server.js

```
const Server = require('./framework/server-class'); // Server framework
class

let mainServer = new Server.Server('localhost','8080',{}); // Instantiate
Server
require('./database/init.js').initDB().then(() => { // Wait for the DB to
be initialised
    mainServer.openDB(__dirname + '/database/dev.db'); // Initliase DB in
mainServer
    // Placeholder db path for production db
    mainServer.run(); // Run the server
}); // Initialises DB and runs server
```

database/db-mgmt.js

```
const sqlite3 = require('sqlite3').verbose();
const sqlite = require('sqlite');
/** Class used to perform database operations and store connection */
class dbManager {
    /**
     * Opens the database and stores connection
     * @param {String} dbPath - An absolute file path to the DB
     * @constructor
     */
    constructor (dbPath) {
        this.db = sqlite.open({
            filename : dbPath,
            mode: sqlite3.OPEN_READWRITE,
            driver: sqlite3.cached.Database
        }).then((res) => {
                res.on('trace', (data) => {
                    console.log(data);
                }); // For debuging, prints any SQL statements recieved by
the
                    // DB to the console.
                res.exec('PRAGMA foreign_keys = ON;'); // Enables foreign
keys
                return res;
        }).catch((err) => {
            console.log('error in opening db');
            console.log(err);
            process.abort(); // Fail - failed to open DB - probably invalid
file path
        }); // Opens the DB - NOTE
            // THIS IS ASYNC
            // Essentially if a request is passed to this object AS it is
```

```
created,
            // or until however long it takes to open the db, it will fail.
    }

    /**
     * Runs some SQL statement and DOES NOT return result
     * @param {String} sql
     * @param {Object} params
     * @returns {Object}
     */
    async _dbExec(sql, params) {
        return await (await this.db).run(sql, params);
    }

    /**
     * Runs some SQL statement and returns the first result row
     * @param {String} sql
     * @param {Object} params
     * @returns {Object} Row
     */
    async _dbGet(sql, params){
        return await (await this.db).get(sql, params);
    }

    /**
     * Runs some SQL statement and returns all result rows
     * @param {String} sql
     * @param {Object} params
     * @returns {Array} All rows
     */
    async _dbAll(sql, params){
        return await (await this.db).all(sql, params);
    }
}

module.exports = {dbManager};
```

database/init.js

```
const sqlite3 = require('sqlite3');
const sqlite = require('sqlite');

async function initDB() {
    const db = await sqlite.open({
        filename : __dirname + '/dev.db',
        driver: sqlite3.Database
    })

    await db.exec(`CREATE TABLE IF NOT EXISTS accountTbl (
            username TEXT NOT NULL UNIQUE,
```

```
            password TEXT NOT NULL,
            salt TEXT NOT NULL,
            PRIMARY KEY (username)
            );`);
    await db.exec(`CREATE TABLE IF NOT EXISTS projectTbl (
            projectID INTEGER NOT NULL UNIQUE,
            username TEXT NOT NULL,
            projectName TEXT,
            PRIMARY KEY (projectID),
            CONSTRAINT fk_accountTbl
                FOREIGN KEY (username)
                REFERENCES accountTbl(username)
                ON DELETE CASCADE
            );`);
    await db.exec(`CREATE TABLE IF NOT EXISTS contentTbl (
            projectID INTEGER NOT NULL,
            type INTEGER NOT NULL,
            content TEXT,
            PRIMARY KEY (projectID, type),
            CONSTRAINT fk_projectTbl
                FOREIGN KEY (projectID)
                REFERENCES projectTbl(projectID)
                ON DELETE CASCADE
            );`);
    // Creates tables if they do not exist
    db.close();
}

module.exports = {initDB};
```

