RESERVOIR PLANNING TOOL

By The Transforming Drainage Project

29.01.2017

Clayton Marshall
Drew Atkinson
Jonah Heeren
Vritant Bhardrwaj

Purpose

Agricultural researchers, drainage contractors, and farmers have all found a need to provide more secure water for crops throughout the growing season. By maintaining adequate drainage during wet periods and limiting nutrient losses from drained agricultural landscapes, crop losses can be minimized. We will help build a web based solution by running user inputted data through the Transforming Drainage Project algorithm and using its output to provide a visual representation of years of data used to plan the construction of more efficient reservoirs (or, also called ponds). This planning can increase crop yield by 20%.

Currently the only product similar to this is the Paradigm suite which includes commercial options to plan the creation of reservoirs, but it primarily addresses engineering concerns. Unlike Paradigm our platform is open source and available for use by anyone seeking to minimize irrigation deficit.

Core Project Requirements

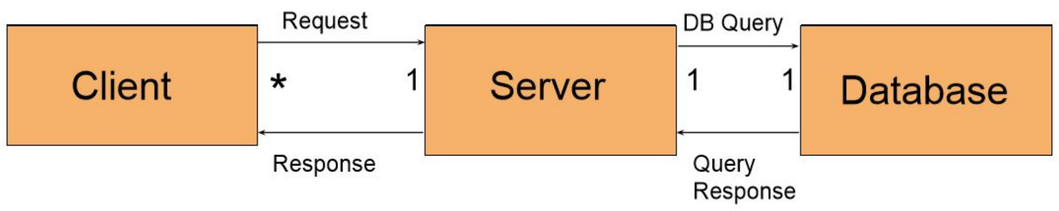
1. Data Input
    - Any user can input data using either their own CSV file whose format will be specified, or by selecting a location on a provided map layout.
2. Calculation
    - User inputted data along with our project owner's research measurements will be sent to the Transforming Drainage Project's algorithm, and as an output the user will receive a downloadable CSV file and graphs allow more efficient planning of a reservoir/pond.
3. Viewing Calculated Data
    - The main output graph will show bypass volume and storage deficit for each pond size increment.
    - Users can also select individual pond sizes and dates to see more in depth data analysis.
4. Saving Calculated Data
    - Users may want to save their calculations for later review, so our platform offers local downloads of either CSV files with the daily data output by the simulation algorithm.
5. Usability
    - An intuitive design that makes the application usable for those who have little to no knowledge of the tool is essential.

- The product will be optimized for users on a desktop or larger tablet screen, since that's where the user is likely to download files locally and view in-depth graphical data.

Design Outline

High Level Overview

Our project will use a client-server model. User data will be passed in through the client, then be sent to the server where we will either parse the user's inputted file and query the database for additional information that the user left out, or we will query the database for all information needed for the Transforming Drainage Project's algorithm. The calculations will then be run on the server and the output will be sent back to the client for graph production.



1.) Client
   a.) The client will be where the user is able to interact with our platform.
   b.) A user will fill out form data to specify an interval of pond size.
   c.) The user will be able to upload a file from here and input a location on our provided map API.
   d.) The data inputted on map, file(if provided), and form will then be sent directly to our server.
   e.) After responses from the server/database the client will display all data returned on an interactive graph.

2.) Server
   a.) The server will handle all communication between the client and the database.
   b.) The server will parse the data given in the provided files sent from the client if a file was uploaded.
   c.) The server will query the database for the data associated with the inputted location of the user. If a file was uploaded by the user the evaporation data returned from the database will be added to the user's file. The server will run the Transforming Drainage Project's algorithm with

all inputs from database and client form, and send them to the client as a response.

3.) Database
   a.) The database will store 30+ years of modeled precipitation data for the midwest region of the United States, that our project owner provided.
   b.) The database will be queried for data corresponding to a given location and will return its findings back to the server for calculation.

Design Issues

Issue: Where should we host our application?

- Option 1: Engineering Computer Network (ECN)
- Option 2: ITaP Firebox Virtual Servers
- Option 3: HUBzero at Purdue
- **Option 4: Amazon Web Services (AWS)**

In our initial meeting with the project owners, we had a multitude of server options presented to us along with our suggestion of AWS.

