**585347**

11

**Movie Review Website**

**An analysis, design, implementation and evaluation of a movie review website**

**Kieran Foxley-Jones**

Table of Contents

[System Process 5](#_Toc293401626)

[Prototyping 5](#_Toc293401627)

[Client Needs and Methodologies 5](#_Toc293401628)

[Prototype One 6](#_Toc293401629)

[Proposal 7](#_Toc293401630)

[Title 7](#_Toc293401631)

[Introduction 7](#_Toc293401632)

[Key Phrases 7](#_Toc293401633)

[Objectives 7](#_Toc293401634)

[Required Resources 7](#_Toc293401635)

[Feasibility 8](#_Toc293401636)

[Software 8](#_Toc293401637)

[Hardware 8](#_Toc293401638)

[Web Browser 8](#_Toc293401639)

[Analysis 9](#_Toc293401640)

[Rottentomatoes.com 9](#_Toc293401641)

[Review System 9](#_Toc293401642)

[Registration System 10](#_Toc293401643)

[Login Page 10](#_Toc293401644)

[User Profile 11](#_Toc293401645)

[Design – Prototype 1 13](#_Toc293401646)

[General Layout 13](#_Toc293401647)

[Registration 14](#_Toc293401648)

[Login 14](#_Toc293401649)

[Movie List Page 14](#_Toc293401650)

[What Is A Hash? 14](file:///C:\Users\Kieran\Documents\My%20Dropbox\EasyPHP-5.3.5.00000000\www\MovieReview_Pro2\Word\Proposal.docx#_Toc293401651)

[Title.php page 15](#_Toc293401652)

[Review Submission 15](#_Toc293401653)

[SQLHandler 16](#_Toc293401654)

[Database design 16](#_Toc293401655)

[Software 17](#_Toc293401656)

[PHP 17](#_Toc293401657)

[Pseudo Code 18](#_Toc293401658)

[Login 18](#_Toc293401659)

[Registration 18](#_Toc293401660)

[Evaluation 19](#_Toc293401661)

[Movie Listing and Details Output 19](#_Toc293401662)

[Prototype Two 20](#_Toc293401663)

[Analysis of Prototype One 21](#_Toc293401664)

[Login and Registration 21](#_Toc293401665)

[Movie Information Output and Storage 22](#_Toc293401666)

[What Is Web Scraping? 22](file:///C:\Users\Kieran\Documents\My%20Dropbox\EasyPHP-5.3.5.00000000\www\MovieReview_Pro2\Word\Proposal.docx#_Toc293401667)

[Review Output and Storage 23](#_Toc293401668)

[Database 23](#_Toc293401669)

[Layout and Structure 24](#_Toc293401670)

[Design 25](#_Toc293401671)

[Layout 25](#_Toc293401672)

[Database 26](#_Toc293401673)

[API Integration 27](#_Toc293401674)

[Movie.search() 27](#_Toc293401675)

[Movie.getInfo() 27](#_Toc293401676)

[Movie.browse() 27](#_Toc293401677)

[TMDb.php – PHP based API Wrapper 27](file:///C:\Users\Kieran\Documents\My%20Dropbox\EasyPHP-5.3.5.00000000\www\MovieReview_Pro2\Word\Proposal.docx#_Toc293401678)

[Search 28](#_Toc293401679)

[Navigation Bar 28](#_Toc293401680)

[General Navigation - $\_GET 28](#_Toc293401681)

[Left Column Content 29](#_Toc293401682)

[Index Page 29](#_Toc293401683)

[Movies Page 29](#_Toc293401684)

[Title Page 30](#_Toc293401685)

[New Review Form 30](#_Toc293401686)

[Miscellaneous Information 30](#_Toc293401687)

[Review Voting System 30](#_Toc293401688)

[Test Plan 31](#_Toc293401689)

[Login Page 31](#_Toc293401690)

[Registration 32](#_Toc293401691)

[Search Submission 32](#_Toc293401692)

[Review Submission 33](#_Toc293401693)

[Evaluation 34](#_Toc293401694)

[Registration 34](#_Toc293401695)

[Review Submission 34](#_Toc293401696)

[Review Voting System 34](#_Toc293401697)

[Movie.addRating() 35](#_Toc293401698)

[API Dependency 35](#_Toc293401699)

[System and User documentation 36](#_Toc293401700)

[Registration 36](#_Toc293401701)

[Login and Logout 36](#_Toc293401702)

[Posting a Review 37](#_Toc293401703)

[Bibliography 38](#_Toc293401704)

[Work Log 39](#_Toc293401705)

# http://as.exeter.ac.uk/media/level1/academicserviceswebsite/aboutus/biss/iws/images/SpiralMethod.jpgSystem Process

# Prototyping

For this project I have decided to use a fast prototyping, I will be using a basic version of the Spiral Methodology, it will allow me to rapidly create prototypes giving me to opportunity to test and evaluate the systems and scripts I will create.

The spiral methodology is an iterative method which allows me to re-evaluate my created systems and redevelop necessary areas.

Each phase begins with analysis of an existing system, or previous prototype allowing you to focus on areas that need serious work, or new concepts and ideas that arise through the process. Design of a prototype follows, allowing me to focus on aspects brought up in the analysis, it also allows me to review what has been created so far. Each phase has analysis of either the initial concept or current created system allowing the developers to evaluate what it is they need to do to hit the deliverable.

The steps in the spiral model are generally as follows:

* The system requirements are defined.
* Design of the intended system
* A first prototype of the new system is constructed; it is usually a basic construction with core systems.
* The first prototype is evaluated and the same steps followed for the second prototype.

## Client Needs and Methodologies

The choosing of a methodology that fits with what the client requires is essential, a client’s needs and what they want will be different between projects and your interaction with the client will need to adapt. For example, if your client needs a system developed that effects multiple parts of the company, you may find that you have multiple people, be they executives, management or users. You will need the input of all these people to create a requirement specification for your system, but they may not truly know what they want, and the overarching system might not be what everyone wants.

Using a methodology such as the Waterfall Model would cause serious problems, because this model doesn’t have a flexible development process, any analysis and questioning take at the start of the process, maybe incorrect, or irrelevant when the development process come to its conclusion, you may find that what you have created does not satisfy any of the people using it or requesting its creation. A more flexible methodology such as the Spiral Model would be appropriate, as you create prototypes and gain insight at multiple stages throughout the process, adjusting or redesigning elements of the system after further analysis of user interactions with the prototypes. The eventual system should be closer to what the client originally envisioned as their input is taken onboard throughout development.

# Prototype One

Proposal

Analysis

Design

Evaluation

# Proposal

## Title

A website that takes user submitted movie reviews and stores them for other people to read.

## Introduction

The project will concern itself with the development of a movie review website, it will include elements of database design, client side and server side scripting, and general web design. There are many site dedicated to movie reviews, but many of these sites emphasise the professional critics point of view, this review site focuses on the average persons experience with a particular film. The finished project would be concerned with its own information and stored data; it should also allow the possible expansion to include off site data. This project will also require me to learn how to create many aspects of the site, for example the querying of a database the use of information queried to generate a new webpage for viewing by a user.

## Key Phrases

Databases, Normalisation, PHP, Javascript, Scripting Security Issues, Website readability.

## Objectives

* To evaluate competitive review websites, and establish what features are critical to similar sites, and what my particular site can do differently to compete.
* Ensure the storage of information is secured, especially personal details and security information (username/passwords)
* Usage of database queries to store, search and categorise user submitted reviews.
* Use of server-side scripting to generate pages from user selection.
* Output of generated pages in a readable format

## Required Resources

PHP 5.3.0, MySQL, Multiple web browsers (Chrome, Firefox, IE), Web server hardware and software (WAMP, XAMPP, EasyPHP).

# Feasibility

## Software

The potential software needed to create this project is reasonably easy to acquire and used. PHP, MySQL, Javascript and general web browsers are all available, for free to anyone who has a computer. Also there are free to use Web Server packages, such as WAMP, EasyPHP and XAMPP that provide a server that PHP scripts run on, and MySQL to create and run a SQL database. The website itself will be very dynamic, meaning a reliance on PHP and MySQL and their interactions with each other. This web server software will need to be installed onto the hardware. PHP itself can be simply created with notepad, although it doesn’t come with formatting for the language. I will personally be using Netbeans IDE with its PHP plugin, this allows me to quickly create new scripts and keep the project together, it also allows me to use a Debug function built into EasyPHP web server, allowing me to step into and see what is happening when the scripts are run.

## Hardware

The website will simple require a powerful enough server which runs the Web Server, servers capable of this are everywhere on the web, and even your home computer can become one by installed a web server. This means during development, I will be able to develop and run the website on my PC without the need to publish to the internet, or be dependent on an online web server. When the site is in a workable state, and live testing will need to be started, publishing to an online web server will be necessary, although a basic server can be had for very little money or even free.

