Web Crawler

For our web crawler, we used ParseHub which is a free to use software with multiple different paid versions offering better services. For this project, it was not necessary to buy any of the paid versions since the scope of the project fell into what was allowed with the free version. While ParseHub does provide a tutorial on how to use it to gather information off of other sites we will also discuss some of the things learned using it. To start, you provide ParseHub with a website that you wish to gather data from. Once we have provided a site, it will open up the webpage and from there we can start selecting data that we want to collect. For this project, we were specifically looking for the name of the movie, the date it was released, the runtime of the movie, the poster, the director, the movie’s budget and the cast. We did find that it sometimes did not always grab the data that we wanted due to the differences in formatting from one page to another. To rectify this, we started making separate select categories for the data that was not selected by the first category. This helped to gather more than what we would have had we not but it is not a perfect fix and as such it is possible that we still missed some data. To that end we are also manually going through the data to try and insert what is missing. It should also be noted that a spot might be blank simply because there was no data to retrieve which is most common with movie’s that either have recently been released or have not yet been released but only announced. It is also possible that the data is not publicly available and as such it is not listed with the movie. The last major thing to note when it grabs data is that it does not convert punctuation or foreign characters well when putting it into a .csv file. This is most commonly found with foreign cast members and directors. Once we set up the templates, we then had the option to either run the entire project or to test it. We first tested it which returned the first five pages and allowed us the option to go for an additional five pages as we saw fit. It was through this testing that we discovered the issues stated above. Lastly, once we had the file, during our manual check, for any empty spot that did not have data and was confirmed to not have data, we filled in N/A into those spots so that it would look and function better when we inevitably import those files into the SQL database.

SQL Database

For our database, we decided to use Microsoft's SQL Server which allows us to install a SQL server onto a computer. We choose this over Oracle because it is free and was easy to install. Starting with installing the server, we first went to<https://www.microsoft.com/en-us/sql-server/sql-server-downloads>. From there, there are two choices; the developer version or the express version. For our project, the express version is more than sufficient for our needs. After that, it was installed on a computer that will constantly remain on so that the database can be accessed not only by us but also by users requesting information. We did a basic install of the express server and left the name as the default which is SQLEXPRESS. From there it asked us if we wished to install some languages such as Python and R which we allowed it to do. Once that was done, the server was installed and several programs were also installed to help manage it like the SQL Server Configuration Manager and some debugging tools.

We then went into the management console to make it available remotely. We did this by going to the SQL Server Network Configuration tab and expanding it so that we saw Protocols for SQLEXPRESS. That then gave us three options to adjust which are Shared Memory, Named Pipes and TCP/IP. By default, TCP/IP is disabled which means the first change to the server was to enable that which then required us to restart the server. Once that was done, we then went and installed the Microsoft SQL Server Management Studio which functions just like how MySQL does only it gives us more access to edit and view things from the database. With it installed on the same computer as the server, we then connected to the server using Windows Authentication. Once connected, we then went down to the managment section and then the SQL Server Logs. We then selected the current log and then filtered out the log to find the port number that the server was currently listening on. Once we have the port number, we can then connect to the server remotely from other computers. We then also created logins by By installing Microsoft SQL Server Management Studio on the other computers, it will enable us to login to the server under different usernames. This was accomplished by going to the security tab and right clicking it. We then go to new and select login. This then brings up the login page where we specify a Windows user or a SQL user. For this project, we are using a SQL user which means we need to provide a usernanme and password. We then have additional options to either have the password be temporary and require the user to change on their next login or to make it a permanent password. For this, everyone starts with a temporary password which they then must change. Once the logins are complete, we can then have users sign in remotely with their username and password and telling it where the server is. This is where the port number is used, we tell SSMS that we want to sign in to a database engine with the name being the IP Address of the server followed by the port number. An example of this is 123.156.0.78,115. We then select SQL Server Authentication so that we can sign in and then put in our username and password.This then gives us access to the server and databases that we have permission to modify.

To setup the database that we will use, we first right click the database tab and select New Database. We then name the database and then click add. From there, we then need to give all of the user accounts that we made for the project access to that database. We go to the security tab and then into the logins tab. We then right click o one of the user accounts and go to properties. From there, we go to the bottom of the first page and change the default database from master to what we named out server. From there, we tehn go to user mapping and check the box next to our server. This will then give permission for the account to view and make edits to tables in that database and only that database unless we need to make a second one. From there, we repeat the process for all of the other accounts .

User Interface

The UI files are written using HTML/CSS for implementation of design elements, while PHP is used for general functionality and validating user input. The main page is marked index, as is customary practice for web page design; however, some design pages have been separately implemented and simply included so as not to clutter the file with an exorbitant amount of code. These files included are labeled header and footer. The header contains the CSS style code, as well as the website logo and black background requested by the client. The footer contains the same black background, as well as a special thanks message to our client. In the current index file, there is a search bar and links to login (a file named log outlines a design for the login capability) or create an account within the system. These will be handled outside of the index file by referencing another PHP document supplied by each group member. In order to test the UI, we will be running Travis CI as our automation framework as it is lauded for its ease of integration with GitHub. Within the Travis CI documentation, it describes how different test functions may be implemented for browsers and web interfaces through the .travis.yml files which are meant to split up and build into multiple jobs. Test planning is scheduled for each functionality at the end of each sprint utilizing the Travis commands and software.

The UI is currently being tested and run locally through the use of Xampp. After connecting to the xampp interface, one must start Apache and SQL. The Apache connection, if valid and accessible from your device, will be used to display the webpage in your browser and link properly to other files referenced within the same folder for dynamic use. The SQL connection would connect the UI with the backend database in order to retrieve information requested using PHP in the UI files.

Installations for implementing the UI locally would be Xampp; however, for users/clients who would like a public, interactive service, this would require the purchase of a specific domain to host the project on the internet.

User Authentication

The user authentication system is written in PHP and HTML. XAMPP is being used for hosting. Multiple PHP and HTML files are working together to ensure a secure login/registration process. The login will look different until it’s integrated into the main user interface, for testing purposes it’s being built separately thus having a different design compared to the main UI.

The register.html file creates 3 fields for the user to register with their username, password and email. Any other fields could be added later on if the client requires so (Name of the user for example) via this file. Upon clicking the submit button, register.php takes care of the next step in registration. First thing it does is try to make a connection to the database. Then it verifies whether or not any fields were empty during the registration, followed by checking to see if the username already exists within the database. If there’s an error at any point in the process, register.php would display a message explaining the error. If everything goes smoothly a success message is echoed.   
 The next step would be to redirect the user to the main login page of our project UI which hasn't been implemented yet.

The Login.html takes in two inputs, the username and password and upon clicking the submit button, forwards the items to authenticate.php. The authenticator first tries to establish a connection to the database. Upon successful connection it verifies if the required data were submitted from login. Then it prepares the SQL statement and binds the parameter to prevent SQL injection. It then checks if the username exists and then verifies the password hash, stored in the database. If the password is verified then a new session is created. Then it redirects the user to a dummy page (home.php and profile.php) which at a later on would be changed to the main user interface.