E.C.H.O. Sort

Capstone 499-02

Amina Bashir, Asif Chayan, Presley John, Jeffrey Young

Roberto is a sixteen-year-old who lives in Brooklyn, New York City. For the past few week, he has been experiencing shortness of breath during regular activities like walking up stairs and vacuuming. There is a tightness in his chest that has been slowly building. He knows he needs to see a doctor, as he can tell his condition is slowly degrading. Being an undocumented immigrant, he does not have insurance and frets going to the doctor, knowing all the examinations will cost his single mother quite a large sum of money. He decides to look up his symptoms on WebMD and after a bit of research, finds out that he might be suffering from emphysema. Knowing that his condition is not a dire situation requiring immediate medical attention, Roberto decides to look up prices for the treatment of emphysema in different hospitals around him. He's frustrated by the lack of information he finds despite Googling away for hours and when he finally happens upon a .gov website, he quickly gets overwhelmed by the thousands of hospitals and all the different numbers and prices that are listed. He wishes there was a coherent way to visualize all the information thrown at him. E.C.H.O. Sort does just that, by bringing relevant information to the user. Users of the web application can choose their ailment from a list and from there, the website will display hospitals nearby to the user. The prices of each hospital to treat that ailment will also be displayed and the user will have the option to sort these hospitals based on price or distance from them. In our case, Roberto can look up hospitals that have a specialty ward that treats emphysema and because he is uninsured, he has the option of displaying a comprehensible list of prices and choosing one that best fits his budget. Economic Choice Hospital Operator (E.C.H.O.) Sort aims to change the way that access to healthcare is offered to people like Roberto. It offers a more convenient and intimate way to get the best care possible.

Like Roberto, most people in the United States do not have sufficient insurance to cover an injury or ailment. Hospitals have different prices for said injury, and patients want the best "bang" for their buck. In most instances, they want to spend the least amount of money possible, as they would be paying out of pocket for the procedure. E.C.H.O. Sort is a web application that aims to allow users to find a hospital based on an injury from a list of given injuries from a diagnosis-related group. They can sort these hospitals by preference of price for given injury or the distance the hospital is from their location (some injuries may require immediate attention), thus allowing them to pick one that suits their urgency as well as economic restrains.

E.C.H.O. Sort is composed of data from data.cms.gov for fiscal year of 2011which was last updated in 2014, making the data relatively recent. In the .gov website, the Inpatient Prospective Payment System (IPPS) gives a large dataset of the provider summary for the top 100 diagnosis-related groups. According to the site, the dataset "include hospital-specific charges for the more than 3,000 U.S. hospitals that receive Medicare Inpatient Prospective Payment System (IPPS) payments for the top 100 most frequently billed discharges, paid under Medicare based on a rate per discharge using the Medicare Severity Diagnosis Related Group (MS-DRG) for Fiscal Year (FY) 2011. These DRGs represent more than 7 million discharges or 60 percent of total Medicare IPPS discharges." A plain summary of the dataset would be the top 100 DRGs, with given provider name, address, city, state, and zip code. There are also the number of total discharges from each hospital, ranging from 11 to 3,383. There is another column which gives the average covered charges, which is the provider's average charge for services covered by Medicare for all discharges in the DRG. The remaining two columns consist of average total payments, which is the average of Medicare payments to the provider for the DRG, and the average Medicare payments. All of these columns can be sorted, from least to greatest and alphabetically (when sorting cities, states, provider names, and DRGs).

There is a significant amount of data in the Inpatient Prospective Payment System, with 163,065 columns of data. As useful as the information is, it is jumbled and confusing to understand. When one first visits the site, more likely than not, they are overwhelmed with information. They are also left to wander the site by themselves, as there are no tutorials or navigational assistance tool in the site itself. The search function is also massively inefficient in that it returns a large amount of results, most of which are irrelevant. In Roberto's case, he searched up the term "respiratory" and received 9,113 results. When he searched hospitals in Brooklyn, he got 877 results. It is safe to state that a person using this site for a serious medical situation would not live long enough to get a plausible result.

