QUEENSLAND UNIVERSITY OF TECHNOLOGY

27/09/2021

ALEXANDER IFTENE

N9657533

CAB432 CLOUD COMPUTING

SEMESTER 2

Table of Contents

Introduction

Mashup Purpose & Description

The Real estate business is booming all around the world now even with Covid-19. People need homes to stay with their families. For this reason, the Real Estate Force web application was invented in order to assist real estate managers simplify their operations in one platform. This app captures all properties managed by a real estate company and identifies each home as a unit. It also captures the personal information of all the tenants occupying the units. All this is done on a very simple interface that does not require a lot of learning to work with. The user interface of this app is very responsive thus it can be accessed by even smaller screened devices with no difficulty that way improving its user experience as well.

Services used

The app makes use on an API endpoint that performs Create, Read, Update and Delete operations on a MYSQL database. It retrieves data in JSON format and accepts data input in JSON format as well. This API was build on JavaScript and node.JS express; this is a popular and latest web technology for making restful APIs. The framework makes it diverse in that it can be installed easily on any server since it works on Centos, Ubuntu, Windows and any other popular servers available on the amazon web services platform or google cloud.

Mashup Use Cases and Services

Below are the user stories that assisted in the design of the application.

As a	Real estate manager
I want	To have records of all tenants

In order to	Simplify my communication with them and easily get financial reports

As a	Tenant
I need to	Get a home space easily and make payments without a big hassle
So that	I do not get unnecessary interuptions and disturbances during my stay

As a	Real estate manager
I want to	Know which tenant lives on which unit without having to vising the residence physically
So that	I can concentrate on providing quality services to other customers while in the office instead of making visits to the residential area everytime.

Retrieve all rental properties	{url:4300}/properties
Retrieve all the tenants	{url:4300}/tenants
Add new rental properties	{url:4300}/properties
Add new tenants	{url:4300}/tenants

Technical breakdown

The backend side of this application was developed entirely on Node.js. Node.js is an open source runtime environment whose base is Javascript and it allows for effortless development of scalable web applications. This backend framework is making trends now on real-time web applications that make use of push technology and not web sockets. Node.js has been a revolutionary framework since in the course of more than 20 years of using stateless web based request and response design, it finally accrued that its possible to have real-time web applications with two way connection whereby the server and the client can initiate a communication which makes them exchange data/information in a free manner. Node.js is technically based on HTML, JS and CSS and runs over the standard port 80 unless defined otherwise. Node js comes with the Node Package Manager (npm) which is a set of reusable components available publicly through an online repository and has easy management of dependencies and version control. For this web application, we made use of a number of npm modules which include:

Express: Express is an insignificant and adaptable Node.js web application structure that gives a strong arrangement of components for mobile and web applications. It gives a slim layer of basic web application highlights, without darkening Node.js feature sets that you know and love. With a heap of HTTP utility techniques and middleware available to you, making a vigorous API is fast and simple.

Mongodb: Provides support for the MongoDB object database.

HttpClient: It creates a service that performs HTTP requests. It is used as an injectable class containing methods that do HTTP requests. Every request method available contains a number of signatures and returns a type depending on the signature that is parsed as an argument. The signatures are mainly the values of the responseType and observe variables.

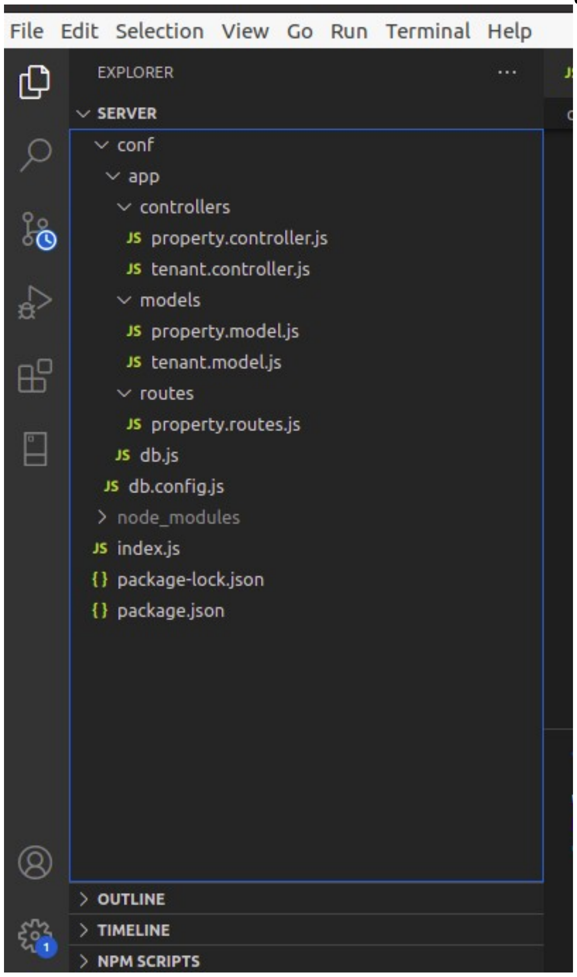
On the other hand, the front end was designed using Angular. Angular JS is a framework that is structural and meant for dynamic web applications. Using Amgular, it is possible to make use of

HTML as a templating language and it further provides for the extension of the syntax of HTML into showing the components of the application in an effortless way.