## Web Browser

The website will need to be developed with multiple web browsers in mind, the numerous browsers in the wild generally behave slightly different. The main example of this is the difference between web standard compliant browsers such as Firefox and Chrome, and Internet Explorer version 6-8. Due to decisions made by Microsoft during the early development of the Web and web technologies running their browsers adherence to the accepted standards and way of doing it, well documented workarounds and coding ‘hacks’ are needed in some circumstances to visually output a site the same way across all browsers. This will need to be taken into account, especially if a lot of CSS is used throughout the site.

An example of this is the way Internet Explorer deals with the CSS box model, the W3C standard defines the width of a CSS box, it is separate from the padding, border and margin. However Internet Explorer 6 interprets the width as including the padding and border. This means all CSS boxes will need to take into account this bug, either by creating the boxes with it in mind, or creating a separate IE6 only style sheet.

# Analysis

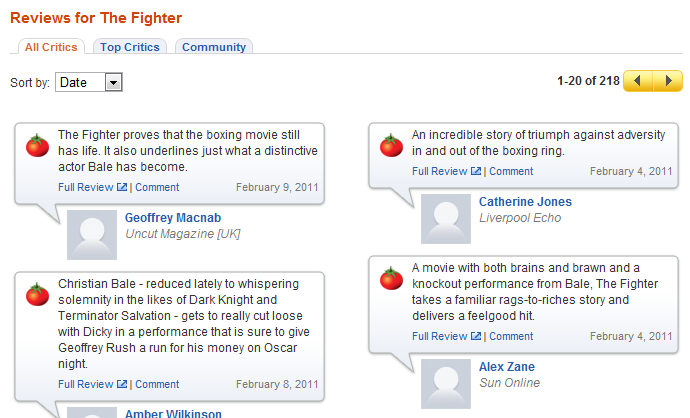
There are a number of other movie review sites that already take and output reviews created by the general public.

## Rottentomatoes.com

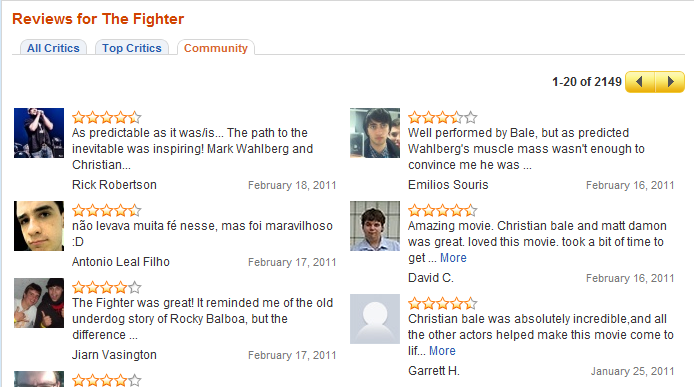
Rotten Tomatoes is one of the biggest review sites on the internet, people tend to refer to this site when they are talking about new or recent movies. A lot of message boards and forums I have researched, such as Something Awful and Reddit, use rotten tomatoes’ fresh and rotten marker as a general quality marker in discussions of movies.

### Review System

Rotten Tomatoes takes an aggregate of review scores from multiple sources, and displays it along with the subject movie, as either fresh or rotten with their percentage mark. These sources include professional reviews from published critics, along with user submitted ones.



When you choose a particular movie, you get a multitude of information, with links to the full reviews of the professional critics and a short summary on the page itself. This allows you to quickly ascertain what professional critics say about the movie, and also head straight to their published review. This gives a nice idea on what you may like about it

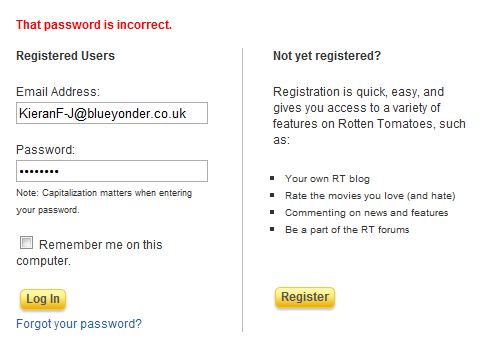


You can also get links and summaries of community posted reviews in a separate tab and what they thought in multiple forms (star rating, words). You can read these community reviews just like the critic reviews. Clicking through into these reviews takes you to the selected persons profile, where you can see what other films they have commented on, which is an idea to consider for my site.

### Registration System

Rotten Tomatoes registration page is simple, and typical for many websites. You enter your email address twice for verification, again with passwords and your first name and last name. There is no username field as your email is used for logins. Gender, date of birth and Country are also included possibly for data collection reasons or some kind of script dedicated to sorting data retrieved when viewing a movie page (reviews by country, DOB etc).

Also present is a checkbox for the understanding of a user agreement and privacy policy. This is important when storing personal details and necessary should any legal or technical problems arise.

I can safely assume these details are stored within a database, and the password itself has some sort of security possibly using some form of hashing and salting.

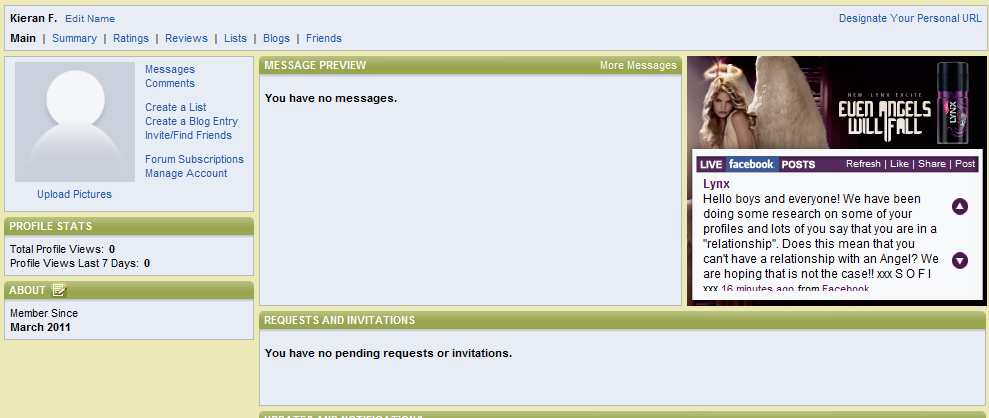
### Login Page

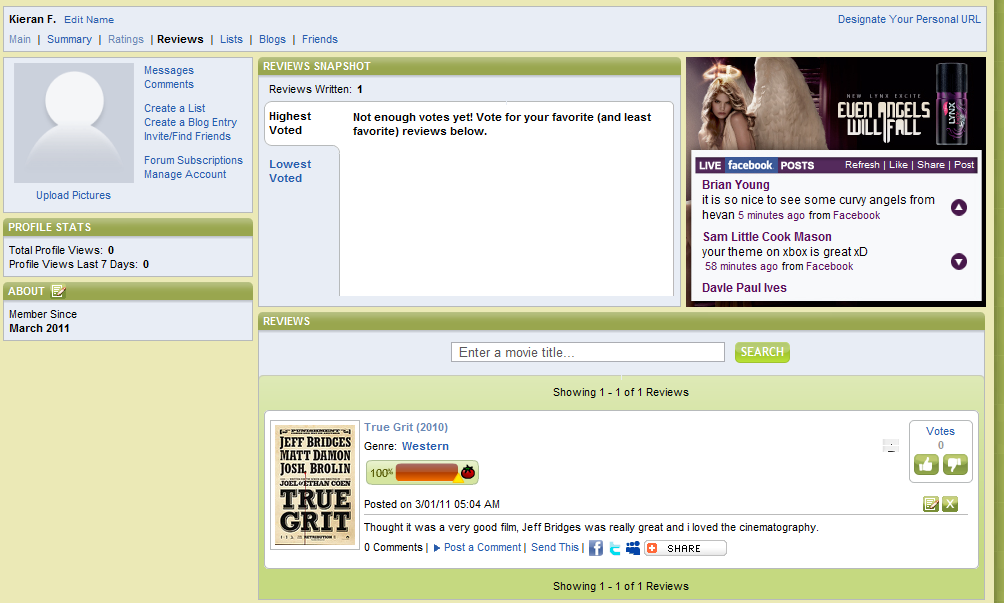
The login page is pretty simple, a simple well laid out login form, with “remember me” check box and a forgotten password link. Also on the page is a link to register an account, this allows the main front page to have less confusing links to a new user. When a new user arrives on the site and wants to make an account, they do not have to choose between a link straight to registration and a link to login.

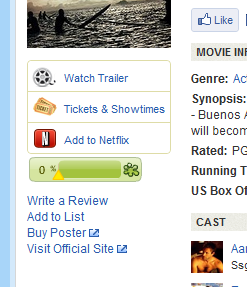
I believe I can safely assume that this login page sends the user inputs to a PHP script, which possibly reverses the process used to provide security for the user passwords (hash, salt), and this hash will be stored into a secured database.

