

Expanded User Documentation

USGS Tidal: Data Visualization Website

Version 1.0 • 7 June 2019



Natasha Ng • Caleb Ouellette • Ken Jung • Cristina Feliberti
Western Washington University

Table of Contents

1. Introducing Our Software	2
1.1 Motivation	2
1.2 Project Description	2
1.3 Project Development Approach	2
1.4 Project Features	3
2. For Product Consumers	4
2.1 Website Consumers	4
2.2 Data Producing Consumers	4
2.2.1 Setup	4
2.2.2 Workflow	5
2.2.3 Manifest file	5
2.2.4 Supported data formats	6
2.3 Hosting the Docker Container	7
2.3.1 Building	7
2.3.1 Running	7
2.3.1 Deploying	7
3. For Product Developers	8
3.1 Design Plan/architecture	8
3.2 Technologies Used	8
3.2 Division of Work	9
3.2.1 Server	9
3.2.2 Graphs	9
3.2.3 Frontend Website	11
3.2.4 Data Packager App	11
3.3 Testing	12
3.3.1 Experimental Results of Validation Test Plan	12
3.3.2 Description of Each Validation Criterion	12
3.4 Unimplemented Features	14
3.4.1 Data size issues	14
Glossary	15
Licensing	16

1. Introducing Our Software

1.1 Motivation

Sean Crosby is a Coastal Geomorphologist working with USGS interested in studying regional wave, wind, and pressure transformation. Part of his research includes displaying high-resolution coastal wave predictions via wave buoy observations. Data is then collected throughout the local Salish Sea area, recording wind direction and speed, wave heights and periods, water level and storm surge (also referred to as non-tidal residual), and changes in air pressure.

Ultimately, Sean would like to depict all of this data through a website that is available and accessible to not only him and his colleagues at USGS, but also to the public. With predicted data visualized in this application, municipal authorities will have enough time to prepare for areas expecting to undergo flooding in anticipation of upcoming storms. This website can also be used for recreational purposes, such as surfers wanting to check the conditions at specific locations beforehand.

1.2 Project Description

We (the USGS team) are tasked with building a website that visualizes the movement of wind, wave, and pressure data across mapped locations over a 48 hour time period. The front-end of the website is built using React and Deck.gl, a JavaScript library from Uber. The main feature is an interactive map that, when clicking different locations, also presents graphs comparing two different categories of data: water level (predicted, observed, and predicted tide) and wind (predicted direction, observed direction, and wind speed). This project is hosted in a Docker container and handed off to Western's WebTech services team.

1.3 Project Development Approach

The Agile software development methodology was employed for this project. This methodology required our team to set goals that are accomplishable within two-week long sprints during the development phase. During these sprints, weekly scrum meetings between the team members, customers, and supervisor were held. These

meetings allowed every person involved with this project to discuss progression and confirm their level of understanding of individually assigned tasks.

1.4 Project Features

The primary feature of this product is a website providing an interactive map for the user.

Secondary features of the map includes the following:

- Zooming in and out of specific regions on the map
- Displaying the change in the movement of wind and wave vectors over time
- Displaying the change of flood and pressure levels over time
- Clickable locations display a graph comparing predicted and observed water level or wind direction and speed data
- Timeline slider for a 48-hour time span
- Controls for layer toggling to view different types of data

2. For Product Consumers

2.1 Website Consumers

Website Consumers are anyone who would like to see the data visualization. The website can be accessed through Safari, Firefox, or Chrome at the url provided by whom ever hosts the docker container. The website is designed to be intuitive and our goal is for end consumers to need no additional documentation to use the product. The Map is navigated through click and drag, and scroll in to zoom. Icons on the app are interactable and will display graphs. Region on the map can be clicked to display wave data for a given area. Additionally there are links in the top left corner for more information about the product.

2.2 Data Producing Consumers

Data producing consumers will be anyone who is uploading data to the server. We have built a python application in order to accomplish this. The application starts by reading a manifest file parsing any files mentioned in the manifest file, and sending the parsed version to the server.

2.2.1 Setup

The Data packager app is currently distributed by pulling the master branch on the repo. The folder is data-packager. No other folder needs to be accessed for this. Python 3 is required along with the following python packages: scipy.io, numpy, requests.

The file config.json will need an API key, which is set by whoever sets up the server, and API URL which will be the IP of the server and optionally a sleep interval. If the sleep interval is not set, the app will only run once and then exit. Otherwise, it will wake up on sleep interval to check for a new manifest file.

