Created by Openai Turbo 3.5

project/

│

├── server/

│ ├── app.js

│ ├── routes/

│ │ ├── login.js

│ │ ├── dashboard.js

│ │ ├── settings.js

│ └── views/

│ ├── login.ejs

│ ├── dashboard.ejs

│ ├── settings.ejs

│

└── client/

├── index.html

├── css/

│ ├── styles.css

├── js/

│ ├── router.js

│ ├── login.js

│ ├── dashboard.js

│ └── settings.js

In this folder structure:

* The **server** folder contains the server-side code, including the **app.js** file for setting up the Express server, the **routes** folder for handling different routes, and the **views** folder for EJS templates.
* The **client** folder contains the client-side code, including the **index.html** file as the main entry point, the **css** folder for stylesheets, and the **js** folder for JavaScript files.

Now, let's write the code for the server-side and client-side components.

# Let's write the code for the server-side components using Express.js

# Filename: server/app.js

server\_app\_code = """

const express = require('express');

const app = express();

const path = require('path');

// Set EJS as the view engine

app.set('view engine', 'ejs');

// Serve static files from the 'client' folder

app.use(express.static(path.join(\_\_dirname, '../client')));

// Define routes

app.use('/login', require('./routes/login'));

app.use('/dashboard', require('./routes/dashboard'));

app.use('/settings', require('./routes/settings'));

// Start the server

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

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

"""

server\_login\_route\_code = """

const express = require('express');

const router = express.Router();

// GET login page

router.get('/', (req, res) => {

res.render('login');

});

module.exports = router;

"""

server\_dashboard\_route\_code = """

const express = require('express');

const router = express.Router();

// GET dashboard page

router.get('/', (req, res) => {

res.render('dashboard');

});

module.exports = router;

"""

server\_settings\_route\_code = """

const express = require('express');

const router = express.Router();

// GET settings page

router.get('/', (req, res) => {

res.render('settings');

});

module.exports = router;

"""

# Now, let's write the client-side JavaScript code for the router and the pages

# Filename: client/index.html

index\_html\_code = """

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>MultiModal App</title>

<link rel="stylesheet" href="css/styles.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="#/login">Login</a></li>

<li><a href="#/dashboard">Dashboard</a></li>

<li><a href="#/settings">Settings</a></li>

</ul>

</nav>

</header>

<main id="app"></main>

<script src="js/router.js"></script>

</body>

</html>

"""

# Filename: client/js/router.js

router\_js\_code = """

function loadPage(pageName) {

const app = document.getElementById('app');

fetch(`/views/${pageName}.ejs`)

.then(response => response.text())

.then(html => {

app.innerHTML = html;

});

}

function navigate() {

const hash = window.location.hash.substring(1);

if (hash === 'login' || hash === 'dashboard' || hash === 'settings') {

loadPage(hash);

} else {

loadPage('login');

}

}

window.addEventListener('hashchange', navigate);

window.addEventListener('load', navigate);

"""

# Filename: client/js/login.js

login\_js\_code = """

// Add login page functionality

"""

# Filename: client/js/dashboard.js

dashboard\_js\_code = """

// Add dashboard page functionality

"""

# Filename: client/js/settings.js

settings\_js\_code = """

// Add settings page functionality

"""

I have prepared the code for the server-side components using Express.js as well as the client-side JavaScript code for the router and the pages. Here are the contents of the files:

### Server-side Code

#### Filename: server/app.js

${server\_app\_code}

#### Filename: server/routes/login.js

${server\_login\_route\_code}

#### Filename: server/routes/dashboard.js

${server\_dashboard\_route\_code}

#### Filename: server/routes/settings.js

${server\_settings\_route\_code}

### Client-side Code

#### Filename: client/index.html

${index\_html\_code}

#### Filename: client/js/router.js

${router\_js\_code}

#### Filename: client/js/login.js

${login\_js\_code}

#### Filename: client/js/dashboard.js

${dashboard\_js\_code}

#### Filename: client/js/settings.js

${settings\_js\_code}

You can organize these files into the folder structure mentioned earlier, and ensure that the server-side files are placed in the appropriate **server** directory, and client-side files in the **client** directory.

Let me know if you need further assistance or additional code examples!

Created by: Phind.com

Prompt: write code with login, dashboard and settings pages. use vanilla js for the client side router and express js in the server. indicate the filenames for the code if it is modular. illustrate the folder structure.

To create a simple web application with login, dashboard, and settings pages using Vanilla JS for client-side routing and Express.js for the server, we'll organize the project into several modules for better maintainability. Here's a basic outline:

Folder Structure/myapp

/client

/js

- main.js (Client-side router)

- login.js

- dashboard.js

- settings.js

/html

- index.html

- login.html

- dashboard.html

- settings.html

/server

- app.js (Express server)

package.json

Client-Side Routing (main.js)

// client/js/main.js

document.addEventListener('DOMContentLoaded', function() {

const routes = {

'/': () => import('./html/index.html'),

'/login': () => import('./html/login.html'),

'/dashboard': () => import('./html/dashboard.html'),

'/settings': () => import('./html/settings.html')

};

function loadPage(path) {

window.location.hash = path;

routes[path]().then(html => document.getElementById('content').innerHTML = html.default);

}

// Example route handling

window.addEventListener('hashchange', () => loadPage(window.location.hash.substring(1)));

loadPage(window.location.hash.substring(1));

});

Server-Side Setup (app.js)