The E.C.H.O Sort website builds upon the IPPS data and returns information that is relevant to the user, in an easy-to-use manner. Simplicity is key for the website, and from the very homepage, it demonstrations that, by giving the user the option to look up hospitals based on price (for non-urgent injuries) or based on distance (urgent injuries require immediate attention and thus, needs the closest hospital). A click of either of these two options would lead to the diagnosis related group page. Currently, there are six ailments to choose from: broken limbs, poison, chest pains, respiratory issues, broken hips, and stroke (the website is of course in beta, and is thus missing a plethora of other DRGs). Once the DRG is chosen by the user of the site, they are redirected to a list of the top five hospitals that treats the ailment that they have picked. If the user picked based on price on the home page, the hospitals will be sorted in an order of least to highest priced hospitals. Along with the prices, there are also visuals attached which helps the user get a better sense of the money they would spend at each hospital. If the user picked based on distance, the user is displayed the nearest five hospitals. The hospitals that are shown are all clickable links which lead to more information about each hospital, including the address and contact information. Instead of giving hundreds, even thousands of results as the .gov website returns, our website returns only the top five results.

E.C.H.O. Sort is a web application written on the Heroku platform. It is composed of a conglomeration of front end languages, HTML5, CSS, and Javascript. For the backend and server portion, we used Python programming language and Django was our web framework. Django is a Python-based web framework that allows users to quickly construct websites and applications while avoiding the technical complexities of web development. When creating a Django project for the first time, some files and folders will be generated for you. These consist of some configuration/settings files, command utilities and web deployment utilities. Django also comes with an installation of SQLite3, an embedded database that exists within all newly created Django projects, which also served as our database for our project.

The command utility file “manage.py” is the core file that allows users to interact with and add to their project. When users wish to add more functionality, by running the python command:

```
python manage.py startapp app_name
```

from their project, a new directory will be created which contains tools for application development within Django. For our project, we created an application called “hospital”.

Without delving too deeply into the inner working of Django applications, the two most important files for development are “views.py” and “models.py”. The latter file is where users define class objects that hold information. As seen in our project, we have two models, Item (which represents the hospitals in our project) and ZipCode (which represents all the zip codes in America). When defining a class’s objects, one must also define variable type and size. Our Item model supports name and location as well as injury with an associated price. Our ZipCode model supports all relevant zip code information, including the zip code itself, the latitude and longitude as well as the city and state of the zip code.

Once these models are defined, they are ready to be created as objects and stored appropriately within our embedded database. There are multiple methods to do this, and each has benefits depending on the situation. The first method used by our team for adding hospitals to our database was through “admin.py”. This small file in your app directory gives access to a web-based GUI for insertion of objects one at a time. Simply open the file and import your models, then register them for access on the site as shown below:

```
from django.contrib import admin
from .models import Item, ZipCode

admin.site.register(Item)
admin.site.register(ZipCode)
```

Now users can manually add objects to their database through their website, as long as the project is being run on a server. Thankfully, Django comes packaged with a locally-hosted server that can be activated at any time using the command:

```
python manage.py runserver
```

This method of insertion is how we input our small list of about 30 hospitals in our database and is appropriate for testing small batches of information. However manual input is unreasonable once your data is large enough and this was the case with our zip code database.

Our search function based on price is relatively simple, as it only needs to detect the lowest price and return those values. The search based on distance is a bit more complicated, as it revolves around Pareto improvement, which is defined to be a change to a different allocation that makes at least one individual better off without making any other individual (another client) worse off. We utilize Euclidean algorithms to optimize the distance function and retrieve the most efficient hospital for the user.

Originally, we planned on using HTML5 geolocation with some special Django location-based libraries along with the Google Maps Distance Matrix API to calculate distances between a user and their desired location, but in practice we found this method to be overly complicated for our needs and instead opted for a simpler solution where we would query users for a zip code and compare it to other zip codes. This required finding a database of all zip codes in America, fortunately through research I discovered The Zip Code Database Project, an open source database which contained all the information we needed as well as a distance formula for use in Python. The database itself is in a .csv (comma separated value) file, and as such can’t easily be placed into our own database, especially not manually. To solve this, I used another method of data insertion, by creating a Python script that runs through the .csv file line by line and inserts each line as an object following the ZipCode model as defined in models.py.

```
from hospital.models import ZipCode

import csv
dataReader = csv.reader(open(csv_filepathname), delimiter=',', quotechar='"')

for row in dataReader:
    if row[0] != 'zipcode': # Ignore the header row, import everything else
```

```python
        zipcode = ZipCode()
        zipcode.zipcode = row[0]
        zipcode.statecode = row[1]
        zipcode.latitude = row[2]
        zipcode.longitude = row[3]
        zipcode.city = row[4]
        zipcode.state = row[5]
        zipcode.save()
```

