

Data Visualizer with Data Mining

Spring 2020

Senior Project Capstone

Final Report

Andrew Nashid

Eduardo Granados

Victor Herman

Giovanni Cabrera

1. Topic

The title of our project is Data Visualization with Data Mining. The overview of this project includes using technology to mine, store and visualize data. This type of scenario is prevalent in the world of data analytics. ACM keywords include Database Administration, Web Mining, Web Applications, and Database Design and Models.

1.1 Problem

In today's world, many things are able to be measured and monitored. This can range from the population of a state, to the number of a certain gender enrolled in college, to the average age of CNN readers. This is possible due to the prevalence of computer technology. But data isn't usually collected for the sake of collecting it. It is often analyzed to solve real-world problems, by both public and private organizations. It can be used to answer questions and make predictions on how or how things can shape. One real-world example is UPS using data and data science to optimize package transport from drop-off to delivery. Its latest platform, Network Planning Tools (NPT), incorporates machine-learning and AI to solve logistics puzzles, such as how packages should be rerouted around bad weather (Rice, 2019).

Regardless of the intent, data collection and analytics has established its place in today's society. Many companies of different sizes, industries, intent, etc. use data in their operations. Also, different departments of said companies use data for their own ends. This can be proven upon using any job board website to search and see. Human Resources will use data differently from, say, a marketing team. As a result, there have been many tools and services offered in the marketplace to assist in those operations. There is a demand for technology that assists in analyzing and storage of data. Some popular examples of software packages include. Tableau, Microsoft PowerBI, Salesforce, Crystal Reports. Examples of popular data storage services include Oracle Databases, IBM Db2, Microsoft SQL server, and AWS ("Top BI Tools", 2020).

Another related topic is data as a service. Often, when companies want to conduct studies, they may not have time to gather the data themselves, so the use of REST APIs come into play. This enables a company to write code that talks to another company's server, and this exchange includes that of data, typically, in JSON or XML format. Once that information is

received by the caller it can be saved into a database and handled by a program. This is useful for all parties involved. Customer companies don't have to waste time in conducting their own surveys and polling. They can simply just buy the data. Depending on the quality of the API, the data sent can be very easy to work with, or may call for more work to be done by the customer company.

While many tools out there are very robust and proven, they also tend to come with a big price tag. This could be an issue for non-profits, start-ups, small companies, and others. Nonetheless, these companies can still find use for data, and data analytics. As a result, they may need custom solutions. Rather than paying for ready-made property solutions, software developers may have to make their own version of data analyzer tools. This can consist of using free software languages, and free database engines, for example. Also, there may be cases where a software product solution doesn't exist, and software developer teams have to craft a makeshift solution.

Also, there is the prospect of data brokering. If a company collects data, and they want to distribute that data (either paid or unpaid), they have to make sure to write stable software to facilitate those transactions. Also, if an employee were making a product for his/her company (as highlighted above), s/he would have to understand what s/he was dealing with, and how to call and parse that data. This is where RESTful APIs come into place: allowing a loose set of standards to facilitate such data transactions.

Another related issue is a working front-end. Tech workers often work alongside those without tech skills. That being said, these non-tech employees still need to use the data crucial to operations, companies can't expect a developer to share source code with a business analyst, and have them work it out on their own. If a custom made data-visualizer is to be made, it must have an easy to use front-end.

These are common problems in today's world of data analytics. We seek to simulate those problems and see how we could solve them. Data Analytics and storage is big these days. Many job titles pertain to it. This includes data scientists, business analysts, data engineers, Business Intelligence Analysts and Database Admins, etc (Theuwissen, n.d.). It is important to recreate such scenarios, as many of us may enter into job roles such as that.

Therefore, we used this project to solve an imaginary need at an imaginary company. We roleplay as a software engineering team operating under a budget. We made a Bloomberg terminal-like or Salesforce-like tool of sorts. Our company requires a working tool for their analysts and traders to use with ease. We require free sources of information, regarding stock, oil, foreign currency, and business needs. We require to store that neatly, using proper scheme. A front-end will need to be neatly crafted to call upon that data. Finally, our company that employs use would like to sell any data in our database. Making an API set for our data was vital.

Because we are working under a budget, we are using common open-source languages. We are using tools that are not new to us. This is to avoid having the company spend money by delay. We are trying to solve the issue, overall, of fabrication of data visualization tools.

1.2

Motivation

Our motivation for this project should be touched on. What we all have in common is our interest in database operations, as well as backend development. Combine this with real-world issues, and we are able to use our interests towards problem-solving and employment opportunities. Through our readings, we noticed the prevalence of open-source software and frameworks in the job force, not just for data analytics, but other things. This includes machine learning, for example. This could be attributed to the cost of paying for proprietary software. So it would be cool for us to see how far we could go in our project, using just free software.

When we first linked up, we all discussed what we knew and what we didn't know. We also discussed how the capstone wasn't our only class, and thus time would be of the essence. This helped in deciding upon technologies and topics we had familiarity with, but also gave us the option to grow. This meant understanding backend development, but now learning more about server configuration topics. This also meant, knowing about how to deploy websites locally, and wanting to see how it would work by using professional web hosting. Also, we could try to see how publishing a mobile version of a project would work, with the Google Play store.

We also discussed our desired jobs down the line, they fell in line with what the project was about: Data and Databases, servers, API, web engineering, etc.

1.3

Description

Overall, this system featured a full stack system: front-end, back-end, and database.

Our website features four types of data: business news, stocks, foreign currency rate, and oil rates. You can see data visuals to depict them.

On the back-end, there are APIs that developers could use for their own end, to fetch JSON data from our system. The API, when called properly, would tell the server to fetch the data from a database, and encode it in JSON format (a standard data interchange format). This would ensure that if the data visualizers suit one's needs, that the data can still be retrieved and used for one's ends.

An android app was put in place to view it while mobile, and serves nothing more than a thin client viewer.

1.4

Scope

Like any project, school or work, there are limits and constraints to be taken into consideration. Thus we could expect what our project will do, what it can't do, and what it maybe can do.

The number one issue and constraint for not just us, but society as a whole was this whole COVID debacle. When our classes became online, we were deprived of the immediate face to face interaction among our classmates. Thus we had to use online collaboration to ensure communication. This included the use of websites like Google Hangouts and Discord. As a result, we were limited in our interactions.

We also had the constraint of time as well. We couldn't make the project too small, yet we only had one semester to get this thing where wanted. We had to make sure our project was robust, but not over the top. Thus that meant we probably wouldn't include account management, or API key authorization. We also had to expedite some parts of the development, and use tools in our Android App creation. Since the App was nothing more than a glorified

mobile viewer, we didn't need to reinvent that wheel, and use MIT App Inventor to help us on that end.

We also had to run our project on a monetary budget. One common issue in our project was the selection of free/cheap, robust APIs. Many APIs cost money, and we had to do some digging to find free, cheap APIs. We also had to make sure our selected APIs gave us what we needed, as some had harder to work with features (e.g. calling one day a time, instead of a year). One of our APIs was used for making calls about oil, and since that cost money, our data may not reflect immediately up to date (although it went back to late 80s).

Finally, we had to make sure we could balance the demands of this class, along with the demands of our other classes.