Once pulled and configured the app can run with the following command in the terminal:

```
python3 ./app.py ./path/to/directory/of/data
```

2.2.2 Workflow

Once the data packager app is setup whoever is producing the data can move files into the data folder (the folder that is given as the second argument when starting the app). The data-packager will look for a manifest.json file. This file should be the last thing copied into the folder to ensure all data is ready to be processed when the data-packager wakes up. The data-packager will parse the needed files and upload everything to the server.

2.2.3 Manifest file

The manifest file is a json file that describes what data is being uploaded. The file must be valid JSON and have the following structure:

```
{
  "dataSetId": 20190305060000,
  "startDateTime": "2019-03-05 06:00:00",
  "forecastHours": 48,
  "SpatialDomains": [{
    "siteDisplayName": "Bellingham Bay",
    "data": [{
      "fileName": "bellinghamWave.mat",
      "dataType": "Wave",
      "dataFormat": ".mat"
    }
  ],
  "NEpoint": [48.786233,-122.399424],
  "SWpoint": [48.548654,-122.701661]
}],
  "PointLocations": [{
    "siteDisplayName": "Bellingham Bay Tides",
    "data": [{
      "fileName": "9449211_twlpred.csv",
      "dataType": "TidePred",
      "dataFormat": ".csv"
    }
  ],
  "Location": [48.7450, -122.4950]
```

```
    },  
  ] }
```

The Json file is an object that contains the following fields:

- forecastHours: a Number of how long each prediction file goes for.
- startDateTime: The time at which all predictions start.
- SpatialDomains: is an array of sites in which there is data in a space on the map.

For a site in Spatial Domains there is a:

- siteDisplayName : Name of the site seen by the public
- NEpoint: the North East point given as an array of [lat, long]
- SWpoint: the South west point given as an array of [lat, long]
- data: an array of data files. Each data file needs the fileName, the datatype, and the data format. Only some combination are supported See supported data formats.

- PointLocations: is an array of sites in which there is data in a graph on the map.

For a site in PointLocations there is a:

- siteDisplayName : Name of the site seen by the public
- Location: the Location given as an array of [lat, long]
- Data: an array of data files. Each data file needs the fileName, the datatype, and the data format. Only some combination are supported See supported data formats.

2.2.4 Supported data formats

Currently we support the following data formats.

Spatial Domains

Type: Wind

Formats: .mat

Type: Wave

Formats: .mat

Point Locations

Type: water_pred

Formats: .csv

Type: water_obs

Formats: .csv

Type: wind_pred

Formats: .csv

Type: wind_obs

Formats: .csv

Note to future developers: If you wish to add new data formats you must write a new adapter and place it in the adapters folder then, add it to parserCatalog in data_parser.py From there the data will be available to the front end via an api call to the site with the type appendend.

2.3 Hosting the Docker Container

The server app is contained in the server folder of our repo. Our server is a self contained docker container and does not need access to a persistent storage. Instead the app simply caches the current data, and deletes it when new data arrives. This makes host relatively simple. If you are not familiar with docker read some documentation before getting started.

2.3.1 Building

To build simply run docker build in the server folder. See the Readme in the folder for the exact command. This will build the docker image that runs the server.

2.3.1 Running

In order to run the image you can use docker run, but you must specify an environment variable `usgstidalapiadminkey`. If not specified, the app will quit on start. In order to interact with the server you will need to expose port `8080`. See the Readme in the folder for exact details.

2.3.1 Deploying

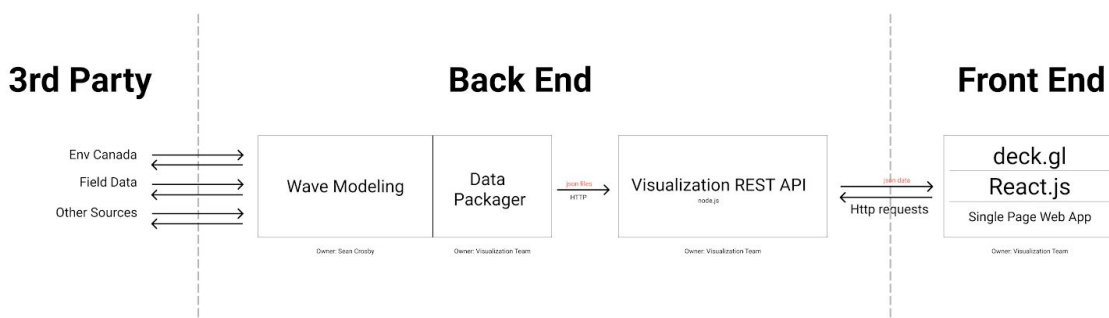
This will vary from host to host and requires some specialty knowledge. There are guides for doing this on AWS, Azure and Google cloud. The command to deploy it to Google Cloud in the Readme.

