**Tornado Devastation Analysis Product**

**Christopher Hayes**

**Western Governors University**

Date:

10/01/2023

LETTER OF TRANSMITTAL

October 01, 2023

Debrah Parks

Nation Weather Bureau

1325 East West Highway

Silver Spring, MD 20910

Mrs. Parks

With tornadoes becoming more common over the years wouldn’t it be beneficial to be able to predict the devastation amount that a tornado would have over a specific state? We here at WhirlWind Analytics have created a solution just for that.

**Project Recommendation**

**Problem Summary**

It’s difficult to predict an “act of god” and how it’ll affect an area, whether that will be with human fatalities, injuries, or property damage. With tornadoes, there are still many unknown variables at play about when they’re forming and where they’ll go. Unlike hurricanes or other storms, tornadoes can quickly form and disappear within 20 minutes. Despite the great strides that meteorologists have made to help understand and predict tornadoes, it’s just difficult to know when and where it’ll happen. As a result, immediate response by medical professionals or disaster recovery may be hours away from helping the people that are affected. Resolving this would have a great impact on our society.

**Solution Summary**

Here at WhirlWind Analytics, we’ve created an ML engine that can take years of data into the algorithm and predict future devastation of a tornado. It can also predict the length that it will travel in miles, the width of the tornado in yards, the injuries, fatalities, and the property loss it will cause. This tool will be available to use if the user has a reliable internet connection. We also have software as a service plan that customers can purchase and be able to use their own data sets that they can run through the algorithm to make predictions. This will open up a whole new customer base including insurance companies, hospitals, emergency services, governments, etc.

**Application Benefits**

The WhirlWind engine currently has 72 years’ worth of tornado public data, and it’s only growing. The engine will predict future devastation of tornadoes in a specific state. The application is very easy to use, with an online interface, so you can use it anywhere that you have internet access. There’s nothing in the current market that is easy to use as this solution.

**Data Description**

We’re currently storing all the data that will be used. But have plans of letting our customers upload their own datasets that will be held on our servers. If the customer has their own data warehouse, they can integrate that into our SaaS application, and they can switch between using their data and our provided datasets. Our dataset is cleaned and is public data that is published by different weather stations across the United States. We are updating this daily.

**Objective and Hypothesis**

The primary objective of this project will create a future dataset of tornado devastation that can be extracted from the site via download to excel. Customers can use our provided data or use their own data. The hypothesis is that we can provide our customers with a prediction of about 95% accuracy of this future data.

**Methodology**

At WhirlWind Analytics we use Agile methodology. We found that our team works best and most efficiently using the scrum method. Having a lead developer be the scrum master and surrounding that person with product management and business analyst works best for us. That way the product and business team can gather requirements from our customers and stakeholders while the development team only needs to focus on developing and engineering the product. The needs of stakeholders and customers can change at a moment’s notice, whether that be including something new or removing something that is obsolete now. With our sprints being 2 weeks, we able to switch tasks and prioritize easily. Using this methodology, the customers are able to see the progress we’re making as every 2 weeks we’ll have something to show them.

**Funding Requirements**

This application will be hosted in Azure cloud app service using Azure database and security. There’s no hardware cost associated with the application. Using Azure’s pricing calculator, it’ll be around $450 a month for hosting the application, database, and security. We have a team of 6 including a product manager, business analyst, a lead developer, and 3 other developers. The estimated time for completing the project to be around 300 hours totaling to around $180,000 with an estimate that each team member makes $100 an hour. Using open-source software for our development environments, and programming languages there is no cost associated with any of the software that we use to develop. We estimate that there will need to be monthly maintenance to fix bugs and implement new features that will cost around $400.

**Stakeholders Impact**

This application is being built using the Scrum methodology. Using this method has constant communication with stakeholders from the customer and the business. Both will have to be in full agreement to make this project successful. Stakeholders will get to see and use a new implementation at least every two weeks while the application is in production. This way, they will be able to immediately see progress and find any bugs or new ideas to add into the development process. The stakeholders must agree that this application will need to be able to generate a profit for both companies. We at WhirlWind Analytics estimate that this application will generate around $1 million of new income once it’s been rolled out to our customers. We anticipate that our customers would like to have this software as a service which will then generate an additional $5 million in income.