framework/database-class.js

```
const dbManagement = require('../database/db-mgmt');
const crypto = require('node:crypto');

/** Class to handle all database operations
 * @extends dbManager
 */
class DatabaseAccess extends dbManagement.dbManager { // inherits
dbManagement.dbManager
    constructor(db_path) {
        super(db_path); // PLACEHOLDER name for production db
    }

    /**
     * Generates a salt of the desired length
     * @param {Number} length
     * @returns {String} salt
     */
    static #generateSalt(length){
```

```
        return crypto.randomBytes(Math.ceil(length /
2)).toString('hex').slice(0,length);
        // https://blog.logrocket.com/building-a-password-hasher-in-node-
js/
    }

    /**
     * Uses HMAC with sha512 and a salt to hash some input data
     * @param {String} plaintext
     * @param {String} salt
     * @returns {String} hashed data
     */
    static #hash(plaintext, salt){
        let hash = crypto.createHmac('sha512', salt);
        hash.update(plaintext);
        return {hashedValue: hash.digest('hex'), salt: salt};
    }

    /**
     * Compares a plaintext password with a hashed password
     * @param {String} inputPassword
     * @param {String} salt
     * @param {String} desiredPassword
     * @returns {Boolean} boolean result
     */
    static #validatePassword(inputPassword, salt, desiredPassword){
        let hashedPassword = DatabaseAccess.#hash(inputPassword,
salt).hashedValue;
        return hashedPassword == desiredPassword;
    }

    /**
     * Creates an account -
     * Can fail, returning an errno of 19 if the username already exists,
     * or an errno of 0 if the password is invalid
     * @param {String} username - Account username
     * @param {String} password - Plaintext password
     * @returns {Object} Either:
     *              - {lastID, changes} for success
     *              - {errno, errdsc} for fail
     */
    createAccount(username, password){
        if (password.length < 8){
            return {errno: 0, errdsc: 'Password must be a minimum of 8
characters long.'};
        } // Fail - password not long enough
        if (username.length > 20 || username.length < 1) {
            return {errno: 0, errdsc: 'Username must be within 1-20
characters long.'};
        } // Fail - username invalid
        const hashInformation = DatabaseAccess.#hash(password,
DatabaseAccess.#generateSalt(128));
        return this._dbExec('INSERT INTO accountTbl (username, password,
salt) VALUES ($username,$password, $salt);', {
```

```
            $username : username,
            $password : hashInformation.hashedValue,
            $salt: hashInformation.salt
        }).then((result) => {
            if (result.changes == 1){
                return result; // Success - created account
            }
        }).catch((err) => {
            if (err.errno == 19){
                console.log("Username already exists.");
                return {errno: 0, errdsc: "Username already exists."};
            }
            console.log(err); // Dont log 'username already exists' errors
            return err; // Fail - unknown
        });
    }

    /**
     * Attempts to login -
     * Can fail, returning an errno of 0 if the password is incorrect or if
there are no accounts found
     * @param {String} username - Account username
     * @param {String} password - Plaintext password
     * @returns {Object} Either:
     *              - {username, success} if success
     *              - {errno, errdsc} if failiure
     */
    login(username, password){
        return this._dbGet('SELECT password, salt FROM accountTbl WHERE
accountTbl.username = $username;', {
            $username : username
        }).then((result) => {
            if (result){
                if (DatabaseAccess.#validatePassword(password, result.salt,
result.password)){
                    return {username: username}; // Success
                } else {
                    return {errno: 0, errdsc: 'Wrong password'}; // Fail -
wrong password
                }
            } else {
                return {errno: 0, errdsc: 'No account found'}; // Fail - no
account found
            }
        }).catch((err) => {
            console.log(err);
            return err; // Fail - unkown
        });
    }

    /**
     * Returns all projects associated with a specific account -
     * Can fail, returning an errno of 0 if no projects are found
     * @param {String} username - Account username
```

```
     * @returns {Object} Either:
     *              - An array of {projectName, projectID} if success
     *              - {errno, errdsc} if failiure
     */
    getProjects(username){
        return this._dbAll('SELECT projectName, projectID FROM projectTbl
WHERE projectTbl.username = $username;', {
            $username: username
        }).then((result) => {
            if (result.length != 0){
                return result; // Success
            } else {
                return this._dbGet('SELECT username FROM accountTbl WHERE
accountTbl.username = $username', {
                    $username: username
                }).then((result) => {
                    if (result){
                        return {errno: 0, errdsc: 'No projects found.'} //
Fail - no projects saved
                    } else {
                        return {errno: 1, errdsc: 'No account found.'} //
Fail - no account found
                    }
                })
            }
        }).catch((err) => {
            console.log(err);
            return err; // Fail - unkown
        });
    }

    /**
     * Attempts to delete an account
     * @param {String} username - Account username
     * @param {String} password - Plaintext password
     * @returns {Object} Either:
     *              - {lastID, changes} if success
     *              - {errno, errdsc} if failiure
     */
    deleteAccount(username, password){
        return this.login(username, password).then((result) => {
            if (result.username == username){
                return this._dbExec('DELETE FROM accountTbl WHERE
accountTbl.username = $username;', {
                    $username: username
                }).then((result) => {
                    return result;// Success
                });
            } else {
                return result; // Fail - login failed
            }
        }).catch((err) => {
            console.log(err);
            return err; // Fail - unkown
```

```
        });
    } // Attempts to delete account

    /**
     * Creates a new project or updates an existing one
     * @param {String} username - Account userame