### User Profile

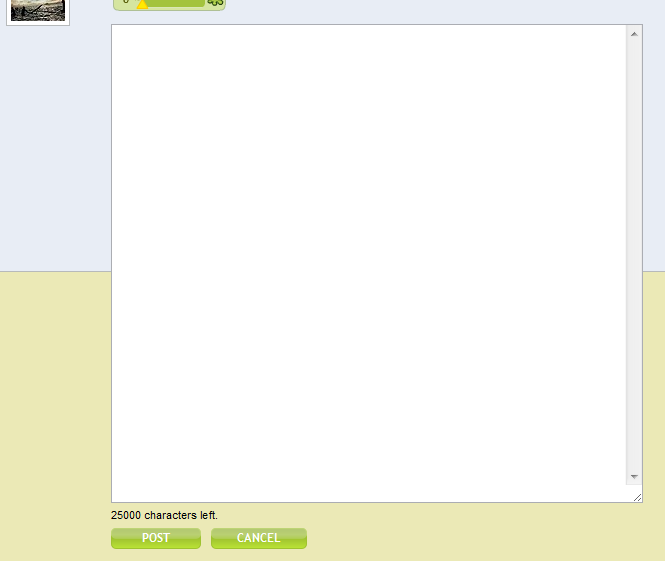
The user profile is pretty standard for sites like RottenTomatoes, it provides you with access to your public available information, reviews you have posted, statistics on ratings you have given, your own mini blog and access to friends list.

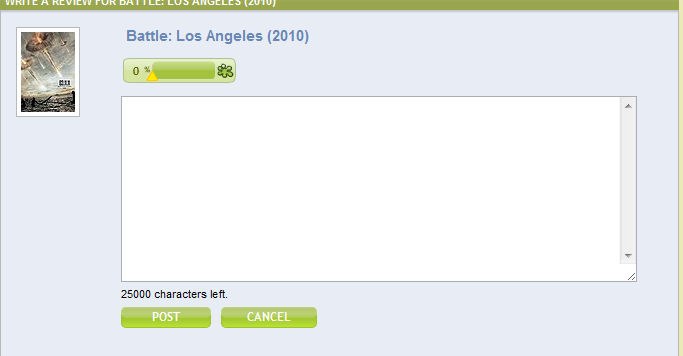


Any reviews you have posted show up on the reviews part of your profile, allowing you to edit or delete them at will. This system is also used for the friends reviews page, except you cannot edit or delete those reviews, but you can comment and vote up or down the review. This system is more than adequate for a website of this type, and it presents the information in a clean, accessible way.

**Posting a Review**

This is simply handled by going into the movie you want to post a review of and clicking on the “Write a Review” link. This will take you to another page with a text area and flash based, rating percentage chooser. You simple type your review, choose what percent you give the movie, and submit. There is a character limit, with 25000 but this seems more than sufficient for any review a standard user would submit. The one downside I would point out, is given that this is website based around reviews of movies, the actual “Write a Review” link isn’t in the most prominent position. It took me a while to notice where it was on my first time at the site.

One useful feature of the text area used is that you can dynamically resize it, and make it a lot longer than it initially is.

Design – Prototype 1

The first prototype will be very simple and will concentrate on creating the bare essentials, the visual design of the site will be sparse, and barely there. The intention is to finish this prototype with:

* a working registration and login system
* session control so users can move from one page to another without repeated logins
* a basic review upload system allowing users to write reviews and save them to the server
* have a movie page output any reviews stored for it

## General Layout

The general layout of the first prototype will be a simple affair; it will initially be built to incorporate my intended layout with a header bar, small navigation bar below that, a left column and main content area, and footer bar. Specific sizes of these areas will likely change throughout the prototyping process, but the basic layout will stay as such.

The navigation bar will house links to login, registration and display whether you are already logged in or not.

Things which may not make Prototype 1 before the deadline will include a search box, genre listing, and any kind of visual logo or SEO. This is to ensure that the focus of this prototype is the core backend, with emphasis on the review input and output.

## Registration

### What Is A Hash?

*A hash (also called a hash code, digest, or message digest) can be thought of as the digital fingerprint of a piece of data. You can easily generate a fixed length hash for any text string using a one-way mathematical process. It is next to impossible to (efficiently) recover the original text from a hash alone. It is also vastly unlikely that any different text string will give you an identical hash - a 'hash collision'. These properties make hashes ideally suited for storing your application's passwords. Why? Because although an attacker may compromise a part of your system and reveal your list of password hashes, they can't determine from the hashes alone what the real passwords are.*

(McGlinn)

This prototypes registration page will be simple, I do not intend to implement many security and validation features at this early stage, JavaScript validation on user inputs may come in future version, and the same can be said for a image based Captcha system to avoid repeated registration from automated external sources.

The registration form itself will be left in an unformatted state whilst I concentrate on the important task of creating essential systems for the website. Registration will consist of a Username and Password field, when the user submits the required information; it will be posted to a PHP script. This script will take the values for password, and apply a sha256 hash to it; it will also generate a random string of letters and use that as a salt for the now hashed password. This salt and hash combo will again be hashed and the result will be stored as the password within the database.

This provides some security to an outside attack, if someone gains authorised access to the database, all they will have available to them is a string of 64 characters which would not give any hints to what the original password is. Combine this with the salting of the password, and even brute force attacks, which know what the end hash is, would be problematic.

## Login

Login will appear to be a simple affair, with a login form available on all standard pages (index, movie list, movie specific etc) should a user not be logged in. The act of login will send the user inputted username and password to a login script; this script will search the user table of the database for the username. It will then take the password hash and stored salt and use them to check the user input password. The script will take the password, hash it, add the salt to this hash, and then hash again, just like in the registration script. This hash is compared to the stored hash and a decision is made to whether it is a valid password.

If it is valid then the script also creates a session and keeps the user logged in during their visit.

## Movie List Page

The movie list page will have a small script which queries the database for movie names and MovieID’s, it will then output this data as hypertext links. The movie name will make up the visual link, however the link itself will contain the movie ID, using PHP you can append to a URL a PHP variable as so:

<a href="title.php?Mov\_ID=<?php echo $movieNames[$i]['Mov\_ID'];?>">

This will create a URL of titple.php?Mov\_ID=1 if the Mov\_ID variable is 1, this allows the title page to pick up this variable using $\_GET[‘Mov\_ID’]; allowing the page to output dynamically. The movie list itself will have a loop outputting every single movie name and id stored within the movie database.

## Title.php page

This page contains the most potential content, the intention will be to take the Mov\_ID variable passed to the script through the use of $\_GET and the URL, as explained in the movie list page section. This Mov\_ID will then be used to extract all the information stored within the database relevant to the movie selected.

Within the database the movie title, description, year of release and other information will be stored, and it is the job of the Title.php page scripting to query the database the retrieve this information, and output it to the page. The reviews will be output in much the same way, however to keep the correct reviews showing up, the SQL query uses table linking though the link table.

## Review Submission

The form for submitting reviews will be located within the title.php page. It will appear at the bottom of the page after all of the review output. When the user has typed up their review, they then select the rating they want to give the movie and press submit. The inputs will be then sent to the reviewup.php script via $\_POST. This script take the content of the review and rating, the userID from $\_SESSION and the movieID from the URL and $\_GET. It then stores the content and rating into the review database, which auto generates a review ID, this gets returned through the use of the SQLHandler to the reviewup.php script. The script then stores the review ID, movie ID and user ID into the link table of the database.

Due to potential errors caused by punctuation marks combined with the SQL statements I will need to implement a function to parse the input string and escape any problematic characters, in this case apostrophes. The PHP function, mysql\_escape\_string(), parses the string a adds a \ to any apostrophes that need it, this stops the string contents from interfering with the SQL statement when it is passed to the SQL server.

## SQLHandler

The SQLHandler script was given to me by Kevin Woodard, it contains functions that automatically call a connection to the SQL database, with information stored within the script itself, and it also contains functions to handle all types of queries that would crop up, also handling any errors that may appear. This allows you me to use one script file to connect to and query throughout the site, rather than having to retype database connection code and any error handling. This will make the code through the site easier to read for everyone.

## Database design

When a user posts a new review, an insert query inserts the body of content from the review into the review table, along with the user rating. This auto generates a Review ID number, stored in the Rev\_ID field. This is then retrieved through a select query and input into the link table, along with Movie ID (Mov\_ID) and User ID (User\_ID). This provides the necessary link between users, reviews and movies.

