**Date(s): 11/8/18**

**Prepared by: Freddie Birr, Jose Rodriguez, Justas Vikonis, Shayne Bloom, Khaja Ameenuddin**

**1. Introduction**

This document contains the system requirements for Gilligan’s Coconut TV These requirements have

been derived from several sources, including our client, Professor Rossi. Each header will have a representation of an aspect of the project, and how our group tackled the issue.

**1.1 Purpose of This Document**

This document is intended to guide development of Gilligan’s Coconut TV***.*** It will go through several

stages during the course of the project.

**1.2 Scope of the Product**

*.*

The product that we intend to create is a movie streaming service site that allows users to preview films on our database, create an account, and choose a tier in which to subscribe. The intent here is to create a service with the framework of Netflix, but utilizing a less clunky search bar, as well as improvements in other aspects. Our end goal is to create a streaming service using all the user approved features from competitors service, but in the most competitive and desirable way.

**1.3 Business Case for the Product**

The product is requested to move our client from and a brick and mortar approach to movie rental to a streaming service to provide customers with easier access and usability. By creating this application, it will grow our business to include strengths in building websites, servlets, and development of information by cliental.

**1.4 Overview of the Requirements Document**

Target franchise: Everyone with access to the internet

Child section(Includes PG and G potentially. Allow parental controls of what ratings a title can be)

Have ratings attached to titles; Use MPAA ratings

* Ratings and genres used for recommendations of titles.

- Search Bar: Filter must return all movies with the search in the title.

- Search by title and genre

- No preferred layout : run the layout verically

Implementation: Support on all browsers, (safari, explorer, firefox),

* mobile version not needed

No adds/add space needed in design

Recommendations given to the user:

* Ratings
* Genres
* Prior downloaded content

Navigation: No more then two clicks from homepage to desired title

Promote new films on home page

Subscription:

* Won’t need a subscription to browse
* Needed to watch, no preview of film

Subscription: 3 levels, each has different benefits and promotions

* Bronze: 1 movies rented out at a time, small promotional benefits
* Silver: 2 movies rented out at a time, larger value
* Gold: 3 movies rented out, highest promotional value

Design prefernces

* Use green and red color scheme.
* Netflix feel, less clunky search bar

Main account features

* 3 users per account
* Profiles managed by head account (Account attached to credit card)
* Primary language: English other languages?

Customer service aspects

* Charge up front, automatically deduct money from account after billing period passes
* Contact user who is subscribed that hasn’t used their account in awhile. Period of time for this?

**2. General Description**

*.*

The project Gilligan’s Coconut TV is a new streaming service taking movie rentals to a whole new level introducing the market to a streaming service. With this product, we intend to move the movie rental business to a whole new market, showing the power of the internet. The convenience of online streaming will be the selling point of our product. Renting a movie at the store requires you to visit the store, and once you finish watching, you need to return the movie or risk fine. With the online streaming you can have what movie you seek with just a few clicks of the mouse. Also, when you finish you film, your just a few clicks away from returning the title. The time wasted doing this all but disappears with Gilligan’s Coconut TV. Any person young or old with whatever tastes can utilize this product so long as they have internet access.

Constraints faced in this developmental period were factors in completion. First the amount space available in the database holding our titles was hard to limit as we try to streamline speed of retrieval as well as holding the largest collection of titles as possible.

**2.1 Product Perspective**

We chose to develop this product for our client as we believe fervently that what they intend to create has an audience and can be profitable. This product will serve the people who want to watch movies, but don’t want the inconvenience of getting the film and returning it at their own expense. The primary stakeholders in this case are those funding the project, as well as our development team. All users as well as stakeholders will benefit from the success of this enterprise.

**2.2 Product Functions**

*.*

The products main purpose is to provide a streamlined outlet for viewing films. Users can watch many films; account administrators can limit the scope of certain profiles watching ability. Also finding a title should be really simplified as opposed to competitors as well as more user friendly and compatible. Also, a ratings function will appear with each title so that each user can tell if a movie is highly recommended or not.

**2.3 User Characteristics**

The final users of this product would be anyone able to use a computer/browse the web. Instead of putting in the disc to watch these movies, users of our site are going to need the ability to navigate the website. The level of interaction we hope to show is the lowest level of technology understanding.

**2.4 General Constraints**

Constraints are using java through eclipse, database through SQL, and the code shared through Git-Hub.

