COMS32500 Web Technologies Report

Overview

The Source game Engine has a notably level of customisation that it affords users, which has led to the creation of sites that allow creators to share their content. This includes maps, stored as (.bsp, or binary space partition) files, scripts that execute console commands called configs, custom models, and many more.

I wanted to build a site that supports uploading, downloading, favouriting and commenting on such content. A desire to understand more about server-side handling encouraged me to take a raw approach to development, not using Express or any other framework. I used SQLite 3 as an embedded database.

I worked by myself (jw17943).

Running the server

The server runs on the port 8080.

I have used a script `reset_database.bat` to delete the database and recreate it with some dummy users and uploads for the sake of development. This includes creating folders and copying over some test files and images.

There are 6 test users: `admin`, `testuser`, `tim`, `bob`, `alfie`, and `dean`. Each has the password '`password`'.

There are also 4 test uploads and 7 test comments. The users and comments are listed by logging in as `admin`, who is the only default moderator, and visiting /admin or clicking the link in the navbar.

Estimated Grades

- A for HTML
- A for CSS
- A for JS
- C for PNG
- A for SVG
- A for Server
- B for Database
- A+ for Dynamic pages

HTML

I developed my HTML pages first with template data which was then replaced with integrated HTML generated by server requests. To do this, I had to work with loading scripts and CSS in HTML as well as generating well-formed HTML in JavaScript.

Consistent use of XHTML made it easy to validate that my HTML was well-formed.

CSS

The CSS can be found in `public/resources/css/`.

I have written a range of stylesheets, including two global stylesheets to handle the page and navbar. This includes styling tables in `grid_content.css`, flexboxes in `page_content.css`, and several forms, for example in `upload_content.css`.

Animation was done in CSS in `page_content.css` to navigate uploaded images, although this was done using https://www.w3schools.com/howto/howto_js_slideshow.asp as a reference.

JS

The JS can be found in `public/resources/scripts/`.

I made heavy use of client-side JavaScript to interact with the server, for example in `insert_navbar.js`, to dynamically switch between upload categories in `upload_formopener.js`, and to navigate an image slideshow in `handle_slideshow.js`.

As well as this, I store the `user_id` and `sessionkey` values in the browser `sessionStorage`, to keep track of the user currently logged in as they navigate the website.

All requests to the server were made using `XMLHttpRequest`s.

PNG

PNG images can be found in `public/resources/img/`.

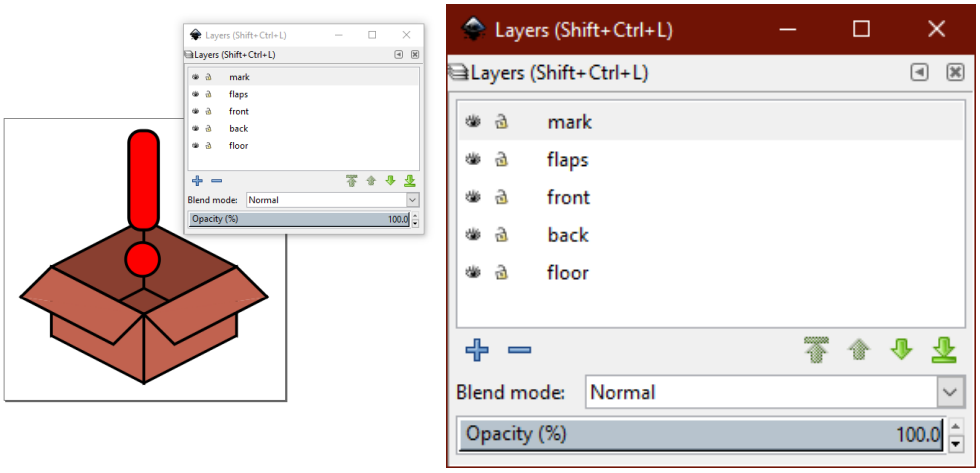
I created several PNGs with SVG counterparts for use as a thumbnail on uploads with no accompanying images, such as `cfg_png.png`:

SVG

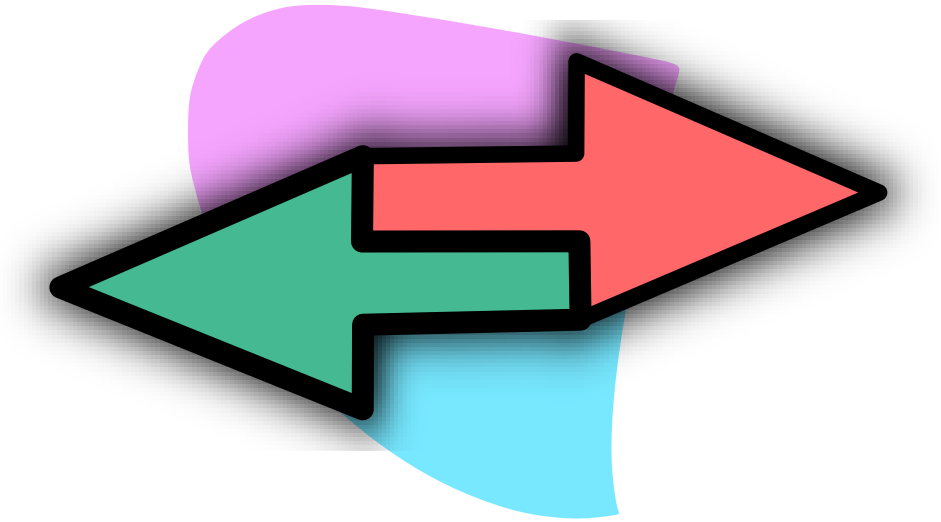
SVG graphics can also be found in `public/resources/img/`.

I used Inkscape to create a range of SVGs which can be loaded by visiting /logo or clicking the link in the navbar. Freeform drawing was used in `missing.svg`.

To create the images, layers helped me order and group segments to create clean and images:



I am particularly happy with the use of careful layering and design to create `emptybox.svg`, as well as `logo2.svg`:



Server

I programmed with a raw server using let http = require("http");

This included server-side validation of all user inputs, handling of POST requests including file uploads and downloads, and managing user access around the website.

I am particularly happy that I managed to get the address /home redirecting to the current user's logged in page in a way that reuses the code for handling /user/x, and that I managed to get all post requests parsed and sent back to a single function with the help from this snippet:

https://itnext.io/how-to-handle-the-post-request-body-in-node-js-without-using-a-framework-cd2038b93190?gi=d6a8f3e99295

I also made use of several nodeJS packages:

- `multiparty`, to parse `multipart/form-data` POST requests
- `querystring`, to parse POST requests
- `mkdirp`, to recursively make folders
- `rimraf`, to recursively delete folders
- `ncp`, to recursively copy folder contents

Although I was not able to implement some of the features I would've liked to due to time constraints, such as self-certifying or hosting, I am still happy with the achieved server considering I was working alone.

Database

I use three tables: users, uploads, and comments, to store submitted data in a relational database.

All necessary fields are tagged with NOT NULL, and each has a PRIMARY KEY id.

Where a user is interacting with the database, I only interact with prepared statements to prevent SQL injection.

The comments table was implemented to resolve a many-many issue, where many users can comment on many uploads. It has two foreign id keys to link each comment to its respective user and upload.

Dynamic Pages

I am particularly happy with the dynamic structure of my website, done through a combination of both inserting data into templates by the server, and requesting data from the server and inserting it.

Some pages like `index.html` use templates, whereas a page like `user.html` must also handle the request for /home. The server does not know when handling /home who the user is, so this request is made after loading the page.

Each page also requests the server for a dynamic navbar. No current user will result in a login option, where a current user will give options to upload, visit their profile, and log out. Admins will also see a link to /admin.

The /admin page makes a request to the server that returns two tables, for users and uploads. I had some difficulty parsing the result of updating these tables; however, using dynamic ids to identify each user I was able to do it cleanly and efficiently. Dynamically the first user can also not be deleted or removed as a moderator. Using a script, the tables can also be hidden, if they get too long, which works very effectively.

Links on pages like the landing page are also all dynamically loaded, meaning navigating the site is smooth and works very well. Links to users lead to links to their content which leads to comments which leads to more users…

Lastly, I am proud of the responsive updating of favouriting and downloading content.

I have learnt a lot about raw website design and scripting with JavaScript, which I have not used before and somewhat enjoyed, particularly the modularity of Node JS packages.