This script imports the appropriate models as well as csv library included in Python, and uses a basic for loop to save the codes to the database using the csv.reader function to read the file. Because the .csv file contains over 30,000 zip codes, this line-by-line saving process took over 2 hours to complete. The database also provides us with a simple distance formula that when adapted, allows us to easily calculate miles between any two zip codes.

```python
from math import *

def calcDist(lat_A, long_A, lat_B, long_B):
  distance = (sin(radians(lat_A)) *
              sin(radians(lat_B)) +
              cos(radians(lat_A)) *
              cos(radians(lat_B)) *
              cos(radians(long_A - long_B)))

  distance = (degrees(acos(distance))) * 69.09

  return distance
```

The way this function works is that it takes 2 latitudes and longitudes as parameters, converts the values from degrees to radians and calculates the sine and cosine of each. This inner functions are imported from the included Python math library. After multiplying and adding the results. The following distance is multiplied to account for mileage and the resulting arc cosine converted to degrees is the final distance in miles. We use this formula to calculate distance between a given hospital and a given user location. Currently we assume the location to my home zip code in Flushing, Queens, NY, however we are looking to expand this to include any zip code later. We encountered a bug where some zip codes are missing from the database (such as the zip code for Hunter College which is 10065) and include these in a case-by-case basis.

For displaying all this information, one must develop the "views.py" file. In this file, a user must define requests that will be prompted from users on the website. A small sample of one a view is:

```python
def about(request):
    return render(request, 'aboutus/about.html')
```

This is the view for the "about us" section of our website. When a user requests the "about" page, the view will see that a request from prompted and return a rendering of the given .html file we have specified to deal with the request, that being "about.html". This means that our project will load (or render) "about.html" on the requested page.

These .html files show what is viewed on the website themselves, and as such employ a complex mix of code for displaying information. For example, if you perform a search by cost on our website and select any ailment (in our example here we will look at broken limbs), the project will interpret the user as calling the view:

```python
def broken_limb(request):
    all_broken_limb = Item.objects.filter(diagnosis_related_group='Broken limbs')
    return render(request, 'res-
brl/broken_limb.html',{'all_broken_limb' : all_broken_limb})
```

This view performs two functions, firstly it loads the correct item from our database by performing a special filter shown by the "Item.objects.filter()" command, which is in turn used within the special .html file "brl/broken_limb.html". This .html implements a series of special .html specific Django commands such as " {% extends '' %}" which call other .html files for the sake of using templates, meaning that you can have a base .html file that all your other specific .html files can reference to ensure style and presentation is uniform.

Our "broken_limb.html" file also contains the specific graph for displaying our information. In our project, we use a special JavaScript library known as D3.js. This library contains a series of tools that allow users to bind any sort of data to specific objects and display them using SVGs (scalable vector graphics). The specific graph we used was adapted from a graph that was provided on the D3.js website. To use the specific tools within D3 library, one must declare the D3 script, which can be done using the appropriate HTML tags.

```
<script src="http://d3js.org/d3.v3.min.js"></script>
```

Once this script is declared, we can implement the appropriate features. Our specific graph uses a frequency array of our given hospital data, and uses it to show off the relevant parts of the data (i.e.: the cost data of the specific injury per relevant hospital). This information is hard-coded within the frequency array for the sake of testing and simplicity.

For the creation of the graph itself, it employs a mix of CSS styling for colors and attributes and JavaScript for the objects and interactive features. SVG allow us to create smooth and scalable images for our graph, and in our situation we declare two SVGs, one for our bar chart and one for our pie chart. Our SVGs contain attributes which allow us to specify the size, and so for the bar chart, the height and width are dependent on the values of our data points and the compression of SVGs are dependent on the number of data points we are using. On the top of our bar chart rectangles we append a text box that has the total value of what the bar chart is representing.