3. For Product Developers

The following sections will cover crucial information necessary for understanding the development process of the team over the nine-month long period. Additional information that may be helpful for future developers is also included.

Throughout this process, three months were spent on drafting a design plan and understanding the requirements for the project. The remaining six months consisted of implementation and testing.

3.1 Design Plan/architecture



Our Client currently produces wave models for the puget sound area. He takes in data from 3rd parties to produce a wave prediction for the next 48 hours. This is the Wave Modeling section of this diagram. From there we will parse his data using the Data Packager app that will send the new data to our Server side code. From there the client side app can make requests to it.

3.2 Technologies Used

The majority of the project was done using JavaScript; specifically, React.js. Parsing for changes in wind and wave data movement uses Python.

C3.js: a D3-based JavaScript chart library used for creating interactive data visualizations for use in web browsers. This is used specifically for creating the water level and wind graphs.

React: a JavaScript framework for building web applications. Developed by Facebook.

Deck.gl: a library to create and manipulate map data visualizations. Developed by Uber.

Python: a high-level programming language that can handle large amounts of data

Docker: a computer program that runs software virtually in containers.

NodeJS: a JavaScript runtime environment.

GitLab: an open-source software development platform with version control

3.2 Division of Work

Development was divided amongst the four team members, with each member focusing on their respective fields:

- Server
- Graphs
- Frontend website
- Data Packager

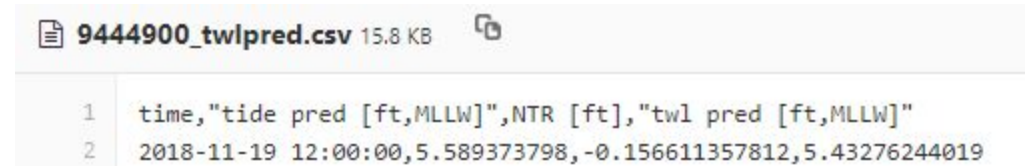
3.2.1 Server

For backend server work we used Express with Node.js. We wrapped this inside a docker container to make it easy to deploy. This is a very popular, lightweight api framework that made it easy to setup HTTP endpoints. We also use Express to host the client side web application. The Application is served from the public/build folder. When you want to deploy a new Client side app you will to put the new build in this folder, then rebuild and redeploy the docker container.

3.2.2 Graphs

First, data is passed in from the client's machine as .CSV files. These files are one of two categories, water or wind, and contain either predicted or observed information recorded at specific Tides or Winds locations, as indicated by the 8-digit NOAA ID preceding the file name. These locations and their coordinates are taken from the manifest file drafted by the client, and is also located in the client's machine.

Four .CSV file formats are supported, each with a different set of headings. Examples of the CSV inputs are below.



9444900_twlpred.csv 15.8 KB	
1	time,"tide pred [ft,MLLW]",NTR [ft],"twl pred [ft,MLLW]"
2	2018-11-19 12:00:00,5.589373798,-0.156611357812,5.43276244019

	9444900_twlobs.csv 13.2 KB	
1	,time,twl	
2	0,2018-11-19 00:00:00,5.5	

	9440910_wind_pred.csv 3.4 KB	
1	time,Direction [deg],Speed [m/s]	
2	2019-02-16 12:00:00,59.022205532,0.582767002589	

	9440910_wind_obs.csv 37 KB	
1	time,Direction [deg],Gusts [m/s],Speed [m/s]	
2	2019-02-13 14:54:00+00:00,92.0,3.8,3.5	

All Tides locations will contain predicted information, but some sites do not have observed information. Handling for this test case is accounted for the respective `Water.js` and `Wind.js` files. The diagram below illustrates interaction between multiple components to parse the CSV data, create a C3 graph depending on what kind of information is desired, and displaying the graph in the application.