2. Team and Tasks

Andrew teed off the whole project, with mentioning his prior project experience in RESTful APIs. He made the suggestions of finding real-world data, and visualizing. He oversaw the coding of PHP and MySQL in establishing the back-end. He drove the initiative of making several programs to call several decades worth of data in oil prices, currency, news, etc. He worked hard to make sure everything was extracted from his API calls, and stored appropriately into the databases. He was really our backend architect, and web-master.

Eduardo took reigns of the app development, and oversight of using MIT App Inventor. He took a good amount of time to understand the tool, and how downloading and setting it up on Google Play would work. He also reviewed the server-side code, and database scheme, to ensure nothing was afoul. Eduardo provided excellent feedback on selection of API choice, and alternatives, in case we needed to switch. Having a solid foundational experience in the full-stack, he floated to other duties and consultation roles, as well.

Victor used his experience in several JavaScript libraries to help make the front end, and visualize the data. Since JS just needs a browser engine, no extensive configuration was required. Victor also assisted in choosing viable web-hosting options, and finding ones that streamlined and integrated well with our XAMPP stack. This would mean choosing GoDaddy, over

alternatives, such as AWS and GoDaddy. Out of the entire group, Victor handled the bulk of our coding efforts, and really carried the visualization. He provided much needed leadership and wisdom on how to achieve our project goals, in a timely manner.

Giovanni helped in the coding efforts with Victor, and leaned on his past classes in databases and data visualization to finish any task delegated to him. He also played a great role in getting Andrew up to speed on what front-end tools would be needed. He also used his knowledge of design tools, to help in making cool logos, should the need have arisen.

3. Project Breakdown

3.1 Architecture

Our project uses a multitude of languages and tools. It's hard to find the right place to start. That being said, we will randomly begin with using PHP. PHP was the language we used for our server side. It stands for "PHP: Hypertext Preprocessor". It is a popular general-purpose scripting language. We used it to make our own APIs with, as well as using it to call other APIs, and storing the calls. We also used the language to make scripts that would be called by Linux Cron jobs, daily. These scripts would make daily API calls to keep our database up to date.

Apache HTTP Web Server was the server on which the code was stored. It is a free and open-source language. It allowed for easy storage of files, and configuration. It is also a popular and common web server, and one we had experience from prior coursework. It was originally released in 1995 and has maintained popularity since then.

We used MariaDB. It is a very popular and free open-source relational Database. MySQL was the query language used for our database queries. MariaDB is a fork of the MySQL RDBMS. It was put in place, after Oracle acquired it, and allowed a free and open-source option to developers (Pearce, 2013). Very often used together with Linux. It was initially released back in 2009. When it came to using XAMPP and cPanel, phpmyadmin was the tool of our database administration. Another similar example would be MySQL workbench. While one can use the command line to administer the database, we decided to use the GUI to help create users, databases, tables, and columns. We also can and have used it to create users and permissions. It also included the features of import and export files, to help share SQL data. What we also like

about it was that every time you achieved a command through the GUI, it would show the equivalent command for the command line or SQL command, or PHP code line.

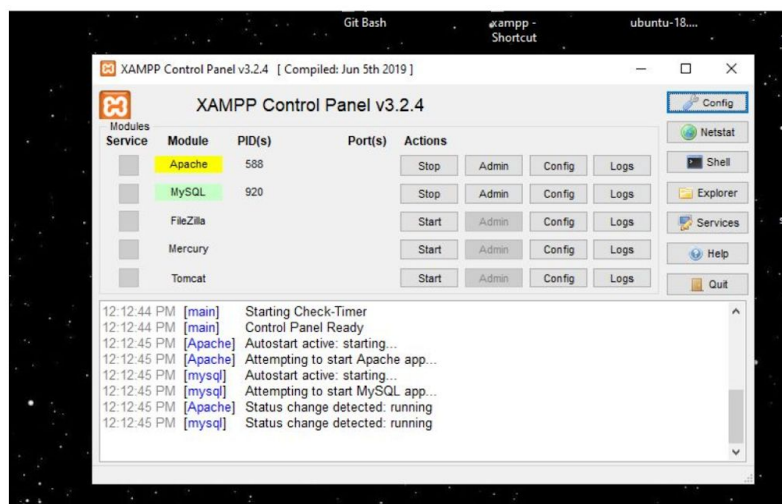
The front-end consists of many technologies. For starters, we used HTML. HTML is how we would present text. We also used CSS to help in the look and feel of the website. When it came to JavaScript, we used a multitude of JavaScript Libraries. JQuery was used, along with AJAX for html document traversal and manipulation. AJAX is used for exchanging data with a server and can update a portion of the page, without reloading the whole thing. Loader.js was used to load data in JSON form and make graphs with it.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is very easy for humans and machines to parse. Our code was to convert any SQL results retrieved, into JSON. This would make it easy for other developers to work with, and is similar to how data is brokered in real-world for-profit API plans. We also sought out APIs that dealt with JSON, as opposed to XML. This was as JSON was more pleasant to work with.

XAMPP consolidated many tools into one. It is a FOSS and multi-platform web server solution stack package developed by Apache Friends. It includes Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl languages. Having this multi-tool made it easy for us to incorporate those technologies into one.

Working with XAMPP

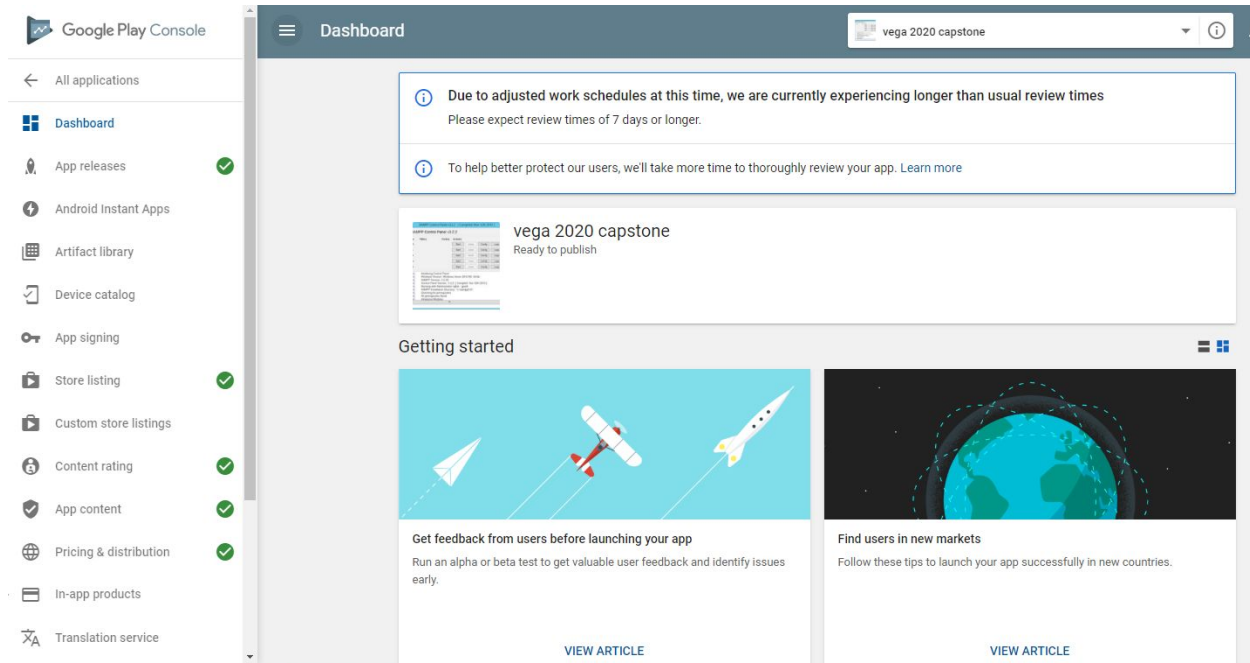
- Allows us to turn our computer into server (localhost), and run server side operations.
- Same for database events.
- Open Control Panel ShortCut
- Click start on Apache and MySQL parts