D3 also allows us to add interactive features that will updates the charts in real-time depending on the parameter. At a glance, the bars represent the total costs of the hospitals in relation to one another and the pie chart shows the breakdown of all the hospitals varying costs in relation to their total. By moving the mouse over the any bar, it will display the three costs that make up that particular in the pie chart, likewise hovering over any part of the pie chart will show the specific cost breakdown for all the hospitals in the bars. The final result allows for an in-depth viewing of the information outside of basic lists. These transitions are possible through D3 functions which update SVGs on certain actions.

Heroku lets developer(s) deploy, run, and manage applications written in many different languages, including Python, our language of choice. The Python language is fast, has more user-friendly syntax and uses fewer lines of code, making it ideal for getting things to the web quickly; it's emphasis on readability and simplicity is vital for beginners. It is also very powerful and works well in object-oriented designs, and was ideal for the back-end of our site. The back end of web development is essentially the engine of a site– the user doesn't see it or directly interact with it. It consists of the server, database and sever-side applications; it provides the behind-the-scenes functionality. We had to ensure that the data or services requested by the front-end system were delivered through programmatic means. We also created and maintained the entire back-end while simultaneously performing the testing and debugging. The implementing and design of data storage solutions, building of reusable code and libraries that are robust and expandable, optimization of the website for maximum speed and scalability, whilst ensuring security and data protection was all key in a smooth functioning website.

As the back-end web developer, Presley's primary function was to take front-end instructions and give it working functionality. This included tasks such as making drop down menu possible by building the infrastructure that pulls values from the database. SQLite3 was chosen to create the database; a hospital table was given an address as well as other fields like name, DRG, covered charges, total and Medicare payments. Our six DRGs were created and each given to multiple hospital of varying locations. Our web framework, Django, emphasizes reusability; rapid development and the principle of cutting out repetitions. One of the best features of the framework is a lightweight web server for development and testing; this pre-configured server restarts whenever the code is modified. URLS, Views and Models are three of the most important forms in the framework. Basically the developer provides the model, the view and template maps it to a URL and Django serves it to the user. The file urls.py controls what is served based on URL patterns: written as regular expressions. When a user makes a request for a page Django's controller looks for the corresponding view and then returns the HTML response. The views.py file handles what the end-user "views" or interacts with: a view function is a python function that takes a request and returns a web response. HTML contents of web pages are the usual response, as well as a XML document, an image, a 404 error or a redirect. The models.py file is the database structures and the metadata. Another crucial file is the manage.py, this is like the local Django-admin for interaction with the project via command line (starts the development server, sync the database, and send a test email to a specified recipient).

To ensure that our website was fully responsive on many different devices like desktops, laptops, tablets, and cell phones, Bootstrap was implemented into the website. Bootstrap is a front-end web framework used to design websites. Responsive web design is about creating web sites, which automatically adjusts itself to format all devices. Bootstrap was instrumental in allowing quick and easy construct of prototype new designs. It also added consistency to design and code between projects and between developers, prevents repetition between projects and ensures cross- browser compatibility. Bootstrap framework includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation and image carousels. It was very instrumental in the building and debugging phase. Following on with the concept of no repeat elements like the navigations bar, footer that is “static” (would be present on all webpages) was designed on a base, which was then re-generated on the many different templates.

The website was deployed on Heroku: a cloud as a platform service for developers who want to get their applications online without having to worry about infrastructure details. There is an Instant Deployment with Git push; it gives our project a place to live on the Internet. The website domain www.echosort.com was acquired through google domains.