Architecture and Data Flow

The general architecture of the application is well organized in a simple and well understandable way. I have two directories that hold two distinct purposes. One is the server directory and the other is the client directory, they hold source code for the server side and client side respectively. An explanation of the structure of each of the directories is shown below:

As aforementioned, the server was build using Node JS, the entire directory looks like the figure attached below.



```
SERVER
  conf
    app
      controllers
        property.controller.js
        tenant.controller.js
      models
        property.model.js
        tenant.model.js
      routes
        property.routes.js
      db.js
    db.config.js
  node_modules
  index.js
  package-lock.json
  package.json
```

It is made up of the **routes**, **controllers** and **models**. The routes define how the web application’s end points make responses to requests from clients. The routing is defined by methods of the express object which corresponds to HTTP methods; a good example is app.post that handles POST requests and app.get that handles GET requests. These methods also define a call back function that is called every time the application receives requests to a specified route and HTTP method. The example source code below shows an example of a route that was used in the web application.

```
module.exports = app => {
const properties = require("../controllers/property.controller.js");
const tenant = require("../controllers/tenant.controller.js");
// Create a new Property
app.post("/properties", properties.create);
app.get("/properties", properties.findAll);
app.get("/property/:owner", properties.findCustom);
app.get("/property-name/:property_name", properties.findPropName);
app.get("/properties/:propertyId", properties.findOne);
// Update a Property with propertyId
app.put("/properties/:propertyId", properties.update);
// Delete a Property with propertyId
app.delete("/properties/:propertyId", properties.delete);
// app.delete("/properties", properties.deleteAll);
// Tenants
app.post("/tenant", tenant.create);
app.get("/tenants", tenant.findAll);
app.get("/tenants/:tenantId", tenant.findOne);
};
```

The controllers on this application are callback functions which make correspondense to routers in order to handle requests.

The tenant controller was created as shown in the source code below.

```
const Tenant = require("../models/tenant.model.js");

// Create and Save a new Tenant
exports.create = (req, res) => {
// Validate request
if (!req.body) {
res.status(400).send({
message: "Sorry the tenant body not be empty!"
});
}
console.log(req);
// Create a Tenant
const tenant = new Tenant({

first_name:req.body.first_name,
last_name:req.body.last_name,
unit_number:req.body.unit_number,
phone: req.body.phone,
email: req.body.email,
resident_center_status: req.body.resident_center_status,
text_message_status: req.body.text_message_status
});
// Save Tenant in the database
Tenant.create(tenant, (err, data) => {
if (err)
res.status(500).send({
message:
err.message || "Some error occurred while creating the Tenant."
});
else res.send(data);
});
};
```

In this tenant controller, we create the create function to handle the POST '/tenant' request. Which creates new tenants in the database. Furthermore, we exported this function in order to be able to import it to our routes/property.routes.js, as shown earlier.

The Model is a representation of a collection of documents available in the database where anyone can scan; the instances of models represent all unique documents that can be retrieved and saved. In this framework, all CRUD operations are made synchronously by supplying a callback that is executed when the operation is completed. This API makes use of a good convention known as the error-first argument style which means that every call back argument that comes first is always an error or a null value, any result that is returned will come as a second argument. The model in this scenario can make database operations directly by making use of the mysql module imported into the project. This way, it is possible to write SQL select statements in Javascript. The source code below illustrates how the tenant model was made

```
const sql = require("../db.js");
// constructor
const Tenant = function(tenant) {
 this.first_name = tenant.first_name;
 this.last_name = tenant.last_name;
 this.unit_number = tenant.unit_number;
 this.phone = tenant.phone;
 this.email = tenant.email;
 this.resident_center_status = tenant.resident_center_status;
 this.text_message_status = tenant.text_message_status;
 };
 Tenant.create = (newtenant, result) => {
 sql.query("INSERT INTO new_tenant SET ?", newtenant, (err, res) => {
 if (err) {
 console.log("error: ", err);
 result(err, null);
 return;
 }
 console.log("created new_tenant: ", { id: res.insertId, ...newtenant });
 result(null, { id: res.insertId, ...newtenant });
 });
 };
 Tenant.getAll = result => {
 sql.query("SELECT * FROM new_tenant", (err, res) => {
 if (err) {
 console.log("error: ", err);
 result(null, err);
 return;
 }
 console.log("tenant: ", res);
 result(null, res);
 });
 };
```

Deployment and the Use of Docker

To deploy the app using docker, the server first needs to have docker installed. For my case, I am making use of the Ubuntu server; Docker was simply installed by running the command below in the terminal.