**Data Precautions**

The current data that is used is stored in the Azure database in the cloud. It’s only accessible to WhirlWind Analytic employees with specific access. We get our data from public sources and clean it and upload it to our database which is password protected with two factor authentication. With our SaaS product, the customers will have to use their own database solutions. Password information for this SaaS product will be stored using verified hashing.

**Developer Expertise**

Our team will be comprised of a product manager who has over 12 years of experience in this field and will be our subject matter expert, a business analyst who has 8 years of experience. The development team all have over 2 years of experience with the team lead having 10. All team members have built products before, and all have a wealth of knowledge and experience of building different systems. Me having a bachelor’s of science in Computer Science and over 3 years of professional development experience will be working with this team to get this project completed.

**Prompt B**

**Project Proposal**

**Problem Statement**

Currently, tornadoes are getting more common with the changing weather in the world. There is currently no way to predict tornadoes and where they will be, how long they will last, and how much damage they can do to a specific state. Emergency services could take hours or days to respond to a tornado disaster. Accurate predictions of tornadoes could solve these issues and help save more people from being stranded or help them be more prepared.

**Client Summary**

This application is intended to be used by nation weather forecasters, Insurance companies, emergency services, and governments. Its purpose is to reinforce assurance of a disaster happening. It’s not a matter of if, it’s a matter of when. This application will present the users with accurate information about when a tornado is expected and for how long it will last. It will give estimates of how many people will be injured and the fatalities. It will also estimate the magnitude and the severity of property loss.

**Existing Systems Analysis**

The client currently has their own datasets which are used by data scientists to clean and make weather forecasts. They do a wide array of weather forecasts and tornadoes are a small part of what they forecast. They do well with this, but they cannot make accurate forecasts this way as tornadoes isn’t their focus. Using our services, it’ll free up the data scientists to focus on other weather patterns to make it more accurate. We take care of all tornadoes, while they focus on other weather-related tasks. We will do all the heavy lifting by even providing current data. And if the clients want to upload their own data, they can do that as well. This way, someone not from the analyst team can run their own reports.

**Data**

Currently, all our data is stored in an Azure database. We upload them via csv spreadsheet into the data warehouse that is provided by various government weather agencies for each state. We then sort this data into their own tables by state. We then query this data through our Go server via REST APIs. For any incomplete data we dispose of as we want the most accurate predictions available.

**Project Methodology**

At WhirlWind Analytics all our teams use Agile methodology. We found that our team works best and most efficiently using the Scrum method. Having a lead developer be the scrum master and surrounding that person with product management and business analyst works best for us. That way the product and business team can gather requirements from our customers and stakeholders while the development team only needs to focus on developing and engineering the product. The needs of stakeholders and customers can change at a moment’s notice, whether that be including something new or removing something that is obsolete now. With our sprints being 2 weeks, we able to switch tasks and prioritize easily. Using this methodology, the customers can see the progress we’re making as every 2 weeks we’ll have something to show them.

The first step is to create a backlog - This is the product roadmap and guides the team in creating user stories and requirements. In the backlog, is a list of features that the team must deliver. The Product Manager decides which features will make up the backlog.

The second step is backlog release - This is when the PM and team collaborate to decide which user story will make it into each sprint.

The third step is the sprint work - In a sprint, the team members complete a set of backlog tasks within the 2-week time frame. During this time, the developers build the product features.

The team holds a stand-up meeting daily for 15 minutes. During this meeting they discuss solutions to daily work issues

The fourth step is the burndown chart – Team members use a Jira to create a burndown chart. This shows original time estimates compared to real-time activities, which shows where expectations or team resources need to be adjusted. The PM and BA usually handle this and will be using this information to pass it on to the stakeholders.