Another major part of the project was the front end. The front end was an essential part of the project because the platform selected for our project was a website. The font end served as a bridge between the developers and the user and ultimately defined the project. The aim of website was to develop something that will change the way these hospital searches were performed and information was presented to the users. The data.gov website, which is where the data was retrieved from, has the same database as this website. However, that website presented the information in a bland manner. It, in a way, “threw” tons of information at the user without sorting it and left the user struggling to search through the plethora of information on his/her own. In addition, the website had no design and was simply a database, where the user would have to go line by line to find what he/she wants. This website, however, aims to change that. It attempts to rearrange a lot of information, in the database, and display it to the user in a simple way, which is easy to interpret and helpful in more ways than one. It provides simple search engines, which help users navigate the website with ease, and presents sorted data in the form of lists and graphs so that the user can understand the information easily and use it to make a decision. The website’s goal is to provide a platform that users are comfortable using and that is convenient for people. It aims at a wide and eclectic audience; therefore, its design had to be one that was appealing to a wide group of people, including those that are good with technology and those that are not and people from all age groups, excluding children. If the website fails to attract people, then the back end and the all the other work loses its worth; thus, while developing the front end of the website, several aforementioned things were taken into consideration.

The front end of the website was developed using HTML, CSS and JavaScript. To begin with, HTML was used to develop the structure of the website; therefore, it was the backbone of the front end. HTML was used to create the header. The template for the header was taken from an online website: getbootstrap.com. However, the header was not completely functional, especially since this website required dropdown menus. Therefore, additional features were coded into the header using HTML and CSS and it was adapted into the project. The header was used to link all the pages in the website together; it functioned as the search engine. It used dropdown menus and categorized similar pages together to help the user search with ease. They made it less complicated for the user to find his/her desirable page The dropdown menus also were beneficial in that they truncated the amount of options the user initially sees; this furthers the goal of a simple website with easy searches because it prevents the user from being intimidated by more than necessary options. In addition, HTML was also used for the body structures of each page. Mainly div tags were used to divide the body into columns; this helped spread the information across the page rather than keeping it in one corner of it.

```html
<div class="row">
            <ul>
            <li><div class="darkestBG col-sm-3"> <img src="http://www.iconsdb.com/icons/preview/black/html-xxl.png"><div></li>
            <li><div class="darkestBG col-sm-6">
            <strong>Web Designer</strong><br></br>Asif Chayan is responsible for the front end of web development, which is HTML and CSS based. He is currently a student at Hunter College.
            <br></br>
            Email: asif_chayan@yahoo.com</div></li>
            </ul>
        </div>
```

They helped create a more uniformed and even look for the pages. This was evident in the page that used pictures to show DRG options. Div tags were used here to evenly display the images across the page. This helped make the website more interactive for the user and again simpler to use.

```html
<div class="col-md-4">
            <a href="{% url 'stroke' %}" class="img-rounded thumbnail img-responsive img-rounded" width=230 height=230>
                <p> Stroke </p>
                <img src="https://ercraigranch.com/wp-content/uploads/2016/06/stroke-symptoms1.jpg">
            </a>
        </div>

        <div class="col-md-4">
            <a href="{% url 'broken_limb' %}" class="thumbnail img-fluid img-rounded" width=230 height=230>
                <p> Broken Limbs </p>
                <img src="http://topnews.in/health/files/fixing-broken-bones.jpg" class="img-rounded thumbnail img-responsive img-rounded" width=230 height=230">
            </a>
        </div>
```

HTML, overall, proved to be the best language for this project as it was simple to use and flexible enough for page structures that were designed. In addition, it is easy to modify later on by others because it is a popular base language. Therefore, it provides an opportunity for others to add to this project with ease. This, again, furthers an important goal of developers. The developers want this to be a platform that users can use comfortably and is intimate with the users. Hence, the team's emails are provided so that users may contact team members with suggestions on how to enhance and alter the website so that it provides the best service and remains a convenient option for users. With HTML, not just the team, but also others can manipulate the front end, the search engines and display of information and the website with ease, if need be. Since the design of the website aims to be simple, HTML has all the properties and is flexible enough to add to it.

In addition to HTML, CSS was used to create the design. It was used to manipulate the structure and enhance it to create the mood of the website and make the website appealing to users. The template for the header was an adaptation of an online one; however, because it was not completely functional, we used CSS to add more to it and adapt it to this website. CSS was used to make the dropdowns menus work when hovered over. It added features using colors and styling techniques to help users search with ease.

```
nav ul li a:hover{
    color:#ccc;
    text-decoration: none;
}

nav ul li:hover ul{
    display: block;
}
nav ul ul {
    display: none;
    position: absolute;
    background-color: #333;
}
```

