Разработка интерактивного редактора кода “Online IDE”

Групва: ПМ-04
Студент: Федоров В.Е.
Преподаватель: Арыков С.Б.

Новосибирск
2024

1. Содержание

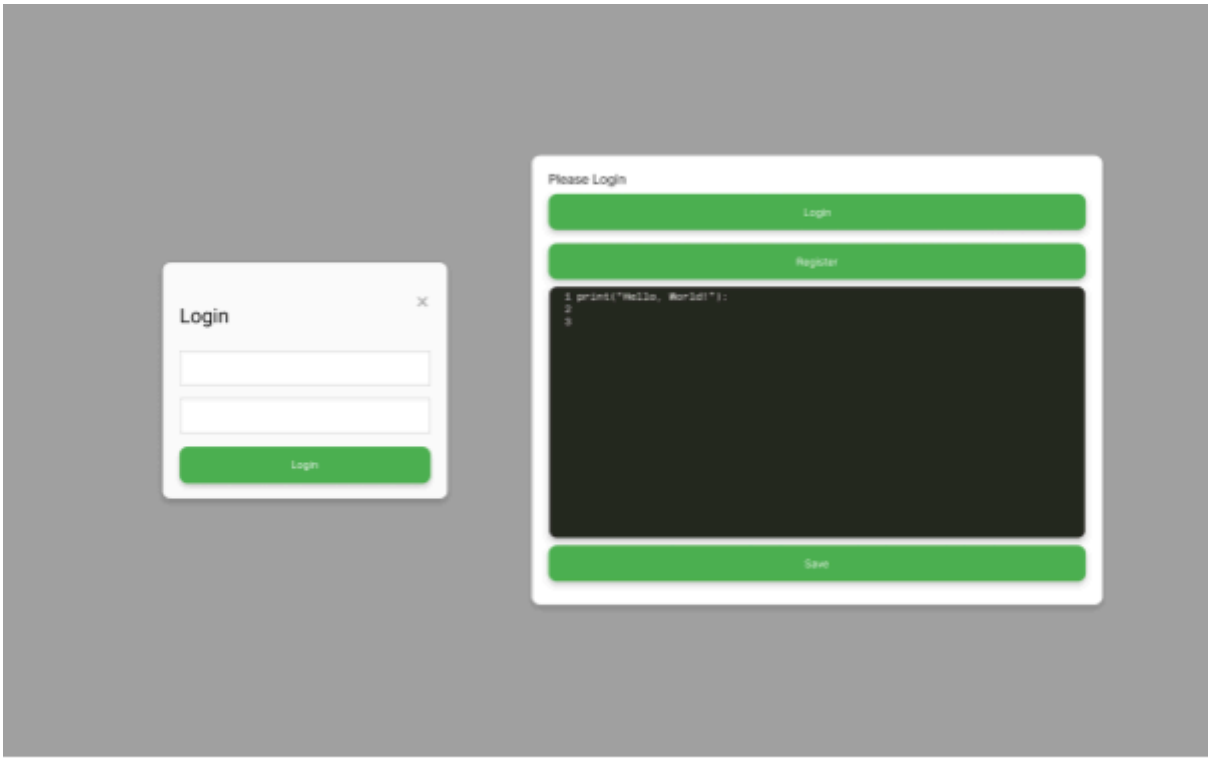
2. Введение

Этот проект будет реализован в рамках университетского курса "Проектная деятельность". Выбор за основу темы "Разработка интерактивного редактора кода" послужил началом для создания онлайн-среды разработки. Основная цель интерактивного редактора кода, который является веб-сайтом для работы с кодом в браузере, а не в настольном приложении, – упростить процесс программирования.

На сегодняшний день существует не так много сайтов для кодирования, которые могут похвастаться удобством и интуитивностью в использовании, обеспечивая при этом бесперебойную работу с кодом.

Разрабатываемый редактор кода нацелен на то, чтобы программисты могли легко и быстро вносить изменения в свой код на ходу, тем самым увеличивая его надежность и уменьшая вероятность появления ошибок.

3. Требование к программе



(Рисунок 1)

Используя следующие технологии: HTML, CSS, JS, Node.js, Express.js, SQLite нужно реализовать Rest API приложение с асинхронностью и AJAX для редактирования кода.

Приложение представляет собой редактор кода. Макет изображён на рисунке 1. Сообщения должны выводиться в виде всплывающих окон. Всего у приложения есть 5 кнопок: Login, Register, Logout, Delete User, и Save. В центре расположено поле для редактирования текста. Максимальная допустимое количество символов в одной строке 80, если пользователь превысил данный лимит, то текст переносится на новую строку. Редактор должен поддерживать подсветку синтаксиса для языка программирования Python, автоматическую нумерацию строк и прокручивание поля редактирования.

Если пользователь не вошёл в свой аккаунт, то ему доступно следующие три кнопки: Login, Register, Save и сверху расположена надпись просящая пользователя войти. Когда пользователь не вошёл для него доступно редактирование общего файла единого для всех посетителей.

