Social Media Clone

Oliver Handford – Social Network

# Brief

The brief for this assignment was to build a social media web application that has the usual features you would expect from a social media platform. The platform can be created in any programming language and should be responsive where possible. The social media clone should also use a database to hold information.

# Specification

The specification from the brief was:

* Ability to register/login/logout.
* Ability to reset their password.
* Ability to view a user’s profile.
* Ability to edit their details/own profile.
* Ability to make posts that include: Text, Images and Tags
* Ability to reply to posts.
* Ability to like posts.
* Ability to delete content that the user has posted.

With the other requirement being that the website is secure from SQL Injections and other XSS techniques.

The technology that I have chosen to complete this project is: PHP & MySQL for the backend and VueJS for the frontend.

# Design and Technical Implementation

The application is split into two parts, the frontend, and the backend. I will break each down individually.

## Backend

The backend of my social media clone is written in PHP, using MySQL as a database to hold data. The project is setup to mimic the style of a MVC framework, however due to the simplicity of this application the MVC setup is not very thorough and only contains the required features.

Index.php is the entry point to the web app, and all traffic is directed to this file via the .htaccess file, which tells the Apache Web Server the rules for this application. Index.php also includes all of the controllers and classes for the application, making them immediately available to the rest of the application. This file also defines a few global variables and imports the routes.php file, containing all of the routes the application.

Routes.php contains all of the URLs that the application uses. This allows the application to know what page or information is being requested. When user attempts to access a page, Apache converts this to make the URLs clean but also passes the information to the PHP application. For example, when accessing the login page, the user sees <http://url.com/login> when in reality, they are actually accessing <http://url.com/index.php?url=login>.

The classes/Route.php file checks which route has been requested by using the GET ‘url’ parameter provided, and then uses the Magic Method \_\_invoke(). The invoke method allows the application to use the callback provided to process the route.

The controllers handle all of the application logic, such as registering or resetting a password. Every control extends the base Controller.php which has one function in, view(), which serves a view. This kind of setup allows the application to be easily extendable where required and could easily implement many more controllers for different logic.

The most important controllers in the application are the AuthController.php and the APIController.php. The AuthController handles all authentication for the application, such as registering, logging in, logging out and resetting a password. The APIController handles the fetching and POSTing of data for the VueJS frontend, and implements key logic such as making a post or deleting a post. The APIController always returns a valid JSON response, with the correct HTTP code to ensure smooth integration with the frontend.

One of the classes is Email.php, which is an implementation of <https://github.com/snipworks/php-smtp> to allow the application to easily send emails, which is required for password resets.

The DB.php class handles the database implementation. The DB class connects to the MySQL database via PDO, and then has multiple functions to interact with the database. Most of the functions implement the classes own insert() method, however the reason for this is to allow the application code to be more verbose and future proof in case additional functionality was needed to be implemented, such as deleting or modifying records in high-speed storage like Redis as well as the database.

## Frontend

The frontend is split into two parts, views are handled by PHP, and a controller will return exactly which view needs to be processed. A view implements a page in the application and will include a copy of the header and footer of the application as to no repeat the same code across every view.

All authentication related views and logic are handled by PHP, as in this instance it is not beneficial to use JavaScript.

All application logic, such as making posts or liking posts is done via VueJS. VueJS is a modern frontend JavaScript framework that allows for two-way data binding and lots of other features to speed up development of interactive frontend elements.

In the application I also make use of the package Axios. Axios is a HTTP client, that allows for simple requests to webpages and APIs, which is perfect for this web app. Together, Axios and Vue make a really powerful frontend system.

Below is an example of the function that submits a post:

        submitPost()

        {

            axios.post('/post', *this*.postThread)

            .then(*function* (*response*) {

*this*.success = true

*this*.error = false

*this*.successMessage = response.data.message

*this*.loadPosts()

            }.bind(*this*))

            .catch(*function* (*error*) {

*this*.error = true

*this*.errorMessage = error.response.data.message

            }.bind(*this*))

        },

It takes the data which is held in this.postThread and POSTs it to /post, which is http://url.com/post . This request is then handled by PHP, such as inserting it into the database, and then PHP responds with either an error message or a 200 HTTP code to signify that the request has been successfully completed.

Upon successful completion in the .then() callback, the application updates a few more variables to show a success message to the user and then reloads the posts on the page so the latest post is now visible.

### Uploading Pictures

I have used a different method to store picture data, instead of the traditional way of storing the file on the server. I decided that the best way to deal with this, as we are only dealing with pictures, is to convert the images to a Base64 string and then store that in the database. My reasoning behind this is it makes it much more difficult for somebody to upload a potentially dangerous file, the Base64 encoding is done on the frontend which reduces load on the server and all of the application data is held in one place.