**2.5 Assumptions and Dependencies**

The finished product needs to be delivered over the internet. We assume the user has a computer with internet accessibility. It is also assumed the user of our product could navigate a simple, commonly used website. The website will be designed to ensure that the most pedestrian user can navigate comfortably. Another assumption made is that the admin can access and use our administrative tools to add and remove items from the database as well as view tables from the database by entering the tables name.

**3. Specific Requirements**

This section of the document lists specific requirements for Gilligan’s Coconut TV Requirements are

divided into the following sections:

*1.* *Functional requirements.*

*.*

Functional website with minimal clicks to get from one point to the next. Should be able to get to the desired title with ease. Ease of access shold be very important. The easier to use, the broader the audience that can view the material. The User of the application should be able to sign up by choosing a subscription plan right for them or login from the main page. They should then be able create a profile for an individualistic experience. From this page they can add profiles, edit profiles, and manage the account. The User should be able to see movies they have “Favorited” on their home screen as well as their most recently downloaded titles. From here the user can either select a movie to watch that displayed by clicking a picture of the movie image, or search for the intended title in the search bar. Outside of the consumer there are options for the website administrator to update infromation such as adding and removing movies from the selection available to the consumer.

*2.* *Non-Functional and System requirements.*

The web application written was written using .css and .jsp pages for displaying different web pages and content. Site navigation was handled through the html. 3 packages of different code was written for the application to retrieve data from the database, update the database, and display to the user the intended infromation. The first package contains the classes, and the methods associating with that particular class. The classes written were used after the initalization of a javabean for the page written for storing the data (which user selected the movie, what movie it was, the subscriber that movie belonged to, etc.) passed to the page to display information or execute Java code. However, in some cases, when the code was two long to write for the java bean itself, we turned to our third package of code, Servlets. Written to interact with certain buttons/operations, the servelts would tackle certain tasks requiring a longer amount of code. For example, when submitting a rating, if the user clicks submit, the page is redirected to a servelt named properly in the html layout on the original page to execute the intended code. In the case of submitting a rarting, the servlet captures the value the user stores, updates the database with appropriate values, and then updates the website to reflect the changes provided. The final package written for our project was the “database” package, which is filled with .java files that were written with purpose of altering the database as intended. Having these written saves significant time, as the methods split the tasks up of database entry and retrieval quite nicely.

3. *System Models. These are requirements emphasize the interface requirements and*

*relationships among application domain objects and the dynamic nature of individual*

*objects (state diagrams) and the dynamic interaction of users with the system and*

*between.*

**3.1 Functional Requirements**

*.*

User must have a computer with access to our website.

**3.2 Non-Functional and System Requirements**

*List detailed system requirements here. If your system is large, you may wish to break this into*

*several subsections.*

Gilligan’s Coconut TV needs to use SQL to access data located in a database. With this, the program needs to run on Apache Tomcat server. The project uses servlets, jsps, and javabeans, which means there is necessity to have JVM installed that can use these.

**3.3 System Models**

**3.3.1 Scenarios**

*Show several storyboards or specific kinds of uses of your system. Use this to develop your Use*

*Cases of how users interact with the system*

The layout of our project is as follows:

Start at the ‘Start’ page. From this page the user has the ability to sign up, sign-in as well as view the prices/perks of each subscription.

When signed in the next page will be the profile selection screen where a user can select profile they would like to use as well as update the user profiles and account settings.

After a profile is selected, the next page is a profile home screen. From the screen the user is met with their favorite movies, as well as recently downloaded in an eye-appealing display across the screen. The user can then use the search bar to find a title they would like, use the ‘TV’ button on the top left to get a shortcut to their favorite genres of film, or click the button next to it to return to the profile’s main page. Also avaialble on the right size is the ‘person’ button to direct the user to switch/manage profiles. Also present is a logout button to log out.

When a movie is clicked, the ‘Heart’ button underneath the Title’s description provides the user the ability to add the movie to their personal profile’s favorites list. Along with this is a download button that allows the user to download whatever title is selected.

**3.3.2 Use Cases and Interface Requirements**

**3.3.2.1 Scenarios and Use Cases**

*Show all Use Cases and show interface requirements in this section. Where appropriate*

*include screen mockups. If you use mockups, be sure to relate to scenarios and/or Use*

*Cases and explain major features or functions with narrative to avoid confusion or*

*omission of desired features.*

Case: Download a specific title: 21.