Если пользователь вошёл в свой аккаунт, то ему доступно следующие три кнопки: Logout, Delete User, Save и сверху расположено имя пользователя. У каждого вошедшего посетителя для редактирования открыт свой уникальный файл.

Имеется три вида всплывающих меню: Login, Register, Massage. Меню Login и Register оба имеют по два поля для ввода имени пользователя (его почты) и пароля, а также кнопки для входа и регистрации соответственно. Меню Massage же просто содержит текстовую информацию об успешности или повальности действий. Любое меню можно закрыть с помощью кнопки в виде крестика в правом верхнем углу экрана.

При операциях с данными пользователя на сервере должна проводиться их валидация: (проверка, что поля формы не пустые; проверка, что введенный пароль не менее 6 символов и содержит буквы и цифры; проверка, что указан корректный e-mail (хотя бы что там присутствует символ ‘@’ и в конце два слова разделены точкой)). И если что-то было нарушено вместо действия должно всплывать окно с сообщением об ошибке, например: “ Fields must not be empty”, “ Incorrect email or pasword”.

При нажатии на кнопку Login выходит одноимённое всплывающее меню для входа с двумя полями email и password и кнопкой входа. Есть валидация данных и проверка наличия пользователя, если данные совпадают с базой и пользователь существует, то нужно войти. Закрывается общий тестовый файл и открывается файл конкретного пользователя. Должно всплывать текстовое меню с успешностью входа или сообщение об ошибке.

При нажатии на кнопку Logout если пользователь находиться в системе он должен выйти. Текущий открытый файл пользователя должен закрыться и открыться общий файл. Должно всплывать текстовое меню с успешностью выхода или сообщение об ошибке.

При нажатии на кнопку Register должно выходить всплывающее меню для регистрации с двумя полями email и password и кнопкой регистрации. Должна быть проведена валидация данных и проверка наличия пользователя, если в базе нет совпадающего пользователя, то нужно добавить его и его файл в систему. Должно всплывать текстовое меню с успешностью регистрации или сообщение об ошибке.

При нажатии на кнопку Delete User если пользователь находиться в системе он должен выйти, и пользователь и его файл должен быть удалён из базы. Текущий открытый файл должен быть закрыт и удалён, вместо него должен открыться общий файл. Должно всплывать текстовое меню с успешностью удаления или сообщение об ошибке.

При нажатии на кнопку Save file все изменения в текущем открытом файле должны сохраняться на сервере. Должно всплывать текстовое меню с успешностью сохранения или сообщение об ошибке.

4. Разработка

Для начала разработаем frontend приложения. Мы будем использовать HTML, CSS и JavaScript. Для плавности приложения используем AJAX чтобы сделать все взаимодействия асинхронными. Для красивого подсвечивания кода, а также нумерации строк будет использовать библиотеку для JS CodeMirror.

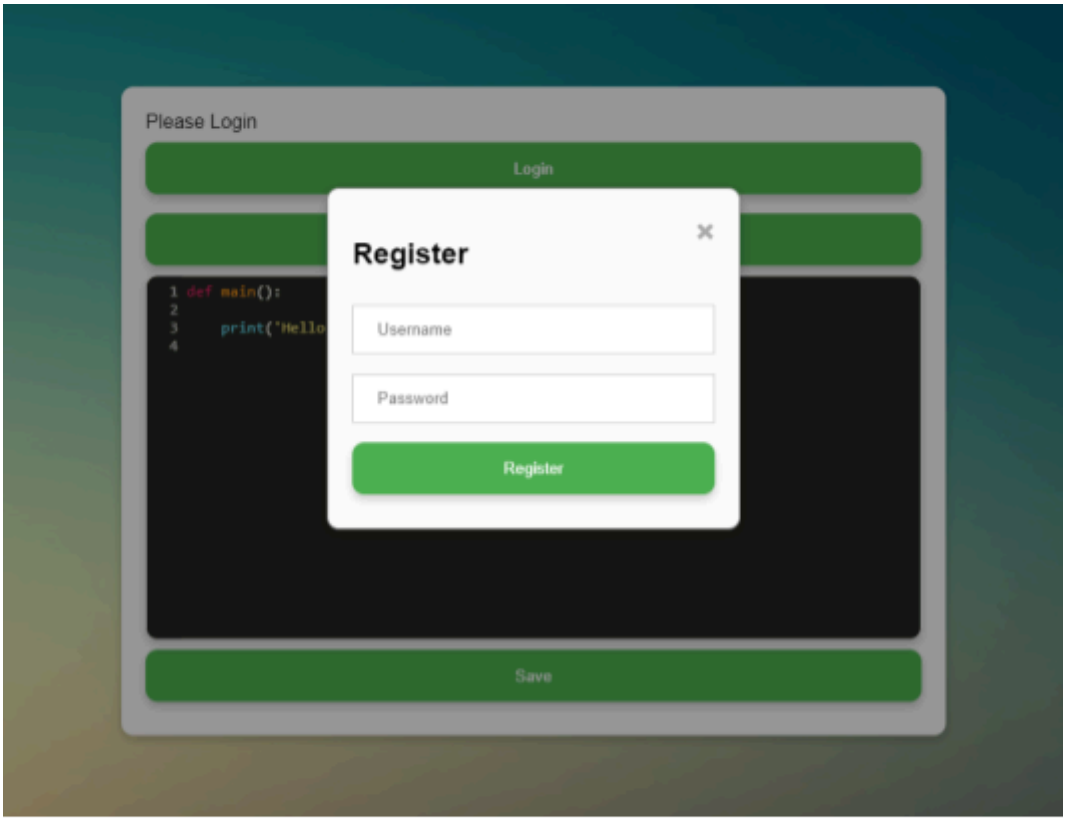
Ниже приведены примеры дизайна интерфеса приложения.