Hosting on ECN was quickly dismissed because not having root access would limit our usage of web frameworks. We encountered similar issues with Firebox virtual servers, the language restrictions of the platform would have hindered both quality and speed of development. HUBzero would've been simply too expensive, the starter hub package came with a price tag of $72,177 for the first year and $47,465 for subsequent years.

AWS was chosen because it is not limiting upon languages and frameworks, it has a strong ecosystem of services, and the price was sustainable for this project.

Issue: Which mapping API should we use?

- Option 1: Leaflet
- Option 2: Openlayers 3
- **Option 3: Google Maps**

Leaflet and OpenLayers 3 are both open source, extensible, and allow for grid overlays which are essential to our project. Had we chose either of these JavaScript libraries, we would have needed a middleman API to translate addresses into latitudes and longitudes for DB lookup.

The key feature though that Google Maps had that swayed our decision was the built in address search functionality. Beyond that, Google Maps is simply the most developed and mature platform available for mapping. Since our software will be open source and not for profit, Google Maps also does not pose any additional cost, barring an influx of users.

Issue: What CSS framework should we use?

- Option 3: Materialize
- Option 4: Semantic UI
- **Option 4: Bootstrap**

To style the website, we chose to use Bootstrap as it is well known in the developer community and easy to learn, and it has a big community as well. Compared to Semantic UI, it is much easier for junior developers, and compared to Materialize, it is much easier to customize.

Issue: What JavaScript framework or library should we use?

- AngularJS
- ReactJS
- **Jquery**

Our project owners aim to pass on the project to undergraduate students who would further develop and maintain the website. Since AngularJS and ReactJS are relatively hard to learn, and updating them can cause things to break, we decided against using them and chose to go with JQuery. JQuery is really easy to pick up and has a larger community than the other two.

Issue: What back end language should we use?

- Option 1: Python
- Option 2: PHP
- **Option 3: Node.JS**

We considered both Python and PHP because they would've been required depending on where we chose to host our application. However, we chose Node.JS because most of our team members are familiar with it and it also allows us to have a JS commonality between front and back end. This commonality between the front and the back end should also help our project owners find future developers to continue this project since less languages/frameworks will be used.

Issue: Responsive or Adaptive design?

Our audience would mainly use either laptops, desktops or tablets to use this website, since it is for graphing years of data, and uploading their own. This would mean that the layouts would be similar for all devices but tablets. And so we have chose to go for an adaptive design, where the larger screen devices will have a fixed layout and tablets would have a different layout.

Issue: How should we handle parsing user data files?

A user will upload a new data file in CSV format which will have either 5 or 6 columns per row. The optional column is a value for open water potential evapotranspiration (PET), as we do not expect most users to know this value. However, in the interest of robustness, we will still accept this column. In cases where the data file does not include the open water PET, we will query the database for regional data to synthesize into the user's dataset.

Issue: Should we allow users to create accounts and log in?

When we initially discussed the project concept with our project owners, we mentioned that it would be possible to allow users to create accounts that would then store their upload data. As we discussed all of the requirements for the project further, we decided to keep user accounts out of the initial scope of the project in order to conserve development time. If we end up having extra time, user accounts will be added in a subsequent release.