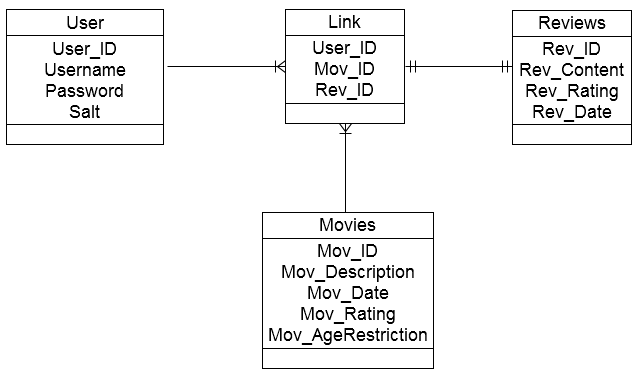
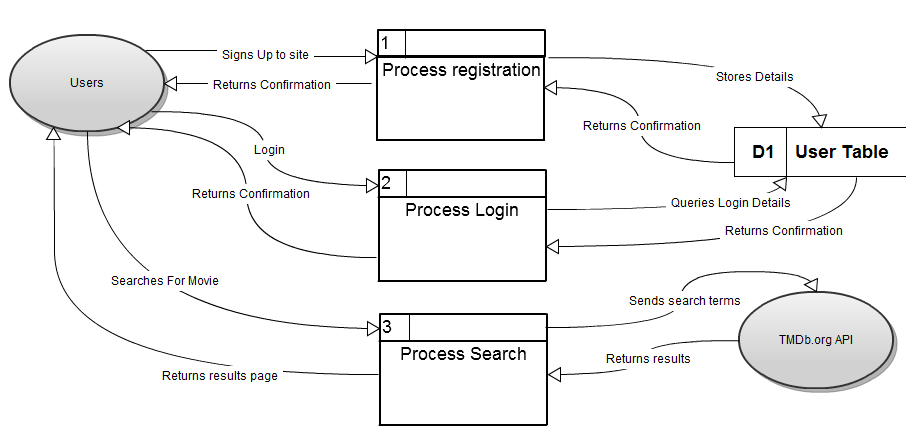
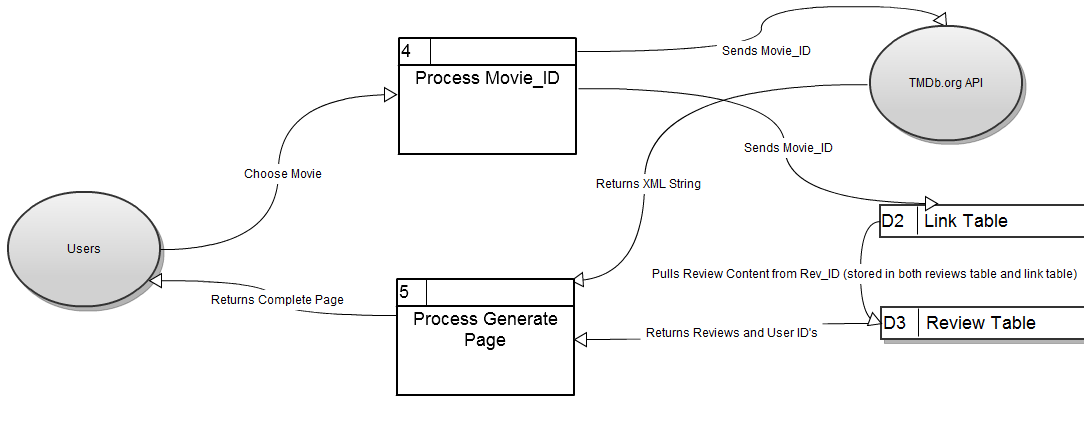
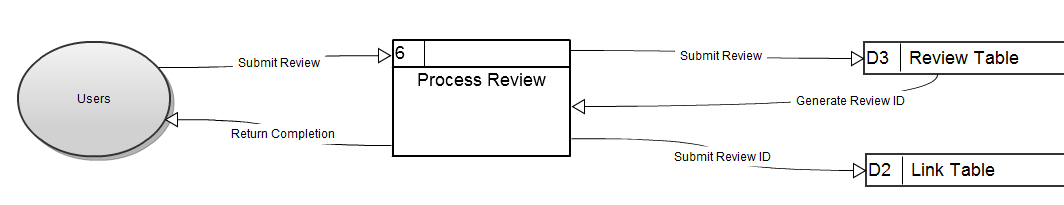


Figure ERD diagram

# Data Flow Diagram



# Software

## PHP

PHP is the main server-side scripting language used in modern web development; it is used extensively on many website that contain a personalised experience. One of the biggest websites in the world, Facebook, uses PHP throughout the site, their PHP scripts take data from databases and formats outputs the result a HTML page, which is then sent to the clients web browser.

If you were to look at the source of this page, you would just see a standard HTML page possibly with some Javascript or similar client side scripting language. This highlights one advantage of PHP, because it is server-side the client never sees the actual script being run on the server, and so a malicious user can’t alter the code to do something detrimental to the server.

The language also has great support due to its nature as open source; the main PHP website (PHP.net) for example has the ability to make comments on features and functions of the language, providing real life examples of how they are used.



Version Changes are one disadvantage of PHP, as the language evolves and major updates are released, functions and other assorted features get changed and deprecated, this could cause problems with your website, should you upgrade to a new version, although this can be avoided by just not updating unless necessary. The error handling of PHP also leave a lot to be desired, generally just outputting a generic 404 of 500 error should the page fall over, this can be alleviated by using a pre-existing error handling script, many of which are free to use from the internet, or creating your own.

# Pseudo Code

### Login

Input Username and Password

Connect to Database

Create query (select password, salt where username = input username)

Send query to MySQL server

Receive results

If number of results < 1 then

Error, no such user

Else

Hash user input password

Add salt to hash

Hash salthash

Check subsequent hash against hash received from database

If scripthash != databasehash

Send page to registration

Else

Create session,

Send username, password and user ID to session control

Send page back to index.php

End script

### Registration

Input desired username and password

Connect to database

Create Salt (random 3 characters)

Hash password

Add salt to password

Hash saltpassword

Create query storing username, hash and salt

Return to Index.php

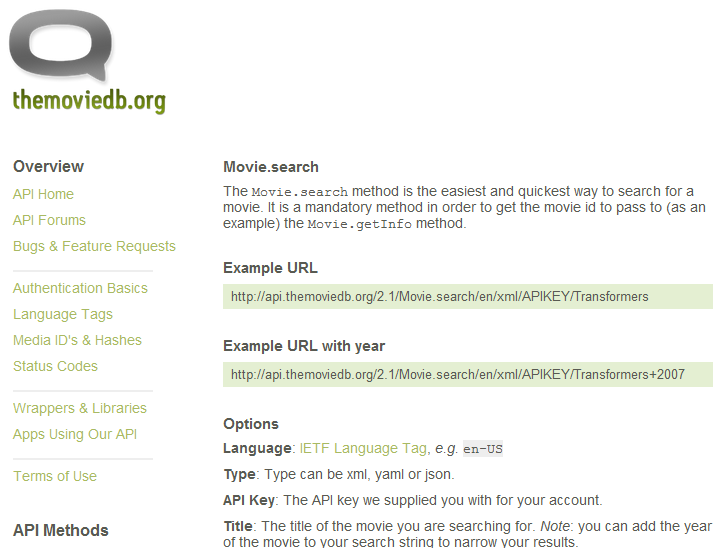
# Evaluation

The development of prototype one was a massive learning experience, as I had, previous to this project, very little PHP experience outside of simple login scripts, and virtually no experience with SQL and relational databases. I have had to practically learn from scratch all the elements needed to create this first prototype, and given the circumstance I believe I have done extremely well. That is not to say there are not problems with Prototype One, but the core fundamentals are present and working.

## Movie Listing and Details Output

The idea when designing the site was to store all information, data, images etc of individual movies on my own server and within my database. This seemed sensible at the start of the project, but it soon became apparent, as the project ran, that this would be unfeasible for a one person project. The sheer amount of data entry, and storage requirements would be far too much for me to handle, considering I also had to develop the other core systems of the site. I came to the conclusion that there are other online resources that could provide me with what I need, I first looked at the International Movie Database (IMDB.com) website, which is considered the standard other movie information websites are measured by. IMDB.com however does not supply their database access to external entities for free, so I looked into some kind of script that scrapes the information from the generated WebPages themselves.

There were some readymade scripts that did this available on the internet, such as one created by Fabian Beiner, based off an original script by David Walsh. This script allowed the user to take all the information from an IMDB.com generated page, including description, dates, actors, image links etc. The problem the script creator highlighted is that IMDB.com does not give permission to do this, so I had to drop the idea, although I did take what information I had received from the script and used it for basic testing.

One idea to overcome this major issue is to use the Open Movie Database (TMDB.org) API to retrieve the information I need. TMDB.org allows open use of its resources, including all information you are able to glean from its freely available API, it also provides PHP wrappers (along with other languages) to access it, which is perfect for my use. This API would allow access to Movie search, output of movies lists by date, genre etc, fetching of information and all manner of methods. The only requirement is that I would need to implement some kind of reference back their site and attribution for all content used gleaned from their database.

# Prototype Two

Analysis

Design

Evaluation

# Analysis of Prototype One

I used prototype one as a learning experience to teach myself how to use PHP and MySQL effectively. Many backend systems were created, such as the database and its constituent tables, the review input and output scripts and movie information scripts. All of these systems worked well enough for inclusion into the prototype, but many of these systems had shortcomings.