(Рисунок 2)

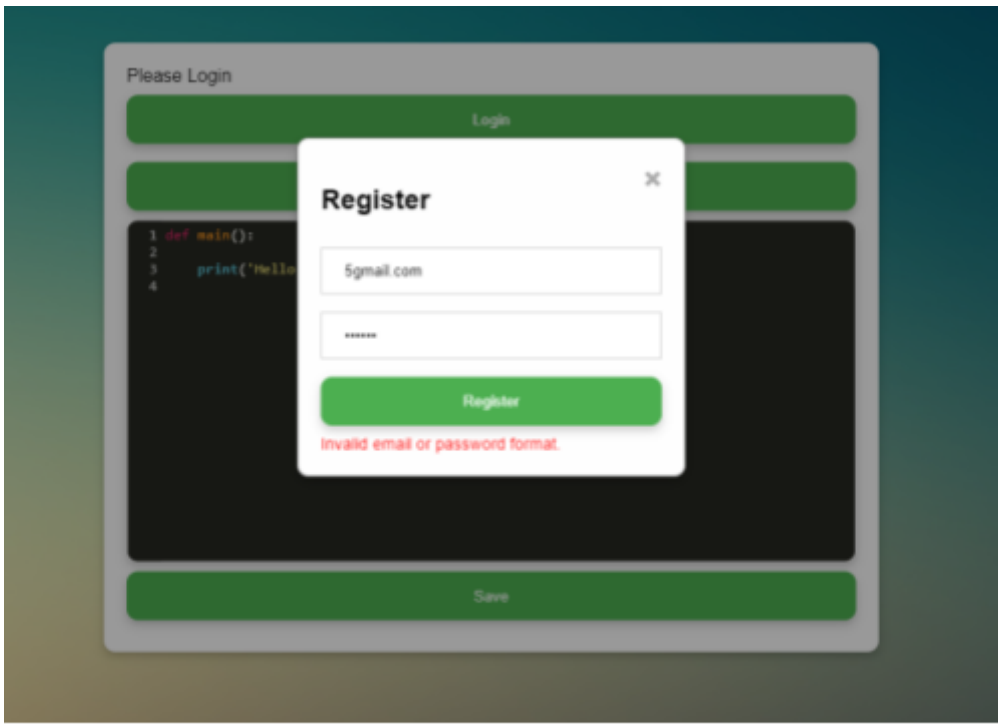
На рисунке 2 изображён вид приложения для пользователя не вошедшего в систему. Открыто редактирование общего файла, доступны кнопки для входа, регистрации и сохранения.

На рисунке 3 изображён вид приложения для пользователя вошедшего в систему. Открыт личный файл конкретного пользователя, доступны кнопки для выхода из аккаунта, удаления текущего пользователя и сохранения.

(Рисунок 3)



(Рисунок 4)