// server/app.js

const express = require('express');

const app = express();

const port = 3000;

app.use(express.static('client'));

app.get('/', (req, res) => res.sendFile(\_\_dirname + '/client/html/index.html'));

app.get('/login', (req, res) => res.sendFile(\_\_dirname + '/client/html/login.html'));

app.get('/dashboard', (req, res) => res.sendFile(\_\_dirname + '/client/html/dashboard.html'));

app.get('/settings', (req, res) => res.sendFile(\_\_dirname + '/client/html/settings.html'));

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

HTML Template (index.html)

<!-- client/html/index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>My App</title>

</head>

<body>

<div id="content"></div>

<script src="/js/main.js"></script>

</body>

</html>

This setup uses ES Modules for dynamic imports in the client-side router (main.js) and Express.js for serving static files and routing on the server (app.js). Each page has its own HTML file, and main.js dynamically loads them based on the URL hash. This approach keeps the code modular and easy to manage.

=================================================

**Created by OpenAI Assistant in Playground using gpt-4o-2024-5-13**

### **Folder Structure**

project-root/

├── client/

│ ├── index.html

│ ├── app.js

│ ├── login.html

│ ├── dashboard.html

│ ├── settings.html

│ └── styles.css

├── server/

│ ├── app.js

│ ├── routes/

│ │ ├── auth.js

│ │ ├── dashboard.js

│ │ └── settings.js

├── package.json

└── package-lock.json

**Client**

**1. index.html**

This will be the entry point of our application.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>SPA with Vanilla JS</title>

<link rel="stylesheet" href="/styles.css">

</head>

<body>

<div id="app"></div>

<script src="/app.js"></script>

</body>

</html>

**2. login.html**

The login page.

<div>

<h2>Login</h2>

<form id="loginForm">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<button type="submit">Login</button>

</form>

</div>

**3. dashboard.html**

The dashboard page.

<div>

<h2>Dashboard</h2>

<p>Welcome to the dashboard!</p>

</div>

**4. settings.html**

The settings page.

<div>

<h2>Settings</h2>

<p>Manage your settings here.</p>

</div>

**5. styles.css**

body {

font-family: Arial, sans-serif;

}

#app {

padding: 20px;

}

**6. app.js**

document.addEventListener("DOMContentLoaded", () => {

const appDiv = document.getElementById("app");

const loadContent = (url) => {

fetch(url)

.then(response => response.text())

.then(html => appDiv.innerHTML = html)

.catch(err => console.error('Failed to load content', err));

};

const routes = {

"/": "/login.html",

"/dashboard": "/dashboard.html",

"/settings": "/settings.html"

};

const onRouteChange = () => {

const path = window.location.pathname;

const resource = routes[path] || routes["/"];

loadContent(resource);

};

window.onpopstate = onRouteChange;

const handleRouteChange = (event) => {

event.preventDefault();

const path = event.target.getAttribute("href");

window.history.pushState({}, path, window.location.origin + path);

onRouteChange();

};

document.addEventListener("click", event => {

if (event.target.matches("a")) handleRouteChange(event);

});

onRouteChange();

appDiv.addEventListener("submit", (event) => {

if (event.target.id === "loginForm") {

event.preventDefault();

const formData = new FormData(event.target);

fetch("/login", {

method: "POST",

body: JSON.stringify({

username: formData.get("username"),

password: formData.get("password")

}),

headers: {

"Content-Type": "application/json"

}

})

.then(response => response.json())

.then(data => {

if (data.success) {

window.history.pushState({}, "/dashboard", window.location.origin + "/dashboard");

onRouteChange();

} else {

alert("Login failed!");

}

})

.catch(err => console.error('Login failed', err));

}

});

});

### **Server**

**1. server/app.js**

The main server file.

const express = require('express');

const app = express();

const path = require('path');

const bodyParser = require('body-parser');

const authRoutes = require('./routes/auth');

const dashboardRoutes = require('./routes/dashboard');

const settingsRoutes = require('./routes/settings');

app.use(bodyParser.json());

app.use(express.static(path.join(\_\_dirname, '..', 'client')));

app.use(authRoutes);

app.use(dashboardRoutes);

app.use(settingsRoutes);

app.get('\*', (req, res) => {

res.sendFile(path.resolve(\_\_dirname, '..', 'client', 'index.html'));

});

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

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

**2. server/routes/auth.js**

Authentication routes.

const express = require('express');

const router = express.Router();

router.post('/login', (req, res) => {

const { username, password } = req.body;

if (username === 'user' && password === 'password') {

res.json({ success: true });

} else {

res.json({ success: false });

}

});

module.exports = router;

**3. server/routes/dashboard.js**

Dashboard routes.

const express = require('express');

const router = express.Router();

router.get('/dashboard', (req, res) => {

res.json({ success: true, message: 'Welcome to the dashboard!' });

});

module.exports = router;

**4. server/routes/settings.js**

Settings routes.

const express = require('express');

const router = express.Router();

router.get('/settings', (req, res) => {

res.json({ success: true, message: 'Manage your settings here.' });

});

module.exports = router;

*Install Dependencies*

*In project-root/, run:*

*npm init -y*

*npm install express body-parser*

*This sets up a basic structure for a single-page application with Vanilla JavaScript for the client-side routing and Express.js for the server-side framework. The login mechanism is very basic and is for demonstration purposes only. In a real application, you would add more sophisticated user authentication and session management.*