At login screen, enter password and username to gain access to account. If the user has no account, click sign up, enter information, and click submit. User will be at profile management screen. User selects their profile or adds a profile if their account allows for them to (base on selected plan) and clicks submit. From here the user is directed to a screen that allows the user to select a movie from a generated recommendation or a favorites list if a user has added favorites to their profile. At this point the user can click the search icon, enter the title ‘21’ into the bar, hit enter, and find the movie as the top result. From here the user clicks on the movie which will direct the user to the movie selected page displaying the movie info. This includes the director, the top 2 actors in the film, the description, as well as a link to the movie trailer. From this page, the user can then click the download button to have their title downloaded.

Case: Delete a person from using the subscribers account.

Login with username and password from login screen. From the profile management screen click the link at the bottom reading “Manage your Profiles”. When clicked each profile will have a button displayed that allows the user to either delete the selected user or update their information.

Case: View the top Action films.

Login in with username and password and select the user profile. From here click the button of the TV at the top corner to produce a drop-down menu. Click ‘Action’, and a list of desired titles will appear.

**3.3.2.2 Summary of Screen Navigation and Screen Mock-ups.**

*Emphasize the relationship between the screens and show a hierarchy of screens where*

*appropriate.*

The screens we use in the project act in system, with the intended desire to help each user browse and select movies with a personalized experience. The first two screens are designed to stress the individualistic experience design. By logging in and selecting a profile, the user now has full capabilities of the site as though the site was designed for them. They can add movies and download features at their leisure, and all the information is passed in front of them as they are the focus of the site. The home page, which is the page the user is taken to directly after the profile selection screen acts as the focal point to the whole application. Once the user gets to the home page, they will use this page as a base to start from, and thus navigate from. Our pages designed to capture the user’s information such as the welcome page, the signup page, and the login page all have the purpose to lead to the home screen. Once the home screen is accessible to the user, it becomes the main page of navigation. All pages can be in some form derived through it, and all paths start from it.

Place Hierarchy Diagrams here:

**3.3.3 Object Model**

*.*

The main menu bar will be present across all pages in the application. It will feature the following buttons that navigate to different pages.

1. Home button: Whatever page you are on, it will return you to the selected profile’s home page.
2. Movies button: will produce a drop-down list that the user can use to select an intended genre, and after selection produce a list of those titles.
3. The search bar: when used will take to the user to a list of titles fitting the values that user provided. Will return a list of movies to choose from based on keyword provided in the search.
4. Profile Management button: When clicked on will take the user to the profile selection screen. At the screen the user can switch profiles, update settings of the account or edit profiles themselves.

The download button: This button will have the universal symbol for downloads, and when clicked will download the title intended if applicable.

The “Favorites” button: This is a heart shaped button that when clicked, will update the user’s profile to reflect that movie in their 3 favorites. There will only be three favorites at a time, so the movie that was favorited the longest time ago will be removed from the user’s favorites list.

The logout button is an arrow pointing the right and will log the user out of their account.

Class UML diagram attached to the bottom.

**3.3.3.1 Data Dictionary**

***These are all the abstract data fields needed for your project. They may or may not map to the design and***

***implementation fields of your code.***

For Clarification: Each class mentions a “Getter and setter” method for each class. In the context of our project a getter returns what value is stored into that variable. For example, getLine1() will return the String value stored in m\_line1. A setter method will take in an object type that is the same as the object type of which the setter sets to, and then sets the variable equal to the passed in object. For example, setLine1(“190 S Prospect”) will set m\_line1 equal to “190 S Prospect”.

Classes:

Address –

Variables: String m\_line1, m\_line2, m\_city, m\_state, m\_zip

Methods: a getter method and setter method for each variable stated above.

CreditCard – Extends the Person class

Variables: **int** m\_cardID, m\_ccv, m\_expYear, m\_expMonth

String m\_ccNumber, m\_ccType

Methods: a getter and setter method for each variable stated above.

Crew –

Variables: **int** m\_crewID

String m\_firstName, m\_lastName

Methods: a getter and setter for each variable stated above.

Constructor: The constructor creates a new Crew member and assign a new ID by using the database.

LoginCredentials-

Variables: String m\_email, m\_password

Methods: a getter and setter for each variable listed above.

passwordChecker(String) – uses a string object to check if what is stored in the password variable is equal to the String the user passed in. Returns a Boolean, either true or false.

Movie-

Variables: **int** m\_movieID

String m\_genre, m\_title

String m\_description, m\_movieImage, m\_movieTrailer

Crew m\_director

String m\_actor1, m\_actor2, m\_date, m\_MPAARating

**int** m\_views, m\_ratingSum, m\_ratingCount, m\_ratingAvg

Methods: A getter and setter for each variable.