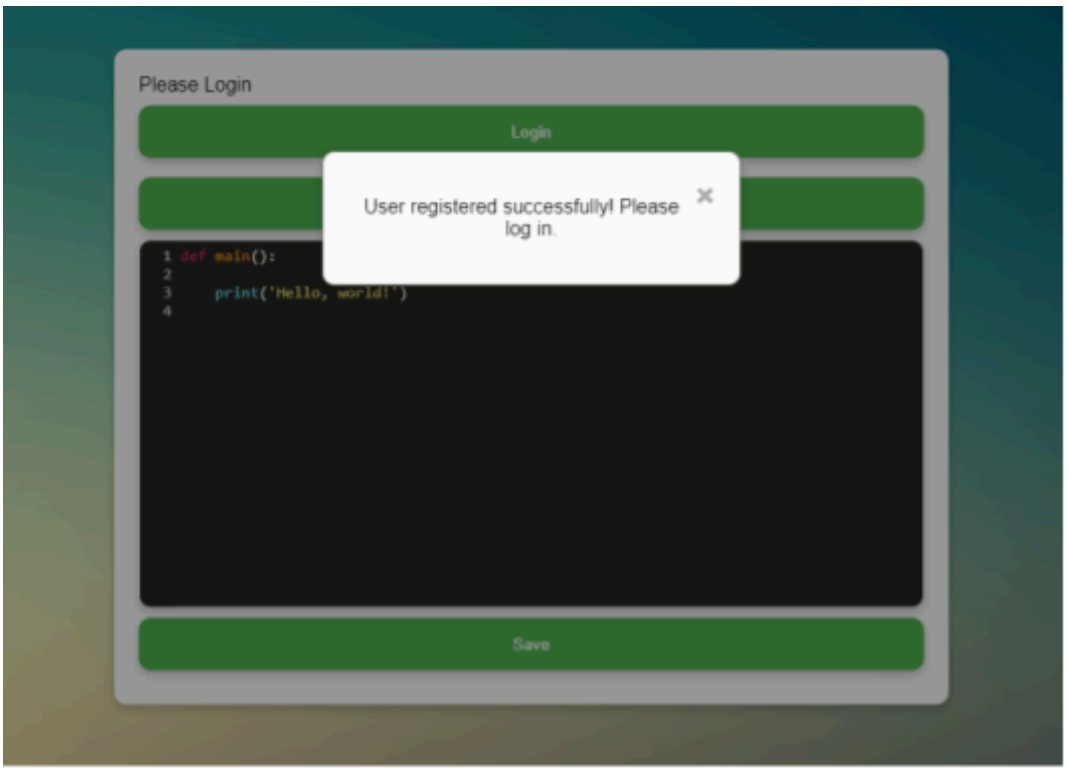


(Рисунок 5)

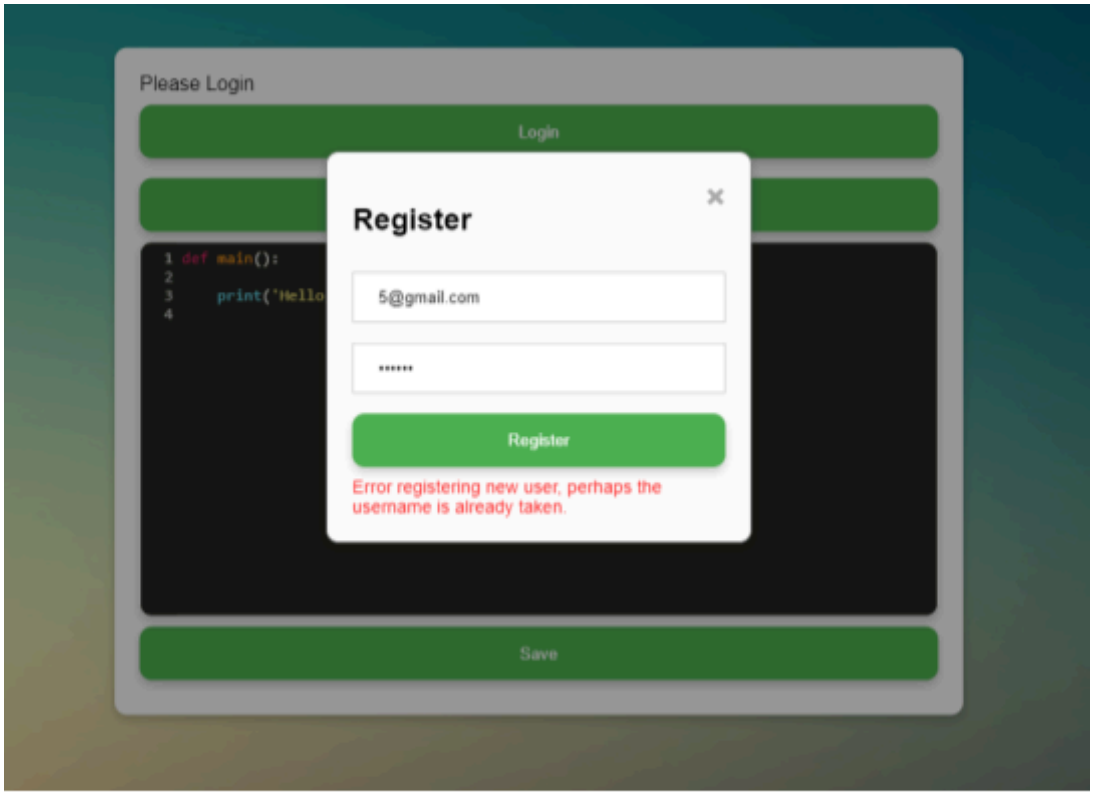
На рисунке 4 изображено меню для регистрации, а на рисунке 5 изображенно собщение о том что введёные данные не соответствуют требованиям валидации данных. Аналогичное сообщение выводиться при попытке ввести пустые поля, а также на рисунке 7 можно видеть сообщение о том что данное имя пользователя уже занято.

На рисунке 6 изображенно сообщение об успешной регистрации, аналогичные сообщения выводяться при успешной входе, выходе, сохранении файла и удалении пользователя.

