

# SOFTWARE PROJECT: TECHNICAL REPORT

## GRADIENT WEBSITE

---

<b>1. Executive Summary</b>	<b>2</b>	3.1.4 Operating Environments	6
<b>2. Introduction</b>	<b>2</b>	3.1.5 General Constraints	6
2.1 Background	2	3.1.6 Assumptions & Dependencies	7
2.2 Project Overview	2	3.2 System Requirements Specification	7
2.3 Target Market	3	3.2.1 External Interface Requirements	7
2.4 Possible Risks	3	3.2.2 Functional Requirements	7
2.5 Glossary	3	3.2.3 Non-Functional Requirements	10
2.6 Technologies	3	3.3 Design & Architecture	11
2.6.1 PHP	4	3.3.1 Database Design	11
2.6.2 JavaScript	4	3.3.2 Activity Design	11
2.6.3 Ajax	4	3.3.3 System Architecture	12
2.6.4 MySQL	4	3.4 Graphical User Interface (GUI)	12
2.6.5 Laravel 5	4	3.5 Implementation	13
2.6.6 Bootstrap	4	3.5.1 Models, Views & Controllers	13
2.6.7 Apache Server	4	3.5.2 Grouping Algorithm	14
2.6.8 Amazon Web Service (EC2)	4	3.5.3 Colour Matching	15
2.6.9 GitHub	4	3.6 Testing	16
2.6.10 Hardware & Software	4	3.6.1 Integration Testing	16
2.7 Project Breakdown	4	3.6.2 Load Testing	17
2.7.1 Breakdown	5	<b>4. Conclusion</b>	<b>17</b>
2.7.2 Delivery Stages	5	<b>5. Further Development</b>	<b>17</b>
<b>3. System</b>	<b>6</b>	<b>6. References</b>	<b>18</b>
3.1 User Requirements Definition	6	<b>7. Appendix</b>	<b>18</b>
3.1.1 Products Perspective	6		
3.1.2 Product Functions	6		
3.1.3 User Characteristics	6		

# 1. Executive Summary

This report details the development methods and solutions involved in the production of Gradient, a social media website. Gradient's aim is to reduce cyber bullying and the influx of negative comments that come along with posting on a social media platform. To achieve this Gradient uses an algorithm to dynamically share the contents of your status updates with other like minded users and users with similar interests.

Gradient is built using the Laravel 5 framework on a PHP codebase and deployed on a Amazon Web Service Server. Detailed information on class structure and system architecture can be found within this document. As well as some of the more important code snippets.

We believe that the small change to how relationships and content sharing works on our website truly sets Gradient apart from the competition in terms of safe spaces and friendly user interaction. Gradient frees users from the burden of being judged about their niche interests and promotes freedom of self expression.

## 2. Introduction

The goal of this project was to develop a software product in an area of our choosing. The software must offer useful functionality and have a high level of innovation, either filling gaps in a current technology area or branch into an entirely new one.

### 2.1 Background

The idea for this project came to me as a result of my own changing interests over the years. Take Facebook for example, you add a friend on Facebook and you receive all of their updates and posts even posts you may not be interested in. This leads to users posting updates that get no likes, no comments and no views. Even more

so it opens up the opportunity for negative comments or bullying from "friends" that do not enjoy your posts about computer games or obscure TV shows you enjoy.

The aim here is to create a space where users can update their page and are guaranteed to have it viewed by people with the same interests regardless of friend or follower count.

### 2.2 Project Overview

Gradient is a social media website that dynamically shares your content with other users of the same interests. Every time a user posts an update to their page we parse the text and extract the most relevant keywords and data from the text and use this data to determine who to share this information with.

Likewise the user who shares content will receive content based on the contents of their latest updates and or posts. This results in an ever changing flow of relevant information based on your current and up to date interests. There are no friends or followers on this site, all interactions are dynamic and ever changing.

All users will be dynamically placed in groups based on the content of their post history. Groups are formed based on a user's top three interests e.g. football, music, TV, if a user stops talking about one of these topics they will eventually be moved from this group and get pushed into a different group e.g. football, golf, TV.

Groups only form when two or more people have the same interests. If there is no matching group for a user's three most common interests they will be put into a group that most closely matches their interests usually with a 2/3 match. Users can only see posts from that group as long as they remain in said group. The more frequently you post about different topics the more likely you are to move through different groups.

The more you talk about a specific subject or topic the more likely you are to see relevant posts about that subject. The interesting thing about this idea is that the user does not select who the content is displayed to; that is all taken care of in the back end. Because we keep track of all your posts and interactions we know that your interests have changed even if you don't.

Each user is also given a colour based on their current interest which is attached to their profile and avatar. This colour is generated from their three most common interests and their weights using the RGB colour profile. This colour will indicate how closely your interest match with other people in your group at any given time.

## 2.3 Target Market

The target market for this website would be any user that likes to engage in open discussions about their hobbies or interests with like-minded individuals. A place where your posts don't live and die by your popularity or friend count, but rather by its content.

It's also for those users that are threatened by other social media sites where you actively have to look for friends and followers. Our site takes the pressure of finding like-minded people off the user's shoulders and lets them focus on what really matters, the message content.

## 2.4 Possible Risks

The main risk with this project is lack of privacy. Users have no control over who views their content because it is accessible by every other user that is part of their group. Having said that, this website is not for sharing private information it is more of an open discussion forum.

## 2.5 Glossary

Term	Definition
Database	A collection of all information from which the system can access and store information.
User	The person using the application.
Anonymous User	A person who has not created an account in the application.
PHP	PHP is a server-side scripting language designed primarily for web development but also used as a general-purpose programming language.
JavaScript	JavaScript is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites.
Laravel 5	Laravel is a free, open-source PHP web framework, created by Taylor Otwell and intended for the development of web applications following the model–view–controller (MVC) architectural pattern.
GUI	Graphical User Interface
Post	A small passage of text that the user enters into the website to describe something about themselves or their daily lives.
Comment	A small passage of text left by a user on another user's status.
Amazon Web Server	Amazon Web Services is a subsidiary of Amazon.com that offers on-demand cloud computing platforms.

## 2.6 Technologies

Most of the technologies used to develop this application were chosen because of ease of use and or familiarity. PHP was the exception as we had no programming experience with this scripting language.

Fortunately there was an abundance of resources and documentation online on the

subject. The Laravel 5 framework was also a great help as it simplified a lot of the boilerplate work that goes into developing a website such as this. Such as user accounts and sessions. Below is a brief rundown of all the technologies used throughout the development of the project.

### 2.6.1 PHP

The main bulk of the website was written in PHP as it offers good performance and is easy to integrate into most servers. PHP code is also clean and easy to understand with lots of resources available.

### 2.6.2 JavaScript

JavaScript was used to add some dynamic elements on the front end as well as any client side validation that needs to be done. JavaScript libraries such as jQuery were used where needed for design and accessibility features.

### 2.6.3 Ajax

Ajax was used to update portions of the website dynamically. It was also used to add pagination to a user's news feed to improve load times and responsiveness.

### 2.6.4 MySQL

An online MySQL database was used to store all data pertaining to the project.

### 2.6.5 Laravel 5

Laravel 5 is a PHP framework that was used to streamline some of the more common website tasks such as sessions and cookies. It also has a lot of helper functions to manage database calls for saving, updating or displaying data.

### 2.6.6 Bootstrap

Where possible the website used the Twitter Bootstrap framework to minimise HTML and help the site respond to changing screen sizes.

