.*

Generate an initial design document to create a plan for deciding the implementation and workflow of the application and discuss design decisions, including but not limited to database, page structure, and any potential development issues. In addition, populate a Gantt chart with all relevant components and tasks.

- Generate the general structure of the web application, mapping the user flow of the application. This will help in allowing us to gather a good idea about user use cases and will enable us to generate a useful structure to meet user requests.
- Further specification and structure for both model, and instance JSON files. along with Django logic to handle either Python or C code.

As shown in Figure 5.1 significant phases of "Planning implementation" include:

- Hand Drawn page layout
- Initial design implantation, and the Design Document
- Peer review 1
- Model specification. JSON design

Front and back-end implementation: *Jan. 2019 – Feb. 2019.*

The front and back-end will be implemented simultaneously with tasks to design pages, model graphing capabilities, JSON dictionary structure, and other tasks to achieve required code operations.

Front-end tasks will focus on:

- Creating the user interface that will utilize the back end.
- Creating a user account management system for forum posts and model uploading.
- Graphing implementation interface to manage and create models.

While back-end tasks will focus on implementing:

- PostgreSQL database queries to managing user activity such as user profile management.
- Further specification and structure for both model, and instance JSON files. along with Django logic to handle either Python or C code.
- Model flat file functionality, along with Firejail to safely handle uploaded user code.

As shown in the figure, major phases of "Front and back-end implementation" include the following:

- Environment Set up
- Black Box python and C implementation
- Functional Forum operations
- Model specification page (creates JSON files)
- Graphing implementation
- Peer review 2
- Alpha Due

Testing and requirements verification: *Feb. 2019 – Mar. 2019.*

Once the application has the front and back end implemented, user testing will begin. This will be done with a small group of users to test the EON web application. Once they have completed their testing, they will fill out a survey about the web application. Surveys conducted after testing will be used to implement suggested interface layouts to improve the user experience.

Throughout development, the requirements document will be reviewed and used to verify that all requirements have been satisfied.

As shown in the figure, significant phases of "Testing and requirements verification" include the following:

- WebApp Testing
- Backend Refinement and UI refinement is time to fix any issues or include suggestions that test users might provide.

Finalize design and prepare for delivery: *Mar. 2019 – Apr. 2019.*

Finalize design and preparation of delivery will consist of preparing for the closing assignments and taking steps to ensure the effectiveness of the product. This includes making last-minute changes and requirements to polish appearance and accomplish any reachable stretch goals outlined in the requirements document. Alongside this development of the final deliverable documents will begin to ensure a detailed project report and user manual.

As shown in the figure, significant phases of "Finalize design and prepare for delivery" include the following:

- Team reflection Document
- Finish Team website
- Peer Review 3
- Capstone Presentation
- Design review presentations 3

Final presentation and completion of project: *Apr. 2019 – May 2019.*

Finish the final project report, the final presentation of Pandemic Processing's EON implementation, and ensure proper documentation of the product to the client within the user manual.

As shown in the figure, significant phases of "Final presentation and completion of the project" include the following:

- User manual & check out sheet
- Final project report
- Peer review 4

6 CONCLUSION

Infectious diseases are a major health challenge and affect modern day society. Hundreds of thousands die every year from preventable diseases. The use of data-driven disease modeling in modern epidemiology can dramatically reduce the number of lives lost. A model must be created of a particular disease in order to properly plan, and epidemiologists construct these models so that they can determine the optimal preventative measures to be taken. The problem is that in creating these models, these scientists have a difficult time exchanging information before publication. Thus, EON will bridge this gap in the model development process by giving epidemiologists a public place for them to present and review their ideas. EON will become a public repository for epidemiological models, and a forum through which those models can be critiqued and refined.

This document will act as a blueprint for the remainder of this project and is a live document. By laying out detailed versions of the implementation and architectural plans for EON we could better capture design flaws early on, which will save time as development continues.

EON is integral in an age where information spreads quickly and diseases move even faster. EON will be the solution that can revolutionize epidemiology and could be applied on an immense scale and generate a large future impact. It has the potential to save millions of lives around the world. Currently development of an alpha prototype is occurring, that will be completed by early March, 2019.