Person –

Variables: String m\_firstName, m\_lastName

Methods: A getter and setter for each

Subscriber:

Variables: **int** m\_accountID

String m\_phoneNumber, m\_accountStatus, m\_levelName

Date m\_createDate

Address m\_address

List<Users> m\_userProfiles

LoginCredentials m\_loginInfo

CreditCard m\_paymentInfo

Methods: a getter and setter for each variable stated above.

Users –

Variables: String m\_username, m\_favoriteGenre

**int** m\_accountID

List<Integer> m\_favorites

List<Integer> m\_recents

Crew m\_favoriteCrew

**int** m\_userID

String m\_ageRestriction

Methods: A getter and setter method for each variable listed above

addFavorite(int): this method takes the integer passed in which will be the movie that will be added to favorites and adds it to the front of the favorites list.

removeFavorite(int)- Uses the integer which represents the movie ID number that is to be removed, finds it on the favorites list and then removes it.

addRecent(int)- Checks if the movie ID number is already in the recent’s list and if it is not, will add it to the recents list.

removeRecents(int)- checks to see if the movie ID number passed in is in the recents list, and if so, removes it from the recents list.

DataBase:

Tables:

Card: card ID type int

Account ID type int

creditCardCCV type int

creditCardNumber type int

cardHolderFirstName type String

cardHolderLastName type String

expYear type int

expMonth type String

ccType type type String

Crew: crewID type int

crewFirstName type String

crewLastName type String

Favorites: userID type int

username type String

genrePrefeerence type String

favorite1 type int

favorite2 type int

favorite3 type int

crewperson type String

recent1 type int

recent2 type int

recent3 type int

ageRestriction type String

FilmCrew: movieID type int

crewID type int

actor type int

producer type int

director type int

Level: levelName type String

levelNoDVDperMonth type int

levelChargeperMonth: type String

Movie: movieID type int

movieGenre type String

movieTitle type String

movieDescription type String

movieImage type String

movieTrailor type String

movieReleaseDate type String

movieMPAARating type String

director type String

actor1 type String

actor2 type String

view type int

ratingSum type int

ratingCount type int

ratingAvg type int

Queue: accountID type int

movieID type int

queueSequence type int

movieTitle type String

Quotes: quoteID type int

quoteString type String

movieTitle type String

Subscriber: accountID type int

levelName type String

firstName type String

lastName type String

billAddressLine1 type String

billAddressLine2 type String

billCity type String

billState type String

billZipCode type int

phoneNumber type String

emailAddress type String

memberPassword type String

accountCreateDate type String

accountStatus type String

Users: accountID type int

User1 type int

User2 type int

User3 type int

Classes designed to interact with the DataBase: Each method will have a getConnection method to return the connection value of the database

AdminDB

Methods:

ResultSet getTableData(String tableName)

This method selects the table of whatever name is passed into tablename and returns that table as a resultset

ResultSet getTableData(String tableName, String columnName, String restrictor)

This method similar to its predecessor, returns a result but more scoped, return a specific value being passed in that values location

String[] getTables()

This method returns an Array of Strings of all the table names in the database

crewDB

Methods:

Crew getCrew(String name) **throws** SQLException

Returns a Crew object with all the info from the database of the Crew member with whatever name is passed into it.

**void** addCrew(Crew newCrew) **throws** SQLException

This method adds a Crew member newCrew to the DataBase

favoritesDB

Methods:

**void** addFavorite(Users favUser) **throws** SQLException

This method adds a new a favorite instance to the database from the information provided by the user.

**void** deleteFavorite(Users favDeleteFromUser) **throws** SQLException

Deletes a row from favorites when called using favDeleteFromUser as to which one gets deleted.

**void** deleteFavorite(**int** userID) **throws** SQLException

Same function as the deleteFavorite above method, but uses the users ID number as a way to find the user to be deleted.

**void** updateFavorites(Users updatedUser) **throws** SQLException

This method updates the favorites tables with whatever information the user has stored in it when it is passed to the method

Users getUsersObj(**int** userID) **throws** SQLException

This method returns a user with whatever ID is passed into it

movieDB:

Methods:

**int** getViews(**int** movieID) **throws** SQLException

This method returns the number of views that is stored in the DataBase of the movie with the ID passed in

**void** setViews(**int** movieID, **int** views) **throws** SQLException

This method updates the views in the database with whatever is populated in the movieID with the addition of views.