CSS was also used to add the colors, which helped create a seemingly common theme and look for the website. The three main colors used for that were blue, dark gray and white. The color choices were made very carefully because not only did the colors have to support the hospital theme, but they also had to attract users to the website, help them navigate through it and become comfortable with it. Thus, the color combination and choices helped add to the simplicity and convenience of the page. This was evident especially on the homepage, where uniformity and color coordination were momentarily paused as orange texts were placed in a blue-black-white page to bring the user's attention to two search buttons, which help them directly search according to distance and cost. Those are placed there for convenience and to save the user's time. They automatically attract the users' attention and are a means to speedy results in cases where time if of the essence. Again, this adds to the simplicity and convenience goals the website aims to achieve. CSS was a great option for this project, similar to HTML, because it is a popular and simple language; therefore, the code can easily be interpreted, changed and enhanced by others, if need be. In addition, since the goal of this website was to provide a simple yet efficient front end, CSS proved to be flexible enough to meet the demands.

Lastly, JavaScript was used to make the website easier to access and more interactive. The website used a bootstrap program so when it is used on mobile devices the dropdown menus and the search engines still work. Bootstrap also ensures that when the screen changes, the way the information is presented adjusts automatically. It also made the website more interactive and flexible so that it may work and deliver what it promised by adjusting/adapting to the space it is provided. This allows people who do not have access to computers to open the website on their phones or any other device with a different screen size; hence, our website becomes more accessible and convenient for the users. Bootstrap helps ensure that the website provides the optimal service in any given environment.

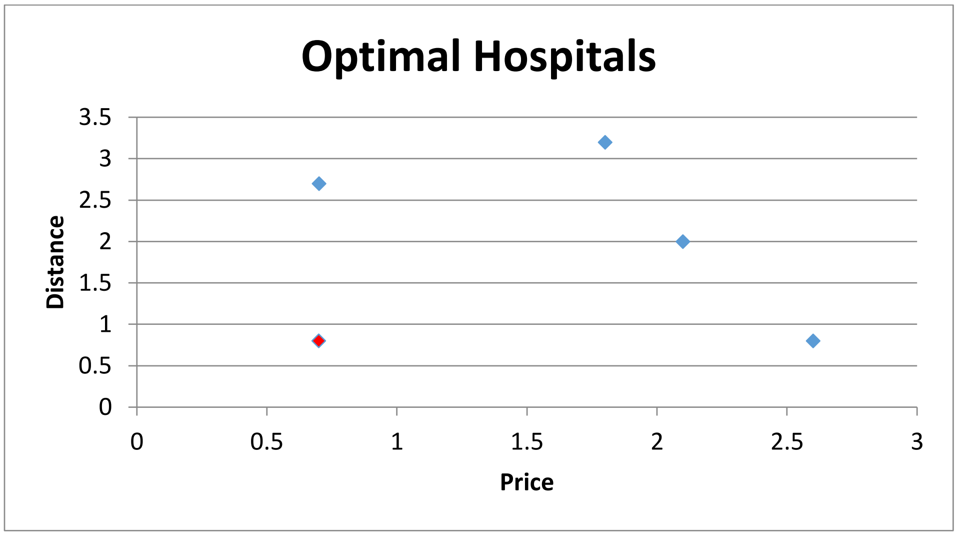
```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">

        <!-- Optional theme -->
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css">
```

JavaScript was also added to make sure the backgrounds and the features on the pages load right away and there is no lag amid page-to-page transition. This allows for better service and helps people get to their desirable page faster without any hindrance. It saves the user time and loads the information on the page right away. This is crucial since this website aims to be working with a large database and expand to include more search options and methods. Other than the graphs and the charts, this is what JavaScript was mainly used for.

Since this project was designed according to the formats presented by the web framework Django, the format of the HTML files was different. Instead of creating several different pages, each with a header and body of its own, a common base page was used. The base page included the header file and the background, both of which were the same in all the files. Instead of repeating that code in every file, only one file contained the header and background, while the other files were extensions of that base and contained information, which was specific and relevant to only those particular files. This format proved to be beneficial in several ways. If something were to be modified, it only had to be modified in one file as opposed to all the files. This made adding changes and altering the front end easier. This is also a more organized and simple coding structure, which will prove to be convenient for anyone in the future attempting to enhance and/or modify the code in any way. However, this does prevent each page from being uniquely different from the others, as the base remains consistent throughout. However, in this case, a common search engine is desirable because it adds to a consistent, simple and easy to use design, which happens to be one of the main goals of this website.

