User Document

USGS Tidal: Data Visualization Website

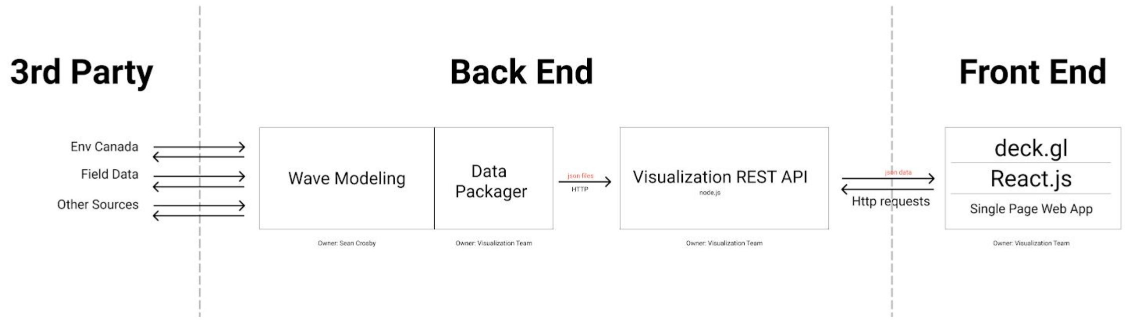
Version 1.0 • 7 June 2019

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

Table of Contents

Summary

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 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 on a server provided by Western's WebTech services team.



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.

Features Implemented

Features	Description
User-navigable Map	User can navigate the map by clicking and drag, and using scroll to zoom.
Adjustable time slider	Once a user selects a site, a time slider is displayed
Displaying the change in the movement of wind and wave vectors overtime	As the user scrolls the time slider the updated data is shown for each given hour.

……
…

Zooming in and out of specific regions on the map	The user can click to zoom into a region and can click the close button to exit that region.
Layer toggling to view different types of data	Selecting specific regions will display options for showing the change in motion of wind and wave data across time. Selecting any of these options will present different data on the map.
Locations display a graph comparing predicted and observed water level, non-tidal residual (storm surge), wind speed, or wind direction data	Icons are located on the map at specific locations. Hovering over the icon displays a window where the user can choose one of several types of data graphs to view and interact with.
Parsing wave, and wind into compressed format to be consumed on the front end	The format that the data comes in is not usable for web development. In order to make json files parsers were written for each file type.
Automatically upload data from data folder to server	The data-packager app parses data and sends it to the server. This ensures the latest predictions are displayed on the website.
Built server inside docker container	The server runs inside a docker container. This makes deploying and hosting the server easy.
Send data to client app from server	The server sends data as the client requests it. This ensures the page loading is quick and no extra data is sent to the client.

Features Not Implemented

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.

....
...

Pressure layer	This feature requires working with contour layers, which is an additional level of difficulty.

Maintaining the Product

Uploading new data

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 start by reading a manifest file parsing any files mentioned in the manifest file, and sending the parsed version to the server.

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 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 by ran with python3 ./app.py ./path/to/directory/of/data.

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.

Server Hosting

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.

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. Once again see the Readme in the folder for exact details.

Bugs and Errors

Long load times

Site data is larger than we want. We are working on a way to trim this data down to improve load times and performance, but do not have this completed. The easiest way to do this would be to switch from point data to contour data. An alternative to this would be to divide up predictions into smaller segments. Instead of sending 48 hours of data only send 10 hours at a time.

Error handling

Error handling could be improved. If all data is present and data is passed as expected everything works. We have not done extensive testing of what happens when data is out of date or poorly formatted.

Handling for locations missing an observation file

In the earlier stages of development, creating the graphs was done separately from the main project. Handling for file existence was working as desired and the graphs displayed included the appropriate fields. For example, if a location was missing a corresponding observation file, the graph would only contain predicted information, with no markers indicative of any observed data. However, when this feature was brought

....

into the React application, the method of which the error handling was conducted did not correctly function anymore. Now, for locations that are missing an observation file, the program believes that the file exists. A graph is created with most of its features, but time labels on the x-axis are NaN and there is a marker for observed data.

Tasks to be Done

The following tasks are labeled with the degree of effort required to continue their development. Low effort tasks require a single person's work for several hours. Medium effort tasks are a single person's work for several days. High effort tasks require two or more people.

Task Description	Reason Why To Be Done	Degree of Effort
Deploy the project	To make the project available, we will need to deploy it to western's servers.	Medium
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.	High
Swell data	Extra feature that was requested halfway through the project. It is similar to the wave data implementation.	High
Contours	Would improve load times and performance of app.	High

Technical Details

Deploy the project

Deploying the project is relatively easy, but you will have to work with the web team at WWU. The Server is already in a docker container, and the web team is familiar with this method of deploying. You will have to work with them to understand how they want to docker container delivered. Two things to keep in mind when deploying is that 8080 is the port to be exposed and an environment variable named usgstidalapiadminkey will need to be set in order for the app to run. This environment variable is used as the password.

Live upload of About us page

This could be done using the existing architecture. Something would need to be added to the data packager to upload the new page. Something would need to be added to the server to send the new page. Finally something would need to be added to the client application to display the new page. This would be a lot of work and we do not currently have a plan to implement.

Swell Data

The swell data is similar to the wave data implementation in that it takes in a lat-lon coordinate and the height of the water. The difference is that the data is plotted along the shores of a given area. The process to add the layer is the same way as the wind and wave layers. Parse the binary files into JSON, pass the file to the server, and the SwellLayer program will plot the points and their heights onto the map. The parsing would be done in Python to go through and get the heights and coordinates. The SwellLayer would be written in javascript with the Deck.gl framework. The layer type can be any point-based plot layer. This would be layers like GridCellLayer, or HexagonLayer. A key point about swell data is that the time intervals are different from the rest of the datasets (48 hours). The time interval can be changed in the Controls.js and will have to be dynamic depending on the toggling of the layer. This would be a quick check on which layer is toggled and changing the value displayed above the slider.

Contours

In order to speed up performance and load time, switching from point data to contour data is the next step forward. We have prototyped this out using matplotlib but have ran into issues with getting nice contours to form on all data. With the large data sets some artifacts are created that are not correct.

....
...