The fifth step is testing – During testing, the team demonstrates product functionality for the stakeholders. They give feedback to the scrum team for any needed changes or new features that complement the current product.

The sixth step is Sprint Retrospective and follow up – This is the final phase of the Jira workflow. A sprint retrospective are post-mortems on the previous workflow. This is the stage in which the team questions what they did well and what didn’t go to plan and what changes need to be made for the next sprint.

Deployment of our app in a docker container onto Azure app services to host the website. Using docker and Azure makes the app more accessible and available. We feel it’s the easiest way to update and maintain.

**Project Outcomes**

The finished application will be hosted via Azure and the clint will be provided a URL that the customers can then host themselves over their own network. The webapp will be set up using OAuth2 for security purposes so that the customers are able to log into their own portal. The application will be using our own database with our data provided to them. Customers are able to upload their own datasets to use and run through our algorithm to predict their own data. For documentation purposes, they can visit the documentation tab in the application to get all the documentation for the application.

**Implementation Plan**

The development of the application will start with gathering of the data and how we’re going to utilize each dataset. This will give us a better understanding of how to develop the REST APIs and the front-end of the application. Once the data is set up in our tables correctly, we’ll build out the backend of the application using Go. We chose Go because it’s easy to implement and fast to develop as its standard library has everything that we need. It’s also one of the faster languages out there. It doesn’t have any ML standard libraries, but the standard libraries that it comes with are easy to create our own algorithms to create one. Once the backend is set up, we can start implementing the front-end and the Azure web App Services. We chose to go with Azure as we’re already using it for other applications and will be easy to implement another application. We’re also using the Azure Data Services for our data warehouse. For the front-end we chose the React JavaScript framework as our development team is familiar with it and can get a functional app up quickly. During this time, we’ll be setting up meetings with stakeholders to test the application.

We have set up a Continuous Integrate and Continuous Development pipeline to get the application up and running with the latest changes faster and it will automatically deploy the latest code changes. Once all the development is complete and tested, we’ll be ready to roll out the production environment. Once that environment is implemented, we can then send it out to our customers. There are no physical servers on our end as everything we do is in the cloud and hosted by Microsoft. Once we hand the production environment off to our customers, they can choose to host it locally on a physical server or not. A payment plan will be decided, and we’ll always provide maintenance for the life of the application. We’re able to roll out updates on the fly with little down time with our CI/CD pipelines.

**Evaluation Plan**

Starting from the beginning of development, we are going to include the stakeholders for this project. They will be included in our end of sprint meetings and will be updated daily on the progress of the project by the product manager and business analyst. This will ensure that the stakeholders are engaged in the development process. We will have user testing after every deliverable. This will ensure that no bugs are released into the production release. It will also help stakeholders decide on new features or remove features that aren’t needed any more. The stakeholders will also validate the data to make sure that it’s exactly what is needed.

**Resources and Costs**

No physical hardware is needed for this project besides the company issued laptop that each employee has. We use Azure to host everything and for this project, we anticipate spending about $500 a month for hosting this application. The timeline to get this project complete and production ready is about 300 hours or 2 months. With 6 employees on the project, each costing about $100 an hour, we anticipate spending around $180,000 a month on employee salary.

Since we’re using free and open-source programming languages to develop the application, there are no licensing fees. Go, and ReactJS

We anticipate around $400 monthly for maintenance. This is for changing plans in Azure or fixing bugs in the code. Adding new features at the request of the customer will cost around $4000 a month.

**Timeline and Milestones**

Development through production ready deployment is around 300 hours of labor over the course of 2 months. This schedule is continuous development by our team that is dedicated to working just on this project.