## Login and Registration

Login and Registration is a very basic affair within Prototype One, the current system actually allows you to log in even if you do not enter anything into the text boxes provided, a serious problem which will need to be addressed. This affects the review system as it then allows a user with a User\_ID of ‘0’ to actually upload reviews, which then get output. The login and registration also only deal with just the Username and Password, basic requirements, but I feel the site needs more user information. Some sites allow the user to pick their current location, and also upload a small image as a Profile picture or Avatar for their account.

This User profile page should then give me a good basis to create an interface for the user to see all of their submitted reviews, and allow them to edit. I could also implement some kind of social feature allowing them to add users as friends, and have those users’ recent reviews, be listed for reading. This would be best left for development after Prototype Two is completed due to its non-core functionality for the website, it would be a good idea to have, but it is not essential to the running of the website.

## Movie Information Output and Storage

### What Is Web Scraping?

*Put simply, web scraping involves taking content from another website and republishing it on your own website. This can be done through a variety of means, some of them more underhanded than others:*

*You could do a manual cut and paste, taking the content from another website and pasting it up on your own website with your own name on it, offering no credit to the website where the material originally came from.*

*You could also grab someone’s RSS feed and publish it on your website in its entirety, thus at least giving credit to the originating website, though not giving your readers much reason to go and visit them (after all, if I can see all the content here, why go there?)*

(Hammer)

The main problem with the storing of movie information is that there are a lot of movies to store information of, be it title, dates, actors, description, images etc. This would require a massive amount of work to input all this information for tens of thousands of movies. Each movie would need to be individually attended to, to allow the users to submit reviews, you cannot submit a review to a released movie if it doesn’t current exist in the database. One solution to this problem was to ‘scrape’ information from the International Movie Database (IMDB.com) web pages, going through each individual movie ID and storing the required information into my own database. The problem with scraping IMDB.com’s website is that the specifically do not allow anyone to do this, this is noted in their own Terms and Conditions. IMDB also doesn’t provide any public way for people to access their database, with private access being sold for +$15,000. These two problems killed the idea of using scraping, even though it worked surprisingly well.

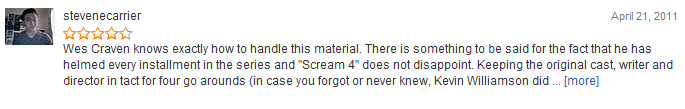
The next idea, during some research into the problem, was to use an open movie database, such as the once located at The Movie Database (TMDB.org). TMDB is an open source database maintained primarily by users who submit, update and edit movie information. TMDB offer free and open access to their database, funded through donations, they have also created an API for people to access the information.

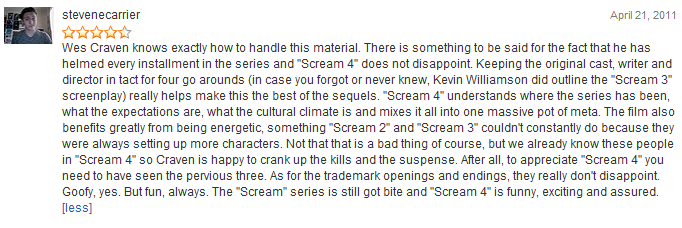
Through researching this API I believe it will be a very good idea to implement this into my review website, it provides all the information Prototype One uses, plus information of genres, actor listings, links to posters and movie images. Implementing this API will allow me to quickly provide content on my site, but also easily expand the site. The API itself allows the searching and browsing of the database, returning results based on information sent to it. The return is sent as a sting in an XML format, and this is easily converted to XML for easy processing in PHP.

## Review Output and Storage

Storage of the review should continue in prototype two as it is in prototype one, a couple of new field will be required however due to my intention to implement a review rating system. This will allow viewers of the website to vote up or down a specific review they have read, allowing that review to change position when it is output to the movie page. This will have the effect of having the highest rated reviews to be output first on a movie’s page, and lower rated one will be much lower down.

I will also need to put in a client side system that truncates a review, having the full sized reviews fully visible will stretch the page height far too much, dropping lower reviews far down below the page fold, especially if there are many reviews with large amounts of text. Truncating the review outputs will allow the user to skim the initial paragraph of a review and allow them to click a button or link to automatically show the rest of the review. This could be achieved using Javascript and CSS library, allowing the text to be revealed elegantly and without a page load.





## Database

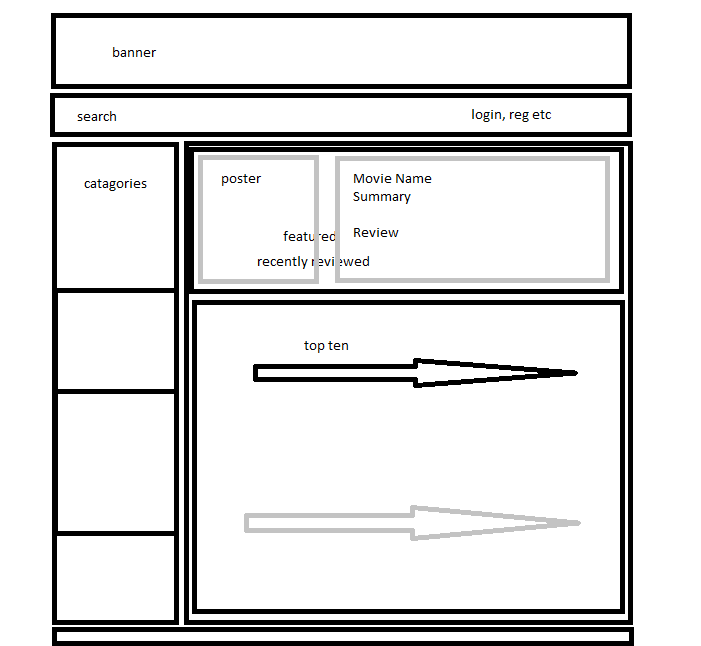
The database for prototype two will need some changes to adapt it to the new API. The Movies table as it stands will cease to exist, as I will not be initially containing the movie information, although at a later data I may change this to cache any movie that has a review posted to it, for expedience sake and to save bandwidth. Some of the other tables will need new field to incorporate new features, such as the review rating system.

## Layout and Structure

Prototype One had a very messy structure within the PHP scripts, it very much represented my continued learning during its construction with code snippets and a generally odd, unstructured feel to code. Although the code itself worked to an extent, it definitely felt untidy. With Prototype Two, intend to create a solid HTML and CSS structure before porting over the code to it, separating the PHP code into their own functions and files, removing critical code from the layout where possible.

The general CSS layout was a bog standard header/left column/content/footer setup, with barely any formatting apart from what was necessary. Scripts outputting into the DIVs would regularly break the formatting of the site, and I feel it is paramount to get the width and height of the various elements set in stone.

A way to navigate around the site will also be essential, as I will need to list links to genre categories and alphabetical lists, containing this within the existing left column element would eventually make the page far too high, and push a lot of the essential links down below the general page fold. A solution to this would be to use an initially hidden drop down, or accordion style menu to hold the links, keeping the layout of the site tidy, whilst allowing easy, quick access to the menus. This could all be contained within the NavBar just below the current header area, leaving the left column area for outputting of items of interest (Top Reviews, most recently reviewed, upcoming movies etc).

Also essential is setting the elements to provide enough space for the outputs, and providing space for any future expansion. The site will be dynamic, so search results could quickly move into double digits with an inaccurate search query.

# Design

## Layout

Using the existing layout as a general guide, I have decided to go with this general layout:

This will allow me to contain the outputs into their relevant divisions.

## Database

Main Content

Footer

Left Column

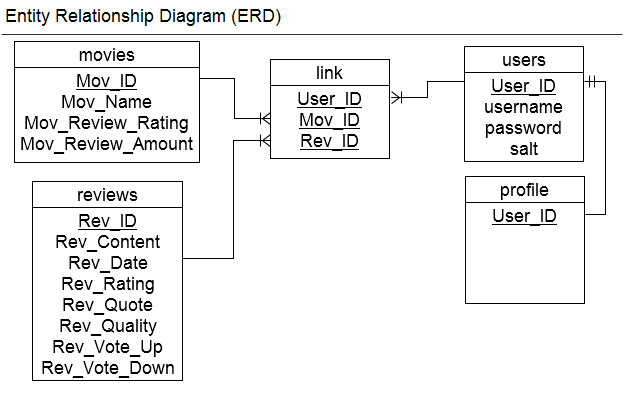
Navigation Bar

Header

With the plan to implement the TMDB.org API, the existing database will need to go through some significant changes, the existing movies table will be dropped and a new movies table put in its place. All code relating to the previous movies table will not be needed, instead the function will be supplanted by the TMDB.org API, the new movies table will hold basic information on the Movie ID gleaned from the API, its name, total amounts of reviews, and the sum of their ratings. This will eventually be used to output an overall rating using the value stored in rating , divided by number of reviews, so a movie with five reviews and an over rating of 40 will have, on output to its webpage, a star rating of 8 stars.