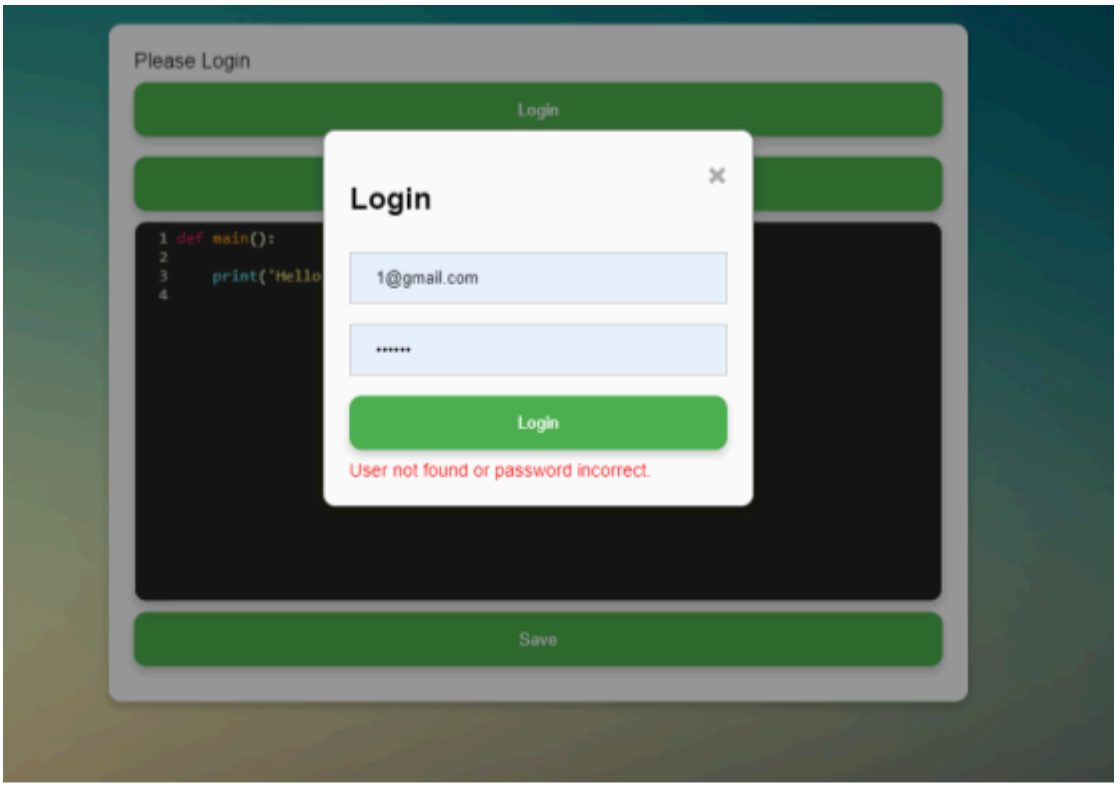
На рисуне 8 изображенно сообщении о том что пользователя с таким именем не существует



(Рисунок 6)



(Рисунок 7)



(Рисунок 8)

Для реализации backenda использовался Node.js, Express.js, JSON файлы и база данных SQLite. Все запросы выполняются асинхронно, для каждого пользователя заводиться сессия.

Было реализованно следующее Rest API:

№	Тип	Путь	Описание
1	POST	/register	Регистрация нового пользователя
2	POST	/login	Аутентификация пользователя
3	POST	/logout	Выход пользователя из системы
4	POST	/deleteuser	Удаление пользователя

5	GET	/check	Проверка статуса входа пользователя
6	GET	/load	Загрузка данных (текста)
7	POST	/save	Сохранение данных (текста)

Примеры вызова:

Путь	Тело запроса	Возвращаемое значение

/register	<pre>{
	  "username": "test@example.com",
	  "password": "password123"
	}</pre>
		<pre>{
		  "message": "User registered successfully!"
		}</pre>

/login	<pre>{
	  "username": "test@example.com",
	  "password": "password123"
	}</pre>
		<pre>{
		  "message": "User logged in successfully!",
		  "username": "test@example.com"
		}</pre>

/logout	<pre>{
	  "message": "User logged out successfully."
	}</pre>
		<pre>{
		  "message": "User logged in successfully!",
		  "username": "<username>"
		}</pre>

/deleteuser	-	<pre>{
		  "message": "User deleted successfully."
		}</pre>

/check	-	<pre>{
		  "loggedin": true,
		  "username": "test@example.com"
		}</pre>

/load	-	<pre>"print('Hello, world')"</pre>

/save	<pre>{
	  "text": "print('Updated Hello, world')"
	}</pre>
		<pre>"File saved successfully"</pre>

Далее выложим код на GitHub и разместим его на хостинге railway.app: https://editor-production-1614.up.railway.app/

5. Литература и источники

1. https://html5book.ru
2. https://code.mu/ru/markup/book/prime/
3. https://colorscheme.ru/html-colors.html
4. https://learn.javascript.ru/
5. https://www.figma.com
6. https://wireframe.cc

