ECHO Documentation

ECHO is a collaborative effort of Jeffrey Young, Presley John, Amina Bashir and Asif Chayan. The main goal of this project is to make an application that uses the Inpatient Prospective Payment System (IPPS) Provider Summary database and allow users to access that information, as well as make recommendations about their searches. Users will be able to query an injury and our application will sort the database appropriately and display the relevant information. The primary information that needs to be shown to the user is the cost of an injury type (by DRG) and the distance from user's location to local hospitals. Once this information is known, the application will run an efficiency algorithm and display the best choices.

We broke the project down into four major sections: exporting data, accessing database, developing user interface and presenting the results. First, we will export data from the site using one of the options that it provides. Next, the struggle will be to access and store that data into our SQL database. Once it's stored, we will access that database using python and connect it to the website. After these two parts, we have to develop the website. We will be using HTML/CSS for the website. The website will have two major functions: injury and hospitals. We first help the user match to an injury using DRG; if that is not successful, we will link them to WebMD. Once the injury is confirmed, we will give the user the option of performing a single search or a multisearch. A single search will find the hospital for that injury based on one criterion. A multisearch will allow that search to take place based on more than one criteria. This search will find the results for each criterion arriving at an optimal option. The results will be presented to the user. In addition, distance and price are two ways to search for a hospital. We will also use the data to draw some general conclusions.

The database itself is from a study performed in June 2014 acquired from data.cms.gov. This database contains numerous pieces of important financial information for over 160,000 hospitals in America sorted by DRG. DRG stands for Diagnosis Related Group and is a system of classifying a patient's possible diagnosis into about 20 major body systems and subdivides them into many more smaller groups for the purpose of financial payment. Each object in the database contains the DRG, the hospital name, ID, address, city, state, zip code, region, average charge cost for DRG, average covered by Medicare, average total covered, and amount of patients used to determine the averages.

The proper orchestration of a hospital entails many people working together in different sectors; so instead of cataloging all of these intricacies we built our database with specific attributes that are important to our project. More than one hospital can have the same name so to uniquely identify each one we used I.D. Instead of creating attributes for street number, zip code, city and state for each hospital we created the multi-valued address to show the location of a hospital. Average covered charges is the amount charged to a patient by the hospital, while average total payment was the actual amount paid to the hospital by the patient and average medicare payments is the amount paid to the hospital by medicare for a patient.

For this application we have chosen a web page for our platform. The central functions of the site will be a few different systems for determining injury type and location, as well as simply letting the user search through our database for anything in particular. Simplicity is key in our design, and so website navigation will be done most likely with a series of questions to guide the user to the desired result.

The application will have two components, the web page where the user will make queries into the database, and the server where the database is stored. The creation and maintenance of the server will be stored locally on one of our team member's Raspberry Pi computers. We don't expect huge traffic so this shouldn't become an issue. The database will be written using MySQL, a popular database language that our team is learning. Currently the database is stored on data.cms.gov, and so we will export it into a file type that we can later import into our own custom database. Structure Query Language is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS).The scope of SQL includes data insert, query, update and delete; schema creation and modification, and data access control.

The Raspberry Pi itself is a credit card sized computer. The board features a processor, RAM and typical hardware ports found on most computer. It doesn't have as much power as a desktop PC but their price US $40 makes for great little computers to play around with and incase it is broken they're not too expensive to replace. They are also great at doing things that don't really need a pricey computer to do, such as hosting web servers, media centers and Network Attached Storage (NAS). In our case, we are going to use a Raspberry Pi as our web server. A web server can be a computer or an appliance that stores a website's component files (e.g HTML documents, images, CSS stylesheets, JavaScript files, databases) and delivers them to the end user's device. They also includes several parts that control how web users access hosted files; they map URL requests from a web client (typically a browser) to a resource that will handle

the request and return a response to the client, the web client and the web server uses HTTP (the protocol browsers use to view webpages) to communicate over a TCP network.

The back-end of our web application will most likely use Python as the base language. Python is a widely used high-level, general-purpose programming language, it's designed to emphasize code readability making it easy to read and simple to implement. It is also a scripting language and is often used for creating web applications and dynamic web content. These features makes it perfect for our project as it allows us to work more quickly and integrate our different systems more effectively. In the application, Python is going to be used for interacting between the web page and the web server for retrieving data and performing sorting algorithms when choosing hospitals.

The front-end of our web application will most likely be a mix of HTML5, CSS and Javascript. HTML for basic site formatting, CSS for styling, and Javascript for the bulk of the site features as needed. In order to perform our location services, we will use Google’s Maps API for directions and static maps (the implementation being in Javascript). CSS is used for styling because it is very efficient. It is consistent: any change made to the website's CSS style sheet, you can automatically make it to every page of your website. Therefore, it helps save time. It also reduces bandwidth: it separates your website's content from its design language thus, you end up reducing the file transfer size. Since the CSS document is stored externally, it will be accessed only once when a visitor requests the website, not every page will be accessed with each visit. Lastly, it increases the website's adaptability to the different browsers and ensures that more visitors will be able to view the website in the way you wanted. Thus, CSS is an ideal choice.

The purpose of our project is to not only provide the most economic/feasible hospital choices but also to determine from the data which areas (regions, cities) have cheaper hospitals. We will attempt to determine this once the search algorithms are in place. We will use the sorted data as well as calculate the average of each region and state to draw this conclusion. In order to present this information to the the user, graphs, charts and lists are being considered. The trouble arises mainly with graphs and charts. Those will require many other elements of the web page to be in place before their implementation can be determined. Similarly, the results of the searches might also be presented to the user using charts and graphs. However, the implementation for that will have to be determined after the initial setup is in place.

The possible difficulties we foresee include trouble implementing the map to determine the distance factor. While we know which API we are going to use, it will still be hard since we will not be working with a static map. Not only that, but the implementation of the decision making algorithm might prove to be difficult. However, we can't attempt that until we have certain other things in place. We also will, as mentioned previously, run into some uncertainty when it comes to implementing graphs and charts if we decide to use those instead of or in addition to lists. Lastly, reading the exported data into the SQL database might prove to be a bit troublesome as well since this is the first time all teammates are working on something of this magnitude.

However, we are confident that we'll be able to overcome these difficulties. In our team we each have our focuses. Presley's strong point is in handling SQL databases and working with them. Jeffrey will be handling the backend web development while Asif and Amina handle the frontend. This isn't definitive as we plan on helping each other in all components as necessary, in order to create the best environment for a well-rounded education.