### 2.6.7 Apache Server

A local Apache web server was used to run the PHP code during the development of the website.

### 2.6.8 Amazon Web Service (EC2)

Everything from the PHP to the database and uploaded images was stored on an Amazon cloud based server during the deployment stage.

### 2.6.9 GitHub

GitHub was used to maintain and manage the project's codebase.

### 2.6.10 Hardware & Software Requirements

- Desktop PC or Laptop
- Apache Server
- Sublime Text
- GitHub
- MySQL
- Amazon Server

This project was completed using a medium spec desktop PC and or laptop computer. If work had to be carried out during class time in college the latest project files could easily be pulled from GitHub. All development was carried out on a local server.

The editor of choice for this project was Sublime Text because it offered some nice plugins that paired well with the Laravel framework. The finished project was deployed to an Amazon Web Server for presentation and testing purposes.

## 2.7 Project Breakdown

Here is a brief overview of all the work carried out during the product's development and how much time was dedicated to each task.

### 2.7.1 Breakdown

Stages	Tasks	Hours
Stage 1	Analysis & Design	
1.1	Mock Ups	6
1.2	Requirements Analysis	12
Stage 2	Dataset	
2.1	Source suitable data for project testing	6
2.2	Database Design	8
2.3	Extract, transform and load selected data to storage.	18
Stage 3	Implementation	
3.1	Website Design	40
3.2	Database functionality	20
3.3	Login and user management functionality	30
3.4	Post, edit, delete and view message functionality	24
3.5	Image and video uploading functionality	20
3.6	Dynamic clustering of users and messages functionality	40
3.7	Colour matching functionality	10
3.8	Additional pages of content, browser support	12
Stage 4	Testing & QA	
4.1	Website testing and bug fixing	24
4.2	Server side load and speed testing	12

### 2.7.2 Delivery Stages

Stage 1 Analysis & Design	
Tasks	Sketch mock ups for UI design and gather functional and non-function requirements.
Deliverables	Mock up feedback and Requirements document
Total Hours	18
Estimated Work	3 Days

Stage 2 Dataset	
Tasks	Source suitable dataset. Design database schema. Extract, transform and load data for website testing.
Deliverables	Working database of usable relevant data.
Total Hours	32
Estimated Work	5 Days

Stage 3 Implementation	
Tasks	Website design and functionality. Database methods and functionality for basic operations. User login management. Sharing and uploading of content. Creation of grouping algorithm. Colour matching algorithm. Additional website features.
Deliverables	Working website with possible bugs.
Total Hours	196
Estimated Work	25 Days

Stage 4 Testing & QA	
Tasks	Testing and bug fixing. Server load tests.
Deliverables	Fully working and responsive website.
Total Hours	36
Estimated Work	5 Days

Stage 5 Deployment	
Tasks	Deployment of website to web server.
Deliverables	Fully working website on deployment server.
Total Hours	4
Estimated Work	1 Day

### 3. System

Below is a breakdown of the whole system starting with the requirements including functional and nonfunctional. The system architecture including database design, the GUI, the system implementations and functions and the system testing procedures and processes.

#### 3.1 User Requirements Definition

The following is a detailed list of everything the user should expect the system to do or perform.

##### 3.1.1 Products Perspective

The aim of the website is to allow people to share and receive content with other users dynamically without adding them as a friend or follower. Users of the website can interact with all content that is relevant to them at any given time. The content displayed to the user will change based on the content of past and present user updates. This means that all users will only be shown content based on their current interests.

##### 3.1.2 Product Functions

- The website will dynamically share content based on a user's current interests.
- Users can share text and images with other users.
- Users can comment on any status relevant to their interests.
- Users have full control over their profile and can block other users.

##### 3.1.3 User Characteristics

###### 3.1.3.1 General User

- Has the ability to interact with all functions of the website once an account has been created and is logged in.

###### 3.1.3.2 Anonymous User

- Has the ability to create an account. Anonymous users cannot access the website without an account.

##### 3.1.4 Operating Environments

- This website is designed to run on all devices that have access to a web browser. This includes but is not limited to PC, laptop, mobile phone and tablet.
- This website with its database are hosted on an Amazon web server due to its fast and reliable service.
- Website development was carried out using Sublime Text and an Apache server package for local development.

##### 3.1.5 General Constraints

- The website must work with a mouse and keyboard.
- The website must work on a touch screen i.e. mobile phone.
- The ability to interface with a database must be implemented.
- The ability to save and load file must be implemented.

### 3.1.6 Assumptions & Dependencies

- The user's browser must support JavaScript.

## 3.2 System Requirements Specification

A list of what the system requires to operate and what the system shall provide for the user. This list will be further broken down into external interface requirements, functional requirements and nonfunctional requirements.

### 3.2.1 External Interface Requirements

#### 3.2.1.1 User Interface

- The user interface shall offer the user a logical representation of what the website is asking the user to do. Drop-down menus and buttons should be used where possible to aid the user. Tooltips and other information should be presented to aid user actions.
- The GUI should have the website logo attached.
- The GUI should scale and respond to the most common screen sizes available.
- Certain optional features may be omitted on smaller screen sized devices.
- Allowances should be made for the visually impaired.
- The GUI should have continuity between pages and have a consistent design throughout.

#### 3.2.1.2 Hardware Interfaces

- The system should be operated with a keyboard and mouse, with the mouse being optional.
- The system should be fully operational on any touchscreen device.

### 3.2.2 Functional Requirements

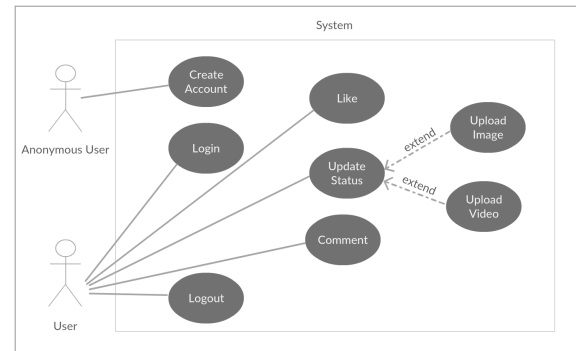


Figure 1. Complete use case diagram

#### 3.2.2.1 User

- All users of the website shall have the ability to create an account which can be used to interact with the website.
- User login is mandatory to access any of the sites features.

#### 3.2.2.2 Creating an Account

- The website should provide an easy to use GUI for creating an account.
- The system should ask for a username and password upon registration.
- Usernames must be unique and passwords must be encrypted.
- The system should notify the user if any invalid characters are used in either the username or password field.
- The system should notify the user if mandatory fields are left blank during registration.

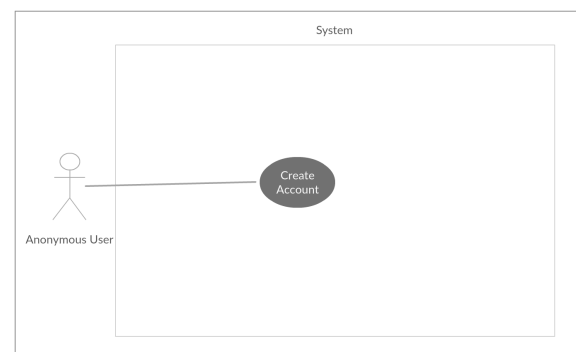


Figure 2. Create account use case diagram

Use Case Element	Description
Use Case Number	1
Application	Gradient
Use Case Name	Create Account
Use Case Description	A user navigates to the website URL and is prompted to create an account.
Primary Actor	Anonymous User
Precondition	The system is deployed and running.
Trigger	The user navigates to the website URL.
Basic Flow	<ol style="list-style-type: none"> <li>1. This use case starts when the user navigates to the website URL and is prompted to create an account.</li> <li>2. The user inputs a username and password into the allotted text fields.[AF1]</li> <li>3. The application notifies the user that the account has been created and grants access to the application.</li> </ol>
Alternate Flows	<ol style="list-style-type: none"> <li>1. The user enters invalid characters into the text field or leaves them blank. The system notifies the user of their error.</li> </ol>

### 3.2.2.3 Login

- The website should provide the user a GUI to allow the user to login when navigating to the website URL.
- The website should prompt the user to enter both username and password into the fields provided.
- If login details are incorrect the website should notify the user of the error.