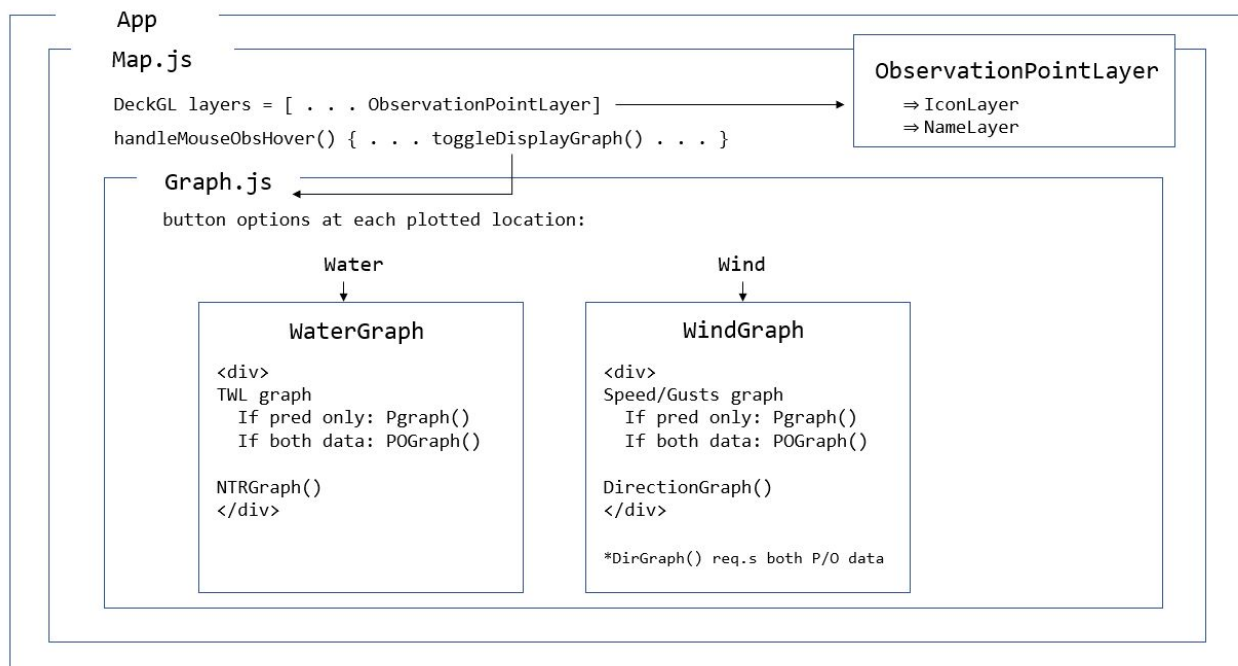


Diagram illustrating interaction between components for graph creation and display

3.2.3 Frontend Website

For the data visualization of wind and waves, we used Deck.gl to plot the height of each individual latitude-longitude coordinate in a specific area. The data for wind and wave is given to us in binary mat files but is preprocessed into JSON beforehand. The wave file contains latitude coordinates, longitude coordinates, height (in feet), hours, and direction (in degrees). The wind file contains latitude coordinates, longitude coordinates, velocity in eastward direction (m/s), and velocity in northward direction (m/s).

The wind data points each lat-lon coordinate and chooses a color based on the given height.

3.2.4 Data Packager App

The data packager is an application designed to handle the various types of data passed into the website. It accepts these different data files as input, parses them into JSON format, and then sends them to the data web API. Parsing is done through several adapters built using Python, of which the adapter used to convert the data to JSON is dependent on the type of data being passed in.

Currently we support the following types of data listed below. The data type, file extension, and respective parser must be included.

```
parserCatalog = {
  "wind": {
    ".mat": parse_wind_mat
  },
  "wave": {
    ".mat": parse_wave_mat
  },
  "tidepred": {
    ".csv": parse_water_pred_csv
  },
  "tideobs": {
    ".csv": parse_water_obs_csv
  },
  "windpred": {
    ".csv": parse_wind_pred_csv
  }
}
```

```
    },  
    "windobs": {  
      ".csv": parse_wind_obs_csv  
    }  
  }  
}
```

3.3 Testing

3.3.1 Experimental Results of Validation Test Plan

#	Title	Testing Status
1	Time slider, layer toggle, and zooming onto sites feature	Passed
2	Script that transforms .mat files to JSON format. Plot the wind and wave data onto the website	Passed
3	Generate graphs on observed and predicted water and wind data via script transformation of .csv files to JSON format	Passed
4	Manifest file to describe file types being sent to the server	Passed
5	Docker container for hosting our website onto the university servers	Passed
6	Using a manifest file to plot clickable location sites that when clicked, a pop-up window containing that location's corresponding wind or wave data graph appears	Passed

3.3.2 Description of Each Validation Criterion

Validation 1

We are utilizing the ReactJS library to implement all UI elements of the website. The time slider feature allows users to click and drag the slider across a 48-hour period and view vector visualizations of wave and wind movement at certain times along that timeframe. The layer toggle feature is conducted through buttons, labeled with names such as “Wind” and “Wave”, that when a button is clicked, the only vectors that appear on the map are for that layer’s data. For example, clicking “Wave” will show only the wave data. The zooming in feature is technically a built-in feature in MapBox, but we are modifying it so that when a user specifically clicks on a plotted site on the map, the user’s map view will zoom into that site and display the wave and wind data there.