Below is the way that the file gets converted to a Base64 string

        getBase64(*file*) {

*var* reader = new FileReader()

            reader.readAsDataURL(file)

            reader.onload = *function* () {

*//console.log(reader.result)*

*this*.postPicture.image = reader.result

*this*.isUploaded = true

            }.bind(*this*)

            reader.onerror = *function* (*error*) {

                console.log('Error: ', error)

            }

        },

This is called by a custom event:

        onPictureChangedEvent (*event*) {

*this*.getBase64(event.target.files[0])

        },

Which automatically converts the image to a Base64 string when it gets changed.

<input *class*="file-input" *type*="file" *@change*="onPictureChangedEvent" *accept*="image/\*">

Using the @change event watcher from Vue. The file upload also has the accept=”image/\*” tag which browsers use to attempt to stop the user uploading a file that is not supported.

In the viewing of images from the database, the Base64 string can then be used as the “src” of the image tag and will be converted back to a picture by the browser.

### Application Design

For the design of the application, I used the Bulma CSS framework to style the basic elements. I decided to keep the design of my application simple and user friendly, with everything having a clear purpose.

The CSS that I have written, and the Bulma import, has all been compiled using SASS and is in the global.scss file.

The application is responsive and can be easily used on all modern devices.

The navigation bar in the application manipulates the use of labels and input checkboxes to allow the application to have a responsive hamburger menu without using JavaScript.

The hamburger menu icon has an input of the type checkbox which is hidden. The label for this checkbox is then the hamburger menu icon, allowing it to check and uncheck the checkbox. Then I used the :checked selector in CSS to allow me to modify the style of the hamburger menu depending on if the checkbox has been checked or not.

# Database Design

Graphical user interface, diagram

Description automatically generated

The database structure for my social network clone is fairly simple. There are only 4 tables that are required.

## Posts

Graphical user interface, diagram

Description automatically generatedThe posts table includes all of the information for any post or reply that is on the social network. It’s PRIMARY key is an id which auto increments. The owner is a foreign key, to the users table, for the person who created the post.

Content is the text from the post. The timestamp is the UNIX timestamp at which the post was created.

isPicture is an INT that holds either 0 or 1, to hold boolean values. This determines if the post contains a picture. pictureData then is a LONGTEXT which allows up to 4GB worth of data to be held, which holds the Base64 string of a picture if the post contains one.

Deleted is an INT holding a boolean value, which determines if a post has been deleted. This is used instead of actually deleting posts to allow for posts be restored or monitored by an admin if required.

isReply is an Int holding a boolean value to determine if the post is a reply to another post. Parentid is a foreign key, to the posts table, which is the ID of the post a post is replying to in the case that it is a reply.

By having this structure and using posts for both posts and replies, it formalises the structure of a post and keeps it consistent across the whole application. It also allows the same functions, such as delete and like a post, to be reused.

## Users

Graphical user interface, diagram

Description automatically generatedThe users table holds all of the information about a user of the social media clone. It’s PRIMARY key is an id which auto increments.

The username is a VARCHAR of max length, that holds the users username that they use to login. The email field is also a VARCHAR which holds the users email address for resetting passwords.

The password field is also a VARCHAR which holds a password hash for the user to be compared with during login. The timestamp is the time of creation of the user.

The apikey holds the identifier that is used in the frontend when accessing the applications data or making a post and is unique.

By using this structure, the user’s information is kept clean and obvious so that it can be extended if required. No unnecessary information is kept in this table, such as post count or like count which can all be retrieved by other queries.

## Reset Token

Graphical user interface, diagram

Description automatically generatedThe reset token table holds a copy of the codes that are emailed to a user when they reset their password, so that the application can verify if the user is the person resetting their password. It’s PRIMARY key is an id which auto increments.

The userid is a foreign key, to the users table, which identifies a user by their ID. This is the user that has requested to reset their password.

Code is a VARCHAR that holds a copy of the code that was emailed.

Timestamp is the time at which a user was sent the code via email, so that the application can identifiy how old the code is and if its been live longer than 30 mins.

This is a simple structure, that keeps a log of all password resets and the time at which they were done, but does not harvest any unnessacary user information.

## User Has Liked

Graphical user interface, diagram

Description automatically generatedThe user has liked table is a link table that has a many-to-many relationship between a user and a post. This table shows when a user has liked a post.

The userid is a foreign key, which relates to a user in the users table.

The postid is a foreign key, which relates to a post in the posts table.

The PRIMARY key for this table is both the userid and the postid comibned, which means that in the application, an INSERT IGNORE query can be used instead of an INSERT query to stop duplicate likes or entries.

This table is important as it creates the many-to-many relationship desired in the application, and makes use of the benefits of a relational database system.