We use Android App Inventor in helping facilitate our mobile web viewer. Android App Inventor is a free tool provided by MIT to assist in creation of making apps. It works by using drag and drop features to design the layout, while it builds the code for you in the background. Since we were making a simple mobile viewer, it allowed us to simply accelerate this process, while focusing on the publishing process.

Snippet of MIT App Inventor

We used Google Play Console to upload the android APK code, to publish the App. It was cool to learn about how the App Publishing process worked, and how it was deployed to different world markets (content rating, taxes, controlling which countries could see, countries where apps were charged to the phone bill, and not credit card, etc). It came with a bunch of useful features like automated publishing, user acquisition, financial reports, development tools, etc.



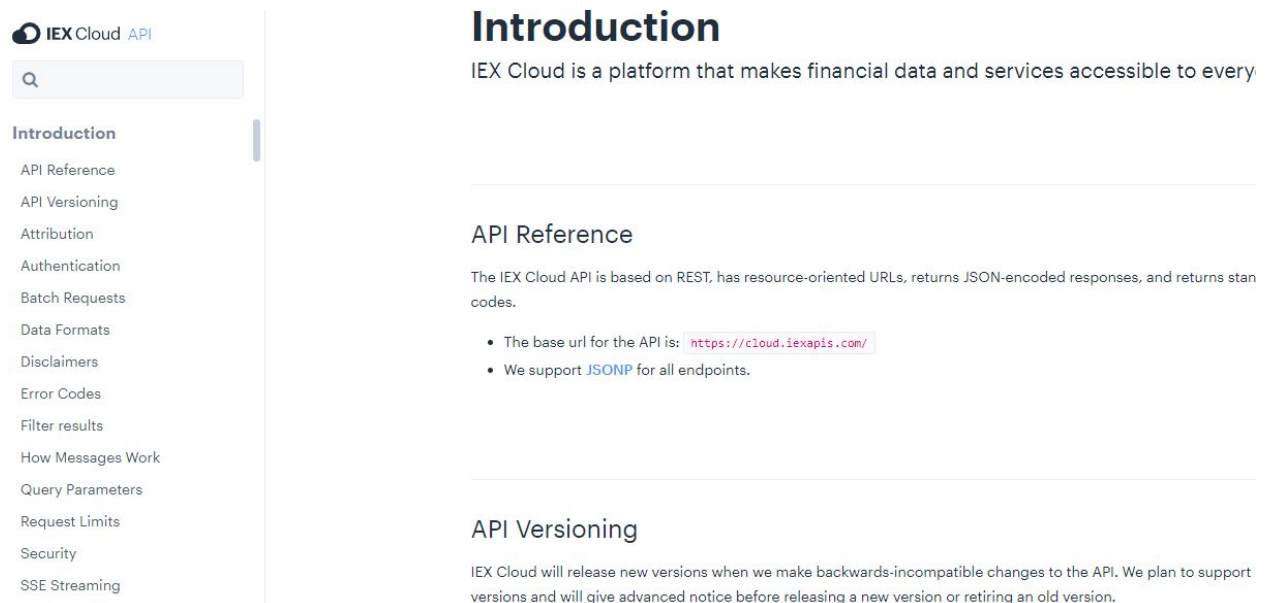
View of Google Play Console

For web hosting we and buying a domain name, we chose to buy from GoDaddy.com. We examined multiple options before deciding upon GoDaddy, and we found GoDaddy to be robust, and friendly for XAMPP developers. With AWS (which Andrew was taking a class in), it could've involved a bit too much time in understanding the ins and outs of it. This was due to the excessive configuration options and features, and for web-hosting, we wanted something that was more plug-and-play. With the use of GoDaddy, it allowed us the use of cPanel. cPanel is common in web hosting solutions, as it makes things easy. This includes easy downloads of logs, firewalls and security, monitoring the network, etc.

When it came to bring the site to life, we would import the code to the cPanel file manager, and the changes would reflect immediately.

cPanel in Godaddy.

We chose our stock API to be from IEX cloud API. Another API choice was the use of this oil one, which came from oilpricepi.com. This one cost a fee, and we only used it temporarily. We were still able to extract decades worth of data from it. Our API for news came from newsapi.org. Our FOREX currency one came from exchangeratesapi.io. We got many years of data from it, but it came out of order, so we had to do some additional coding to fix that.



IEX Cloud API

Search

- Introduction
- API Reference
- API Versioning
- Attribution
- Authentication
- Batch Requests
- Data Formats
- Disclaimers
- Error Codes
- Filter results
- How Messages Work
- Query Parameters
- Request Limits
- Security
- SSE Streaming

Introduction

IEX Cloud is a platform that makes financial data and services accessible to every

API Reference

The IEX Cloud API is based on REST, has resource-oriented URLs, returns JSON-encoded responses, and returns stan codes.

- The base url for the API is: <https://cloud.iexapis.com/>
- We support [JSONP](#) for all endpoints.

API Versioning

IEX Cloud will release new versions when we make backwards-incompatible changes to the API. We plan to support versions and will give advanced notice before releasing a new version or retiring an old version.

Documentation page of IEX Cloud API

An oil price API that is simple, efficient and up to date.

OilpriceAPI is a JSON REST API that provides live* oil prices and historical oil price data in a clean and simple way.

Get your API key

```
curl https://api.oilpriceapi.com/v1/prices/latest \
-H 'Authorization: Token YOUR_API_TOKEN' \
-H 'Content-Type: application/json'

{
  "status": "success",
  "data": {
    "price": 26.65,
    "formatted": "$26.65",
    "currency": "USD",
    "code": "BRENT_CRUDE_USD",
    "type": "spot_price",
    "created_at": "2020-04-20T15:25:01.009Z"
  }
}
```

750K+
DATA POINTS

1-5 min
LIVE PRICE DELAY*

100K+
REQUESTS HANDLED / MONTH

Source of our Oil API

We had the Postman tool to verify our API calls. Postman is a very useful tool, as it helps in testing APIs created or used by software engineers. We used Discord for communication. We used all the listed documentation sources from all our languages and APIs. But in the event if that didn't help, we would use sites like StackOverflow to help with any troubleshooting. For our text editors, we used Sublime Text 3, and Microsoft Visual Studio Code.


apis.com/v1/stock/rfem/logo?token=pk_8172b2d33abb4ad1ab9e57a9391dcf01...