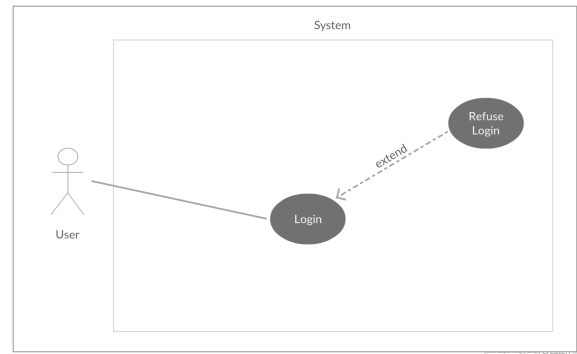


Figure 3. Login use case diagram

Use Case Element	Description
Use Case Number	2
Application	Gradient
Use Case Name	Login
Use Case Description	The user logs in to their account.
Primary Actor	User
Precondition	The user has already created an account.
Trigger	Navigating to the websites URL.
Basic Flow	<ol style="list-style-type: none"> <li>1. This use case starts when the user navigates to the website URL and is presented with a login GUI.</li> <li>2. The user enters their username and password and is allowed access to the system.[AF1]</li> </ol>
Alternate Flows	<ol style="list-style-type: none"> <li>1. The user inputs an incorrect username or password and is notified by the system.</li> </ol>

### 3.2.2.4 Logout

- The system should provide the user a GUI to allow the user to logout of the system.



### 3.2.2.5 Update Status

- The website shall provide the user with means to update their status.
- Status updates can contain text and images.
- The system should store all updates on persistent storage.

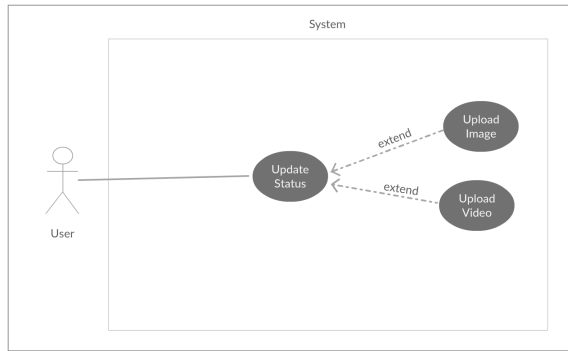


Figure 4. Update status use case diagram

Use Case Element	Description
Use Case Number	3
Application	Gradient
Use Case Name	Update Status
Use Case Description	The user updates their status.
Primary Actor	User
Precondition	The user has already created an account and is logged in.
Trigger	The user clicks the update status button.
Basic Flow	<ol style="list-style-type: none"> <li>1. This use case starts when the user clicks the update status button.</li> <li>2. The user enters their status update in the text box provided by the GUI.[AF1]</li> <li>3. The user clicks submit to post their status to their public profile.</li> </ol>
Alternate Flows	<ol style="list-style-type: none"> <li>1. The user adds an optional image to the status.</li> </ol>

### 3.2.2.6 Dynamic Grouping

- The system should group users together based on the contents of their status updates.
- User can only see updates from other users in their group.
- Users will be removed from the group if the content of their latest updates do not match the group's interests.
- The system shall move users from group to group without any approval from the user.

### 3.2.2.7 Upload Media

- The system shall provide the user with a suitable GUI to upload images.
- All uploaded media should be stored on persistent storage and should be readily available to the user.

### 3.2.2.8 Comment

- The system shall provide the user with a sufficient GUI for adding a comment to another users status update.
- Comments can only contain text.
- All comments must display the user's name, date and time.
- Only logged in users can comment.
- Users can choose to disable comments on any status update.

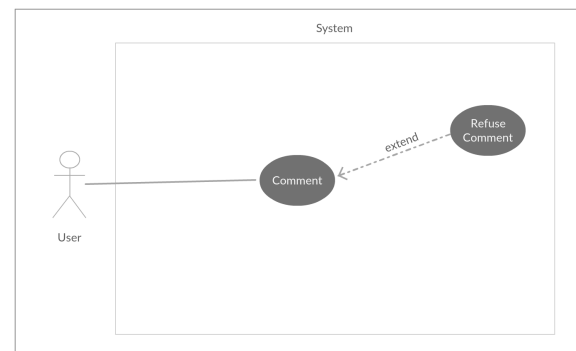


Figure 5. Comment use case diagram

Use Case Element	Description
Use Case Number	4
Application	Gradient
Use Case Name	Comment
Use Case Description	The user comments on their or another users status.
Primary Actor	User
Precondition	The user has already created an account and is logged in.
Trigger	The user clicks the add comment button on a status.
Basic Flow	<ol style="list-style-type: none"> <li>1. This use case starts when the user clicks the add comment button.</li> <li>2. The user enters their comment in the text box provided by the GUI.[AF1]</li> <li>3. The user clicks submit to add their comment to the selected status.</li> </ol>
Alternate Flows	<ol style="list-style-type: none"> <li>1. The user has disable comment on the selected status.</li> </ol>

### 3.2.2.9 Colour Matching

- The system should assign each user a dynamic colour based on the weights of their current interests.
- Colours are used to allow users to identify other users with similar interests.
- Each user's colour should appear on their profile and avatar in status updates and comments.

## 3.2.3 Non-Functional Requirements

### 3.2.3.1 Portability

- The website should run on any device with a capable web browser.

### 3.2.3.2 Reliability

- The system should be extremely reliable and have an uptime of 99.999%.

### 3.2.3.3 Ease of Use

- The website should be user friendly. It should take users no longer than 5 minutes to become comfortable and familiar with the website GUI.

### 3.2.3.4 Speed

- Updating status and adding comments should appear instant to the user.
- The GUI should be responsive and snappy with no lag between pages and button clicks.
- The users feed should fully load in no more than five seconds.

### 3.2.3.5 Size

- The website should be no more than 100 mb in total excluding the database.
- All website images, JavaScript files and CSS should be minified and compressed for better performance and resource conservation.

### 3.2.3.6 Privacy

- Any data retained by the system will be in accordance with the Data Protection Act 1988 and the Data Protection (Amendment) Act 2003.

### 3.3 Design & Architecture

The system makes use of design best practises where possible. This includes but is not limited to UI design, database design and coding patterns. The system is designed to be modular which allows us to add new features and update existing features with ease.

#### 3.3.1 Database Design

A relational database is used to store all persistent data throughout the system. This was chosen as it allowed for complex queries to be carried out with ease. It also provides more efficient storage by reducing redundant data. Figure 6. is a conceptual ERD of the entire database.

Each table represents a logical section of the platform i.e. users, posts and comments. The group type table is a lookup tables which will allow extra groups to be added without the need for cascading updates across the database. Currently the system only allows one image to be attached to a post.