There are also additions to the reviews table, mainly dealing with the proposed review rating system, voting up or down a review will change the values held within Rev\_Quality, Rev\_Vote\_Up and Rev\_Vote\_Down. Rev\_Quote will hold a small, written summary of the user’s review; this will be used so when users are scanning through the review of a movie, they can quickly discern the tone of the review.

Also included on the ERD is a table called profile, this can be used to store information and options the users has chosen to use, one potential field could be user email. This table hasn’t been developed as far as the other tables have due to its apparent separation to the general function of the whole site. If i have time to spare before the end of prototype twos development, I may revisit it, otherwise it will be left until either the final version or subsequent development post-hand in.



## API Integration

### TMDb.php – PHP based API Wrapper

*During my research into the capabilities and usage of the TMDb.org API, I came across a readymade PHP script allowing quick access to the API, by simply passing a value to a PHP function contained within the PHP script. It was created by Jonas De Smet, a Belgian Web developer, and posted up onto GitHub.com for free use located at* [*https://github.com/glamorous/TMDb-PHP-API*](https://github.com/glamorous/TMDb-PHP-API) *. It provides access to the many functions available from the API, among them include movie browsing, search, actor listing, and genre listing.*

The integration of the TMDb.org API will be made easier through the use of a publicly available PHP wrapper. Using this will allow me to use the functions of the API to retrieve and output a large amount of data which I can use to provide content on my website, without the prohibitive amount of time to input or scrape that data myself. The API itself is in constant development, however new features or changes to existing ones happen irregularly so adapting my web site to use it will not necessarily cause many problems.

Specific functions that I will be essential include

### Movie.search()

The search function will allow me to send an input from my website to the API. The API will then send back to my web server an XML string which will need to be converted into an XML object using the PHP function SimpleXML(). I can then output a large amount of data which includes the movie titles, descriptions, images, posters, monetary revenue and budget. The search will primarily be used to output the movie titles and posters into a list for the users to look through and pick out the one which seems more relevant to them.

### Movie.getInfo()

This function will simple take an ID, be it a TMDB.org movie ID, or IMDB.com movie ID and send back in XML format all information contained on that ID. This will be used to output the movie information into the individual movie page, along with any relevant reviews pertaining to that movie.

### Movie.browse()

Movie.browse() can be use to retrieve specific information using a range of requested data, such as a Top 10 list of action movies, the API will send back data on the top rated movies that have been tagged as action movies within its database. You can adjust this to add or remove genres, require a certain amount of individual ratings for movies (i.e. needs 20 ratings to be considered) and limits on how many movies to return information.

### Movie.addRating()

Movie.addRating() can be used to upload an integer between 0 and 10 to the API to effect TMDb’s rating for that particular movie. This may allow me to allow my users to keep TMDb’s rating system up-to-date, causing the knock on effect of changing my rating based outputs over time.

## Search

Movie searching will be easy to implement into the website, due to the use of the TMDb.org API and the PHP Wrapper by Jonas De Smet. It provides a Search function, returning results based on the users submission. My own script will use this wrapper to retrieve the search data from TMDb.org and then output the results in movie poster form, with associated Mov\_ID links and names, allowing the user to click on the poster or name, and move straight to that movie’s page.

I will implement this by including on the Navigation Bar a text box specifically for searching, this will allow the user to quickly search no matter what page they are on. This text box, when submitted will fire off a script that takes the input, and uses the PHP wrapper to send that input to the API. On the return of data from the API I will convert the string into XML format. Using this I can quickly output, using while loops, multiple movie results including Title, Poster images and Movie ID’s allowing me to create links using $\_GET to pass the Movie ID’s to the individual movie page script.

## Navigation Bar

The navigation bar will mainly contain links to specific movie lists, such as the top rated Action movies, or Top 100 movies of all time. Due to the way the site is integrated with the TMDB.org API, these links can be very quickly and easily be created, simply by copying code from one function, to another and changing some attributes (genre numbers, number of ratings, release date, year etc). Due to the potential of this, the development of the navigation bar will be very fluid and creating a set design specifically for it will, in my opinion, be a wasted effort. The only specific design idea to keep in mind is to keep the navigation uncluttered, and this may call for css/javascript dropdown menus.

## General Navigation - $\_GET

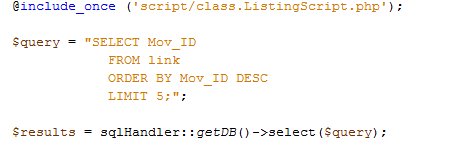
To move from page to page and have the content change depending on what brought you to that page (specifically Movie ID) will involve the use of passing values from page to page using the URL and $\_GET method.

Using PHP you can create URL’s with variables output into them, and when you click on those URL’s , a script at the other end will pick up the variables and then create a page based on that. This works similarly to $\_POST except the variables passed from page to page are located within the URL. To create a URL will simple involve echoing a <a href> tag as such:

Echo <a href=\"title.php?Mov\_ID=".$xml->movies->movie[$i]->id."\">;

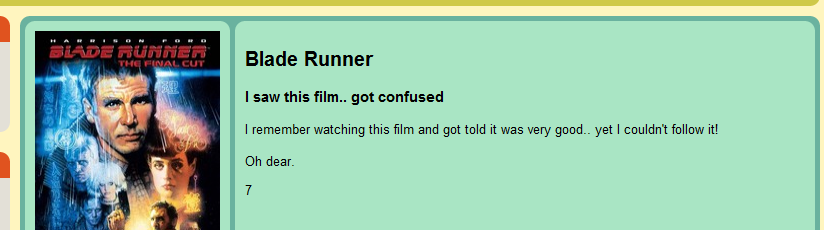
This will create a URL of title.php?Mov\_ID=55555 should the variable contained within =".$xml->movies->movie[$i]->id be 5555. When clicked the web server will take the Mov\_ID variable from the URL and generate the page according to that. So Mov\_ID=5555 will create a page for that movie, including all the information from the API and any review related to that Mov\_ID.

## Left Column Content

The left column will contain some generated content pulled from the database; this will include Top Five films and recent reviews. When a review is submitted the database will be updated with that review. The scripts used in the left column content bar will query the five newest reviews entered into the database, and output the movie name and ID’s based on that query.

The Top Five movie part of the bar will need to query the new movies table in the database, to get the five highest rated movies from the Mov\_Rating column. As this column is updated, so will the Top Five ranking and the output of area.

## Index Page

The index page content will feature the Latest Review first and foremost at the top of the content element. This will simple take the newest entry within the link table, retrieve the Movie ID, Name and the particular review, and output them to that space. Using the Movie ID it will also retrieve the poster for the movie from the API, outputting it to the page, along with the relevant review information next to it.

In the rest of the space on this page, I can output pretty much anything, but my initial idea will be to output a Top Ten movies across all genres, using the API this will be made easy ,especially with the Movies.browse() function. This can easily be changed, so designing a specific function for this area before development commences won’t be necessary and development of this area can be fluid. Eventual content could include, upcoming movies, newest movies, trailers to featured movies, possible advertisement etc.

## Movies Page

The content output to this page will depend on the link used to get to it, for example using the search box to find a movie will take you to a movie.php page that contains the results derived from the API, output in a standard format using $\_POST variables. It will also make use of $\_GET variables when you use the navigation bar to select a genre for example, the script will then output the top ten movies for that particular genre, all output using posters and text with links to the title.php page with a URL variable containing the movie’s ID number.

## Title Page

The title page will consist of the same basic functionality of Prototype One, but with the new overall layout being used throughout the website. There will be some addition and changes to the Title page consisting of:

### New Review Form

There will be a new way of submitting reviews to the website, rather than have a static review form located at the bottom of the page, I will implement a form that is initially hidden away from view using jQuery. This form will be accessed by clicking on a small link or button placed in a prominent position, and when clicked will fade the form into the page, allowing the user to type up their review. The form will require a small summary, 0-10 star rating using images and Javascript to select how many stars, and the review content itself. Also visible will be submit and cancel buttons.

### Miscellaneous Information

This small area will include information such as links to a movie trailer for that movie, budget and revenue, a link to the review form and any other miscellaneous information such as social network links may be contained here.

### Review Voting System

The reviews will need a form of voting so they can be ranked with regards to their relative rating, popular reviews should be output first before the lower down ones, to avoid the problem of having bad or lacking reviews from output as the main content. This will require implementing couple of buttons or elements that fire off some Javascript code that then adds or removes a vote from that review. Through the use of Ajax and jQuery this can be easily implemented, and avoid the problem of having to refresh the page to send the new vote and receive the new total. The Ajax component will send a +1 for vote up or +1 for vote down, to a PHP script which will then update the database accordingly. On completion of this update, it will return a completion message and the new review rating will be visible for the current user.