Send Save

Headers (7) Body Pre-request Script Tests Settings Cookies Code

VALUE	DESCRIPTION	*** Bulk Edit
pk_8172b2d33abb4ad1ab9e57a9391dcf01...		
Value	Description	

Test Results Status: 200 OK Time: 653ms Size: 719 B Save Response

Serialize BETA JSON 

Save as example
Save to a file

age.googleapis.com/iexcloud-h137opg/api/logos/RFEH.png"

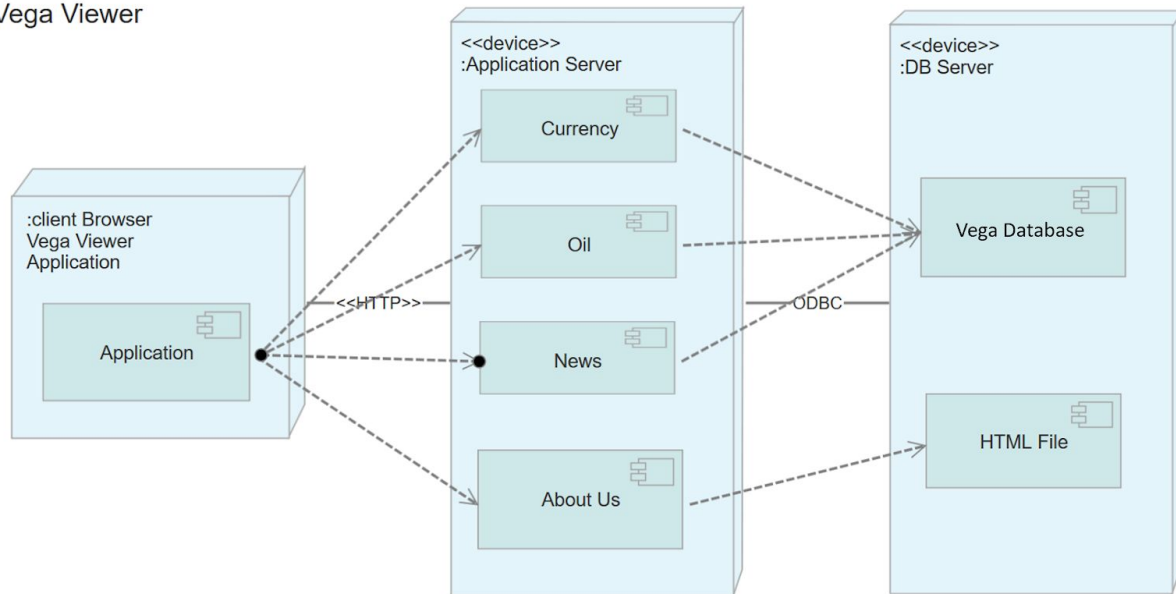
3.2 Data Analysis

- Use Case: Using Vega Site
- Overview: Whenever a User accesses the Vega site, and click one of the tabs, they're taken to current graphs and charts based on what they chose.
- Actors: User, System
- Main Scenario:
 - a. This begins when a user accesses our site
 - b. Then whenever the user clicks one of the tabs, they're taken to point on the site
 - c. That point on the site is a graph or chart with current stats

- d. Or they can go to the site that gives the details about the page creator

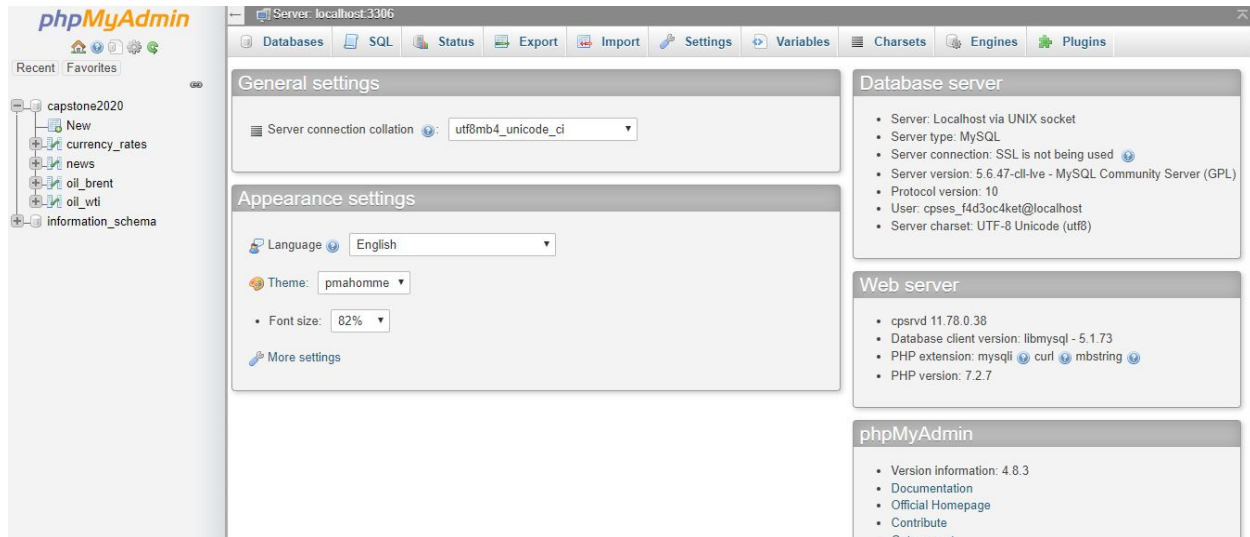
UML Diagram:

Vega Viewer

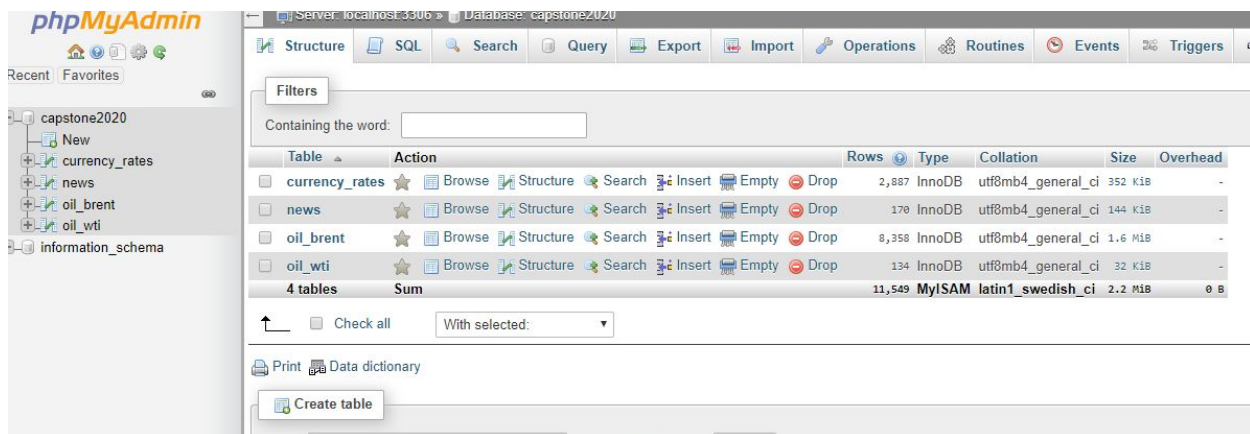


- When the user reaches the site, they're accessing it from some form of web browser.
- To get to our site, a http connection is needed.
- Then once on the site you're shown the main page with tabs to the section you want to view.
- Through an open database connection we can see where the data is being pulled from.