     * @param {String} password - Plaintext password
     * @param {String} projectName - Project Name
     * @param {Array[String]} projectContent - Project content
     * @param {Number} projectID - Project ID
     * @returns {Object} Either:
     *              - {projectID} if success
     *              - {errno, errdsc} if failiure
     */
    saveProject(username, password, projectName, projectContent, projectID)
{
        return this.login(username, password).then((result) => {
            if (result.username == username) {
                if (projectID){
                    return this._dbGet('SELECT projectName FROM projectTbl
WHERE projectTbl.projectID = $projectID and projectTbl.username =
$username;', {
                        $projectID: projectID,
                        $username: username
                    }).then(result => {
                        if (result && result.projectName == projectName) {
// Project already exists
                            projectContent.forEach((element, index) => {
                                this._dbExec('UPDATE contentTbl SET content
= $content WHERE contentTbl.projectID = $projectID AND contentTbl.type =
$type;', {
                                    $projectID: projectID,
                                    $type: index,
                                    $content: element
                                });
                            });
                            return {projectID: projectID}; // Success
                        } else { // Project is being forked (same ID
different name)
                            return this._dbExec(`INSERT INTO
projectTbl(projectName, username) VALUES (
                                $projectName, $username
                            );`, {
                                $projectName: projectName,
                                $username: username
                            }).then((result) => {
                                projectID = result.lastID;
                                projectContent.forEach((element, index) =>
{
                                    this._dbExec('INSERT INTO contentTbl
VALUES ($projectID, $type, $content);', {
                                        $projectID: projectID,
                                        $type: index,
                                        $content: element
```

```
                                    });
                                });
                                return {projectID: projectID}; // Success
                            });
                        }
                    });
                } else { // Project does not already exist
                    return this._dbExec(`INSERT INTO
projectTbl(projectName, username) VALUES (
                        $projectName, $username
                    );`, {
                        $projectName: projectName,
                        $username: username
                    }).then((result) => {
                        projectID = result.lastID;
                        projectContent.forEach((element, index) => {
                            this._dbExec('INSERT INTO contentTbl VALUES
($projectID, $type, $content);', {
                                $projectID: projectID,
                                $type: index,
                                $content: element
                            });
                        });
                        return {projectID: projectID}; // Success
                    });
                }
            } else {
                return result; // Fail - Login failed
            }
        }).catch((err) => {
            console.log(err);
            return err; // Fail - unkown
        });
    }

    /**
     * Gets all project data associated with a projectID
     * @param {Number} projectID - Project ID
     * @param {boolean} metadata - Whether or not to only send metadata
(project name)
     * @returns {Object} Either:
     *              - {projectContent, projectName} if success
     *              - {errno, errdsc} if failiure
     */
    loadProject(projectID){
        projectID = Number(projectID);
        if (Number.isNaN(projectID)) {return {errno: 0, errdsc: 'Invalid
project ID'};}
        return this._dbAll('SELECT content, type FROM contentTbl WHERE
contentTbl.projectID = $projectID;', {
            $projectID: projectID
        }).then(result => {
            if (result.length != 0){
                let projectContent = [0,0,0,0];
```

```
                result.forEach(element => {
                    projectContent[element.type] = element.content;
                });
                return this._dbGet('SELECT projectName FROM projectTbl
WHERE projectTbl.projectID = $projectID;',{
                    $projectID: projectID
                }).then(result => {
                    return {projectContent: projectContent, projectName:
result.projectName}; // Success
                })
            } else {
                return {errno: 0, errdsc: 'No project found'}; // Fail - no
project found
            }
        }).catch(err => {
            console.log(err);
            return err; // Fail - unkown
        });
    }

    /**
     * Delets the given project
     * @param {String} username - Account username
     * @param {String} password - Plaintext password
     * @param {Number} projectID - Project ID
     * @returns {Object}  Either:
     *              - {lastID, changes} if success
     *              - {errno, errdsc} if failiure
     */
    deleteProject(username, password, projectID){
        projectID = Number(projectID);
        if (Number.isNaN(projectID)) {return {errno: 0, errdsc: 'Invalid
project ID'};}
        return this.login(username, password).then((result) => {
            if (result.username == username){
                return this._dbExec('DELETE FROM projectTbl WHERE
projectTbl.projectID = $projectID AND projectTbl.username = $username', {
                    $projectID: projectID,
                    $username: username
                }).then(result => {
                    if (result.changes != 0){
                        return result; // Success
                    } else {
                        return {errno: 0, errdsc: 'No project found on
account'}; // Fail - no project found
                    }
                })
            } else {
                return result; // Fail - login failed
            }
        }).catch((err) => {
            console.log(err);
            return err; // Fail - unkown
        });
```

```
    }
}

module.exports = {DatabaseAccess};
```