6. Приложение

Index.js:

```
const { promisify } = require('util');
const express = require('express');
const bodyParser = require('body-parser');
const bcrypt = require('bcryptjs');
const sqlite3 = require('sqlite3').verbose();
const fs = require('fs');
const path = require('path');
const session = require('express-session');
const SQLiteStore = require('connect-sqlite3')(session);
const dotenvResult = require('dotenv').config();

const PORT = process.env.PORT || 3000;

const app = express();
const db_path = path.join('db', 'database.db');
const db = new sqlite3.Database(db_path);
const dbRun = promisify(db.run.bind(db));
const fsUnlink = promisify(fs.unlink);

app.use(express.static('public'));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    store: new SQLiteStore({ db: 'sessions.sqlite', dir: 'db' }),
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: true,
    cookie: { secure: 'auto', httpOnly: true, maxAge: 7 * 24 * 60 * 60 * 1000
}
}));

async function initializeDatabase() {
    return new Promise((resolve, reject) => {
        db.run('CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY
AUTOINCREMENT, username TEXT UNIQUE, password TEXT)', (err) => {
```

```
            if (err) {
                console.error('Error creating users table', err);
                reject(err);
            } else {
                resolve();
            }
        });
    });
}

(async () => {
    try {
        await initializeDatabase();
        console.log('Database initialized successfully.');
    } catch (err) {
        console.error('Database initialization failed:', err);
    }
})();

function validateEmail(email) {
    return /^[^@]+@\w+(\.\w+)+\w$/.test(email);
}

function validatePassword(password) {
    return password.length >= 6 && /[a-zA-Z]/.test(password) &&
/\d/.test(password);
}

app.post('/register', async (req, res) => {
    const { username, password } = req.body;

    if (!validateEmail(username) || !validatePassword(password)) {
        return res.status(400).json({ error: 'Invalid email or password
format.' });
    }
    try {
        const hashedPassword = bcrypt.hashSync(password, 8);
        await dbRun('INSERT INTO users (username, password) VALUES (?, ?)',
[username, hashedPassword]);

        const file_path = path.join('db', 'files', `${username}.txt`);
        await fs.promises.writeFile(file_path, "print('Hello, world')");
```

```
        res.json({ message: 'User registered successfully!' });
    } catch (err) {
        if (err.code === 'SQLITE_CONSTRAINT') {
            res.status(500).json({ error: 'Error registering new user, perhaps
the username is already taken.' });
        } else {
            res.status(500).send({ error: 'Error creating file'});
        }
    }
});

app.post('/login', async (req, res) => {
    const { username, password } = req.body;

    if (!username || !password) {
        return res.status(400).json({ error: 'Username and password are
required.' });
    }
    const dbGet = promisify(db.get.bind(db));
    try {
        const user = await dbGet('SELECT * FROM users WHERE username = ?',
username);

        if (!user || !bcrypt.compareSync(password, user.password)) {
            return res.status(404).json({ error: 'User not found or password
incorrect.' });
        }
        req.session.loggedin = true;
        req.session.username = username;
        res.json({ message: 'User logged in successfully!', username });
    } catch (err) {
        console.error(err);
        return res.status(500).json({ error: 'Error on the server.' });
    }
});


app.post('/logout', async (req, res) => {
    try {
        await new Promise((resolve, reject) => {
            req.session.destroy((err) => {
```

```
                if (err) reject(err);
                else resolve();
            });
        });
        res.json({ message: 'User logged out successfully.' });
    } catch (err) {
        console.error(err);
        res.status(500).json({ error: 'Failed to log out.' });
    }
});

app.post('/deleteuser', async (req, res) => {
    if (!req.session.loggedin) {
        return res.status(400).json({ error: 'User is not logged in.' });
    }
    const username = req.session.username;
    try {
        await dbRun('DELETE FROM users WHERE username = ?', username);
        const file_path = path.join('db', 'files', `${username}.txt`);
        try {
            await fsUnlink(file_path);
        } catch (err) {
            console.error(`Failed to delete user file for ${username}:`, err);
        }
        await new Promise((resolve, reject) => {
            req.session.destroy(err => {
                if (err) reject(err);
                else resolve();
            });
        });
        res.json({ message: 'User deleted successfully.' });
    } catch (err) {
        console.error(err);
        res.status(500).json({ error: 'Error deleting user.' });
    }
});

app.get('/check', async (req, res) => {
    if (req.session.loggedin) {
        res.json({
            loggedin: true,
            username: req.session.username
```

```
        });
    } else {
        res.json({ loggedin: false });
    }
});


app.get('/load', async (req, res) => {
    try {
        const defaultFilename = 'notepad.txt';
        const filePath = req.session.username
            ? path.join('db', 'files', `${req.session.username}.txt`)
            : defaultFilename;

        // Await on the promise returned by fs.promises.readFile
        const data = await fs.promises.readFile(filePath, 'utf8');
        res.send(data);
    } catch (err) {
        console.error(err); // Logging the error can help in debugging
        res.status(500).send('Error reading file');
    }
});

app.post('/save', async (req, res) => {
    const { text } = req.body;
    const defaultFilename = 'notepad.txt';
    const filePath = req.session.username ? path.join('db', 'files',
`${req.session.username}.txt`) : defaultFilename;

    try {
      await fs.promises.writeFile(filePath, text, 'utf8'); // Write the file
with UTF-8 encoding
      res.send('File saved successfully');
    } catch (err) {
      console.error(err); // Log the error for debugging
      res.status(500).send('Error saving file');
    }
});

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Code editor</title>
    <link rel="stylesheet" href="style.css">
    <script src="script.js"></script>
    <!-- CodeMirror CSS -->
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.0/codemirror.min.
css">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.0/theme/monokai.m
in.css">
    <!-- CodeMirror JS -->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.0/codemirror.min.j
s"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.0/mode/python/pyth
on.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.0/addon/lineNumber
s/lineNumbers.min.js"></script>
</head>
<body>
    <div id="app"  class="center">
        <div id="userInfo">Please Login</div>
        <button id="loginBtn">Login</button>
        <button id="registerBtn">Register</button>
        <button id="logoutBtn" style="display: none;">Logout</button>
        <button id="deleteUserBtn" style="display: none;">Delete User</button>
        <div id="notepad"></div>
        <button id="saveBtn">Save</button>
    </div>

    <!-- Login Modal -->
    <div id="loginModal" class="modal">
```

```
        <div class="modal-content">
            <span class="close" id="closeLogin">&times;</span>
            <h2>Login</h2>
            <input type="text" id="loginUsername" placeholder="Username"
required>
            <input type="password" id="loginPassword" placeholder="Password"
required>
            <button id="loginButton">Login</button>
            <div id="loginError" class="error"></div>
        </div>
    </div>

    <!-- Register Modal -->
    <div id="registerModal" class="modal">
        <div class="modal-content">
            <span class="close" id="closeRegister">&times;</span>
            <h2>Register</h2>
            <input type="text" id="registerUsername" placeholder="Username"
required>
            <input type="password" id="registerPassword"
placeholder="Password" required>
            <button id="registerButton">Register</button>
            <div id="registerError" class="error"></div>
        </div>
    </div>

    <!-- Message Modal -->
    <div id="messageModal" class="modal">
        <div class="modal-content">
            <span class="close" id="closeMassage">&times;</span>
            <p id="messageText" style="text-align: center;"></p>
        </div>
    </div>
</body>
</html>
```