3.3 Supporting Images



We used phpmyadmin to administer our online automated database.



Our tables used in our database. Note thousands of rows.


```

1  <?php
2
3  require('mysqli_connect.php'); // Connect to the db.
4
5  //Documentation link
6  //www.https://exchangeratesapi.io/
7
8  //base currency is USD for comparison
9
10 //API call for getting many years of data.
11 $API_call =
12 'https://api.exchangeratesapi.io/history?start_at=2009-01-01&end_at=2020-02-10&symbols=AUD,CHF,CNY,INR,JPY,RUB,S
13 //can change start here^^^^
14 //can make this call go back over 10 years!!!
15
16
17 //API Call for daily update, once initial fill has been done
18 //API_call = 'https://api.exchangeratesapi.io/latest?base=USD&symbols=AUD,CHF,CNY,INR,JPY,RUB,SEK,DKK,SGD,CAD,E
19
20 //=====
21
22 ?>
23
24 <?php
25
26
27 $curl = curl_init();
28 curl_setopt_array($curl,
29 [
30     CURLOPT_RETURNTRANSFER => 1,

```

Snapshot of PHP code used to fill our database with currency values.

```

25
26 $curl = curl_init();
27 curl_setopt_array($curl,
28 [
29     CURLOPT_RETURNTRANSFER => 1,
30     CURLOPT_URL => $API_call
31 ]);
32
33 $response = curl_exec($curl);
34 curl_close($curl);
35
36 $response1 = json_decode($response, true);
37
38 foreach($response1['articles'] as $article)
39 {
40
41     $id = $article['source']['id'];
42     $id = mysqli_real_escape_string($dbc,trim($id));
43
44     $name = $article['source']['name'];
45     $name = mysqli_real_escape_string($dbc,trim($name));
46
47
48
49
50

```

Snapshot of PHP code that would help with daily updates of news articles.

```

<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
  google.charts.load('current', {'packages':['table']});
  google.charts.setOnLoadCallback(drawVisualization);

```

JavaScript library that enables the use of Google Charts. Also loads the selected package, which in this case is 'table'.

```

function drawVisualization() {
    var data = new google.visualization.DataTable();
    data.addColumn('string', 'Name');
    data.addColumn('string', 'URL');
    data.addColumn('string', 'Publishing Date');
    data.addRows([
    <?php
        include "dbconfig.php";
        $con=mysqli_connect($server,$login,$password,$dbname);
        $query1= mysqli_query($con, "select * from news");

        if(mysqli_num_rows($query1)>0){
            while($row = mysqli_fetch_array($query1)){
                echo "[".$row['name'].",".$row['url'].",".$row['publishedAt']."]", ";
            }
        }
    ?>
    ]);
    var chart = new google.visualization.Table(document.getElementById('chart'));
    chart.draw(data, {width: '100%', height: '100%'});
}
</script>

```

JavaScript code that includes php to both load a Google Table as well as pull data from the database.

Javascript functions named “splitToChunks” and “properForm” have a very important role in formatting the data before it can be loaded into a Google chart. “splitToChunks” takes in the data in Json format and splits it into equal portions. These portions are then inserted into a multidimensional array because the format needed requires each entry to be in an array. “properForm” could be considered as the most vital function to properly feed the graph. This

function takes the rows of arrays created by “splitToChunks” and formats the values so that the date is a string and the values are integers.

```
function drawChart() {
    var jsonData = $.ajax({
        url: "getData.php",
        dataType: "json",
        async: false
    }).responseText;
    var x = JSON.parse(jsonData);

    console.log(x);

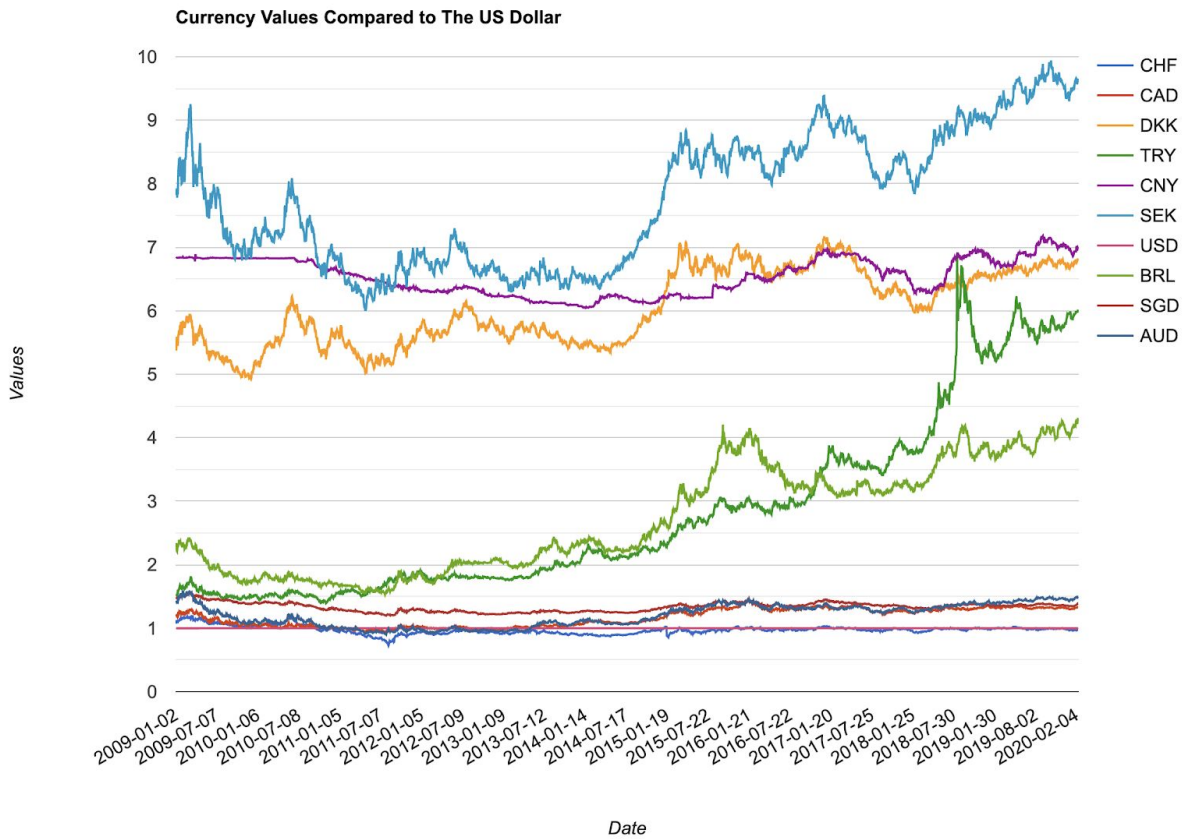
    Test12 = properForm(x);
    console.log(Test12);

    var data = new google.visualization.arrayToDataTable(Test12);

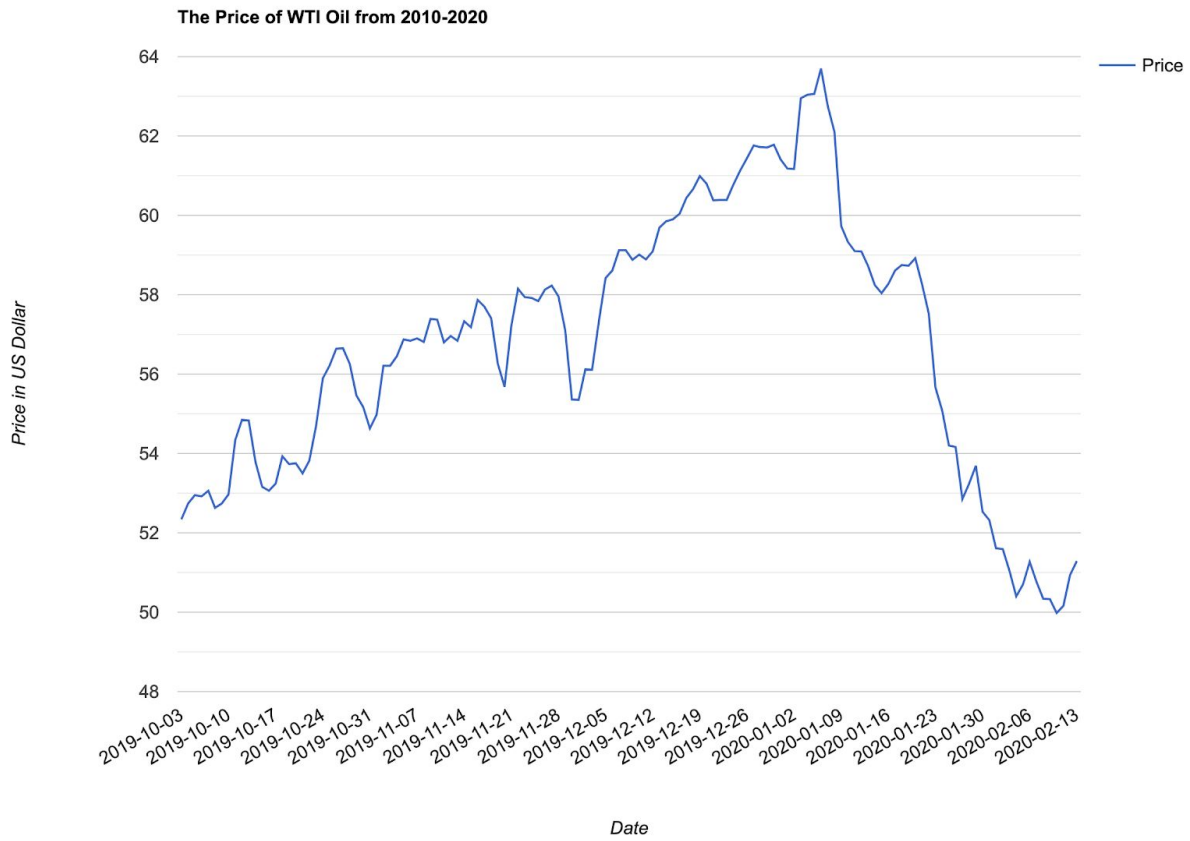
    var options = {
        title : 'Currency Values Compared to The US Dollar',
        chartArea: {width: 9000},
        chartArea: {margin:{left:5}},
        selectionMode: 'multiple',
        zoomable: true,
        pannable: true,
        vAxis: {title: 'Values'},
        hAxis: {title: 'Date'},
        series: {5: {type: 'line'}}
    };

    var chart = new google.visualization.ComboChart(document.getElementById('chart'));
    chart.draw(data, options);
}
```