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

To dockerize the app backend/server, these were the steps:

Creating an empty docker file by running; **touch Dockerfile** on the terminal. Then created a folder/directory to keep the application source code in the docker image and will become the working directory of our web application as well.

# Create app directory

WORKDIR /usr/src/app

Ensure that pacakage.json are well copied by having in the dockerfile this.

COPY package*.json ./

Install all node js modules by:

RUN npm install

To bundle the app's source code in the docker image do:

COPY . .

Then expose the port 4300 where the app binds to by:

EXPOSE 4300

Make a definition of the command used to execute the server app, that is:

CMD ["node", "index.js"]

That is the docker file for the server side, moreover, we created an **.dockerignore** file in the same directory are the Dockerfile. It contains the content to ignore when copying debug logs and local modules to the docker image.

Dockerfile for the client side will look like below:

WORKDIR /usr/src/app

Ensure that pacakage.json are well copied by having in the dockerfile this.

COPY package*.json ./

Install all node js modules by:

RUN npm install

To bundle the app’s source code in the docker image do:

COPY . .

Then expose the port 4300 where the app binds to by:

EXPOSE 4200

Make a definition of the command used to execute the server app, that is:

CMD [“ng”, “serve”]

Finally, the Docker image was built buy running on the terminal the command below:

docker build . -t <username>/server

After image creation, the image can be executed by running:

```
docker run -d <username>/server
docker run -d <username>/client
```

Test plan

TASK	OUTCOME EXPECTED	RESULT
Show table of all tenants	Table of all tenants shows on main page	PASS
Show table of all rental properties	A table of all rental properties shows on the main page	PASS
Create new Rental Properties	A new rental property can be created by filling the form and clicking create on the main page	PASS
Create new tenants	A new tenant can be created by filling the form and clicking create on the main page	PASS

Difficulties / Exclusions / unresolved & persistent errors

One difficulty experienced during the course of this project is data collection. This was a big challenge considering the covid19 restrictions. It was not easy getting information on what kind of systems the Real estate market is using right now and what improvements they would love to be included or recreated in the system. I resolved this challenge by making a lot of calls to real estate managers I found on the internet.

Extensions

This web application will need a lot of improvements in the future. Among the improvements to be done is:

1. Creation of a rental roll that contains a number of features that may include; payment confirmations, lease addition, renew lease and bulk charges.
2. Linking the web app with an financial accounting system so that it can create invoices and a number of journal reports for the managers.

User guide

To run this web app it is simple. Just use docker to start the server by running: docker run -d <username>/server. And by using docker also to start the front end by running docker run -d <username>/client. All done from their respective directories.

Statement on Assignment Demo

The demo for this assignment will be presented face to face.

References

Hoque, S. (2020). *Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node. js*. Packt Publishing Ltd.

Huang, Z., Wang, S., & Yu, K. (2018, September). Angular Softmax for Short-Duration Text-independent Speaker Verification. In *Interspeech* (pp. 3623-3627).

Jensen, P. (2017). *Cross-platform desktop applications: Using node, electron, and Nw. js*. Simon and Schuster.

Kubala, J. (2017). Progressive web app with Angular 2 and ASP. NET.

Mardan, A., Mardan, & Corrigan. (2018). *Practical Node. js*. Apress.

Mardan, A. (2018). Using Express. js to create Node. js web apps. In *Practical Node. js* (pp. 51-87). Apress, Berkeley, CA.

Mardan, A. (2018). Getting Node. js Apps Production Ready. In *Practical Node. js* (pp. 331-364). Apress, Berkeley, CA.

Mardan, A. (2018). Boosting Node. js and MongoDB with Mongoose. In *Practical Node. js* (pp. 239-276). Apress, Berkeley, CA.

Saks, E. (2019). JavaScript Frameworks: Angular vs React vs Vue.

Sepas-Moghaddam, A., Haque, M. A., Correia, P. L., Nasrollahi, K., Moeslund, T. B., & Pereira, F. (2019). A double-deep spatio-angular learning framework for light field-based face recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, *30*(12), 4496-4512.