COSC 360 M4 - System Description

Joy Musiel-Henseleit | *Samantha Scully* | *Ian Steyn*

Site Architecture

File Organization

A good way to get an overview of our code is to understand how our files are organized. For more detailed info, see the files themselves; they are well documented.

`/public_html/` - All files that are served directly to the client, including `css/` stylesheets, JS `scripts/`, SVG `vector-icons/`, and `index.php`.

`/config/` - Abstracts away application configuration code since it can change based on the specific server and database. Also contains utilities for DB connection and URL generation.

`/app/` - contains core application logic; this is where most of our PHP files are.

- `models/` encapsulate database interaction logic.
- `views/` encapsulate presentation logic. Mostly HTML with PHP variables.
- `controllers/` handle client requests. Mediators between the model and view layers.
- `services/` encapsulate logic for specific domains (errors, access control, authentication status)
- `helpers/` contain common utilities for models, views and controllers respectively
- `routing/` encapsulates routing and redirection logic

High-level Overview

The system is built with pure PHP, HTML, CSS, and JavaScript (no Jquery, and no front-end or back-end frameworks). Our project uses a Model-View-Controller architecture, which separates the concerns of data, presentation, and request handling. It also uses a routing system; instead of directly calling different PHP script files, requests are made to specific routes (nice looking URLs) which call controller functions. The *controllers* are the core logical components. They handle user requests and decide on the response, by calling on *models* to send and fetch data from the database, and passing data to *views* for presentation.

Routes

Routes are encoded in the form `<base-url>/?route=/example/path`. Encoding routes as a query parameter like this has some limitations, but it was a necessary work-around because the university server does not allow request rewriting with `.htaccess` files, and we had no control over its configuration. For this reason, please always leave the route query parameter in the URL when testing the site. For example, do not test `<base-url>/fake`, as it will serve only a default Apache 404 page. Instead, try `<base-url>/?route=/fake`, and you will see our custom 404 page.

Root File

Since there is no other file name in the route, each page or action request actually calls `index.php`, and this is the file that is ultimately rendered into an HTML document. It sets up everything the application needs: it resumes the current session, opens a database connection, includes common service and utility classes, initializes controllers, defines all the routes, and finally matches the request URL with the correct route. From there, a controller function is called, which will likely call some service and model functions, and then display a view or redirect to another route.

Authentication and Error handling

Because both actions (like deleting a post) and pages (like the admin dashboard) are requested with simple client-readable routes, there is potential for abuse. For this reason, controllers make use of our access control, authentication, and error handling services, which will redirect unauthorized users to login or error pages if they try to make invalid requests. See PR #193 for some example routes to try.

Front-end Scripting

Our back-end is supported by a small set of front-end scripts, which perform a variety of tasks: Functionally, they validate forms, send asynchronous requests to the server, handle server responses, and provide user confirmation dialogues for irreversible actions. Cosmetically, they load the colour theme, and set the nav's highlighting and collapse state.

Feature Checklist

A checklist of implemented features. Items are numbered according to the Milestone 4 rubric.

1 & 2 - Security

- ☑ **Database:** All model functions use PDO prepared-statements
- ☑ **File access:** All application and configuration files are outside of `/public_html/`
- ☑ **User accounts:** Registering with a username/email that is already taken is not allowed. Password required to update user settings. Access control for many pages (see above).
- ☑ **Input validation:** JS input validation for emails, usernames, passwords, images and required fields
- ☑ **XSS prevention:** helper function `sanitizeData()` is used to escape all data for views
- ☑ **Protecting passwords:** Password input-type is used. Passwords are hashed using the *bcrypt* algorithm.