**int** getRatingAvg(**int** movieID) **throws** SQLException

This method returns the Rating Avg from the datatable of the provided movieID

**void** setRatingAvg(**int** movieID, **int** newRatingAvg) **throws** SQLException

This method updates the Rating Average in the database

**int** getRatingCount(**int** movieID) **throws** SQLException

This method returns the number in the database that holds the Rating count

**void** setRatingCount(**int** movieID, **int** count) **throws** SQLException

This method updates the rating count in the database

**int** getRatingSum(**int** movieID) **throws** SQLException

This method returns the RatingSum of the movie provided in the parameter from the database

**void** setRatingSum(**int** movieID, **int** newSum) **throws** SQLException

This method updates the ratingSum in the database

List<Movie> search(String keyword, String genre, String MPAARating, Integer userRating) **throws** SQLException

This method searches the database for titles with similarities to what is passed in. The results are returned as a list

**void** addRating(**int** movieID, **int** userRating) **throws** SQLException

This method adds a rating value to the database

Movie getMovie(**int** movieID) **throws** SQLException

Returns a Movie object with the values stored in the row that contains the movieID passed in.

**void** addMovie(Movie newMovie) **throws** SQLException

Adds a movie to the database

**void** deleteMovie(**int** movieID) **throws** SQLException

Deletes a movie from the Movie database

QueueDb

Methods

**int** getNumberofRentals(**int** accountID)

This method returns the number of rentals that particular account has already checked out

**boolean** movieCheckedOut(String movieTitle, **int** accountID) **throws** SQLException

This method checks to see if the user has already checked out the passed in movieTitle or not

**boolean** movieCheckedOut(**int** movieID, **int** accountID) **throws** SQLException

Has the same function as the method above, but passes in the movieID instead of the title

**void** rentOutMovie(**int** movieID, String movieTitle, **int** accountID)

This method updates the database that the user has rented out the selected Film

**void** returnMovie(**int** accountID, **int** movieID) **throws** SQLException

This method deletes the information of the movie from the users downloads

QuotesDB

Methods

**int** getRandomQID() **throws** SQLException

This method returns a random quote from the database

String[] getQuote() **throws** SQLException

This method returns a String array pairing the quote from the mvie with the correct movie title

SubsriberDB

Methods

Subscriber getSub(**int** accountID) **throws** SQLException

This method returns all the information of the subscriber from the database from the given accountID

**void** updateSubscriber(Subscriber changedSub) **throws** SQLException

This method updates the database with the Subscriber changedSub’s new information

**void** addSubscriber(Subscriber addedSub) **throws** SQLException

This method adds a new Subscriber to the database

**void** deleteSubsriber(**int** accountID) **throws** SQLException

This method deletes a subscriber object from the database

**void** updateStatus(Subscriber statusChange) **throws** SQLException

This method updates the database with a new status change

**void** updateLevel(Subscriber levelChange) **throws** SQLException

This method updates the the database with any changes to the Subscribers level

**int** loginCheck(String email, String password) **throws** SQLException

This method was used in testing to ensure that the database was fetching the correct results from the login

UsersDb

Methods

**void** updateUser(List<Users> changedUser, **int** accountID) **throws** SQLException

This method updates the database of the specific user with any changes that may have occurred in the application

**void** addUser(List<Users> addedUser, **int** accountID) **throws** SQLException

This method adds a user to the database

**void** deleteUser(**int** accountID, **int** userID) **throws** SQLException

This method deletes a user from the database

String getUserTest(**int** accountID) **throws** SQLException

This method was used in testing as well to make sure the database was returning the correct results

List<Users> getUserList(**int** accountID) **throws** SQLException

This method returned a list of users from the database that matched the subscribers accountID

**3.3.3.2 Class/Object Diagrams**

***These refer to the application domain objects. Use Class and Object UML diagrams. Label all nodes and***

***show the relationships on the arcs. Show aggregation where appropriate and generalization where***

***appropriate****.*

Pictures of our layout, use cases, and UML diagrams at the bottom of the document.

**3.3.4 Dynamic Models**

***These refer to the application domain objects. Sequence and State Diagrams where appropriate.***

**4. Appendices**

*If you wish to append any documents, do so here. You may wish to include some or all of the*

*following:*

• *Personas and scenarios developed for this project*

• *Transcripts of user interviews, observations, or focus groups*

• *Copies of communications which contain user requirements*

• *Original project proposals or other historical documents*

• *Lists of similar projects or products, with notes about how they differ from yours*

• *A list of requirements which were "wish-listed" or marked unfeasible at present*

• *Original screen mockups, if they are relevant*

Wish List:

List of favorite movie quotes displayed at the bottom of every page. Project if the rest of the project is completed in a timely manner. Was able to implement and is displayed on the front page.