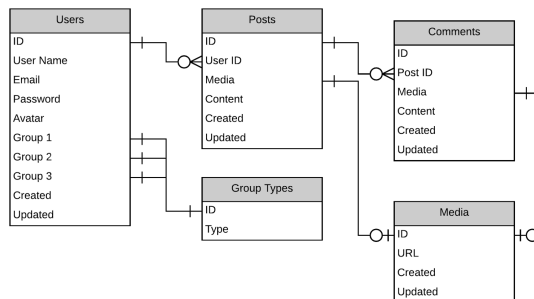


Figure 6. Conceptual ERD

#### 3.3.2 Activity Design

Each function of the platform can be modelled with an activity diagram. Figure 7. is a diagram of the add new post activity. A user can choose to add media to a post if they wish. When a user adds media to the system the activity branches and the system process slightly changes.

All media is stored on the server with the associated URL added to the database post. Either way the results is always the same for the user. They are redirected to the post feed where all the latest posts are found.

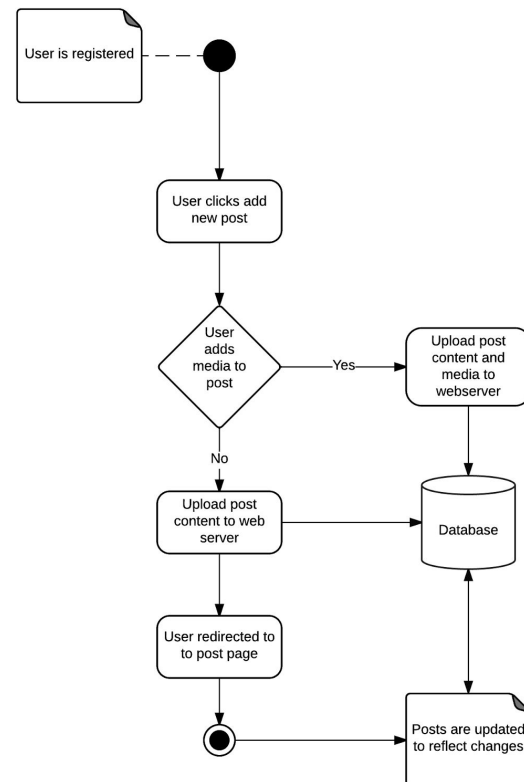


Figure 7. Create post activity diagram

### 3.3.3 System Architecture

The system uses the model view controller (MVC) architecture design pattern. Enabling a separation of concerns between the data, UI and functionality. This was chosen because it reduces the complexity of each component and streamlines the testing process.

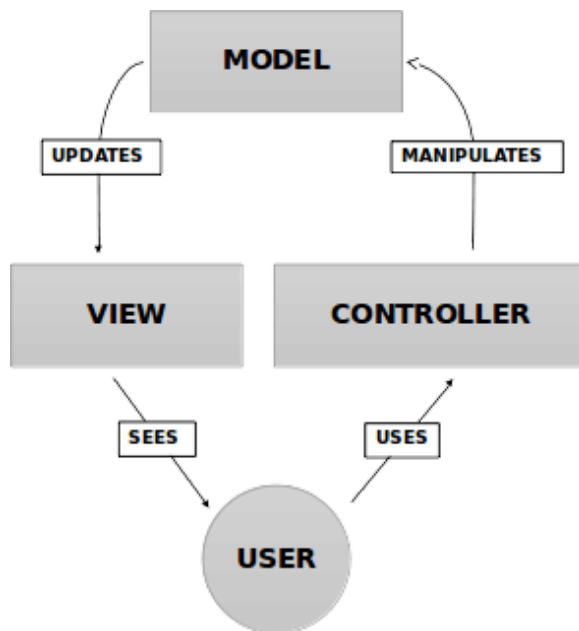


Figure 8. MVC architecture diagram

### 3.4 Graphical User Interface (GUI)

The login page is as simple as possible and will be the only thing a user sees when navigation to the websites sign in URL. The login form will ask for the users username and password to be entered before allowing them access to the rest of the platform.

New users will also have the ability to register a new account from this page too. Users who forget their passwords can also request a new password from this page.

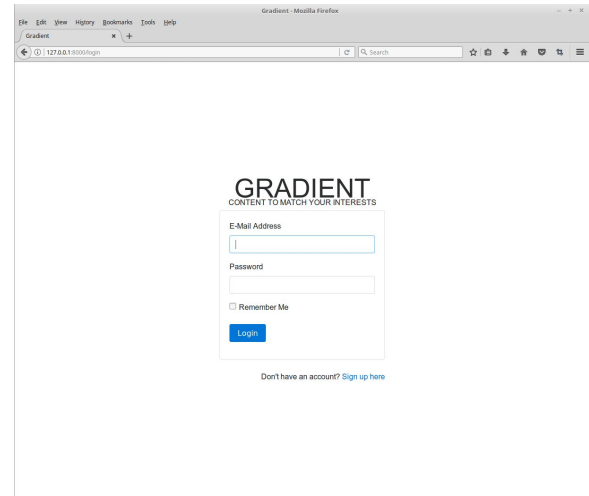


Figure 9. Gradient login GUI

The main page of the site has a fixed navigation bar at the top of the page with links to all major parts of the site. This page also shows all relevant posts to the user in a list ordered by date posted starting from the latest.

Each post, along with all comments and media will be bound by a border and separated from the post below. Each post will feature a thumbnail image, the post title, the name of the author and date posted.

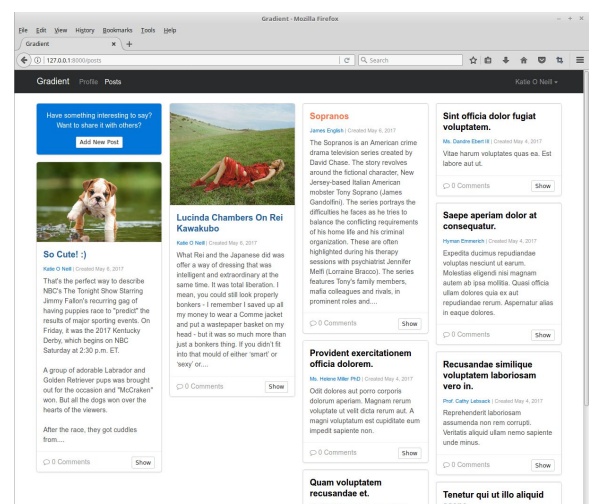


Figure 10. Gradient all posts page

Comments will be positioned in a nested fashion under each post. Only the twenty latest posts are shown to the user on page load. The post

page will feature pagination that will allow the user to click through all posts available 20 posts at a time.

A user can also create a new post from the main page by clicking the new post button from the top of the page. The new post interface will present the user with a form to fill out. This form will ask the user to add a post title, the post content and an optional image. Users will also have some extra option when posting a new post such as disabling comments.