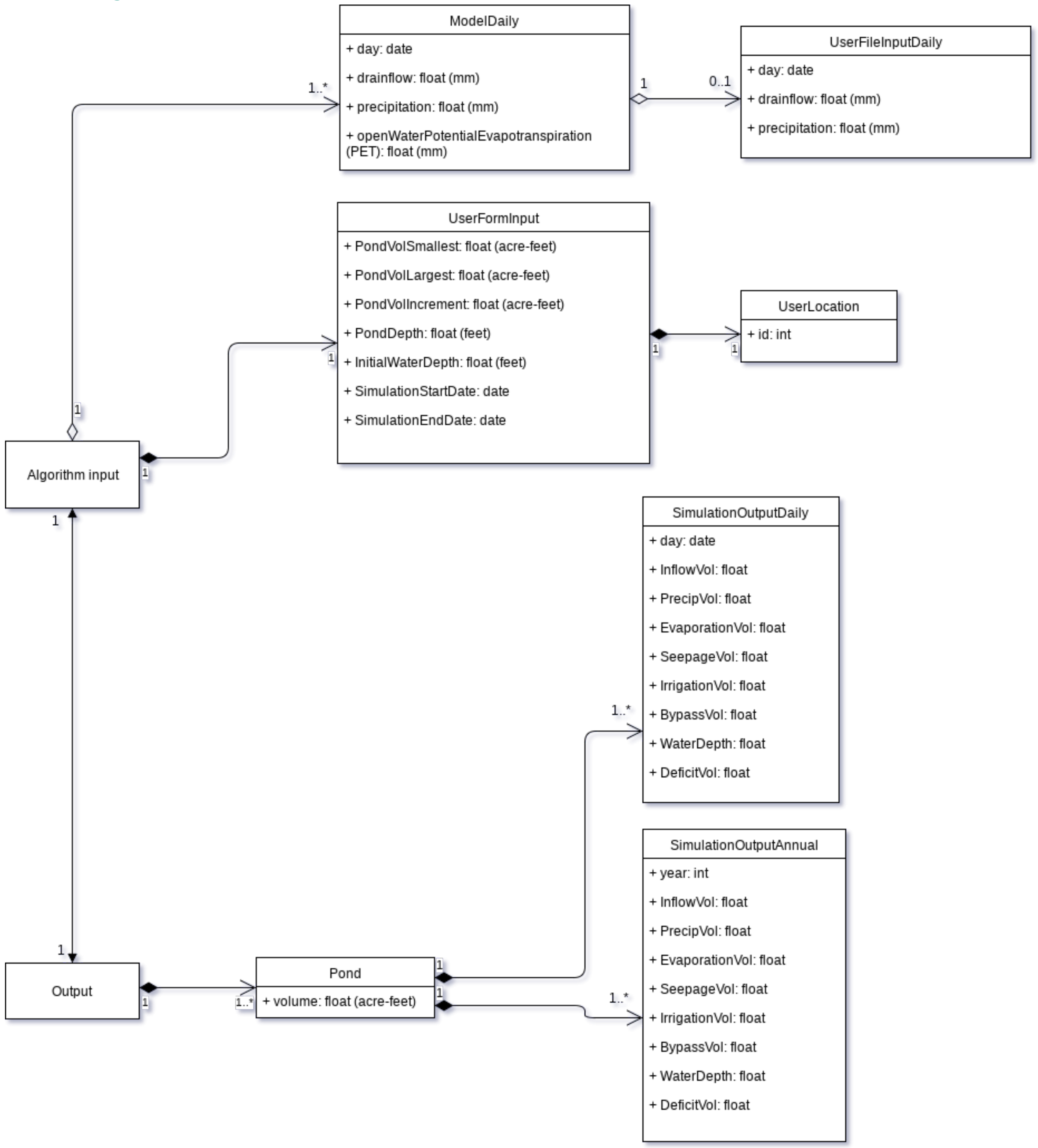
Issue: What chart library should we use?

- Highcharts
- Chartist
- D3
- **Google Charts**

For our JavaScript charting library, we initially considered Highcharts because it has fantastic browser compatibility and was free for non-profits. We didn't choose D3 because of it's steep learning. Chartist did not have the features we desired for, such as clickable data points. Later on, we chose Google charts since it had much more documentation than Highcharts, which would make it easy for future developers to catch up with.

Design Details

Class Diagram



Class Details

Algorithm Input

The main logic of our application is structured around an algorithm developed by the Transforming Drainage Project which calculates bypass flow and storage deficit at a variety of different pond volumes. The goal of these calculations is to find the pond volume with the lowest cost that will still retain adequate water levels. The algorithm input is comprised of user specific input from our web form (UserFormInput) and regional daily data. Regional data will be pulled from a simulated thirty year model of water flow at any location across the midwest (ModelDaily).

UserFormInput

This class holds the initial data for the simulation, gathered by form input. PondVolSmallest, PondVolLargest, and PondVolIncrement, in units of acre-feet, will be used to calculate a set of pond volumes to run simulations against. PondDepth and InitialWaterDepth give information about the depth of the pond. Finally, it will include a date range that the simulation will run for. Part of the UserFormInput object is a single UserLocation object, which is gathered by the map.

UserLocation

This class is a representation of the location where the user wants to run the simulation. The map will output an ID corresponding to the closest polygon in a GeoJSON layer. This location ID will be used to look up the model's daily data (ModelDaily) to use in the simulation. Each location ID will correspond to a Table with rows consisting of ModelDaily objects.