Script.js:

```
document.addEventListener('DOMContentLoaded', () => {
    document.getElementById('loginBtn').addEventListener('click', () => {
        showModal('loginModal')
```

```
    });
    document.getElementById('registerBtn').addEventListener('click', () => {
        showModal('registerModal')
    });
    document.getElementById('closeLogin').addEventListener('click', () => {
        closeModal('loginModal')
    });
    document.getElementById('closeRegister').addEventListener('click', () => {
        closeModal('registerModal')
    });
    document.getElementById('closeMassage').addEventListener('click', () => {
        closeMessageModal()
    });

    document.getElementById('loginButton').addEventListener('click', async ()
=> {
        await login();
        await initializeAndLoadEditor();
    });
    document.getElementById('registerButton').addEventListener('click', async
() => {
        await register();
    });
    document.getElementById('logoutBtn').addEventListener('click', async () =>
{
        await logout();
        await initializeAndLoadEditor();
    });
    document.getElementById('deleteUserBtn').addEventListener('click', async
() => {
        await deleteUser();
        await initializeAndLoadEditor();
    });
    document.getElementById('saveBtn').addEventListener('click', async () => {
        await saveText();
    });
});

document.addEventListener('DOMContentLoaded', async () => {
    await initializeAndLoadEditor();
    await checkLoginStatus();
});
```

```
function showModal(modalId) {
    document.getElementById(modalId).style.display = 'block';
}
function closeModal(modalId) {
    document.getElementById(modalId).style.display = 'none';
    //document.getElementById(modalId + 'Error').innerText = '';
}
function closeMessageModal() {
    document.getElementById('messageModal').style.display = 'none';
}
function showMessage(message) {
    document.getElementById('messageText').innerText = message;
    document.getElementById('messageModal').style.display = 'block';
}
function displayError(modalId, message) {
    document.getElementById(modalId).innerText = message;
}

async function checkLoginStatus() {
    try {
        const response = await fetch('/check');
        const data = await response.json();
        if (data.loggedin) {
            document.getElementById('userInfo').innerText = 'Logged in as ' +
data.username;
            document.getElementById('userInfo').style.display = 'block';
            document.getElementById('logoutBtn').style.display = 'inline';
            document.getElementById('deleteUserBtn').style.display = 'inline';
            document.getElementById('loginBtn').style.display = 'none';
            document.getElementById('registerBtn').style.display = 'none';
        } else {
            //document.getElementById('userInfo').style.display = 'none';
            document.getElementById('userInfo').innerText = 'Please Login'
            document.getElementById('userInfo').style.display = 'block';
            document.getElementById('logoutBtn').style.display = 'none';
            document.getElementById('deleteUserBtn').style.display = 'none';
            document.getElementById('loginBtn').style.display = 'inline';
            document.getElementById('registerBtn').style.display = 'inline';
        }
    } catch (error) {
        console.error('Error:', error);
```

```
    }
}

async function login() {
    const username = document.getElementById('loginUsername').value;
    const password = document.getElementById('loginPassword').value;
    try {
        const response = await fetch('/login', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ username, password }),
        });
        const data = await response.json();
        if (data.error) {
            displayError('loginError', data.error);
        } else {
            await checkLoginStatus();
            closeModal('loginModal');
        }
    } catch (error) {
        console.error('Error:', error);
    }
}

async function register() {
    const username = document.getElementById('registerUsername').value;
    const password = document.getElementById('registerPassword').value;
    try {
        const response = await fetch('/register', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ username, password }),
        });
        const data = await response.json();
        if (data.error) {
            displayError('registerError', data.error);
        } else {
            showMessage('User registered successfully! Please log in.');
            closeModal('registerModal');
            document.getElementById('registerUsername').value = '';
            document.getElementById('registerPassword').value = '';
        }
```

```
    } catch (error) {
        console.error('Error:', error);
    }
}

async function logout() {
    try {
        const response = await fetch('/logout', {
            method: 'POST',
        });
        await response.json();
        await checkLoginStatus();
    } catch (error) {
        console.error('Error:', error);
    }
}

async function deleteUser() {
    try {
        const response = await fetch('/deleteuser', {
            method: 'POST',
        });
        const data = await response.json();
        showMessage(data.message);
        await checkLoginStatus();
    } catch (error) {
        console.error('Error:', error);
    }
}

async function saveText() {
  const text = editor.getValue(); // Get text from editor
  try {
    const response = await fetch('/save', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ text }),
    });
    const data = await response.text(); // Assuming the server response is
plain text
```

```
    showMessage(data);
  } catch (error) {
    console.error('Error:', error);
  }
}

async function initializeAndLoadEditor() {
  try {
    const response = await fetch('/load');
    const data = await response.text();
    if (window.editor) {
      window.editor.setValue(data);
    } else {
      window.editor = CodeMirror(document.getElementById('notepad'), {
        value: data,
        mode: "python",
        theme: "monokai",
        lineNumbers: true,
        lineWrapping: true,
      });
    }
  } catch (error) {
    console.error('Error initializing editor or loading content:', error);
    if (!window.editor) {
      window.editor = CodeMirror(document.getElementById('notepad'), {
        value: "",
        mode: "python",
        theme: "monokai",
        lineNumbers: true,
        lineWrapping: true,
      });
    }
  }
}
```