framework/server-class.js

```
const http = require('node:http'); // set to https later
const fs = require('node:fs');
const path = require('node:path');
const dataBaseClass = require('./database-class');

/**
 * Class containing HTTP server logic
 */
class Server {
    /**
     * @constructor
     * @param {String} hostname
     * @param {Number} port
     * @param {Object} options - Options to be passed to the HTTP server
     */
    constructor (hostname, port, options){
        this.hostname = hostname;
        this.port = port
        this.options = options;
    }

    /**
     * Creates an instance of DatabaseAccess to use for DB operations
     * @param {String} db_path - Absolute path to database file
     */
    openDB (db_path) {
        this.dbAccess = new dataBaseClass.DatabaseAccess(db_path);
    }

    /**
     * Gets all contents and MIME types of files in the specified directory
and returns them as an object
     * @param {String} filePath - Path to search directory
     * @returns {Object} An object containing the contents of the provided
directory and any subdirectories
     */
    static #recursiveReadDir(filePath){
        let data = {}
        fs.readdirSync(filePath).forEach((file) => {
            if (fs.statSync(filePath + file).isDirectory()){
                data[file] = Server.#recursiveReadDir(filePath + file +
'/');
            } else {
                data[file] = {content: fs.readFileSync(filePath + file),
```

```
                    type:
Server.#findMIMEType(path.extname(file).slice(1))};
            }
        });
        return data
    }

    /**
     * Follows a search path provided in searchArray through a provided
object
     * E.g: ['dir1','dir2','file'] corresponds to 'dir1/dir2/file'
     * @param {Object} obj Object to be searched
     * @param {Array} searchArray Search array
     * @returns {Object} An object containing the contents and MIME type of
a file
     */
    static #recursiveObjSearch(obj, searchArray){
        try { var returnObj = obj[searchArray[0]];}
        catch { return null;}
        if (searchArray.length == 1){
            return returnObj
        } else {
            return Server.#recursiveObjSearch(returnObj,
searchArray.slice(1));
        }
    }

    /**
     * Converts file extensions into the appropriate MIME type
     * @param {String} fileExt The file extension
     * @returns {String} MIME type
     */
    static #findMIMEType(fileExt){
        switch (fileExt){
            case 'js':
                return 'text/javascript';
            case 'html':
                return 'text/html';
            case 'css':
                return 'text/css';
            case 'png':
                return 'image/png'
            default:
                return 'text/html'
        }
    }

    /**
     * Sends a resource specified in the url with http
     * @param {http.ServerResponse} res
     * @param {Object} resourceDirectory An object produced by
Server.#recursiveReadDir()
     * @param {String} url The target url
     */
```

```
    static #getResource(res, resourceDirectory, url){
        url = decodeURIComponent(url);
        if (url == '/'){
            url = '/account_page.html'; // Default page
        }
        let urlArray = url.split('/').slice(1);
        let searchIndex = urlArray[0].indexOf('?');
        if (searchIndex != -1){
            urlArray[0] = urlArray[0].slice(0,searchIndex);
        }
        let tempResource = Server.#recursiveObjSearch(resourceDirectory,
urlArray);
        if (!tempResource) {
            let fofResource = Server.#recursiveObjSearch(resourceDirectory,
['404-page.html']);
            res.writeHead(404, {'Content-Type':'text/html'});
            res.end(fofResource.content);
        } else {
            if (tempResource.type == null){
                Server.#error(res, 500);
            } else {
                res.writeHead(200, {'Content-
Type':`${tempResource.type}`});
                res.end(tempResource.content);
            }
        }
    }

    /**
     * Handles a post request (probably by passing information to
this.dbAccess) and sends a response with http
     * @param {http.ServerResponse} res
     * @param {Object} body Body content of the request -> This is assumed
to be JSON encoded
     */
    async #postResourceJSON(res, body){
        const reqBody = JSON.parse(body);
        console.log(reqBody);
        let resultContent = {};
        switch (reqBody.method) {
            case 'create-account':
                if (reqBody.username && reqBody.password) {
                    resultContent = await
this.dbAccess.createAccount(reqBody.username, reqBody.password);
                }
                break;
            case 'log-in':
                if (reqBody.username && reqBody.password) {
                    resultContent = await
this.dbAccess.login(reqBody.username,reqBody.password);
                }
                break;
            case 'get-projects':
                if (reqBody.username) {
```

```
                    resultContent = await
this.dbAccess.getProjects(reqBody.username);
                }
                break;
            case 'delete-account':
                if (reqBody.username, reqBody.password) {
                    resultContent = await
this.dbAccess.deleteAccount(reqBody.username, reqBody.password);
                }
                break;
            case 'save-project':
                if (reqBody.username && reqBody.password &&
reqBody.project_name && reqBody.project_content.length == 4 &&
Array.isArray(reqBody.project_content)) {
                    resultContent = await
this.dbAccess.saveProject(reqBody.username, reqBody.password,
reqBody.project_name, reqBody.project_content, reqBody.projectID);
                }
                break;
            case 'load-project':
                if (reqBody.projectID) { // This will fail if project ID ==
0, but as the ID's start from 1, not a problem
                    resultContent = await
this.dbAccess.loadProject(reqBody.projectID);
                }
                break;
            case 'delete-project':
                if (reqBody.username && reqBody.password &&
reqBody.projectID) {
                    resultContent = await
this.dbAccess.deleteProject(reqBody.username, reqBody.password,
reqBody.projectID);
                }
                break;
            default:
                break;
        }
        if (Object.keys(resultContent).length == 0){
            Server.#error(res, 500);
            return;
        }

        res.writeHead(200, {'Content-Type':'application/json'});
        if (resultContent.errdsc) {
            res.end(JSON.stringify({error: resultContent, stmtResult:
null}));
        } else {
            res.end(JSON.stringify({error: null, stmtResult:
resultContent}));
        }
    }

    /**
     * Generic error method to send the given error code as the http
```

```
response
     * @param {http.ServerResponse} res
     * @param {Number} code Error code, i.e 405 - method not allowed etc.
     */
    static #error(res, code){
        console.log(`Error ${code}`); // DEBUG
        res.writeHead(code);
        res.end();
    }

    /**
     * Initialises the http server, NOT THE DB, and runs it.
     */
    run() {
        this.publicFiles = Server.#recursiveReadDir('./public/');

        this.server = http.createServer(this.options, (req, res) => {
            req.on('error', (err) => {
                console.log(err);
                Server.#error(res, 400);
            });
            res.on('error', (err) => {
                console.log(err);
                Server.#error(res, 400);
            })
            console.log({'method':req.method,'url':req.url});
            switch (req.method){
                case 'GET':
                    Server.#getResource(res, this.publicFiles, req.url);
                    break;
                case 'POST':
                    if (!this.dbAccess) {
                        console.log('Tried to access DB before
initialisation!');
                        Server.#error(res, 500); // DB is not initialised
                        break;
                    }
                    // https://nodejs.org/en/docs/guides/anatomy-of-an-
http-transaction
                    let body = [];
                    req.on('data', (chunk) => {
                        body.push(chunk);
                    }).on('end', () => {
                        body = Buffer.concat(body).toString();
                        switch (req.headers['content-type']) {
                            case 'application/json':
                                this.#postResourceJSON(res, body);
                                break;
                            default:
                                console.log(body);
                                Server.#error(res, 500);
                                break;
                        }
                    });
```

```
                    break;
                default:
                    Server.#error(res, 405);
                    break;
            }
        });

        this.server.listen(this.port, this.hostname, () => {
            console.log(`Server listening at
${this.hostname}:${this.port}`);
            // Some Exit logic here
        });
    }
}

module.exports = {Server};
```