ModelDaily

A ModelDaily object will hold data from one day of Transforming Drainage Project's geolocated water flow model. The fields include drainflow, precipitation, and open water potential evapotranspiration (PET), all measured in millimeters. The algorithm will have as many of these objects as there are days in the date range of the form input.

UserFileInputDaily

If the user decides to upload a CSV file to use their own measured data instead of the model's daily data, each ModelDaily object will be associated with a UserFileInputDaily object from the same day. This causes the algorithm to overwrite the model's information for drainflow and precipitation, and use the user's data instead.

Algorithm Output

Each algorithm input object will have an associated algorithm output object. The algorithm is composed of a pond for each possible volume, and each pond will have both daily and annual data.

Pond

The algorithm will output a pond object for each simulated pond volume, given by PondVolSmallest, PondVolLargest, and PondVolIncrement in the UserFormInput class. The pond object holds the volume of its respective simulation. Each pond is comprised of annual and daily output.

SimulationOutputDaily

For a given pond volume, for each day the algorithm will calculate Inflow Volume, Precipitation Volume, Evaporation Volume, Seepage Volume, Irrigation Volume, Bypass Volume, Water Depth, and Deficit Volume. This data will later be used in the output graphs and compiled with the user's original data into a CSV file for later research.

SimulationOutputAnnual

SimulationOutpuAnnual will hold the exact same data fields as a daily output, but instead hold annual averages and a year. This data will be used for the graphs in the user interface.

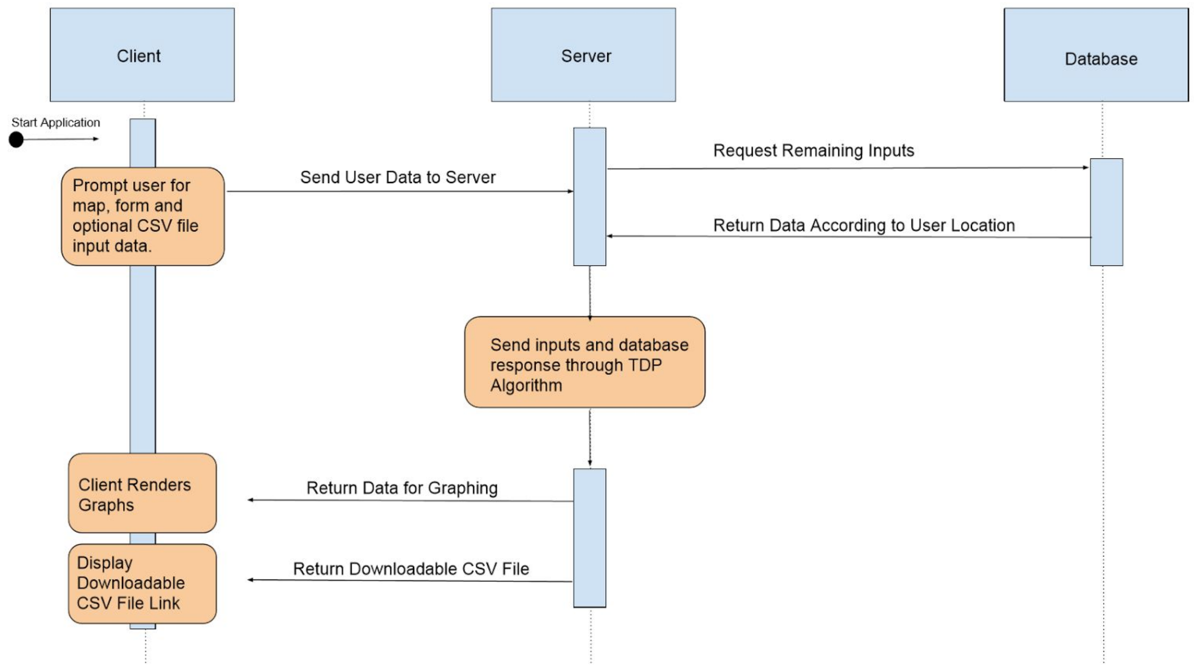
Sequence Diagram Explanation

When using our application, a user must input a location, provide input to a data form for reservoir sizing constraints, and an optional CSV file upload. This optional file will be used to replace our researchers data with a user’s own measurements.

Once all data has been sent to the server, the server will parse the CSV file if it was provided. Once this is done the server will query the database for remaining input information based on the selected location. If a user did upload a file, the only data to be used from the database will be evaporation data for that area.