The milestones below will give a specific graph

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Milestone | Pre-Requisites | Activity | Resource Assigned | Hours | Start | End |
| 1 | None | Requirement Approval / Gather Requirements | Product Manager | 20 | 10/01/2023 | 10/03/2023 |
| 2 | 1 | Architecture and Database design | Lead Software Engineer | 30 | 10/03/2023 | 10/07/2023 |
| 3 | 1 | Azure Services | Lead Software Engineer | 15 | 10/07/2023 | 10/08/2023 |
| 4 | 2,3 | Database data generation | Software Engineer | 8 | 10/09/2023 | 10/10/2023 |
| 5 | 4 | Backend Creation (REST API) | Software Engineer | 20 | 10/11/2023 | 10/14/2023 |
| 6 | 5 | Frontend Creation | Software Engineer | 40 | 10/15/2023 | 10/20/2023 |
| 7 | 6 | ML Model creation | Software Engineer | 40 | 10/21/2023 | 10/26/2023 |
| 8 | 7 | Test Application Deployment and configuration | Product Manager / Software Engineer | 30 | 10/27/2023 | 10/31/2023 |
| 9 | 8 | Testing | Software Engineer (QA) | 20 | 11/01/2023 | 11/04/2023 |
| 10 | 9 | Bug fixes | Software Engineer | 20 | 11/05/2023 | 11/08/2023 |
| 11 | 10 | Testing | Software Engineer (QA) | 10 | 11/09/2023 | 11/10/2023 |
| 12 | 11 | User Testing | Stakeholder | 30 | 11/11/2023 | 11/15/2023 |
| 13 | 12 | Production Application Deployment | Product Manager / Software Engineer | 30 | 11/16/2023 | 11/20/2023 |
| 14 | 13 | Application Verification | Software Engineer | 10 | 11/21/2023 | 11/23/2023 |
| 15 | 14 | Customer Training | Product Manager | 20 | 11/24/2023 | 11/27/2023 |
| 16 | 15 | Final Product Delivery | Whole team | 10 | 11/28/2023 | 11/30/2023 |

**Prompt C**

**Application Files**

**URL:** [**chrishayesc964.wafflecoder.net:8080/**](chrishayesc964.wafflecoder.net:8080/)

The main project folder is called C964

The \* indicates a state abbreviation (i.e.. NJ\_Tot.csv)

\C964

go.mod – is created when starting a Go project. Lists the specific versions of the dependencies that the project uses.

go.sum – provides checksums for the exact contents of each dependency at the time it’s added.

main.go – the main server program entry point. Provides the REST API to the front end. Used to run the regression algorithm on the data.

\server

\data

tornados.csv – Has all data for each state

decades.csv – Has all the tornado data for each decade

\stateData

\decadeTotals

\*\_Decades.csv – Has the totals of each decade of data

\totals

\*\_Tot.csv - Has the totals for each state

\*.csv – Has all the data for each state

\pkg

\model

types.go - struct creation (similar to an object)

\regression

regression.go – The ML algorithm used

\web2

Package-lock.json – Created on npm project start

dockerfile – used to create the docker container

run.sh – the docker file scripts to run in the dockerfile on creation

\tornado – where the front end project is held

.gitignore – used to ignore specific files so they wont get added into the GitHub repository

Index.html – html file to render in the browser

\*.config.js – config files on npm project creation

\components – where each of the components are

TableComponent.tsx – A typescript ReactJS file that has all of the GUI components and API calls

\types – the model for the front end objects

ChartData.ts – Typescript file for each chart data object

DropDown.ts – Typescript file for the Dropdown objects

Tornado.ts - Typescript file for the tornado objects

Api.tsx – Typescript ReactJS files to set the URL calls and retrieve the JSON to pass to the objects to display on the frontend

App.tsx – Typescript ReactJS file to render the component file to the main.tsx

App.css – A CSS file to set the styling of the components

Index.css – A CSS file to bring in TailwindCSS package for styling the components

Main.tsx – A typescript ReactJS file to render everything into the browser

Vite-env.d.ts – A config file for the vite tooling

**Prompt D**

**A Business Vision**

We at WhirlWind Analytics went out to solve a future predicting algorithm through machine learning.

**Datasets**

We have over 68,000 rows of raw data starting in the 1950s up to 2021.