# Test Plan

Due to the nature of the project, a static test plan performed at set times and dates will be unworkable, many systems and scripts need to be tested as they are created to make sure they actually work, and because they have been tested before formal implementation they will tend to not need thorough testing again.

There are not many user inputs on the website, as most links and navigation are system generated, however the user inputs that are available will need to be thoroughly tested, especially the login system, to ensure that no one can login without a correct username/password combination.

API testing will be a simple affair, as the movie ID numbers and titles all come from the API results together in XML format, it would be very hard to accidently attribute one movies ID number with another’s title/poster etc. So the ID number for “Blade Runner” will always come with a “Blade Runner” Poster, and its correct Title. The same can be said for the movie’s information when it is output to the title page. Outputting reviews to their relevant movie’s page will also need to be tested along with the review voting system, with the lack of inputs this should be quite simple, but essential to make sure everything it output correctly.

### Login Page

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ref No. | Test | Data | Expected Outcome | Actual Outcome | Date |
| 1 | Login System  Correct input | User:  admin  Password:  password | Login accepted | Login accepted | 10/5/11 |
| 2 | Login System  Incorrect input | User:  incorrect  Password:  incorrect | Login Failed – sent to registration form | Login Failed – sent to registration form | 10/5/11 |
| 3 | Login system  Incorrect input | User:  admin  Password:  incorrect | Login Failed – sent to registration form | Login Failed – sent to registration form | 10/5/11 |
| 4 | Login System  Large input | User:  Too long to fit  Password:  Too long to fit | Login Failed – sent to registration form | Login Failed – sent to registration form | 10/5/11 |
| 5 | Login System  Null values | User:  Password: | Login Failed – sent to registration form | Login Failed – sent to registration form | 10/5/11 |

### Registration

Due to the nature of the developed registration form, it wouldn’t be able to discern whether an email is a valid one, as long as it has an @ and a period at some point, it should be accepted as valid. The exception to this is the inclusion of specific ASCII characters, generally related to punctuation (/ , # ; [ ] etc).

As long as the input is of the standard accepted formation (e.g. [something@somewhere.com](mailto:something@somewhere.com)) it should be accepted as valid.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ref No. | Test | Data | Expected Outcome | Actual Outcome | Date |
| 1 | Correct input | User: Kieran  Pass: Jones  Email: Kieran@email.com | Accepted | Accepted | 10/5/11 |
| 2 | Incorrect input | User: Kieran  Pass: Jones  Email: Kier[]/@email.com | Failed | Accepted | 10/5/11 |
| 3 | Null values | User:  Pass:  Email: | Failed | Accepted – null value user created, Null value login now works | 10/5/11 |

### Search Submission

Searching works based off the TMDB.org API, so when you submit a search term to my website, all it will do is send that query to the API and then receive the response, creating a page based on that. An invalid submission would either return a result that the API deems as relevant or no response at all, meaning the created page will have no returned results and thus no content.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ref No. | Test | Data | Expected Outcome | Actual Outcome | Date |
| 1 | Correct input | “Blade” | Accepted | Page generated with results relevant to “Blade” | 10/5/11 |
| 2 | Incorrect input | dghfghfnccbvb | Accepted – No page content | Accepted – No page content | 10/5/11 |
| 3 | Null values |  | Accepted – No page content | Accepted – No page content | 10/5/11 |

### Review Submission

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ref No. | Test | Data | Expected Outcome | Actual Outcome | Date |
| 1 | Correct input | Summary: Test  Rating: 5  Content:  Test  User: Admin | Accepted | Accepted, review, output correct | 10/5/11 |
| 2 | Incorrect input | Summary: Test  Rating: 5  Content:  Test  User: Admin | Accepted – Review doesn’t appear on movie page | Accepted – Review doesn’t appear on movie page, appears on index page in featured review area, and in the database | 10/5/11 |
| 3 | Null values |  | Correct:  Error – please fill out field | Correct:  Error – please fill out field | 10/5/11 |

# Evaluation

## Registration

During testing an issue came up where by leaving all text boxes blank during sign up, would create a null valued user account. This account could then be logged in through the standard login form, this is a serious issue that needs to be fixed, during registration a valid submission should include all text boxes to contain at least something, if they are invalid on processing, and error should be displayed, however a completely blank submission shouldn’t even be allowed to be submitted. The use of Javascript could provide the answer to this problem, by not allowing submission unless all required fields are filled in. It could also be used to pre-empt invalid submission, by checking each field as they are completed, and throwing up some kind of error prompting the user to correct it.

### Review Submission

When a review is submitted, and the user submitting it is not logged in (i.e. a null user – see registration) the review sent to the database, it then can be output to the index.php webpage in the form of the Latest Review, taking up the main feature area. This shouldn’t happen, if a user is not logged in, submission of a review should be denied, through the use of more validation this can be achieved. A simple Javascript check for a valid login should be sufficient, it would also avoid the future problem of every single generated page being unique for each user, allowing possible caching of repeated pages should the website generate a lot of traffic.

## Review Voting System

The basic system work, however it does have one serious shortcoming, stemming from its late inclusion into my design. When a user votes up or down a review, it accepts the input as planned; however a user can refresh that page, and vote for the review again, there are no checks on whether the user has voted before or not. Guests to the site who don’t have an account or even non-logged in users can also do this, opening up the possibility of infinite voting happing. What can be done about this is to implement a system which logs an IP everything a vote is made, banning that IP from voting for that review again. Alternatively this can be tracked through the use of cookies, storing on a user’s PC what they have voted for, either way continued development will be needed.

## Movie.addRating()

When researching the feasibility and features of the TMDB.org API for my analysis of Prototype Two, I came across a feature within the API that allowed my website to upload user ratings from my site, to the API, affecting the ratings on the TMDB.org site. This seem like an essential system to implement, and as users review and rate movies on my website, the ratings change and the outputs of top 10 scripts on my site would theoretically change. This function allows me to avoid the potential of needing to store, and access the rating data from my own database, and also update the TMDB.org’s site with relevant data.

Whist designing prototype two, I had this feature firmly in my mind, and even went so far into the design that I started to develop a rough framework of the website using the API. When it came to do a rough implementation of the addRating function, I found that its requirements were not what I thought. It required a unique user to log in, not only on my website, but also onto the TMDB.org to allow them to upload their rating to TMDB.org. This involved using my website API Key to create a token, which then created a valid Session ID for TMDB.org. But to do this required my site to redirect the use to TMDB.org for the login. Although it was a powerful function, and I could implement it, it did not feel elegant moving from one site to another and back again.

## API Dependency

There is a major issue with my current developed project, it is beholden to TMDB.org for all of the movie information and movie ID’s. My reviews stored within my database, are linked to movies through movie ID numbers, using those numbers I can use the TMDB.org API to pull relevant information from it and output that information to my web pages using PHP scripts. When it works, it works spectacularly well, however when the API goes down for any reason, my whole site ceases to function, causing PHP and MySQL errors throughout the site rendering it impossible to even get any information to display. The most that displays is the header and logo, and login form and that is it.

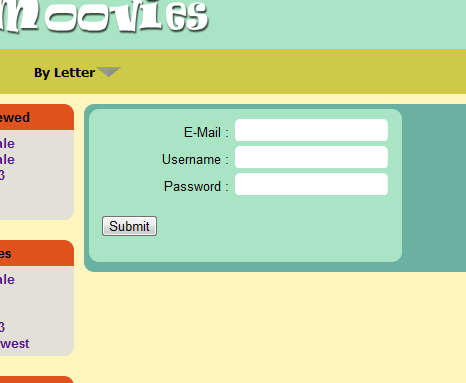
The problem also extends to testing of the site, if the API is down, I cannot work on small unrelated features because of all the missing information required, causing the scripting to break and the page to be output with errors before even the code I am to test is even run.

This of course is a serious issue to my project, as from the outset I developed it to potentially be published and used, but if I cannot find a way to work around this issue, it may never happen, although it is workable for just an IT project.

One potential workaround is to cache any information that I receive from the API to my own database, so my PHP scripts pull the information from my database rather than the API if there is an absence of access to the API or even if the information does exist in my database, accessing it from there will take less time and less bandwidth than if I were to use the API. This could be implemented by simple storing the XML object when it is accessed from TMDB.org, but care has to be taken to always check for up-to-date details

# System and User documentation

## Registration

On all pages is a button that allows you to sign up to the website. Click this and you will be taken to the sign up page.

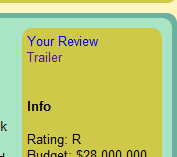
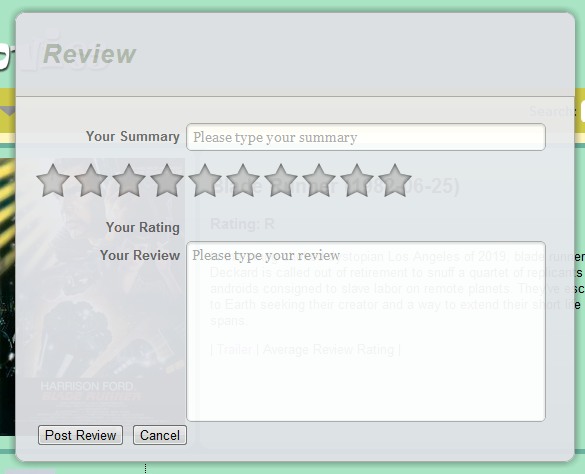
This page requires you to input your E-Mail address that you want associated with your account, a Username you would like to use, and a Password. An account will be created with the submitted details and you can now login.