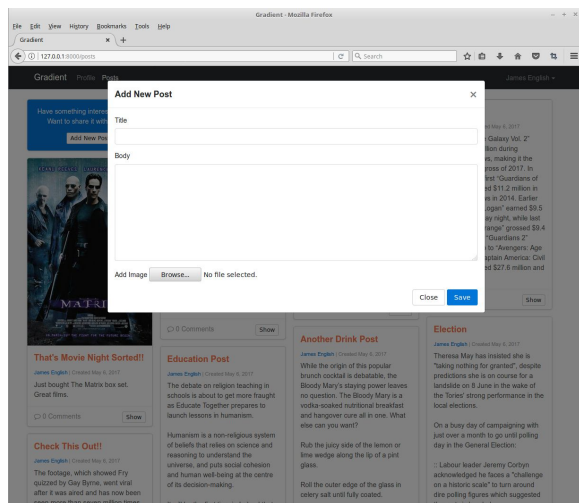


Figure 11. Gradient add new post form

Once a post has been submitted the user will be returned back to the main page and will see their post listed with all other posts from their group. Each user will also have access to a dashboard where they can see all their post history and comments. Users can also delete any comments or posts that they have created from their personal dashboard.

## 3.5 Implementation

### 3.5.1 Models, Views & Controllers

The Laravel framework uses an MVC architecture. The database is created with code using migration files and relationships are handled with models. Each table in the database

must have a corresponding model in the framework.

Controllers are basic PHP files that manipulate the database models using a series of functions and return a collection of data to a specific view. These functions can be access using the web.php file through a series of routes. Routes are responsible for the actions of a controller's function when a specific URL is accessed. Laravel uses RESTful routes to provide a mapping between HTTP verbs and the controller functions. A brief explanation of how posts are displayed follows.

First a migration file is created that will build the 'Posts' database table. This migration file has all the scheme information needed to build a fully qualified MySQL table. When the command '\$php artisan migrate' is ran Laravel will call the up() function and the framework will access that database and create the table.

```
class CreatePostsTable extends Migration
{
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id');
            $table->integer('topic_id');
            $table->string('title');
            $table->text('body');
            $table->string('image_url')->nullable();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('posts');
    }
}
```

Figure 12. Post table migration file

Once the database is set up relationships can be defined in the 'Post.php' model file. Laravel uses an ORM (Object-relational Mapping) and active record approach to database management. Several relationships are set up between our 'Posts' model and other models in our system.

```

class Post extends Model
{
    public function comments(){
        return $this->hasMany(Comment::class);
    }

    public function user(){
        return $this->belongsTo(User::class);
    }

    public function topic(){
        return $this->hasOne(Topic::class);
    }

    public function addComment($body){
        $this->comments()->create([
            'body' => request('body'),
            'user_id' => auth()->id()
        ]);
    }
}

```

Figure 13. Post model file

With the relationships firmly established data can now be added to the database. To display a list of all posts in the system a route must be set up to call a function in our 'PostsController'. Routes map controller functions to URL's. Here we send a GET request to '/posts' which will call the index() function in the 'PostsController'.

```
Route::get('/posts', 'PostsController@index');
```

Figure 13. Post index route

The index() function in the 'PostsController' Will grab a collection of all records from the database posts table.

```

public function index()
{
    if(UserTopics::where('user_id', '=', Auth::id())->exists()){
        $topics = UserTopics::where('user_id', '=', Auth::id())->latest()->first();
        $posts = Post::latest()->where('topic_id', '=', $topics->first_topic)
        ->orWhere('topic_id', '=', $topics->second_topic)
        ->orWhere('topic_id', '=', $topics->third_topic)
        ->paginate($this->posts_per_page);

        if($posts->count() > 20){
            return view('posts.index', compact('posts'));
        }
    }

    $posts = Post::latest()->paginate($this->posts_per_page);
    return view('posts.index', compact('posts'));
}

```

Figure 14. Post controller index() function

In the image above various queries are being performed on the collection of data but the most important part is the return statement. The return statement returns a view which is a PHP file located in 'post/index.blade.php' as well as the collection of posts.

The view file 'index.blade.php' uses the Laravel Blade templating engine which allows PHP and HTML to interact. In this view we can access our '\$posts' variable and display the content.

```

@foreach($posts as $post)
    <h4>{{ $post->title }}</h4>
    <p>{{ $post->body }}</p>
    <small>{{ $post->created_at }}</small>
@endforeach

```

### 3.5.2 Grouping Algorithm

The grouping algorithm is an implementation of the Naive Bayes classification and is incorporated into the project using the Bayes PHP package [1]. We use the Naive Bayes classification method to determine what topic the users post should be classified as.

To achieve this the users post is compared to a set of static training data which covers a range of topics. The algorithm's job is to predict what topic the post should fall under. When a user submits a new post to the system we call the getTopic() function.

```

public static function getTopic($text){
    $topics = Topic::all();

    $tokenizer = new WhitespaceAndPunctuationTokenizer();
    $classifier = new Classifier($tokenizer);

    foreach($topics as $topic){
        $classifier->train($topic->topic, $topic->data);
    }

    $result = $classifier->classify($text);
    $topics = Topic::where('topic', key($result))->first();
    return $topics->id;
}

```

Figure 15. Get topic function

The text of the users post is passed as an argument to the function, we also grab a collection of all topics from the database (also know as training data). A sample of the political topic training data is shown below.

Topic	Training Data	
Politics	Economic	Aristotle
	Diplomacy	Nationalism
	Populist	Liberalism
	Feminism	Conservatism

The training data is then cleaned of all punctuation and separated into tokens. From here the post text is evaluated against the training data and the topic probability is determined. The topics id is then returned and stored in the database along with the user's post.

Now that each post has the most probable topic attached to it we can use this information to display specific posts to specific users. Here the same algorithm is used again but this time it is used to determine the most probable topics over the user's last ten posts and update the user\_topics table in the database with the information.

Now it's just a matter of comparing post topics to the user's current topics and displaying the posts that meet the query.

### 3.5.3 Colour Matching

The colour matching algorithm takes three strings, in this case the user's current topics of interest and converts each character of the string into its ASCII value representation using PHP's ord() function [2]. The sum of each character value is added until it reaches 255 where it resets to zero and the remainder is added. The result of this algorithm returns three values ranging between 0 - 255 which can be used to represent an RGB (Red, Green, Blue) colour code. This code is then used throughout the website in various UI elements for the user.

```

public static function getColours($user_id)
{
    if(!App\UserTopics::where('user_id', '=', $user_id)->exists()){
        return "black";
    }

    $user_topics = App\UserTopics::where('user_id', '=', $user_id)->latest()->get();

    $first = App\Topic::where('id', $user_topics[0]->first_topic)->get();
    $second = App\Topic::where('id', $user_topics[0]->second_topic)->get();
    $third = App\Topic::where('id', $user_topics[0]->third_topic)->get();

    $colours = array($first[0]->topic, $second[0]->topic, $third[0]->topic);
    $rgb = array();

    for($i = 0; $i<sizeof($colours); $i++){
        $value = Utilities::decodeWord($colours[$i]);
        $sum = 0;

        for($j = 0; $j<sizeof($value); $j++){
            if ($sum + $value[$j] > 255) {
                $sum = ($sum + $value[$j]) - 255;
            }else{
                $sum = ($sum + $value[$j]);
            }
        }

        array_push($rgb, $sum);
    }

    $rgba = "rgb(" . $rgb[0] . ", " . $rgb[1] . ", " . $rgb[2] . ")";
    return $rgba;
}

```

Figure 16. Get colours function



## 3.6 Testing

### 3.6.1 Integration Testing

Integration tests were written and performed using PHPUnit. These integration tests were written to verify the main components of the system were active and engaging.