public/account.js

```
// Different Modal Contents
const signInModalContent = `<h1>Please Sign In</h1> <hr> <br>
<label>Username:</label>
<input type="text" id="username" /><br>
<label>Password:</label>
<input type="password" id="password" /><br> <br>
<button id="submit-login" onclick="login()">Submit</button>
<button id="switch-mode" onclick="showCreateAccountModal()">Create
Account</button>
<div id="modal_output"></div>`;

const createAccountModalContent = `<h1>Please Create An Account</h1> <hr>
<br>
<label>Username:</label>
<input type="text" id="username" /><br>
<label>Password:</label>
<input type="password" id="password" /><br>
<label>Re-enter Password:</label>
<input type="password" id="password-re-entry" /><br> <br>
<button id="submit-login" onclick="submitCreateAccount()">Submit</button>
<button id="switch-mode" onclick="showLoginModal()">Sign In</button>
<div id="modal_output"></div>`;

const accountMgmtModalContent = `<h1>Account Management</h1> <hr> <br>
<button onclick="logout()">Sign Out</button> <br> <br>
<button onclick="showDeleteAccountModal()">Delete Account</button> <br>
<br>
<button onclick="hideModals()">Close</button>`

const deleteAccountModalContent = `<h1>Delete Account</h1> <hr> <br>
<p>Deleting an account will permenantly delete all saved projects.</p>
```

```
<label>Password:</label>
<input type="password" id="password" /><br>
<button onclick="deleteAccount()">Delete Account</button>
<button onclick="showLoginModal()">Cancel</button>
<div id="modal_output"></div>`;

// Initialising constants and fetching account info
const modal = document.getElementById('login-modal');
const modalContent = document.getElementById('login-modal-content');
const projectList = document.getElementById('project-list-ul');

let accUsername = window.sessionStorage.getItem('username');
let accPassword = window.sessionStorage.getItem('password');

if (!accUsername) {showLoginModal();}
else {
    loadProjects();
    document.getElementById('page-info').innerText = accUsername;
}

// Modal Management
function showLoginModal(){
    if(!accUsername){modalContent.innerHTML = signInModalContent;}// Sets
content if not logged in
    else{modalContent.innerHTML = accountMgmtModalContent;}
    modal.style.display = 'block'; // Shows sign-in modal
}

function showCreateAccountModal(){
    modalContent.innerHTML = createAccountModalContent; // Sets content
    modal.style.display = 'block'; // Shows sign-in modal
}

function showDeleteAccountModal(){
    modalContent.innerHTML = deleteAccountModalContent; // Sets content
    modal.style.display = 'block'; // Shows sign-in modal
}

function showDeleteProjectModal(projectID){
    modalContent.innerHTML = `<h1>Delete Project</h1> <hr> <br>
    <p>Deleting a project will permenantly delete all associated data, are
you sure?</p>
    <button onclick="deleteProject(${projectID});hideModals()">Yes</button>
<button onclick="hideModals()">Cancel</button>`;
    modal.style.display = 'block';
}

function hideModals() {
    modal.style.display = 'none'; // Hides modal
}

function modalOutput(output){
    document.getElementById('modal_output').innerText = output;
}
```

```
// Database operations
async function submitCreateAccount(){
    const passwordInput = document.getElementById('password');
    const usernameInput = document.getElementById('username');
    const passwordReentryInput = document.getElementById('password-re-
entry');
    if (passwordInput.value != passwordReentryInput.value){
        modalOutput('Passwords do not match.')
        return;
    }
    let inputed_username = usernameInput.value;
    if (inputed_username == ''){
        modalOutput('Please enter a username.');
        return;
    }
    const dataToSubmit = {
        method: 'create-account',
        username: inputed_username,
        password: passwordInput.value
    }
    req('/', dataToSubmit).then(result => {
        if (result.error) { // If account creation fails
            if (result.error.errno == 19){
                modalOutput('An account with that username already
exists.\nPlease try a different username.');
            } else if (result.error.errno == 0) {
                modalOutput(result.error.errdsc);
            } else {
                modalOutput('UNKOWN ERROR - Account creation failed.');
            }
        } else {
            hideModals();
            saveAccountInfo(usernameInput.value, passwordInput.value);
        }
    }).catch(error => {
        modalOutput('UNKOWN ERROR - Account creation failed.');
    });
}

function login(){
    let inputed_username = document.getElementById('username').value;
    if (inputed_username == ''){
        modalOutput('Please enter a username.');
        return;
    }
    let inputed_password = document.getElementById('password').value;
    if (inputed_password == ''){
        modalOutput('Please enter a password.');
        return;
    }
    const dataToSubmit = {
        method: 'log-in',
        username: inputed_username,
```

```
        password: inputed_password
    }
    req('/', dataToSubmit).then(result => {
        if (result.error){ // If login fails
            if (result.error.errno == 0) {
                modalOutput(result.error.errdsc);
            } else {
                modalOutput('UNKOWN ERROR - Sign in failed.');
            }
        } else {
            hideModals();
            saveAccountInfo(result.stmtResult.username,
document.getElementById('password').value);
        }
    }).catch(error => {
        modalOutput('UNKOWN ERROR - Sign in failed.');
    });
}

function deleteAccount(){
    let inputed_password = document.getElementById('password').value;
    if (inputed_password == ''){
        modalOutput('Please enter a password.');
        return;
    }
    req(url = '/', {
        method: 'delete-account',
        username: accUsername,
        password: inputed_password
    }).then(res => {
        if (res.error){
            if (res.error.errno == 0) {
                modalOutput(res.error.errdsc);
            } else {
                modalOutput('UNKOWN ERROR - Delete account failed.');
            }
        } else {
            logout();
        }
    });
}

function loadProjects(){
    projectList.innerHTML = '';
    if (!accUsername){
        return;
    }
    req(url = '/', {
        method: 'get-projects',
        username: accUsername
    }).then(res => {
        if (!res.stmtResult){
            if (res.error.errno == 0){
                projectList.innerHTML = `Uh oh - looks like you dont have
```

```
any saved projects!<br><a href='/editor.html'>Make a new project?</a><br><a
href='http://localhost:8080/editor.html?projectid=1'>View an example
project?</a>`;
                return;
            } else if (res.error.errno == 1){
                window.sessionStorage.clear(); // Invalid account username
in session storage
                accUsername = null;
                showLoginModal(); // Reload page
                return;
            }
        }
        res.stmtResult.forEach(project => {
            projectList.innerHTML +=
projectListTemplate(project.projectName, project.projectID);
        });
        projectList.innerHTML += '<br><button
onclick="editProject(null)">Create New</button>';
    });
}

function deleteProject(projectID){
    req(url = '/', {
        method: 'delete-project',
        username: accUsername,
        password: accPassword,
        projectID: projectID
    }).then(res => {
        if (res.error){
            if (result.error.errno == 0) {
                alert(result.error.errdsc);
            } else {
                alert('UNKOWN ERROR - Delete account failed.');
            }
        } else {
            loadProjects();
        }
    });
}

// Manages account information
function logout(){
    saveAccountInfo(null, null);
    showLoginModal();
}

function saveAccountInfo(username, password){
    accUsername = username;
    accPassword = password;
    // Could do something with cookies here to make account info
    // persist through pages, but not in project scope.
    if (username === null || password === null) {
        window.sessionStorage.clear();
        document.getElementById('page-info').innerTEXT = "Not signed in.";
```

```
    } else {
        window.sessionStorage.setItem('password', password);
        window.sessionStorage.setItem('username', username);
        document.getElementById('page-info').innerText = username;
    }
    loadProjects();
}

// Project list management
function projectListTemplate(projectName, projectID){
    return `<li class="project-list-li"">
                <span onclick="editProject(${projectID})" class="project-
name">${projectName}</span>
                <button onclick="showDeleteProjectModal(${projectID})"
class="project-delete-button">Delete</button>
            </li>`;
}

// Redirects to editor page
function editProject(projectID){
    if (projectID == null){
        window.location = '/editor.html';
        return;
    }
    window.location = `/editor.html?projectid=${projectID}`;
}
```