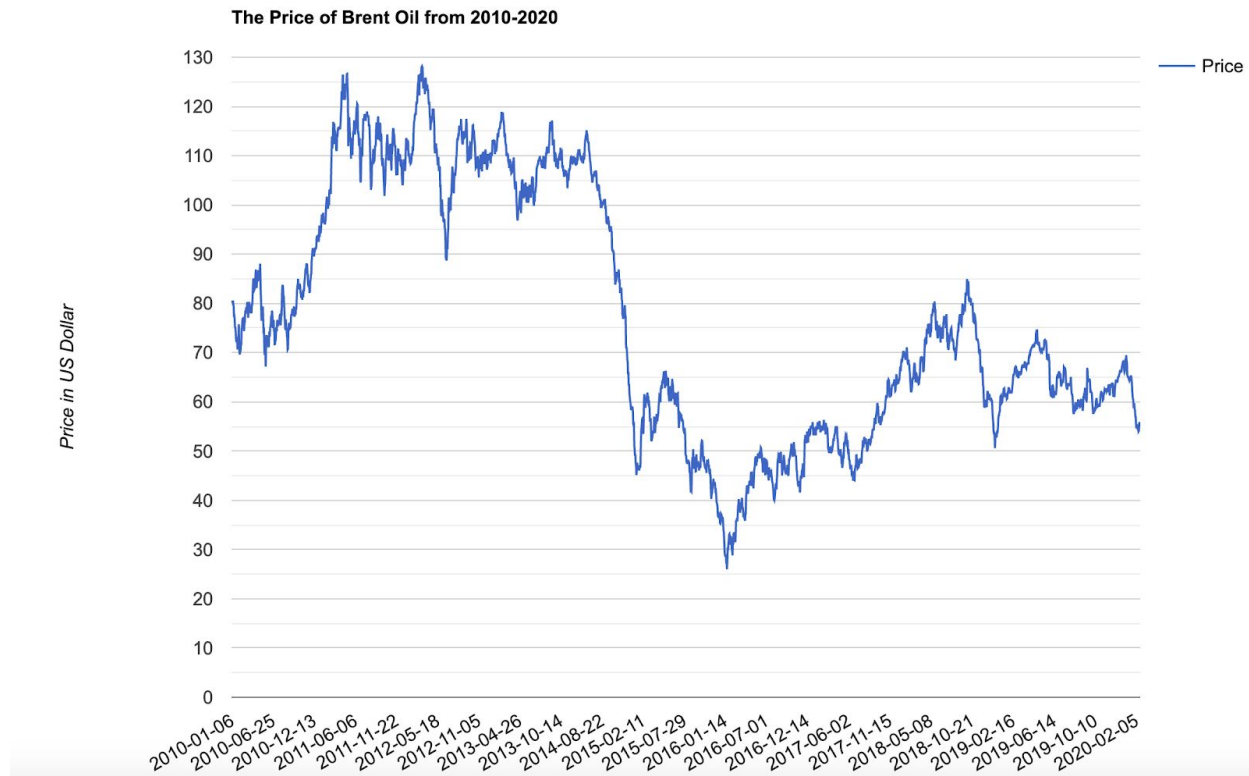
Javascript code which uses ajax to read in Json data and fed to the function “properform” in order to properly format the data which is loaded into the Google chart. Various options are used to allow the user to select points on the graph, zoom in on the graph, and adjust the size of the graph.



The graph produced from currency values which include ten currency values compared to the US dollar. This graph is interactive allowing the user to place points on a line which will show the date and value of that point. The graph is also zoomable which allows users to zoom in as close as they like to see the differences in values.