3 & 4 - Database Storage

- ☑ **(3) General:** Users, posts, comments, likes, and saves all stored in database
- ☑ **(4) Images:** User and post images are stored as `MEDIUMBLOB` objects in the database

5 & 6 - Version Control, Testing, and Server Deployment

- ☑ **Project management:** We used a GitHub Project board to manage and assign issues/tasks
- ☑ **Version control and review process:** All changes were made in separate branches, and were *required* to be reviewed before merging. Reviewers tested the website locally, and looked at the code itself, often requesting changes if problems were found. There are lots of good examples, but some fairly representative PRs are #76, #80, #127, #149.
- ☑ **User testing:** once we deployed on the remote server, we sent the site to several people and asked them to mess around and find bugs. This is why there are a bunch of random posts on the website.
- ☑ **Other forms of testing:** We also did HTML Validation early on (#84), and intentionally used the dependency injection design pattern for our models and controllers (which set it up well for a theoretical future unit testing framework).
- ☑ **Configuration**: to make developing with both remote and local servers easier, we created AppConfig to set the correct base URL, DB connection info, and production/dev favicon, based on the current server.

7 & 8 - Admin Functionality

- ☑ **(7) General Admin Functionality:** Admin dashboard displays site analytics including post and account information, and admins can delete the posts and comments of other users
- ☑ **(8) User Search:** Admins can see a list of all users, and search for users by username

9, 10, 11 - Core User Functionality

- ☑ **(9) Account Creation:** Users can register with username, email, password, and profile picture, and log in with email and password.
- ☑ **(10) Viewing and Editing Profile:** Profile page which shows your bio, picture, created and saved posts, and allows you to change password, profile picture, and add/edit bio.
- ☑ **(11) Content Creation:** Registered and admin users can create and comment on blog posts.

12, 13, 14 - Specialized Functionality

- ☑ **(12) Asynchronous Updates:** *Likes* and *saves* are updated asynchronously (although posts are not), and *user search* updates as you type (see `post-interaction.js` and `admin.js` for AJAX code)
- ☑ **(13) Log-in State:** maintained using `$_SESSION` variables and `Auth` service classes
- ☑ **(14) Topic Grouping:** Not completed, however: we believe that we have so many additional features under item 24, that full marks should still be considered for this section.

15, 16, 20 - Additional Items (specified by rubric)

- ☑ **(15) Search and analysis for topics/items:** post titles can be searched by space-separated keywords on a dedicated search page.
- ☑ **(16) Hot item tracking: "**Popular" tab shows the most liked posts of the past month, and admin dashboard shows the top 5 most liked posts.
- ☑ **(20) Collapsible items without page reloading:** Side-nav is collapsible without page reload.

24 - Additional Items (of our choice)

- ☑ **Saving:** users can save posts and view them on home or profile page, sorted by most recently saved.
- ☑ **Liking:** users can like a post, contributing to its popularity. Admins see exact like numbers of top posts.
- ☑ **Editing posts:** users can edit their posts. Edits are stored in the database without changing the post_id.
- ☑ **Deleting:** Both comments and posts can be deleted by their owners, or by admins.
- ☑ **Viewing other user profiles:** Registered users can view the profiles of other users, including their saved posts. You cannot edit the profile of other users.
- ☑ **Color theme options:** Fully implemented theme switching with light, dark, and system default options. Favicon also changes based on the theme!
- ☑ **Contextual navigation:** Side-navigation highlights current page, tabbed pages indicate the current tab, and specific posts show breadcrumbs to user profiles.
- ☑ **Routing and nice URLs:** site uses a routing system, which allows aesthetic and maintainable URLs.
- ☑ **Robust error handling:** Error pages for 400, 403, 404 errors, which actually give the browser the correct HTTP response (open your network tab to verify this).
- ☑ **Robust authentication and access control:** Unregistered users cannot like or save posts, create posts or comments, or view profiles. Non-admin users cannot access the admin page. Users cannot accidentally modify resources, or modify resources they do not own.
- ☑ **General good structure:** Our code-base is very well organized and documented, allowing for theoretical future testability and maintainability.

Known Limitations

- **Image file size limitations:** Due to server configuration out of our control, image files must be under 2 MB
- **Performance:** page-loading can be slow on pages that load a lot of posts. The fix for this would be to implement pagination, so that not all posts are fetched from the database at once.
- **Word-wrapping bug:** a very particular bug that is unlikely to affect most users. Very long unbroken strings in post bodies will not wrap properly.