public/editor_container.js

```
class EditorContainer {
    constructor(documentID, pages, defaultPage){ // Constructor method
        this.editor = CodeMirror(document.getElementById(documentID), {
            mode:  "clike",
            lineNumbers: "true",
            matchBrackets: "true"
        }); // Codemirror 5 used for editor
        this.pageContent = pages;
        this.page = defaultPage;
        this.#loadPage();
    }

    switchPage(newPageID){ // Public method that swaps the currently
displayed page
        this.syncPages();
        this.page = newPageID;
        this.#loadPage();
    }

    syncPages(){ // Public method to sync the currently displayed content
with the stored content for that page
```

```
            this.pageContent[this.page] = this.editor.getValue();
        }

        #loadPage(){ // Private method to change currently displayed content
            this.editor.setValue(this.pageContent[this.page]);
        }
    }
```

public/editor.js

```
// Modal Content
const signInModalContent = `<h1>Please Sign In</h1> <hr> <br>
<label>Username:</label>
<input type="text" id="username" /><br>
<label>Password:</label>
<input type="password" id="password" /><br>
<label>Project Name:</label>
<input type="text" id="project_name" /><br> <br>
<button id="submit-login" onclick="login();">Submit</button>
<button id="cancel" onclick="hideModals();">Cancel</button>
<div id="modal_output"></div>`;

const projectNameModalContent = `
<label>Project Name:</label>
<input type="text" id="project_name" /><br> <br>
<button id="submit-login" onclick="projectName =
document.getElementById('project_name').value;hideModals();sendProjectData(
);">Submit</button>
<button id="cancel" onclick="hideModals();">Cancel</button>
<div id="modal_output"></div>`

// Initialising constants
const modal = document.getElementById('login-modal');
const modalContent = document.getElementById('login-modal-content');
modal.style.display = 'none';

// Fetching account info
var accountInfo = null;
var projectName = null;

if (window.sessionStorage.getItem('username')){
    accountInfo = {
        username: window.sessionStorage.getItem('username'),
        password: window.sessionStorage.getItem('password')
    }
}

// Initialising glCanvas
glCanvas = new GLCanvas("glScreen");

var projectID = null;
```

```
var editorContainer = null;
if (window.location.search){
    let search = window.location.search;
    if (search.length > 1){search = search.slice(1);}
    let searchArray = search.split('=');
    if (searchArray[0] == 'projectid' && searchArray.length > 1){
        projectID = searchArray[1];
    }
} // There is probably a better way to do this, but it works

// Loading project
if (!projectID){
    loadDefaultPages();
} else {
    loadProjectPages();
}

function loadDefaultPages(){
    editorContainer = new EditorContainer("editor_container",
        [GLCanvas.defaultFragment,
            GLCanvas.defaultVertex,
            convertToPrettyString(GLCanvas.defaultVertices, 6),
            convertToPrettyString(GLCanvas.defaultIndices,3)],
        0
    );
    runCode();
}

function loadProjectPages(){
    req('/', {method:'load-project',projectID:projectID}).then(result => {
        if (!result.error){
            editorContainer = new EditorContainer("editor_container",
                [result.stmtResult.projectContent[0],
                    result.stmtResult.projectContent[1],
                    result.stmtResult.projectContent[2],
                    result.stmtResult.projectContent[3]],
                0
            );
            document.getElementById('page-info').innerText =
result.stmtResult.projectName;
            projectName = result.stmtResult.projectName;
            runCode();
        } else {
            alert(`Error loading project ${projectID}`)
            loadDefaultPages();
        }
    }).catch(error => {
        alert(`Error loading project ${projectID}`)
    });
}

// Editor functions
function convertToArray(string_input){
    return string_input.toString().replace(/[\[\] \n]/g,
```

```
"").split(",").map(Number);
}

function convertToPrettyString(array_input, line_interval){
    let output = "";
    for (let i = 0; i < array_input.length; i++){
        output += `${array_input[i]},` + ((i + 1) % line_interval == 0 ?
"\n" : "");
    }
    return output.slice(0, -2);
}

// Switching tabs
function fragmentTab(){
    editorContainer.switchPage(0);
}
function vertexTab(){
    editorContainer.switchPage(1);
}
function verticesTab(){
    editorContainer.switchPage(2);
}
function indicesTab(){
    editorContainer.switchPage(3);
}

// Player controlls
function runCode(){
    editorContainer.syncPages();
    glCanvas.initProgram(editorContainer.pageContent[0],
        editorContainer.pageContent[1]);
    let vertices = convertToArray(editorContainer.pageContent[2]);
    let indices = convertToArray(editorContainer.pageContent[3]);
    glCanvas.initBuffers(vertices, indices);
    glCanvas.renderStart = performance.now();
    glCanvas.render();
}

// Initialising player variables
const debugInfoDiv = document.getElementById("debug_info");
debugInfoDiv.style.display = "none";
var debugInfoVis = false;
var nextAnimationFrameID = 0;
var debugInfoLast = 0;
const performanceInfo = document.getElementById("performance_info");
const uniformInfo = document.getElementById("uniform_info");

const resSlider = document.getElementById("resolution_slider");
const resDisplay = document.getElementById("resolution_display");

// Debug Info
function debugInfo(){
    if (!debugInfoVis){
        debugInfoDiv.style.display = "block";
```

```
        nextAnimationFrameID =
window.requestAnimationFrame(debugInfoUpdate);
    } else {
        debugInfoDiv.style.display = "none";
        window.cancelAnimationFrame(nextAnimationFrameID);
    }
    debugInfoVis = !debugInfoVis;
}

function debugInfoUpdate(time){
    nextAnimationFrameID = window.requestAnimationFrame(debugInfoUpdate);
    uniformInfo.innerText =
        `Mouse: {${Math.round(glCanvas.mouse.pos.x * 1000) /
1000},${Math.round(glCanvas.mouse.pos.y * 1000) /
1000},${glCanvas.mouse.buttons.lmb}}
        Time: ${(Math.round(glCanvas.time - glCanvas.renderStart)) /
1000}`;
    if (time - debugInfoLast > 1000){ // Update every second
        debugInfoLast = time;
    } else{
        return;
    }
    performanceInfo.innerText =
    `FPS: ${Math.round(glCanvas.fps * 100) / 100}
    MSPF: ${Math.round(glCanvas.mspf * 100) / 100}`;
}

// Resolution slider logic
function resChange(){
    newRes = resSlider.value;
    glCanvas.updateRes(newRes);
    resDisplay.innerText = `${newRes} px`;
}

// Save project
function saveCode(){
    if (!accountInfo){
        showModal();
    } else if (!projectID) {
        showProjectNameModal();
    } else {
        sendProjectData();
    }
}

function saveCodeAs() {
    if (!accountInfo){
        showModal();
    } else {
        showProjectNameModal();
    }
}

// Database operations
```

```
function login(){
    let inputed_username = document.getElementById('username').value;
    if (inputed_username == ''){
        modalOutput('Please enter a username.');
        return;
    }
    let inputed_password = document.getElementById('password').value;
    if (inputed_password == ''){
        modalOutput('Please enter a password.');
        return;
    }
    const dataToSubmit = {
        method: 'log-in',
        username: inputed_username,
        password: inputed_password
    }
    req('/', dataToSubmit).then(result => {
        if (result.error){ // If login fails
            if (result.error.errno == 0) {
                modalOutput(result.error.errdsc);
            } else {
                modalOutput('UNKOWN ERROR - Sign in failed.');
            }
        } else {
            saveAccountInfo(result.stmtResult.username,
document.getElementById('password').value);
            hideModals();
            sendProjectData();
        }
    }).catch(error => {
        modalOutput('UNKOWN ERROR - Sign in failed.');
    });
}

function saveAccountInfo(username, password){
    projectName = document.getElementById('project_name').value;
    accountInfo = {
        username: username,
        password: password
    };
    if (username === null || password === null) {
        window.sessionStorage.clear();
    } else {
        window.sessionStorage.setItem('password', password);
        window.sessionStorage.setItem('username', username);
    }
}

function sendProjectData(){
    editorContainer.syncPages();
    if (!projectName) {
        showProjectNameModal();
        modalOutput("Please enter a project name.");
        return;
```

```
    }
    req(url = '/', {
        method: 'save-project',
        username: accountInfo.username,
        password: accountInfo.password,
        project_name: projectName,
        project_content: editorContainer.pageContent,
        projectID: projectID
    }).then(res => {
        if (res.error){
            if (res.error.errno == 0) {
                console.log(res);
                modalOutput(res.error.errdsc);
            } else {
                modalOutput('UNKOWN ERROR - Save project failed.');
            }
        } else {
            hideModals();
            projectID = res.stmtResult.projectID;
            document.getElementById('page-info').innerText = projectName;
        }
    });
}

// Modal operations
function showModal(){
    modalContent.innerHTML = signInModalContent;
    modal.style.display = 'block';
}

function showProjectNameModal(){
    modalContent.innerHTML = projectNameModalContent;
    modal.style.display = 'block';
}

function hideModals() {
    modal.style.display = 'none';
}

function modalOutput(output){
    document.getElementById('modal_output').innerText = output;
}
```