Style.css:

```
html {
    margin: 0;
    padding: 0;
    height: 100%;
    width: 100%;
```

```
    font-family: Arial, sans-serif;
}

body {
    background-image: url("back.jpg");
    background-size: 100% 100%;
    background-repeat: no-repeat;
    background-position: center;
}

.modal-content {
    background-color: #fefefe;
    margin: 15% auto;
    padding: 20px;
    border: 1px solid #888;
    width: 80%;
    max-width: 300px;
    border-radius: 10px;
    box-shadow: 0 4px 8px #00000033;
}

.close {
    color: #aaa;
    float: right;
    font-size: 28px;
    font-weight: bold;
}

.close:hover,
.close:focus {
    color: black;
    text-decoration: none;
    cursor: pointer;
}

input[type="text"], input[type="password"] {
    width: 100%;
    padding: 12px 20px;
    margin: 8px 0;
    display: inline-block;
    border: 1px solid #ccc;
    box-sizing: border-box;
```

```css
}

button {
    background-color: #4CAF50;
    color: white;
    padding: 14px 20px;
    margin: 8px auto;
    border: none;
    cursor: pointer;
    width: 100%;
    border-radius: 10px;
    box-shadow: 0 4px 8px #00000033;
}

button:hover {
    opacity: 0.8;
}

.error {
    color: red;
    font-size: 0.9em;
}

.modal {
    display: none;
    position: fixed;
    z-index: 2;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    overflow: auto;
    background-color: black;
    background-color: #00000066;
}

.close {
    color: #aaa;
    float: right;
    font-size: 28px;
    font-weight: bold;
}
```

```css
.center {
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    padding: 20px 20px 20px 20px;
    background-color: white;
    border-radius: 10px;
    box-shadow: 0 4px 8px #00000033;
}

.CodeMirror {
    border: 1px solid #eee;
    /* Assuming a character width of around 8px at the current font size;
adjust as needed */
    width: 640px; /* 80 characters * 8px per character */
    height: auto;
    min-height: 300px;
    border-radius: 10px;
    box-shadow: 0 4px 8px #00000033;
}
```