This graph is produced from the WTI oil dataset which covers over a year of data. This type of data visualization shows the user how the value of the oil spiked in January of 2020 and quickly dropped by February 2020.



This graph is produced for the Brent oil dataset which covers dates from 2010 to 2020. This dataset had thousands of values which is clearly shown in the sharp edges of the line graph.

Challenges & Takeaway

We started out keeping realistic expectations and predictions of challenges to overcome. We thought long and deliberated before any undertakings. This was to avoid any undesired road bumps. While any additional feature nonimposed wouldn't be too hard to implementate, we were running under a strict schedule.

For starters, the use of making API keys would have to be put to the side. To give an overview of what that is all about, it means that having special values to help monitor the use of API abuse. In real-world scenarios, there are bot programs that can be set up to abuse free APIs by making excessive calls. Having API keys protects against exceeding atolled limits.

By proxy, that would mean it wouldn't adminser accounts. So no account creation protocol or safeguard was set into place. Setting that up, along with authentication could've added burden to our project. We also had to frontload work in finding robust cheap/free APIs. In

some cases, our data may have not been entirely up to date (although very close), since that would've exceeded budget.

In terms of App creation, creating an app from scratch could've been feasible, but wouldn't be necessary. Our App would just serve as a simple mobile client viewer to what appeared on the desktop browser. We weren't intending to make a custom, unique UI experience when it came for that.

Also, what we did was not out of the ordinary, and gave us a chance to practice what businesses do, which is use simple App Maker tools. We used MIT App Inventor, which is free. But other popular alternatives exist, such as AppyPie and buildFire.

We gave great thought towards if we were going to apply any Machine Learning or not. We have decided that it would take perhaps a semester of its own to get up to speed on that, and set up the frameworks and server to handle that. Also, the data would also have to be formatted more specifically. So we decided to drop that notion, but we did learn more about machine learning. So there is that.

4. Future

We are very proud of the efforts we accomplished. We worked hard and tirelessly towards this project. This project allowed for us to build upon our knowledge from the classroom, and perhaps touch upon things not typically taught in the classroom. We also got to experience the business side of things, when researching on how to pick a domain name, and web hosting providers.

Not only did we develop our technical skills, but we developed our soft skills. We, overall, become better communicators, and developed our patience and open-mindedness, with one another.

That being all said, we know that our project had ways to improve upon itself. For starters, we could've made it that users would have to make an account to interact with the website. That would allow us to practice REGEX for password creation. We could've also practiced database encryption means, had we done that.

We could have also included keys for APIs. This would be to safeguard against any possible case of bot abuse, as well as prove a means of profit, should we have wanted to sell the products services. Monthly quotas could have been put in place, had we included keys.

If we had more time, we could have learned about Kotlin language, and built the app from scratch. This would have given us more control of how our product would be rendered on mobile devices. We could've also gained more fluency in Mobile Development.

5. Additional Notes

Doing this project had us looking up more nuanced terms and features of the technology we dealt with. Even if we didn't use all the things we looked up, we still had to have a core understanding of it, in order to contemplate adding it or not. We also feel that if students include bullet point lists (like the one below) it serves as more validation of the capstone journey, especially if the end product seems a bit underwhelming. Below is a copy/paste of a form we compiled during our journey.

What we learned/Need to learn

This sheet will highlight things we learned and things we need to learn, in order to contribute to this Capstone Project. we suggest everyone make their own version of this sheet.

Some of these may be ideas. Some may be links to relevant sources of information.

Some may be links to tools, or software we used. Some may be problems encountered. Some served as general knowledge, but perhaps not directly applied.

What we sought to learn in this was not just software development, but deployment and administration. And using vendor tools to achieve that. For example, we felt good enough about building dynamic websites, so we wanted to get better in things like domain name selection,

cloud web hosting, and using the tools of the provider (like cPanel for goDaddy or BeanStalk for Amazon Web Services).

Another example was mobile development. We understood how it worked from a technical standpoint, but wanted to really learn how App Publishing worked. All while learning even more about the coding aspect. (External links in blue, and we originally used font size 14 for this)

- [Postman API tool - tutorial](#)
- [JSON vs JSONP](#)
- [Fetching HTTP headers current request in PHP](#)
- [finding Errors with PHP cURL](#)
- [Pagination of API results](#)
- Handling nested JSON arrays with PHP
- [VMware interfering with XAMPP](#)
- [JS libraries that can help with data visualization](#)
- [AWS calculator to help business estimate cloud costs](#)
- [List and descriptions of all AWS services](#)
- [Another neat AWS link](#)
- [API gateway with PHP and Lumen](#)
- [REST with PHP](#)
- [cURL vs Postman](#)
- [Idempotent \(Can be used in Math or CS\)](#)
- [some API terms](#)
- [more API terms](#)
- [storage Engines](#)
- [algorithms and storage engines](#)
- [info on embedded database engines](#)
- [examples of embedded database](#)
- [in-memory databases](#)

- [list of in-memory databases](#)
- API documentation providing embedded code samples for multiple languages
- Unique made libraries for multiple languages dedicated to a particular API. This would be featured on github.
- [PHP configuration file](#)
- [Procedural vs Object Orientated PHP](#)
- [PDO vs MySQLi for databases](#)
- [Oil : Brent vs WTI \(used for one our APIs\)](#)
- Learning ACM categorizing system (keywords)
- [CORS](#)
- Servers could possibly deny XAMPP due to it not being a browser (user agents)
- [HATEOAS](#)
- [CGI \(common gateway interface\)](#)
- [time series database](#)
- [XAMPP vs cPanel](#)
- [some popular hosting control panels](#)
- [OAuth vs OpenID](#)
- [Using Discord Chat](#)
- [Discord endorsing third party bots](#)
- [MIT App Inventor](#)
- [BlueStacks Android Emulator](#)
- [Android WebView](#)
- [Ways to Publish Android App](#)
- [Web-hosting benefits](#)
- [Some great web-hosting option](#)
- [Buying a domain name](#)
- [Android APK basics](#)
- [Popular source to download APK to your desktop](#)
- [CNAME](#)

- [Dedicated IP address - benefits with web hosting](#)
- [Domain Hijacking](#)
- [Domain Lock](#)
- [DNS hosting](#)
- [DNS template](#)
- [Linux Cron and cPanel](#)
- [using different default than index.html files](#)
- [.htaccess cheat sheet](#)
- [.htaccess tutorial](#)
- [enable directory listings for a folder on my website](#)
- [Tool to make favicons \(icons on browser tab\)](#)
- [Delegating \(sharing\) your GoDaddy account with other developers](#)
- [h2c client](#)
- [DOM](#)
- [API security and hacking prevention](#)
- [API security testing](#)
- [To Do list before publishing](#)
- [Monkey and Gorilla Testing](#)
- [guide to Google Play console](#)
- Tools like Tableau and Microsoft PowerBI do not need coding.
- [what is business intelligence](#)
- [what is Tableau](#)
- [Power BI vs Tableau](#)
- [SQL server features/components](#)

Group Timeline

This portion of the report will feature a timeline of our group's activities. It goes through the weeks, giving an overview of the accomplishments of the time, the plans set out during that week, the problem areas, and links relevant to the week. This portion was brought from our cumulative weekly status reports.