**5. Glossary**

*Include a glossary of definitions, acronyms, and abbreviations that might be unfamiliar to some*

*readers, especially technical terms that may not be understood by end-users or domain-specific*

*terms that might not be familiar to developers.*

Users: This refers to whomever is using the application, with a personalized profile.

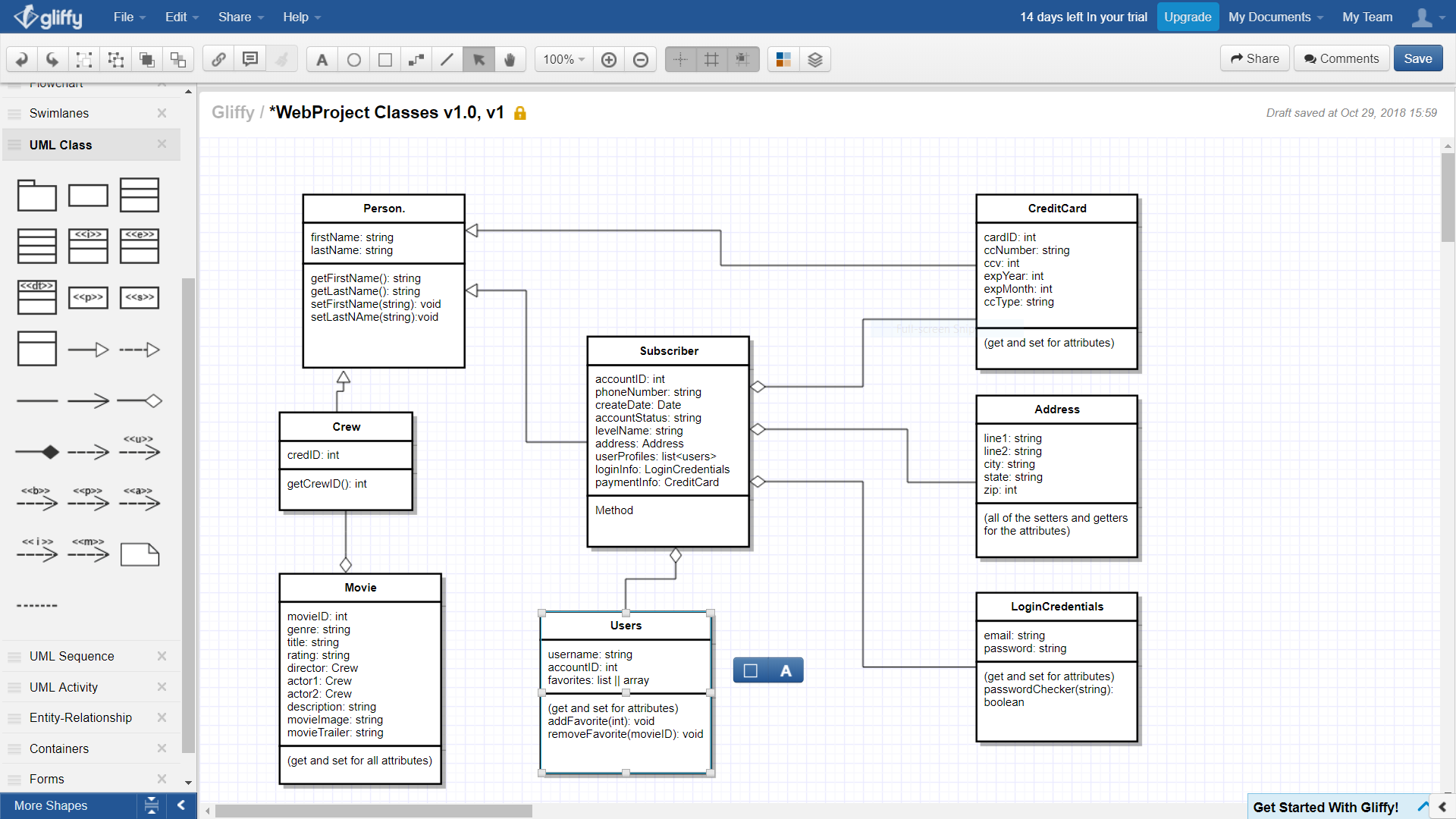
Subscriber: The person who pays for a subscription. They own the account which allows people they decided to have access, have access as a user.