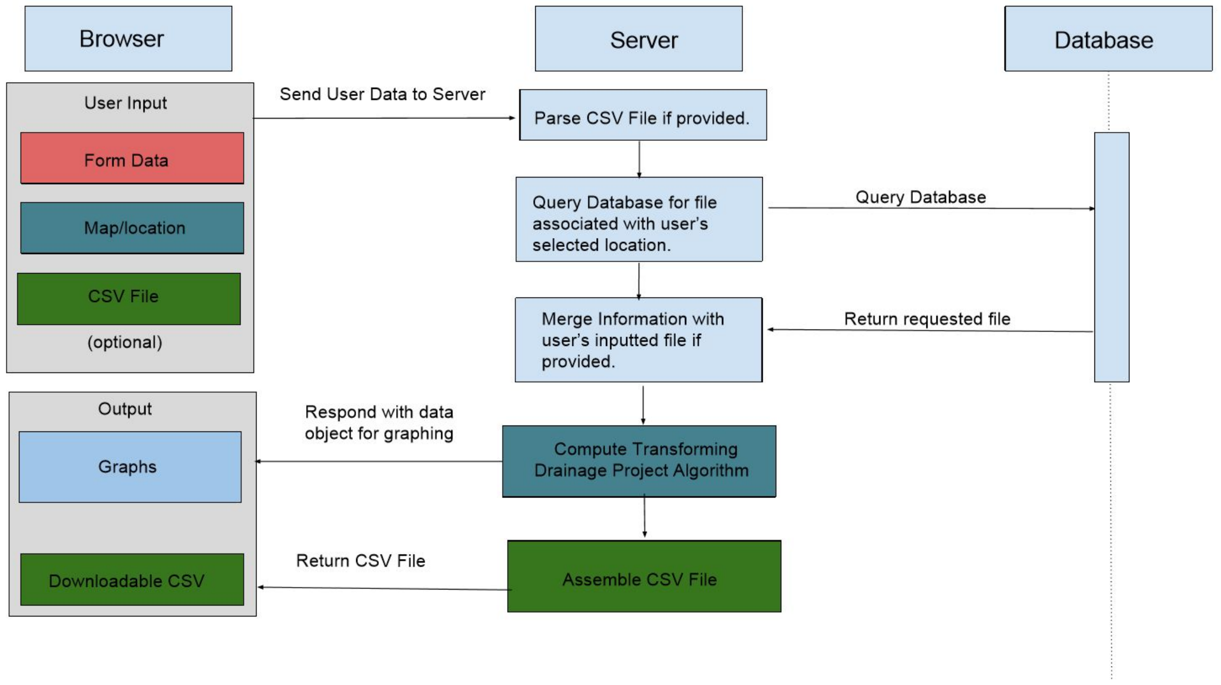
The measurements from the database will then be added to the user’s uploaded file. If no file was uploaded, all extracted information from the database will be used for input. These inputs along with the form data will be given to the Transforming Drainage Project’s algorithm. The output of the algorithm will be sent back to the client in the form of a downloadable CSV file as well as a json response containing aggregate annual data to generate graphical representations of the results.