The main aim of these tests were to verify if the database CRUD functionality using models and controllers were working as intended. Create, read and delete tests were written for both posts and comments.

```
public function testCreatePost()
{
    $user = factory('App\User')->create();

    $post = $user->posts()->create([
        'title' => 'Happy Birthday',
        'body' => 'Today is my birthday, I am happy!',
        'topic_id' => 1,
        'image_url' => 'cake.jpg'
    ]);

    $this->assertDatabaseHas('posts', ['id' => 1]);
}
```

Figure 17. Create post integration test

In the above test a new user is created using a factory to create dummy data. A new post is then created with a relationship to the user. Lastly the database is checked to assert if the new post is available.

To test the delete post function an new user and post are created. The database is then queried to find the new post. Once the post is found the post is deleted and the database is tested to assert the post is missing.

```
public function testDeletePost(){
    $user = factory('App\User')->create();

    $post = $user->posts()->create([
        'title' => 'Happy Birthday',
        'body' => 'Today is my birthday, I am happy!',
        'topic_id' => 1,
        'image_url' => 'cake.jpg'
    ]);

    $found_post = Post::find(1);

    $found_post->delete();

    $this->assertDatabaseMissing('posts', ['id' => 1]);
}
```

Figure 18. Delete post integration test

To test the read functionality of the system a new post was created. The new post was then queried and tested to assert that the title string was equal to the test string.

```
public function testReadPost(){
    $user = factory('App\User')->create();

    $post = $user->posts()->create([
        'title' => 'Happy Birthday',
        'body' => 'Today is my birthday, I am happy!',
        'topic_id' => 1,
        'image_url' => 'cake.jpg'
    ]);

    $found_post = Post::find(1);

    $this->assertEquals($found_post->title, 'Happy Birthday');
}
```

Figure 19. Read post integration test

All tests are carried out in a separate test database. All database tables are created and dropped before and after each test.

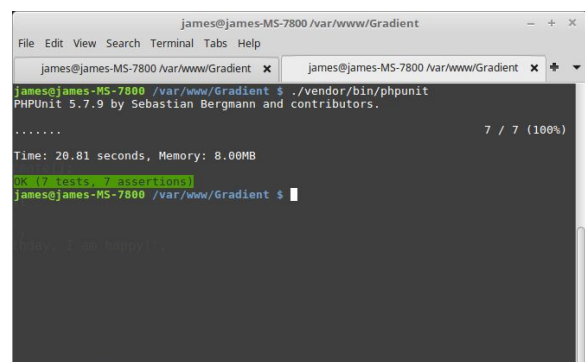
A screenshot of a terminal window with a dark background. The window title is 'james@james-MS-7800 /var/www/Gradient'. It shows the command './vendor/bin/phpunit' being executed. The output indicates that 7 tests passed, with 7 assertions successful. The total time taken was 20.81 seconds and memory usage was 8.60MB. The prompt 'james@james-MS-7800 /var/www/Gradient \$' is visible at the bottom.

Figure 20. Test confirmation output



### 3.6.2 Load Testing

The load testing was carried out on an Amazon EC2 T2 Micro instance running Ubuntu. The Apache Benchmarking tool was used to perform these tests [3]. The test was performed on a page that had multiple CSS and Javascript requests aswell a number of MySQL database queries.

The tests aim was to determine how many concurrent users and page hits the system could handle before failing. The test options were set as follows over the following tests:

```
$ab -l -r -n 600 -c 30 -k -H "Accept-Encoding:
gzip, deflate"
http://ec2-35-166-134-73.us-west-2.
compute.amazonaws.com/posts
```

1. 1 concurrent user / 100 page hits
2. 5 concurrent users / 10 page hits
3. 10 concurrent users / 10 page hits
4. 30 concurrent user / 20 page hits
5. 90 concurrent user / 30 page hits

Test one through four executed perfectly with no errors. The results of test 4 is shown in Figure 21. This test simulates 30 concurrent users doing 20 page reloads each.

```
Server Software:      Apache/2.4.18
Server Hostname:      ec2-35-166-134-73.us-west-2.compute.amazonaws.com
Server Port:          80

Document Path:        /posts
Document Length:      Variable

Concurrency Level:    30
Time taken for tests:  24.701 seconds
Complete requests:    600
Failed requests:      0
Keep-Alive requests:  0
Total transferred:    3356748 bytes
HTML transferred:     2736866 bytes
Requests per second:  24.29 [#]/sec (mean)
Time per request:     1235.069 [ms] (mean)
Time per request:     41.169 [ms] (mean, across all concurrent requests)
Transfer rate:        132.71 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median  max
Connect:  0    0  0.3      0    2
Processing: 300 1224 266.6    1227 3727
Waiting:  297 1202 264.2    1200 3701
Total:    302 1224 266.4    1227 3727
```

Figure 21. Load testing output (Test #4)

Test number five is an enormous load to bare and would cripple any site that didn't employ sophisticated caching. Running the test on the Amazon EC2 instance crashed the server and

had to be restarted from the Amazon Web Service Dashboard.

## 4. Conclusion

Looking back at the project objectively I would say that the biggest problem with this project is it simplicity. The aim of the project was to create a place where users could post and read content that were of interest to them. To this end the project does the job perfectly but it is void of many features that users have come to expect.

Removing the social aspect of friends and followers and removing the arithmetic from posts such as number of likes, number of shares or view count severely limits the amount of interaction a user can have with the site. Users can only post, comment and read. Users who do not post actively will also not get the benefits that the site offers as the more posts a user has the better able we are to determine their current interests.

In theory the idea was sound, a social media site void of ego and number counting would be great but in reality it's just kind of dull. Having said that, working on the project was a great experience. I learned a lot about PHP and the available PHP frameworks. I also got to spend a lot of time working with databases and writing queries which is always good to know.

## 5. Further Development

With more time and research I would like to automate the topics identification system. Currently the project uses manually created training data which helps determine the user's post topic based on a very specific set of topics. New topics and training data must be manually inserted into the database by the administrator. This is both time consuming and monotonous.

A better alternative would be to use Topic Modeling which is a machine learning technique that scans large amount of textual data and groups documents based on common word

clusters and associations [4]. This is a fully automated process with no need for training data. The downside to this approach is the massive resource cost. Each new posts must be added to a corpus containing all posts in the system to determine a specific topic. Performing this operation on a database with millions of posts would have a massive impact on the server's performance.

A solution would have to be found that incorporated both the Naive Bayes algorithm using training data and the use of Topic Modeling to automate the topic associations.

## 6. References

1. Boggiano J. `fieg/bayes` - Packagist [Internet]. [cited 3 May 2017]. Available: <https://packagist.org/packages/fieg/bayes>
2. PHP: `ord` - Manual [Internet]. [cited 3 May 2017]. Available: <http://php.net/manual/en/function.ord.php>
3. `ab` - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4 [Internet]. [cited 5 May 2017]. Available: <https://httpd.apache.org/docs/2.4/programs/ab.html>
4. Topic Modeling [Internet]. [cited 6 May 2017]. Available: <http://mallet.cs.umass.edu/topics.php>

## 7. Appendix

- Monthly Journals
- Project Proposal

# Reflective Journal

Student Name: James English  
Course: BSc in Computing

Student Number: X13118951  
Month: September

---

## Journal #1

My idea is to build a social media site that will dynamically share your content with other users that have shared interests. I pitched the idea at the project pitch presentation and the lectures seemed to like the idea.

Next week I will write up the full project proposal where I will nail down all the features and technologies that the project will have. Right now I'm thinking of using PHP to develop the site. I was researching PHP frameworks and I like the look of Laravel 5. It has some handy build in functions that can manage authentication and sessions for users which should come in handy down the line. It also has a nice templating engine that speeds up page creation and design.