Getter – A method that returns the value stored inside a class’s global variable

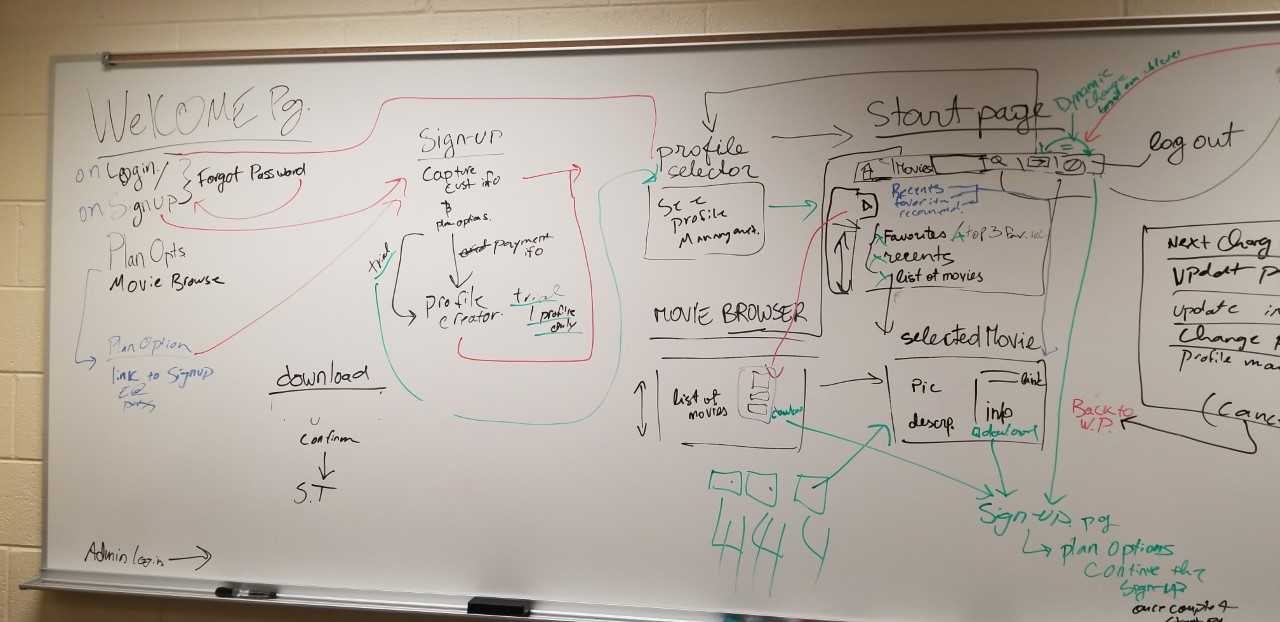
Setter- A method that sets a global variable to the object that is passed into it.

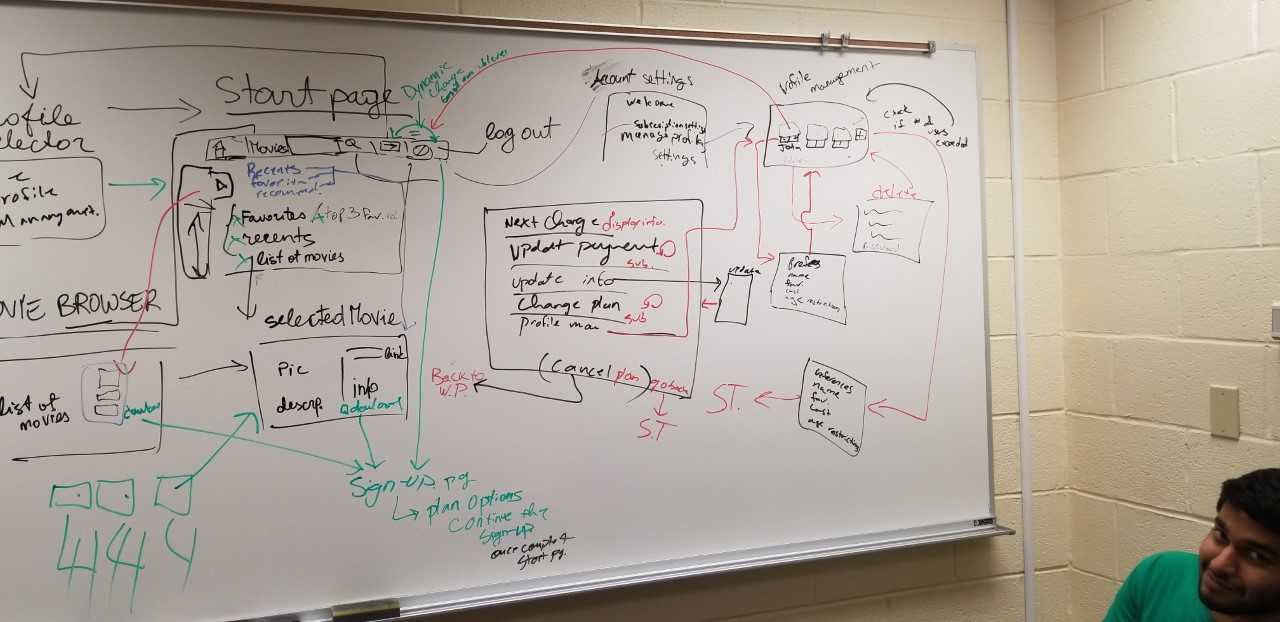
Subscriber- The person who pays for the account, more information is stored here.