public/global.js

```
/***
 * Sends a POST request to the server
 * @param {string} url
 * @param {JSON} data
 * @returns {JSON}
```

```
 */
async function req(url = '', data = {}) {
    const response = await fetch(url, {
        method: 'POST',
        credentials: 'omit',
        headers: {'content-type': 'application/json'},
        body: JSON.stringify(data),
    });
    if (response.ok){
        return response.json();
    }
}
```

public/404-page.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>404</title>
</head>
<style>
    * {
        font-family: Arial, Helvetica, sans-serif;
        text-align: center;
    }

    .fof {
        size: 10rem;
    }
</style>
<body>
    <div id="main">
        <div class="fof">
            <h1>Error 404</h1>
            <p>Resource not found</p>
        </div>
    </div>
</body>
</html>
```

public/account_page_style.css

```
body {
    display: flex;
    flex-direction: column;
    gap:0;
}
```

```
#content {
    display: flex;
    flex-direction: row;
    align-content: center;
    justify-content: center;
    gap: 1rem;
    flex:1;
}

#project-list {
    width: 75%;
    margin-top: 1rem;
}

#project-list-ul {
    list-style: none;

}

.project-list-li {
    border-style: solid;
    border-width: 1px;
    padding: 0.25rem;
    display: flex;
}

.project-delete-button {
    align-self: flex-end;
}

.project-name {
    flex-grow: 1;
    flex-basis: 0;
    text-align: center;
    align-self: center;
}