Apart from a small bit of research not much work has been carried out on the project as of yet. Looking forward my biggest problem is going to be in the design of the database and also the most time consuming. Because of the dynamic nature of the project there will be a lot of database calls so it will have to be well designed and defined.

I'm actually really looking forward to working on this project as this will be the first time I have undertaken a website of this size. Most of my project to date have been desktop applications so this will be an interesting experience.

Hopefully next month after I submit my project proposal and start working on the requirements I will have more to share. But for now everything looks good, I feel confident and eager to get started.

---

Student Signature: James English  
Supervisor Signature:

Date: 08/10/16  
Date: 00/00/16

# Reflective Journal

Student Name: James English  
Course: BSc in Computing

Student Number: X13118951  
Month: October

---

## Journal #2

This month was spent working on my project proposal and project system requirements specifications documents. The project proposal document has been submitted. The requirements documents is almost complete, I just have to create some UI mock ups for the functional requirements.

Now that all pending documentation is out of the way for the time being I can concentrate on the next phase of the project which is the Project Prototype. For the project prototype I am aiming to have the main functionality of the project working. This will include the algorithm that displays all related content to users based on their current interests.

For this phase of the project I will need design and implement a working data base filled with some sample data. I'm aiming to gather this data from Twitter using the streaming Twitter API. I will gather a couple of thousand tweets and store them in the database as random user posts. From here I can implement and test the group sharing algorithm.

For the prototype I will demonstrate how the system dynamically shows the user relevant content based on their current interests. During the live demo I will update certain user profiles to show how the content they see is changed based on what they are currently talking about.

The main focus here is going to be all backend work but if I have time I will try and create a nice front end page to handle the POST request that somewhat resembles the finished design of the website (which has yet to be determined).

Next month I should have a lot more to share once I get started on the practical part of the project and actually start some coding.

---

Student Signature: James English  
Supervisor Signature:

Date: 03/11/16  
Date: 00/00/16

# Reflective Journal

Student Name: James English  
Course: BSc in Computing

Student Number: X13118951  
Month: November

---

## Journal #3

This month wasn't really as productive as I planned it to be. Unfortunately due to numerous CA's, project deadlines and the technical report there wasn't a whole lot of time left to work on the project prototype. However I did manage to get some smaller prototypes working for the presentation.

I have been doing some research into topic models and latent Dirichlet allocation, which is a statistical modelling technique for discovering topics in a collection of text. I have been doing some tests using R and R-Studio and it seems like this technique with some small adjustments will be perfect for classifying user posts under a certain topic for my project. Getting this working as intended is the main bulk of the grouping algorithm for the site going forward.

I also wrote a small algorithm in JavaScript to demonstrate the colour matching function of the website. This algorithm generates three random strings and assigns them an RGB colour value based on the strings Unicode value. This feature will be utilised in the website as to visually represent how closely two different users interests relate.

So for the prototype I'm hoping to demonstrate how users are dynamically grouped together, how each users is given a unique colour and some basic GUI front end navigation. Now that Christmas is just around the corner and all my other projects have been submitted or are just about to wrap up I'm going to make a big drive at getting some serious work done on the project in January and February.

---

Student Signature: James English  
Supervisor Signature:

Date: 11/12/16  
Date: 00/00/16

# Reflective Journal

Student Name: James English	Student Number: X13118951
Course: BSc in Computing	Month: December

## Journal #4

This month I managed to get all the posting, updating and deleting of posts and comments implemented into the project. I also set up the user account service that allows users to register and login to the website. At the moment adding new posts and comments have dedicated pages but I will update this to be more dynamic using AJAX. My aim is to have all of the posting, commenting, editing and liking all to happen dynamically on the one page without page refreshes.

I'm still having a bit of trouble finding the best solution for managing topics. I am researching the Naive Bayes algorithm which looks promising. This algorithm works by comparing text against a set of training data and returns a matching probability. It's exactly what I need for my project but it involves a lot of user management and overview to be effective. I would like to find a more automated solution if possible.

I still need to settle on a design for the website because as of right now its still up in the air. I hope to have a proper design nailed down next month.

Student Signature: James English	Date: 11/01/17
Supervisor Signature:	Date: 00/00/2017

# Reflective Journal

Student Name: James English	Student Number: X13118951
Course: BSc in Computing	Month: January

## Journal #5

Right now I have the post, edit post, liking and commenting all working and updating with AJAX as intended in the same page. I have also went and implemented the Naive Bayes classifier algorithm into the project and it appears to be working great so far after first impressions. Because the algorithm needs training data I will have to create a back end module that can monitor and control all the training data and add and modify data as needed.

This was not the original goal of the project but the Naive Bayes solution is the least resource intensive solution at the moment. Next I will add the image upload feature to the site and also look at adding user profile pictures or use Gravatar's instead. Gravatar's are globally recognised avatars managed by a third party site (gravatar.com). Implementing this third party solution will reduce development time and reduce resource and bandwidth cost for our server.

I'm also in the process of developing the user dashboard, where users can manage their posts, comments and image uploads all in one place. This dashboard will also have some graphs with historical data and a map of the topics and discussions a user was part of plotted on a timeline. Users can also manage their account from here such as (deleting account, blocking other users).

Student Signature: James English	Date: 11/02/17
Supervisor Signature:	Date: 00/00/2017

# PROJECT PROPOSAL

BSHCSD4 | National College of Ireland | E. Nolan

## Abstract

A project proposal document for a social media website that features dynamic grouping of users and content based on common interests.

JAMES ENGLISH  
X13118951



## Table of Contents

Project Description	2
Introduction	2
Project Overview	2
Background	2
Target Market	2
Possible Risks	2
Project Functionality	2
Dynamic Grouping	3
Colour Matching	3
Sharing	3
Accounts	3
Technical	3
System Architecture	3
Technologies	4
PHP	4
JavaScript	4
Ajax	4
MySQL	4
Laravel 5	4
Bootstrap	4
WAMP	4
Azure Cloud Server	4
Hardware & Software Requirements	4
Project Breakdown	5
Breakdown	5
Delivery Stages	5
References	6

## Project Description

### Introduction

The goal of this project is to develop a software product in an area of our choosing. The software must offer useful functionality and have a high level of innovation, either filling gaps in a current technology area or branch into an entirely new one.

### Project Overview

For this project we are going to create a social media website that dynamically shares your content with other users of the same interests. Every time a user posts an update to their page we will parse the text and extract the most relevant keywords and data from the text and use this data to determine who to share this information with.

Likewise the user who shares content will receive content based on the contents of their latest updates and or posts. This should result in an ever changing flow of relevant information based on your current and up to date interests. The more you talk about a specific subject or topic the more likely you are to see relevant posts about that subject. There will be no friends or followers on this site because all interactions are dynamic and ever changing.

### Background

The idea for this project came to me as a result of my own changing interests over the years. Take Facebook for example, you add a friend on Facebook and you receive all of their updates and posts even posts you may not be interested in. This leads to users posting updates that get no likes, no comments and no views. Even more so it opens up the opportunity for negative comments or bullying from “friends” that do not enjoy your posts about computer games or obscure TV shows you enjoy.

The aim here is to create a space where users can update their page and are guaranteed to have it viewed by people with the same interests regardless of friend or follower count.

### Target Market

The target market for this website would be any user that likes to engage in open discussions about their hobbies or interests with like minded individuals. A place where your posts don't live and die by your popularity or friend count, but rather by its content.