Sequence Diagram



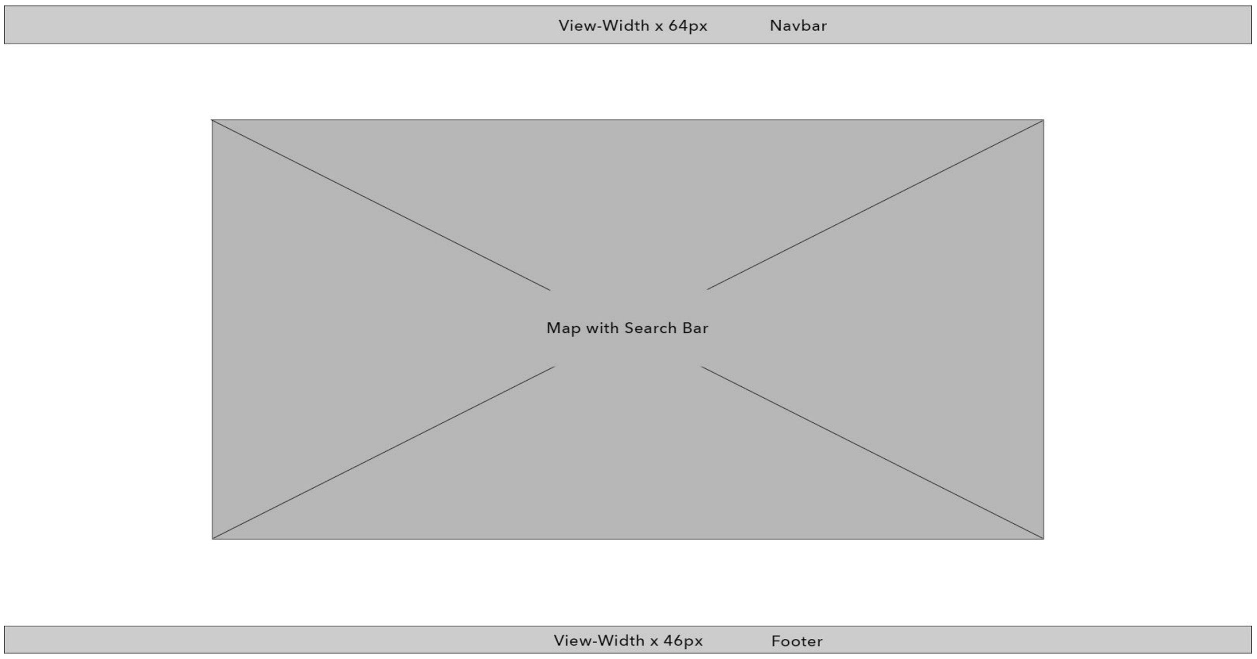
Client
Server
Database
Start Application
Prompt user for map, form and optional CSV file input data.
Send User Data to Server
Request Remaining Inputs
Return Data According to User Location
Send inputs and database response through TDP Algorithm
Client Renders Graphs
Return Data for Graphing
Display Downloadable CSV File Link
Return Downloadable CSV File