.project-list-li:hover {
    border-style: solid;
    border-width: 3px;
    padding: 0.25rem;
}
```

public/account_page.html

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="global_style.css">
    <link rel="stylesheet" href="account_page_style.css">
    <script src="global.js"></script>
    <title>My Account</title>
</head>
<body>
    <div id="header">
        <span id="page-info">
            Not logged in.
        </span>
        <div id="account-info">
            <img src="img/account_icon.png" id="account_icon" alt="Account
Badge" onclick="showLoginModal()">
        </div>
    </div>
    <div id="content">
        <div id="login-modal" class="modal">
            <div id="login-modal-content">

            </div>
        </div>
        <div id="project-list">
            <ul id="project-list-ul">
            </ul>
        </div>
    </div>

    <script src="account.js"></script>
</body>
</html>
```

public/editor_style.css

```
body {
    display: flex;
    flex-direction: column;
    gap:0;
}

#content {
    display: flex;
    flex-direction: row;
    align-content: center;
    justify-content: center;
    gap: 1rem;
    flex:1;
}

#editor_master_container{
```

```
    flex-grow: 2;
    width: 60%;
    text-align: left;
    margin-top: 0.5rem;
    margin-right: 0.5rem;
    margin-bottom: 0.5rem;
    border: 1px;
    border-style: solid;
}

#editor_tabs {
    display: flex;
    align-items: center;
    border-bottom-style: solid;
    border-bottom-color: #dddddd;
}

.span-button {
    position: relative;
    padding:0.25rem;
    background-color: #f7f7f7;
    border-top-right-radius: 5px;
}

.span-button:hover{
    background-color: #dddddd;
}

.span-button:active {
    background-color: white;
}

#debug_info {
    position:absolute;
    left: 1rem;
    top: 1rem;
    background-color: white;
    padding: 0.1rem;
    text-align: left;
}

#editor_controls {
    margin-top: 0.5rem;
    margin-left: 0.5rem;
    margin-bottom: 0.5rem;
}
.CodeMirror {
    height: 100%;
    width: 100%;
}
#glScreen{
    border: 1px;
    border-style: solid;
```

```
    width: 40vw;
}
```

public/editor.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="lib/codemirror.css">
    <link rel="stylesheet" href="global_style.css">
    <link rel="stylesheet" href="editor_style.css">
    <script src="lib/codemirror.js"></script>
    <script src="lib/clike.js"></script>
    <script src="lib/matchbrackets.js"></script>
    <script src="engine.js"></script>
    <script src="editor_container.js"></script>
    <script src="global.js"></script>
    <title>Editor</title>
</head>
<body>
    <div id="header">
        <span id="page-info">
            Untitled
        </span>
        <span id="account-info">
            <a href="account_page.html">
                <img src="img/account_icon.png" id="account_icon"
alt="Account Badge">
            </a>
        </span>
    </div>
    <div id="content">
        <div id="editor_controls" class="flex_item">
            <canvas id="glScreen"></canvas>
            <div id="debug_info">
                <p id="performance_info"></p>
                <p id="uniform_info"></p>
            </div>
            <div>
                <button onclick="runCode()">Run Code</button>
                <button onclick="debugInfo()">Info</button>
            </div>
            <div class="slider-container">
                <input type="range" class="slider" id="resolution_slider"
min="100" max="2000" value="500" oninput="resChange()">
                <p class="inline-label" id="resolution_display">500 px</p>
            </div>
            <div>
                <button onclick="saveCode()">Save Code</button>
```

```
                <button onclick="saveCodeAs()">Save As</button>
            </div>
        </div>
        <div id="editor_master_container" class="flex_item">
            <div id="editor_tabs">
                <span class="span-button no-highlight"
onclick="fragmentTab()">Fragment</span>
                <span class="span-button no-highlight"
onclick="vertexTab()">Vertex</span>
                <span class="span-button no-highlight"
onclick="verticesTab()">Vertices</span>
                <span class="span-button no-highlight"
onclick="indicesTab()">Indices</span>
            </div>
            <div id="editor_container"></div>
        </div>
    </div>
    <div id="login-modal" class=".modal">
        <div id="login-modal-content">

        </div>
    </div>
    <script src="editor.js"></script>
</body>
</html>
```

public/global_style.css

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

button {
    padding: 0.1rem;
}

body{
    color:black;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 1em;
    text-align: center;
    gap: 1rem;
    min-height:100vh;
}

.no-highlight{
    user-select: none;
    -moz-user-select: none;
    -webkit-text-select: none;
```

```
    -webkit-user-select: none;
}

.modal {
    display: none;
    position: fixed;
    z-index: 1;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    overflow: auto;
    background-color: rgb(0,0,0);
    background-color: rgba(0,0,0,0.4);
}  /* https://www.w3schools.com/howto/howto_css_modals.asp */

#login-modal-content {
    background-color: white;
    position: absolute;
    top:50%;
    left:50%;
    margin:auto;
    transform: translate(-50%, -50%);
    width:50%;
    padding: 1rem;
    border-radius: 1rem;
    border-style: solid;
    border-color: black;
    border-width: 0.1rem;
}

#modal_output {
    padding: 1rem;
    color: red;
}


#header {
    padding: 0.5rem;
    background-color: gray;
    display: flex;
    text-align: right;
}

#account_icon {
    align-self:flex-end;
    width: 2rem;
}

#page-info {
    align-self: center;
    flex-grow: 1;
    flex-basis: 0;
```

```
    text-align: center;
}
```

public/index_style.css

```
#content {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    padding: 1rem;
    padding-top: 2rem;
}

#glScreen{
    border: 1px;
    border-style: solid;
    height: 30vw;
}

#header {
    padding: 0.5rem;
    background-color: gray;
}
```