Web-Script Programming
Final Report

August 2018

Contents

1 Introduction

The portal is used by employees of a fictional XYZ company to manage medical leaves. When an employee is sick he informs his supervisor by creating a leave inform request. Later the employee can update the leave with extra details like a medical certificate, how long is he going on leave etc. Supervisors can view the details of the leaves applied by employees, history of medical leaves, the details of the tasks assigned to the employee and can acknowledge the leave inform request.

1.1 Architecture

The application is separated into medical-leave-client and medical-leave-server . Client side of the application is coded in HTML, CSS and Javascript. AJAX is used to talk to the server, to get details and update rows in the database.

The server side of the application is coded in Express js. The server exposes various functionalities required by the client as API endpoints. The data transfer is in JSON format except for file uploads.

I'm using SQLite as the backend database for the sake of simplicity. Three database files are provided along with the application.

1 **database.db** – This is the one used by the application by default

2 **database_with_sample_data.db** – Database with sample employee, clinics and tasks data

3 **database_with_no_data.db** – Database with empty tables

The below diagram summarizes the architecture of the application.

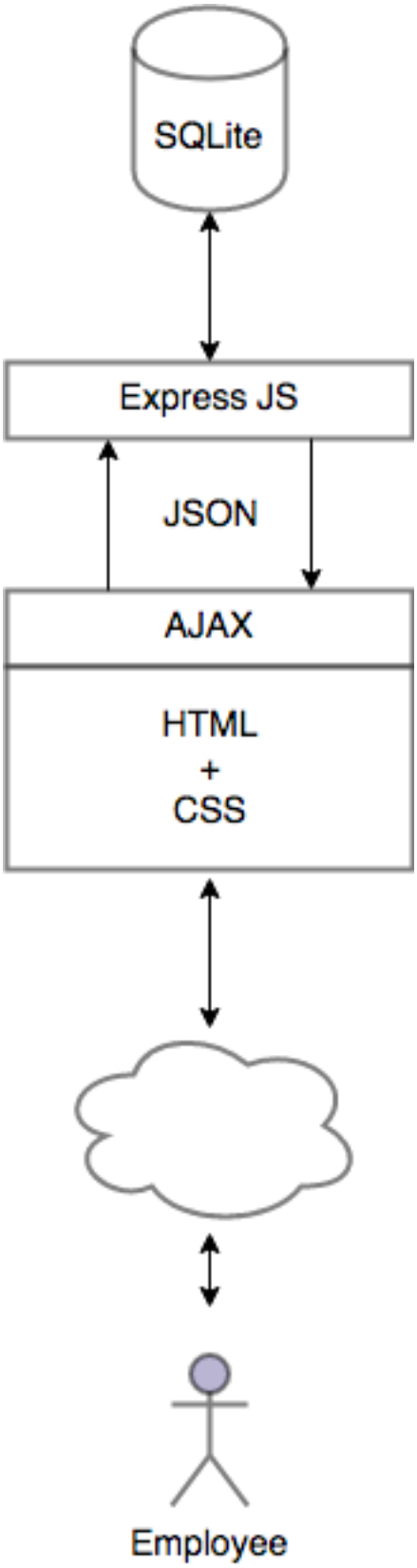

Figure 1: Architecture diagram

1.2 Testing

Test cases are written for the server only. To run the test cases:

```
cd /path/to/medical-leave-server
npm install
npm run test
```

1.3 Running the application

Start the server:

```
cd /path/to/medical-leave-server
npm install
npm start
```

The client application can be served from any webserver. In this example we are using http-server package from npm to serve the client application.

```
npm install http-server -g
cd /path/to/medical-leave-client
http-server
```

This will start an http server listening on port 8080. Visit http://localhost:8080 to access the client application.

1.4 Sample Logins

The following employees are available in the sample database. Use the employee ID as username and 'commonpass' as the password to login.

No	Employee ID	Supervisor	Password
1	500101	500101	commonpass
2	500102	500101	commonpass
3	500103	500101	commonpass
4	500104	500101	commonpass
5	500105	500104	commonpass
6	500106	500104	commonpass
7	500107	500104	commonpass
8	500108	500102	commonpass
9	500109	500104	commonpass
10	500110	500103	commonpass

2 Reflection

The following attributes are handled quite well in the application:

1 Client and Server are completely isolated and developed separately. Going forward both can be maintained independendantly without affecting each other.

2 Avoided using too many frameworks and used vanilla javascript to handle most of the functionalities.

3 Avoided javascript 'callback-hell' in the AJAX calls as much as possible by using async-await functions.

4 Test cases are written for all the base cases.

5 The server side components are organized following best practices. Config files, database connections, helper functions and core functions are all modularised and can be maintained separately.

The following attributes could be improved further:

1 Write tests to cover all the edge cases and preferably use some automated testing for front end as well.

2 Validate all the inputs at server side as well. The db query and parameter validation is handed over to the sqlite package which is good enough for basic validations. However, building an extra layer of validation at the server layer would have been better.

3 Extra client side validations like checking if the user has already informed a sick leave for the day will increase the usability of the application.

4 Improving the user experience further by including search suggestions, auto completion etc.