Sequence Diagram with Dataflow Specifications

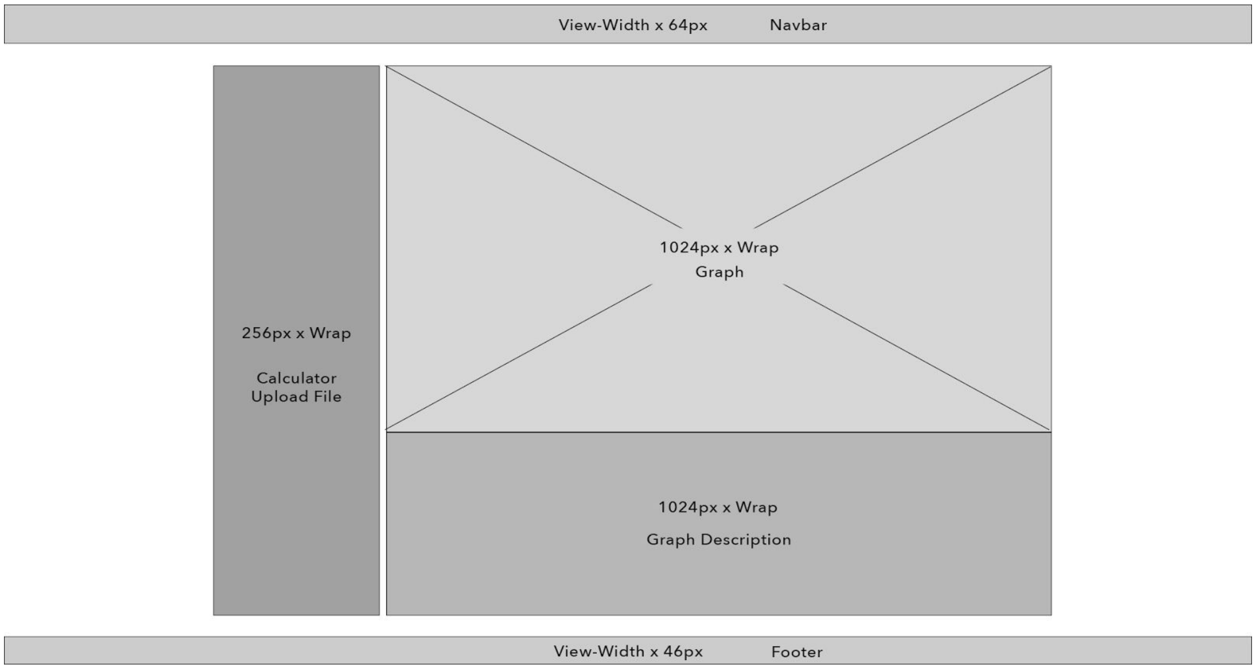

Browser
Server
Database
User Input
Form Data
Map/location
CSV File
(optional)
Send User Data to Server
Parse CSV File if provided.
Query Database for file associated with user's selected location.
Query Database
Merge Information with user's inputted file if provided.
Return requested file
Output
Graphs
Downloadable CSV
Respond with data object for graphing
Compute Transforming Drainage Project Algorithm
Return CSV File
Assemble CSV File

Wireframes

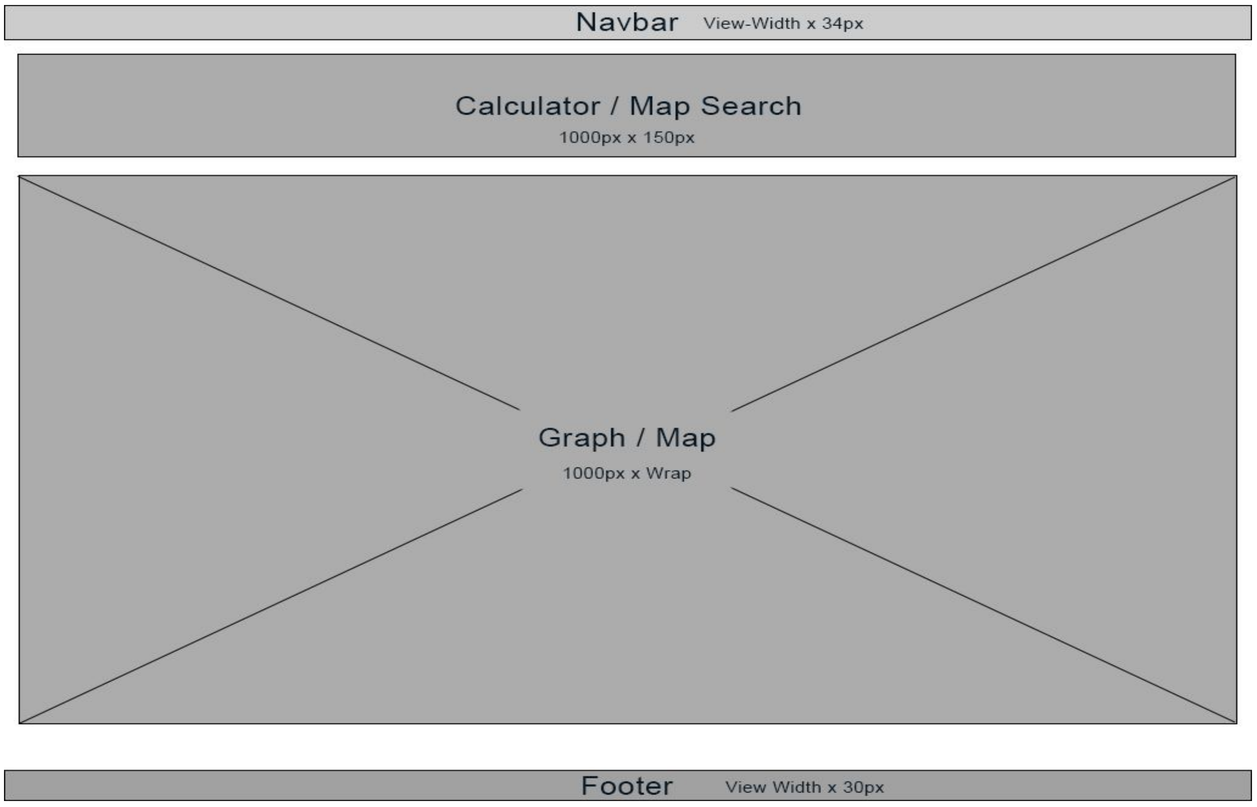
The user will first see a map, using which he/she will select a location.



Once a location has been chosen, the data from that location can be graphed, using the calculator.



The layout for tablets will be slightly different.


Navbar
View-Width x 34px
Calculator / Map Search
1000px x 150px
Graph / Map
1000px x Wrap
Footer
View Width x 30px