- Week 1
 - Accomplishments
 - Formed groups. Got acquainted with one another.
 - Discussed our skill sets.
 - Was orientated to what the class would entail.
 - Envisioned our project; discussed what would entail.
 - Made a shared google folder, and Discord chat room.
 - Plans
 - Come to class prepared; Have suggestions, and offer feedback.
 - Agree upon languages, and project ideas.
 - Delegate tasks to one another
 - Problem Areas
 - Will the technologies we know be enough to achieve our goals?
 - If we need to learn more technologies, can we afford the time?
 - Managing this class, along with our other classes.
 - Links pertaining
 - <https://www.sitepoint.com/4-best-chart-generation-options-php-components/>
 - <https://apilist.fun/>
- Week 2
 - Accomplishments
 - Discussed APIs to use
 - Discussed who would write what pieces of software
 - Talked about how PHP and data visualization would come into play

- Plans
 - Have UI models of website
 - Have basic code templates figured out
 - Decide what types of API sources to use, and what APIs to make.
 - Narrow down choices of web hosting, and domain names
 - Planning on meeting up.
- Problem Areas
 - Making our data visualization as strong as possible.
 - Making our UI as solid as possible.
 - Everyone finds the time to correspond, be it on discord chat, or in person.
 - Finding APIs in our budget
 - Campus internet tends to have some issues.
- Links pertaining
 - <https://openexchangerates.org/>
 - <https://fixer.io/documentation>
- Week 3
 - Accomplishments
 - Created PHP programs to mine data from several APIs.
 - Configured database to neatly store this information.
 - Discussed the selection of another API source.
 - Plans
 - Having developed another PHP file to call our JSON.
 - Started out coding for the interactive front end.
 - Have a fully established, filled database.
 - Problem Areas
 - Finding APIs that fit our budget, and are flexible with calls.
 - Seeing if our schedule permits time for creating our own RESTful APIs.
 - Seeing what ways are feasible for visualizing our data.

- Links pertaining
 - <https://restfulapi.net/rest-architectural-constraints/>
 - <https://www.geeksforgeeks.org/rest-api-architectural-constraints/> ,
 - <https://scottbarstow.com/apis-how-to-read-and-understand-documentation/>
- Week 6
 - Accomplishments
 - Met during our off days, to advance project goals
 - Achieved and learned about Web-hosting
 - Successfully deployed full web-site, to practice for the real one.
 - Plans
 - Finish developing an Android App webviewer
 - Learn about how to deploy our App to the Google Play store.
 - Conceptualize a strong UI.
 - Problem Areas
 - How can we best visualize our data?
 - How effective, with our time constraints, can our UI be?
 - Should we seek out certain JS libraries, or are the ones we are dealing with, effective enough.
 - Links pertaining
 - <https://www.google.com/search?q=mit+app+inventor&aq=chrome.0.0j69i57j0l6.2065j1j7&sourceid=chrome&ie=UTF-8>
 - <https://android.jlelse.eu/publishing-an-android-app-da3502c652af>
 - <https://www.ready4s.com/blog/7-things-you-should-do-before-publishing-your-app>
- Week 7
 - Accomplishments

- Continued our app development side project
 - Continued in correspondence with one another.
 - Developed some APIs of our own.
- Plans
 - Have our data visualization semi-functioning.
 - Have our app on the google play market.
 - Develop our UI front end.
- Problem Areas
 - Finding time to meet.
 - Budgeting time for projects, along other classes.
 - Learn the google play console.
- Links pertaining
 - <https://developer.android.com/distribute/console>
- Week 8
 - Accomplishments
 - Presented our progress
 - Getting a start on the data visualization
 - Getting closer to publishing our app
 - Plans
 - Continue in publishing the app, if not done already.
 - Continue the data visualization coding.
 - Problem Areas
 - Finding the time to meet.
 - Having others learn new things, to keep the pace up.
 - Seeing if we have time to enrich our features
 - Links pertaining
 - <https://en.wikipedia.org/wiki/D3.js>

- Week 9
 - Accomplishments
 - Since we are ahead of schedule, I used the week to decompress a little.
Rested my brain a bit.
 - Published our app.
 - Learned how the google publishing process works.
 - Plans
 - Adjusting for our courses to be online, and getting acclimated for that.
 - Keep coding.
 - Keep constant communication online.
 - Problem Areas
 - Finding places to meet, due to campus shutdown.
 - Dealing with current affairs, and not letting it impede our progress.
 - Links pertaining
 - None this week.
- Week 10
 - Accomplishments
 - Got acclimated to online learning environment
 - Shared our progress with Professor.
 - Plans
 - Try to find a time to meet.
 - Get used to all our classes being online
 - See how the front end is coming along
 - Monitoring the publishing of our android app.
 - Problem Areas
 - Making our front-end pretty, and not merely functional.
 - Not petering out. We started strong, and must not stall.

- Links pertaining
 - None for this week.

- Week 11
 - Accomplishments
 - Learned more about Blackboard Collab
 - Learned how to share my screen to others on Collab.
 - Worked more on the project
 - Plans
 - To continue our pace on the project.
 - Problem Areas
 - No issues encountered during this week.
 - Links pertaining
 - None.

- Week 12
 - Accomplishments
 - Peer review each others code
 - Plans
 - Keep coding
 - Back end is one. Just make the front end visualizers stable.
 - Problem Areas
 - None.
 - Links pertaining
 - None.

- Week 13
 - Accomplishments

- Doing self studies to see how our project ties up in the real world.
- Learned about business intelligence roles, and the tools they use
- Tools like Tableau and Microsoft PowerBI (Business intelligence) are similar to what we are achieving.
- Learning about terms like ETL and ERP, and how they interact with enterprise tools like SAP.
- How database admin skills can be tied into Business intelligence, and how common database admin tools already include code-free visualization tools.
- Plans
 - Start having some of our final paper, and presentation written.
- Problem Areas
 - Staying on top of all our classes.
 - Some of our laptops have had audio issues.
- Links pertaining
 - https://www.reddit.com/r/SQL/comments/4nvk8i/what_the_heck_is_business_intelligence/
 - https://www.reddit.com/r/explainlikeimfive/comments/1l643p/eli5_what_exactly_is_business_intelligence_and/
 - <https://www.quora.com/What-is-Tableau>
 - https://www.reddit.com/r/PowerBI/comments/9r3p11/power_bi_vs_tableau/

After all these weeks, the rest of the semester was dedicated to more logistical tasks. This included drafting our final paper, and presentation. We also uploaded our source code to our web hosting, and practiced how we would present everything that needed presenting.

6. References

List of Top Business Intelligence (BI) Tools 2020. (n.d.). Retrieved from

<https://www.trustradius.com/business-intelligence-bi>

Pearce, R. (2013, March 28). Dead database walking: MySQL's creator on why the future belongs to MariaDB. Retrieved from

<https://www.computerworld.com/article/3463901/dead-database-walking-mysql-s-creator-on-why-the-future-belongs-to-mariadb.html>

Rice, M. (2019, July 23). 17 Data Science Applications & Examples. Retrieved from

<https://builtin.com/data-science/data-science-applications-examples>

Theuwissen, M. (n.d.). The different data science roles in the industry. Retrieved from

<https://www.kdnuggets.com/2015/11/different-data-science-roles-industry.html>