A screen shot of a computer screen

Description automatically generated

We use this raw data to process it into smaller datasets by state.

Here’s a sample of the state of Colorado processed data.

A screen shot of a computer screen

Description automatically generated

Which is then sorted down into decades with specific columns needed to run through the regression algorithm

A screenshot of a computer

Description automatically generated

We retrieve the data through various government weather agencies. Then run that data into our data warehouse to be sorted into their own tables.

We use Azure Data Factory along with Azure security suite to secure our database. Only employees with specific security settings may access the data. We use Microsoft two-factor authentication.

**Data Product Code**

We get the raw data via csv spreadsheets and then we can upload that via Azure Data Factory to input that into the data warehouse. We can then run queries in that data warehouse to move the raw data into the specific tables. We then configure our backend server to run SQL queries against the database to select the data to bring it to our REST API endpoints.

A screen shot of a computer code

Description automatically generated

API Endpoints

A computer screen with text and images

Description automatically generated

On our front end, we can call these REST APIs to bring the data from the database to the user interface.

Calling the API from the front end

A screen shot of a computer code

Description automatically generated

Got the dataA screenshot of a computer

Description automatically generated

Queried more specific data for Arkansas.

A screenshot of a calendar

Description automatically generated

And just below the table are some graphs.

A screenshot of a computer

Description automatically generated

**Objective Verification**

**Effective Visualization and Reporting**

**Accuracy Analysis**

**Application Testing**

**Application Files**

The main project folder is called C964

The \* indicates a state abbreviation (i.e.. NJ\_Tot.csv)

\C964

go.mod – is created when starting a Go project. Lists the specific versions of the dependencies that the project uses.

go.sum – provides checksums for the exact contents of each dependency at the time it’s added.

main.go – the main server program entry point. Provides the REST API to the front end. Used to run the regression algorithm on the data.

\server

\data

tornados.csv – Has all data for each state

decades.csv – Has all the tornado data for each decade

\stateData

\decadeTotals

\*\_Decades.csv – Has the totals of each decade of data

\totals

\*\_Tot.csv - Has the totals for each state

\*.csv – Has all the data for each state

\pkg

\model

types.go - struct creation (similar to an object)

\regression

regression.go – The ML algorithm used

\web2

Package-lock.json – Created on npm project start

dockerfile – used to create the docker container

run.sh – the docker file scripts to run in the dockerfile on creation

\tornado – where the front end project is held

.gitignore – used to ignore specific files so they wont get added into the GitHub repository

Index.html – html file to render in the browser

\*.config.js – config files on npm project creation

\components – where each of the components are

TableComponent.tsx – A typescript ReactJS file that has all of the GUI components and API calls

\types – the model for the front end objects

ChartData.ts – Typescript file for each chart data object

DropDown.ts – Typescript file for the Dropdown objects

Tornado.ts - Typescript file for the tornado objects

Api.tsx – Typescript ReactJS files to set the URL calls and retrieve the JSON to pass to the objects to display on the frontend

App.tsx – Typescript ReactJS file to render the component file to the main.tsx

App.css – A CSS file to set the styling of the components

Index.css – A CSS file to bring in TailwindCSS package for styling the components

Main.tsx – A typescript ReactJS file to render everything into the browser

Vite-env.d.ts – A config file for the vite tooling

The files are organized based on whether they are part of the server or the user interface.

The server directory holds everything needed to run the server.

The web2\tornado directory holds everything needed to run the front end.

You’re able to run the server without starting the front end and vice versa.

But to get the whole application to work, you’ll need to run them both at the same time.

Inside the server directory, I have the main.go which is the main file to start everything for the server. I separated the csv files into the data directory to clean up the files. I separate all of the structs(objects) into the pkg\model directory to make it clean. Same with the regression algorithm.