## Login and Logout

When you have an account, you simply return to the same area that you use to register to the site, but input your user name and password instead. Click Login and the system will log you in. You can now post reviews on any movie you wish.

## Posting a Review

This area also contains a logout link, if you ever want to logout, click this and the site will return you to the index page, logged out.

To post a review you must ensure you are logged in, otherwise the system will not allow you to post. To review a movie, navigate to that movie through searching or finding it and look to the right information bar for a “Your Review” link. Click this and a review form will appear. There is a text box for a review summary, a 10 star rating bar for you to click on, and a main content box. Complete these required inputs, and press Post Review. Alternatively press cancel to remove the review box. Your review will now appear amongst the reviews for that particular movie.

## Searching

To search for a movie, you simple enter what you want to search for in the search text box on the navigation bar, and press Go. The next page will return all the results containing that search term.



# Project Plan Effectiveness

The plan for this project involved many different aspects and was for me a learning experience, it was the first time I had created a plan and stuck to it as best i could, and it did have some problems. From the start of Prototype One the plan was to create a basic site that would allow the creating of a user that then could upload a typed review to the website, and then have that review output correctly to the relevant movie’s web page. This in itself was a learning experience with regards to PHP and MySQL, as my knowledge with both languages was very basic at the start of planning, I created a plan that emphasised the development of the scripting behind the prototype, over the development of the visual front-end. This allowed me to focus on learning the skills needed to create the systems I had planned out.

One example was the creation and inclusion of using security Hashing on user input passwords, adding a Salting string to that hash and then hashing and ultimately storing the resulting string within the database. This was only an idea I had read about and never put into practice before the Plan, and yet I decided to include it. This plan then allowed me the time to actually learn and then implement the needed code into the site.

Prototype Two’s plan was to initially separate the code that runs the various elements of the website away from the layout of the site itself, allowing me to create a layout that could be used in multiple pages without each page being massively different, basically creating template that I could then fill in the gaps with PHP function calls, allowing for a consistent look throughout. This plan work to an extent, as out of the three distinct pages (Index, Movies, Title), two barely had any differences between them, apart from a few lines calling different functions. Title however never ended up in a separated state like Index and Movies, mainly due to time issues, and the page itself contained a lot of the code necessary to output its content. Although the layout framework for the page is the same as Index and Movies, I feel that Title needed the content generation scripts to be moved into their own Class.

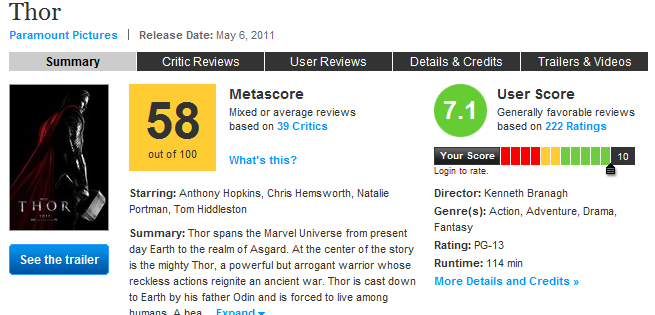
Prototype Two’s plan also focused on the inclusion and implementation of the TMDb.org API, a feature that I had played around within the time between finalising Prototype One and starting planning for Prototype Two.

The plan was to implement multiple functions from the API into the website, including one function called Movie.addRating() which i believed at the time of planning would allow me to upload review ratings from my website to the API, affecting the API’s database and movie ratings, which would in turn effect my website’s top ten movies output. When it came time to implement this function, I quickly found that it would require sending the user from my website, to the TMDb.org website and need them to login there to submit a single rating on my site. This feel very cumbersome to me, as I would not be able to cleanly send people to TMDb.org for login and then bring them back, this would also cause issues with keeping movie ratings the same between both websites. The solution I came up for this was to store the rating myself, simple enough, but it happened at a late stage, and if completely implemented would cause a rewrite on a couple of my already implemented functions.

This was caused by a failure to properly research the function, and assuming it worked the way I initially believed it did, I should have paid more attention to it and planned accordingly, this would have given me the time needed to implement my own rating system before my deadline loomed.

# Potential Service Impact

The impact of the website on users would depend on the user use of the website, if people are deciding on what movie to watch, they could use the website to determine on what to watch. The simple review system will allow people to quickly determine whether watching the movie is worth their time, and the movies are rated by the common user, not a professional critic who may focus on certain aspects of a movie. Generally professional critics do not tend to speak for the average viewer, this can been seen with review scores given by critics on some movies being quite different from the general public consensus, this is easy to see on a website such as Metacitric, where the average Critic score is a lot lower than the user score.

My website could have a similar impact, allowing users to determine, from the ratings given by other users, whether the movie is worth viewing. With continuing work and implementation of a tailored web site experience for users, outputting top rated movies by in genres as preferred by the user, and tracking the social aspect between users (friend’s favourite movies/genres etc). Using an aggregate of a movies overall vote, would allow me to implement a ranking system for movies over, and even sort it my genre sorting films into their respective genres allowing people to quickly search for a movie to view. Potential features for the future could be links to streaming services such as Netflix and Blockbuster online services, sending the user to that service.

An impact on the site itself would be the use of advertising, sites need funding to run, and a site of the potential size would need a substantial server as the user base increases which would need to be funded. The use of adverts would require space on the page to fit said adverts in, benefiting the site by generating income, however this space would need to be taken from other elements of the page. Simply using the space used by the banner for a small advert, or even have page high adverts positioned to the left and right of the content areas, where there is a lot of space available.

# Bibliography

Hammer, E. (n.d.). *www.quantumseolabs.com*. Retrieved from http://www.quantumseolabs.com/blog/website-marketing/web-scraping-helpful/

McGlinn, J. (n.d.). *phpsec.org*. Retrieved from http://phpsec.org/articles/2005/password-hashing.html

Smet, J. D. (n.d.). *Git Hub*. Retrieved from https://github.com/glamorous/TMDb-PHP-API

Walsh, D., & Beiner, F. (n.d.). *Davidwalsh.name*. Retrieved from http://davidwalsh.name/imdb-grabber-class-fabian-beiner

## Work Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Item to do | Date Started | Date Completed | Work done | Sig. |
| Proposal | 15/02/11 | 15/02/11 | Created the proposal, identified the areas needed to focus the design on. |  |
| Analysis | 15/02/11 | 17/3/11 | Looked at a number of potential rival sites, outlined roughly how they work and made notes on what features work well and could be implemented into my site.  Layout Research – 15/3/11 |  |
| Design | 15/02/11 | 17/3/11 | Login, Registration, Movie list, Title.php design  Layout - |  |
| Research | 15/2/11 | 17/3/11 | Databases/relational databases  SQL Insert,Select queries  SQL Join  SQLHandler – Kevin Woodard  IMDB Scraper 15/3/11  TMDB API – 16/3/11  XML research – 16/3/11 |  |
| Development  Prototype 1 | 20/02/11 | 17/3/11 | Created Database 21/2/11  Created basic layout framework (css) 21/2/11  Created login and registration script 21/2/11  Created movie listing page 28/2/11  Created title.php page 7/2/11  Implemented mysql\_escape\_string() – 15/3/11  Layout – started 15/3/11  Implemented sqlHandler.php throughout 13/3/11 |  |
| Evaluation | 16/3/11 | 17/3/11 | Evaluation on prototype 1 21/3/11 |  |
| Analysis | 17/3/11 |  | Prototype 1 Shortfalls - 5/4/11  TMDB.org API and supported functions – 5/4/11  Database – 5/4/11  Review output / Rating - 6/4/11  General research (jquery functions) – 22/3/11 – 3/4/11 |  |
| Design | 17/3/11 |  | API implementation - 6/4/11  Search - 7/4/11  Title Page - 7/4/11  Movie Page - 7/4/11  Navigation - 7/4/11  Index Page - 7/4/11  Left Column Content - 7/4/11 |  |
| Development | 17/3/11 |  | TMDB.org API implementation – 9/4/11  XML integration – 9/4/11  CSS/HTML layout creation 4/4/11  Layout set – 3/4/11  Separation of PHP scripts and layout 3/4/11  Login validation – 3/5/11  Commenting – 17/5/11 |  |
| Evaluation |  |  | Login/Registration Shortcomings – 5/5/11  API Dependency – 5/5/11  User Documentation – 6/5/11  Project Planning Effectiveness – 24/5/11 |  |
|  |  |  |  |  |