Class Diagrams:

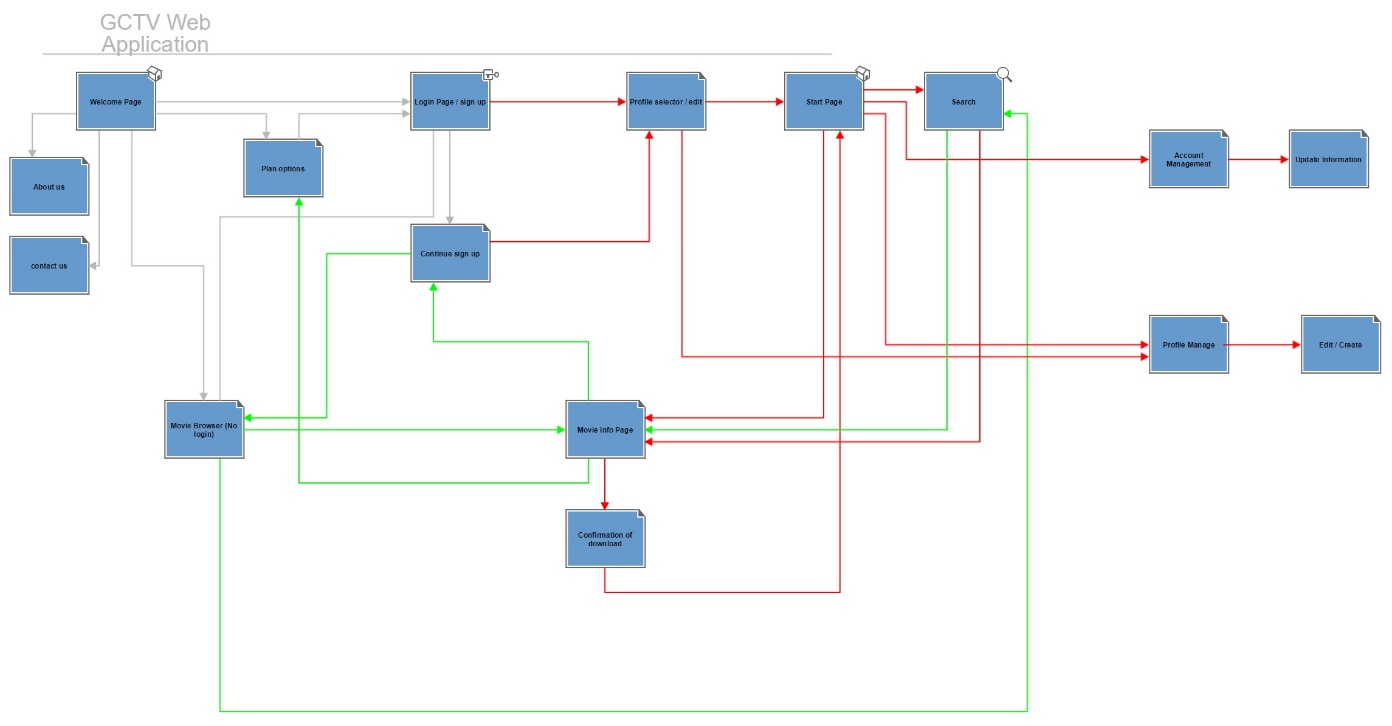


Pictured below is the first draft of the site navigation layout.





This the revised site map layout of the pictures above:



Pic below: A mock-up of the layout our home page will employ.