Inside the web2\tornado directory I have the App.tsx which is used to launch the frontend. I also keep the Api.tsx in there to store all the API endpoint calls. There’s also a lot of config files inside this directory that are created automatically when setting up this React project. I didn’t do too much to the config files except change the port number. I separate the objects into the \types directory and separated the components into the \Components directory to keep things organized and clean.

**User Guide**

**Guide for running it from the web browser.**

1. Go to this URL <http://chrishayesc964.wafflecoder.net:8080/>
2. The page will load the starting data into a table.
3. Use the dropdown menu on the top left to search for a specific state.
4. Hit the “Search” button. This will load the data into the table and the graphs.
5. Use your mouse to hover over the graphs to see tooltips about the graphs.
6. Use the “year” slider below the state dropdown menu to slide to a desired year to predict future data.
7. Click on the “Predict Devastation” button to load the data.
8. You can use the slider freely to bring in more years’ worth of data or remove some years. This will happen automatically.

**Guide for running locally.**

Open and extract the .zip file to a directory of choice.

**Using Git** **– if not using git, skip**

You can git clone the repository from GitHub.

If you don’t have git installed you can install it from this link

<https://git-scm.com/downloads>

Once installed and configured. Open your command prompt in windows.

From the windows start menu type in “cmd” and then click on the “Command Prompt”

A screenshot of a computer

Description automatically generated

Once this is opened, and inside the directory you want to hold the files, you can use this command to clone the repository.

“git clone <https://github.com/chrishayes042/c964.git>”

**Node installation – if you have version 16 or higher, skip**

1. Go to this link to install NodeJS <https://nodejs.org/en/download>
2. Once installed, make sure to check the version by going into a terminal or command prompt and run this command “node --version”. You should see something similar to this.

A close up of a computer screen

Description automatically generated

1. Make sure that npm is installed as well. You can check if it’s installed by running this command “npm --version”



**GoLang installation – If you have installed already – skip**

1. Go to this link to download the language <https://go.dev/doc/install>
2. Once installed, go to a terminal or command prompt and type in “go version”. You should see something like this



This command will bring in all of the files needed to run the program locally.

1. Open the folder in your code editor of choice.
2. Open a terminal or command prompt and move into the web2/tornado directory.
3. Run this command to install all of the node packages that the front uses. “npm install”

A computer screen with white text

Description automatically generated

1. Once the npm is done installing type in the command “npm run dev”
2. This will start the front end.
3. Open a new terminal or command prompt without closing out the first one.
4. Move into the c964/server directory.
5. Type in the command “go run main.go”
6. This will start up the backend. You may get a Windows Security Alert pop up. Click on the Allow Access button

A screenshot of a computer error

Description automatically generated

1. Once the server is started open your browser of choice and type in the Address bar <http://localhost:8080>
2. This will bring you to the front-end user interface.

12. You can use this the same way as the URL steps above “**Guide for running it from the web browser.”**

**Summary of Learning Experience**

I’m a professional software developer for about 2 years now. I’ve never used the React Framework before. We use VueJS at work. I wanted to challenge myself to use something different, so I chose React to build the frontend. I’ve used Go before in personal projects for my backend servers to query databases and encode JSON into the REST APIs. I never used it to query through csv files so I thought I’d give it a try and it was a lot easier than I thought it would be. I use Java with the SpringBoot framework at work, but I never had to extract data from csv files before. So, I had the fundamentals down on what needed to be done to complete this project. I was a little worried about choosing Go as my backend as it doesn’t natively have a good ML library like python does. But I did find a good amount of information about regression and how to create one in Go.

For additional learning I went to YouTube and looked at videos of creators on how they set up a React project and how they utilize folders and components. For anything Go related I used Google and ended up at StackOverflow a bunch to figure out what I needed to do to extract the data from csv files and create my regression algorithm.

I’m always trying to learn new things about programming, and I try to use different libraries and languages than what I use at my job. I never used docker before, so I figured I try it out for this project and I found it pretty simple to set up and I’ll be taking this knowledge to my job and try to implement some containers at work. I have to say, letting us students choose our own way to implement this capstone project is an incredible idea.