Each of these three languages contributed certain qualities and features to the website, together, helping achieve its goals and aims. Mainly HTML and CSS were used. Working with a few languages as opposed to a lot helps make coding more manageable, easy to keep track of and simpler to implement. This is essential for the design of this website because it is a work in progress and is intended to change constantly to match up to user's demands; therefore, it is important for the code to be something that is easy to interpret and work with. The front end happened to be the last major part of the website.

The future of E.C.H.O. Sort is bright. Because there are not many websites out there that aim to do what we do, any adaptation we make to the site will be innovative. In addition to increasing the number of diagnosis related groups and the number of hospitals, we intend on adding features that users will benefit from, all the while keeping the simplicity and ease of use. We realize that our clients would want to find a hospital that is both cost effective and nearby to them and we can try our best to accommodate that by creating a multi-search function. A foundation for this multi-search function is already in place. Our website searches by distance and cost separately, the multi-search function will perform both of these searches and then run an algorithm to combine them in order to present the users with an optimal solution. It would use the algorithm for the Euclidean point, encoded in python, to figure out a list of optimal solutions. Here is how the following algorithm would work. It graphs the points on a cost vs distance graph and then finds the most optimal point. This point, the nadir, will be the cheapest and closest hospital possible.



Here, the red dot would be the calculated nadir, while the blue dots would be the hospitals with distance in miles on the x-axis and cost on the y-axis. From the nadir, outward circles will be calculated by the algorithm until one touches a hospital. Then, that will be presented as one of the optimal choices. This will be calculated in python since Django is a python based framework. The options calculated would be the optimal hospital choices when searched by cost and distance, two variables. Now, another addition that could be added is while searching using these two variables, to also make the results lean more towards lower cost or lower distance. This will be calculated using the same technique, except a cost preference will make the circles calculated more horizontally stretched while a distance preference would make them more vertically stretched. This is an algorithm, which shall be incorporated into the website in order to implement a multi-search function.

In addition, we also take into consideration that some users suffer from more than any one ailment at once, and so we plan on adapting a way to choose multiple DRGs at once. This would be more complex to implement, however can be encoded into the code we already have. While choosing multiple DRGs, we can search each separately and provide the user data in terms of graphs for easy comparisons or we can find hospitals that have both and add the cost of both injuries. Therefore ranking hospitals based on the cheapest sum cost of the combined DRGs. Both options can also be implemented, so the use can compare and have more information to help form an informed decision.

E.C.H.O. Sort is a web application that, in its creation and maintenance, uses a multitude of programming languages, libraries and web site utilities. The backbone of the site is powered by Python and supported by HTML, CSS and JavaScript. Our internal calculations are done using Python scripts. The purpose of the site is to provide a simple and easy-to-use interface for users to access a database of hospital data regarding injuries and pricing in America and provide a specific lens to process the information. We will continue expanding and making E.C.H.O. Sort even greater and hope that our application can help others in the future.

References

"D3.js - Data-Driven Documents." *D3.js - Data-Driven Documents*. N.p., n.d. Web. 18 Aug. 2016. <https://d3js.org/>.

"DashBoard." *Popular Blocks*. N.p., n.d. Web. 18 Aug. 2016. <http://bl.ocks.org/NPashaP/96447623ef4d342ee09b>.

"Documentation." *Django Documentation*. N.p., n.d. Web. 18 Aug. 2016. <https://docs.djangoproject.com/en/1.10/>.

"Getting Started." · *Bootstrap*. N.p., n.d. Web. 18 Aug. 2016. <http://getbootstrap.com/getting-started/>.

"Heroku Dev Center." *Heroku Dev Center*. N.p., n.d. Web. 18 Aug. 2016. <https://devcenter.heroku.com/>.

"The Zip Code Database Project With Distance Calculation!" *The Zip Code Database Project with Zip Code Distance Calculator*. N.p., n.d. Web. 18 Aug. 2016. <http://zips.sourceforge.net/>.