Validation 2

The client's (Dr. Crosby's) files are originally in .mat binary format. We provided a python script that parses through these files and transforms them into JSON format for easy map plotting. The DeckGL library handles all of the data plotting. This includes wave direction, wind direction, and wave heights. This is all color coded and a color bar is displayed to indicate the relevant type of data measurement and its corresponding color.

Validation 3

The client's (Dr. Crosby's) files are originally in .csv format. We provided several python scripts that parse these files and transforms them into JSON format for easy graph creation. The data contained in these files compare predicted and observed wave (time, tidal water level, non-tidal residual, and tide) and wind data (time, direction, gusts, and speed). C3.js is used to create graphs using the JSON data and added to the website's structure using React. The incorporation of these graphs into the application's view is explained in Validation 6.

Validation 4

The manifest is a JSON file format that we built out with our client. The manifest describes the type of data file formats that our client is sending us. The reason behind this is to create a product that is flexible and allows our client to edit the manifest file and add new sites as he creates models for them. Our program will simply read the manifest file and will use that information to parse the files that are being sent to us.

Validation 5

Docker is the standard method that Western uses to host websites. All of our files are put into a Docker container and so that all the university has to do is put it onto the Docker cluster with little to no set-up on their end.

Validation 6

This manifest file is a JSON file format that our client built with and shared with us. It contains point location information for each site that needs to be plotted on the graph. The sites listed in the JSON are either tide or wind locations, and they each specify their .csv file, the data type (TidePred, TideObs, WindPred, WindObs), the file name, and their latitude/longitude for plotting a clickable point on the website's map at that location. When the user clicks on that clickable point, as indicated by an icon, a pop-up window appears containing that location's corresponding wind or wave data graph. For example, clicking on "Cherry Point Tides" will display its wave predictions versus

.....

observation graph. This is done through the work of adapters on the server side, which will pull the manifest data and place a created map in the window.

3.4 Unimplemented Features

Feature Description	Reason Why Not Implemented
Wave data graph comparing predicted periods, observed periods, heights	Initially, three types of data graphs were planned (water level, wind, and wave). During development, it was decided that graphs displaying wave data comparisons was unneeded and the feature was dropped.
Live upload of About us page	Ideally, our client would like to upload an HTML file from his computer that would be automatically updated on the website. However, this was outside the main scope of the project, so it did not have as high a priority as other tasks.
Pressure layer	This feature requires working with contour layers, which is an additional level of difficulty.

3.4.1 Data size issues

The wave data that gets given to us is about 51mb per site. This is too big to pass off to the client. We currently take the files and remove precision of the float. This gets the file closer to 24mb. After we gzip we can get close to 4mb. This is better but is still a little to big to send to the client. The file step we are doing to get the size down is to split this into 4 files that are loaded as needed for a given site. This is a workable approach but is not ideal. The actual solution we have trying to implement is to switch to contour data from point data. We can see this dramatically reduces the size of the files. However when using standard contour functions the data is producing some weird artifacts. We are currently trying to get this fixed in our to give our client the best product. If we can implement contours we should dramatic increase in speed and performance.

Glossary

Agile methodology: a type of project management used in software development. It assists teams with emphasizing short delivery times via sprints.

Back-end: server-side code

Client-side (client): this refers to the user's computer. Interchangeable with front-end

C3.js: a D3-based JavaScript chart library used for creating interactive data visualizations for use in web browsers. This is used specifically for creating the water level and wind graphs

Deck.gl: a library to create and manipulate map data visualizations. Developed by Uber.

Docker: a computer program that runs software virtually in containers.

Front-end: the code that runs on the client's computer. For example, a web browser.

GitLab: an open-source software development platform with version control

json (.json): JavaScript Object Notation. A format for storing and transferring data

Mat (.mat): a compressed data format used by MatLab software. This format is popular in the weather modeling field.

NodeJS: a JavaScript runtime environment.

Python: a high-level programming language that can handle large amounts of data

React: a JavaScript framework for building web applications. Developed by Facebook.

USGS: United States Geological Survey

Licensing

This product is licensed under the MIT License, as defined as such:

MIT License

Copyright (c) 2019 Natasha Ng, Caleb Ouellette, Ken Jung, Cristina Feliberti

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.