It's also for those users that are threatened by other social media sites where you actively have to look for friends and followers. Our site takes the pressure of finding like minded people off their shoulders and lets the users focus on what really matters, the message content.

### Possible Risks

The main risk with this project is lack of privacy. Users have no control over who views their content because it is accessible by every other user that is part of their group. Having said that, this website is not for sharing private information it is more of an open discussion forum.

### Project Functionality

This website will offer users a space to share content with other users who share the same interests. The interesting thing about this idea is that the user does not select who the content is displayed to; that is all taken care of in the back end. Because we keep track of all your posts and interactions we know that your interests have changed even if you don't.

## Dynamic Grouping

All users will be dynamically placed in groups based on the content of their post history. Groups are formed based on a user's top three interests e.g. football, music, TV if you stop talking about one of these topics you will eventually be moved from this group and get pushed into a different group e.g. football, golf, TV.

Groups only form when two or more people have the same interests. If there is no matching group for a user's three most common interests they will be put into a group that most closely matches their interests usually with a 2/3 match. Users can only see posts from that group as long as they remain in said group. The more frequently you post the more likely you are to move through different groups.

## Colour Matching

Each user will be given a colour based on their current interest which is attached to their profile and avatar. This colour is generated from their three most common interests and their weights using the RGB colour profile. This colour will indicate how closely your interest match with other people in your group at any given time.

## Sharing

Users will be able to share text, images and video in any post or comment. All posts are public to their respective groups meaning you can see all post pertaining to or similar to your current interests.

## Accounts

All users will need to register an account to access the platform. Users will have full control of their account including the editing or deleting of posts. All accounts require a username a password to login to the system.

## Technical

### System Architecture

This website will use the classic MVC architecture using the Laravel 5 framework.

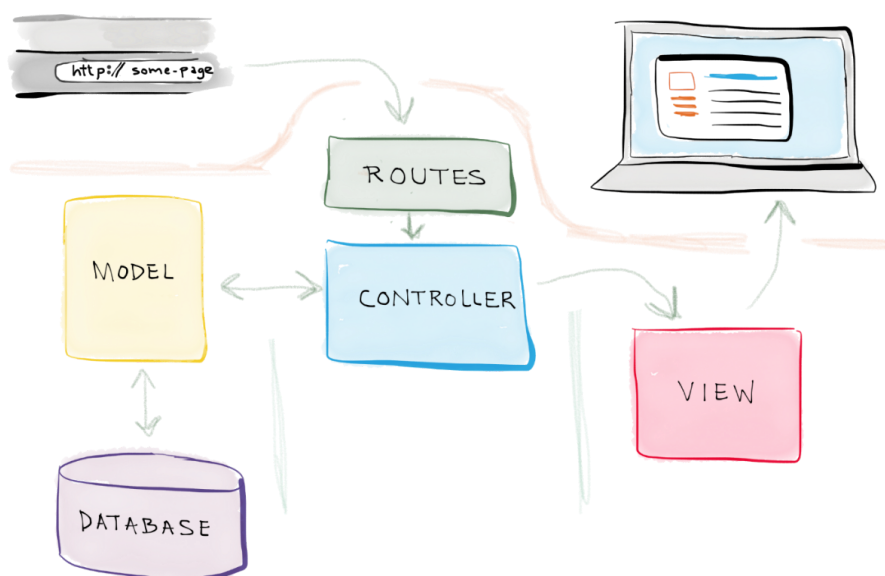


Figure 1 MVC request cycle in a Laravel 5 (Coleman, 2016)

## Technologies

### PHP

The main bulk of the website will be written in PHP.

### JavaScript

JavaScript will be used to add some dynamic elements on the front end as well as any client side validation that needs to be done. JavaScript frame works such as jQuery will be used where needed for design and accessibility features.

### Ajax

Ajax will be used to update portions of the website dynamically. It will also be used to add pagination to a user's news feed to improve load times and responsiveness.

### MySQL

An online MySQL database will be used to store all data pertaining to the project.

### Laravel 5

Laravel 5 is a PHP framework that will be used to streamline some of the more common website tasks such as sessions and cookies. It also has a lot of helper functions to manage database calls for saving, updating or displaying data.

### Bootstrap

Where possible the website will use the Twitter Bootstrap framework to minimise HTML and respond to changing screen sizes.

### WAMP

An all in one local server for running PHP, MySQL and Apache web servers.

### Azure Cloud Server

Everything from the PHP to the database and uploaded images will be stored on an Azure cloud based server during the deployment stage.

## Hardware & Software Requirements

- Desktop PC or Laptop
- WAMP Server
- PHPStorm
- GitHub
- MySQL
- Azure Server

This project will be completed using my personal desktop PC and or laptop computer from home. If work is to be carried out during class time in college I can easily pull the latest project files from GitHub. All development will be carried out on a local server using WAMP. My IDE of choice for this project will be PHPStorm because it offers some nice benefits when writing PHP code. The finished project will be deployed to an Azure Server for presentation and testing purposes.

## Project Breakdown

### Breakdown

Stages	Tasks	Hours
<b>Stage 1</b>	<b>Analysis &amp; Design</b>	
1.1	Mock Ups	6
1.2	Requirements Analysis	12
<b>Stage 2</b>	<b>Dataset</b>	
2.1	Source suitable data for project testing	6
2.2	Database Design	8
2.3	Extract, transform and load selected data to storage.	18
<b>Stage 3</b>	<b>Implementation</b>	
3.1	Website Design	40
3.2	Database functionality	20
3.3	Login and user management functionality	30
3.4	Post, edit, delete and view message functionality	24
3.5	Image and video uploading functionality	20
3.6	Dynamic clustering of users and messages functionality	40
3.7	Colour matching functionality	10
3.8	Additional pages of content, browser support	12
<b>Stage 4</b>	<b>Testing &amp; QA</b>	
4.1	Website testing and bug fixing	24
4.2	Server side load and speed testing	12
<b>Stage 5</b>	<b>Deployment</b>	
5.1	Deploy to server	4

### Delivery Stages

This project will be implemented in the following stages:

Stage 1 Analysis & Design	
<b>Tasks</b>	Sketch mock ups for UI design and gather functional and non-function requirements.
<b>Deliverables</b>	Mock up feedback and Requirements document
<b>Total Hours</b>	18
<b>Estimated Work (Days)</b>	3

Stage 2 Dataset	
<b>Tasks</b>	Source suitable dataset. Design database schema. Extract, transform and load data for website testing.
<b>Deliverables</b>	Working database of usable relevant data.
<b>Total Hours</b>	32
<b>Estimated Work (Days)</b>	5

Stage 3 Implementation	
Tasks	Website design and functionality. Database methods and functionality for basic operations. User login management. Sharing and uploading of content. Creation of grouping algorithm. Colour matching algorithm. Additional website features.
Deliverables	Working website with possible bugs.
Total Hours	196
Estimated Work (Days)	25

Stage 4 Testing & QA	
Tasks	Testing and bug fixing. Server load tests.
Deliverables	Fully working and responsive website.
Total Hours	36
Estimated Work (Days)	5

Stage 5 Deployment	
Tasks	Deployment of website to web server.
Deliverables	Fully working website on deployment server.
Total Hours	4
Estimated Work (Days)	1

## References

Coleman, A., 2016. *Self-Thought Coders*. [Online]

Available at:

<https://selftaughtcoders.com/from-idea-to-launch/lesson-17/laravel-5-mvc-application-in-10-minutes/>

[Accessed 13 October 2016].