1. Contact

Client side:

'/insertContact' method:**Post**, send all data as object to Contact Collection

2.Registration

Client side:

'/checkCustomerEmail' method:**Post** , send(email) to check whether the email exist in customer collection. get Boolean value as return.

'/registration' method:**Post**, send all fields data as object to customer collection

3. login

Client side:

'/login' method:**Post**, send (email, password) to customer collection, get the JWT token as return

https://www.npmjs.com/package/jwt-simple

Usage of JWT in Frontend

The typical workflow for using JWT in frontend is as follows:

* **User Login**: Users log in with their username and password. The frontend sends the username and password to the backend for validation.
* **Backend Validates** Login Information: The backend validates the username and password, generates a JWT (JSON Web Token), embeds some basic user information (such as user ID, username, etc.) as the payload of the JWT, signs it, and returns it to the frontend.
* **Frontend Stores** JWT: Upon receiving the JWT, the frontend stores it in the client's local storage (e.g., localStorage or sessionStorage).
* **JWT Sent with Subsequent Requests:** When making subsequent requests, the frontend includes the stored JWT in the request's header, typically in the Authorization header's Bearer field.
* **Backend Validates** JWT: Upon receiving a request, the backend validates the JWT for its validity and integrity. If the JWT is not expired and the signature is correct, the backend trusts the information contained within it and proceeds with the corresponding operation.

Below is a simple example code demonstrating how the frontend utilizes JWT:

// Login function

async function login(username, password) {

try {

// Send login request

const response = await fetch('/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ username, password })

});

const data = await response.json();

// If login successful, store JWT in local storage

if (response.ok) {

localStorage.setItem('jwt', data.token);

console.log('Login successful');

} else {

console.error('Login failed:', data.message);

}

} catch (error) {

console.error('Login failed:', error);

}

}

// Send request for protected resource

async function fetchProtectedResource() {

try {

// Get JWT from local storage

const token = localStorage.getItem('jwt');

if (!token) {

console.error('Not logged in');

return;

}

// Send request with JWT

const response = await fetch('/protected-resource', {

headers: {

'Authorization': `Bearer ${token}`

}

});

const data = await response.json();

// Handle response

if (response.ok) {

console.log('Successfully fetched resource:', data);

} else {

console.error('Failed to fetch resource:', data.message);

}

} catch (error) {

console.error('Failed to fetch resource:', error);

}

}

// Example: Login and fetch resource

login('username', 'password')

.then(fetchProtectedResource)

.catch(console.error);

// Example: fetch resource

fetchProtectedResource()

.catch(console.error);

4. Appointment

Client side:

‘/getAppointmentTime’ method:**Get**, send(serviceName, apptDate)to get unavailable time value as array from appointment collection as return.(Once when click the data, once when click the next button)

5. Payment

Client side:

‘/insertPayment’ method:**Post**, send paymentObj to payment collection, get payment id as return.

‘/insertAppointment**’** method:**Post**, send appointmentObject to appointment collection, get appointment id as return.

5.1. Appointment Print

??Get,

6. personal information edit

Client side:

‘/insertCustomer’ method:**Post** , send all the field as one object to customer collection

7, Appointment Manage

Client side:

‘/getAppointment’ method:**Get**, send(customerId) to get all appointment objects from appointment collection

1) reschedule,

the same as appointment

‘/updateAppointment’ method:**Put**, send(appointmentId, apptDate, apptTime, createTimeStamp ), to update these fileds in appointment collection

2) cancel

‘/deleteAppointment’ method:**Delete** , send(appointmentId)to delete this appointment from appointment collection

3) feedback

‘/insertFeedback’ method:**Post** ,send all fields as an object to feedback collection, (serviceName,1,2,3,4,5(1))

**!!!If you want to put fetch method in a different file**

To call the contents of a fetch.js file in a regular JavaScript file:

Assuming your fetch.js file defines a function named fetchData, you can export it like this:

// fetch.js

export function fetchData() {

// Asynchronous request logic

}

Then in your regular JavaScript file, you can import the fetchData function and call it:

// your\_file.js

import { fetchData } from './fetch.js';

// Call the fetchData function

fetchData();

This way, you can use the fetchData function exported from the fetch.js file in your regular JavaScript file. Make sure to ensure the correctness of the file paths and use appropriate build tools (such as webpack, Parcel, etc.) to handle module loading.