Find My Property

Technical Document

Contents

Scope of Work

For this project, we are going to create endpoints to add, delete, update and list amenities as well as endpoints to create, list and view details of apartments.

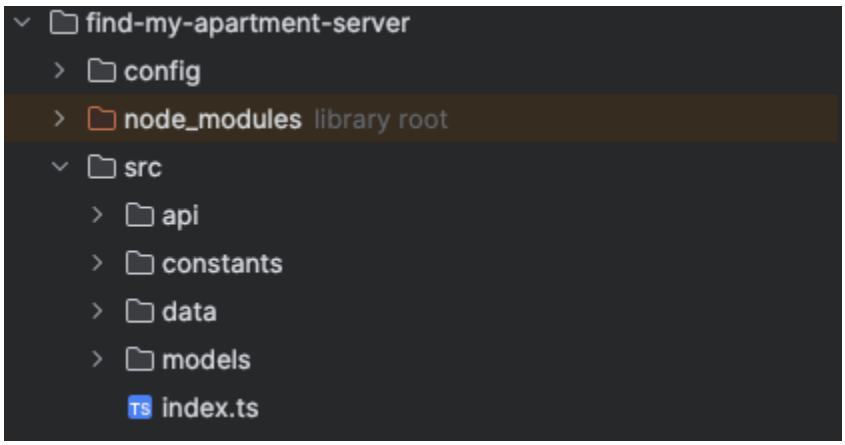
The endpoints are located in find-my-apartment-server.

The server is run on nodejs and contains the src directory where the project files reside.

The server runs on port 4000 and connects to the database on startup.

The server uses Joi library for validations

The folder structure is as follows:



The index.ts is the start of our project.

The config folder contains two files:

1. keys.ts that contains mongo url in case the environment variable `MONGODB_URI` is not provided.
2. mongo.ts that connects to the mongodb using mongo url and populates the database.

The api folder contains 3 directories:

1. controllers that contain controller files that contains the endpoint functions.
2. validations that contain validation files that uses joi to validate endpoint.
3. routers that contain routers for amenities and apartments and calls the controllers and validations.

The constants folder contains two folders:

1. enums folder that contains enums used in the project.
2. network folder that contains response messages and status codes.

The data folder contains two files:

1. dummy-data.json which contains the dummy data used for population.
2. populate.ts which is called upon database connects to use dummy-data.json and populate the tables.

The models folder contains two folders:

1. dto folder that contains the response model.
2. mongo folder that contains mongo models.

Documents

```
import mongoose, { Schema, Document } from 'mongoose';


export interface IAmenity extends Document {
    name: string;
}

const amenitySchema: Schema = new Schema<IAmenity>( definition: {
    name: { type: String, required: true },
});


export default mongoose.model( name: 'amenity', amenitySchema);
```

```
const apartmentSchema: Schema = new Schema<IApartment>( definition: {
    title: {type: String, required: true},
    description: {type: String, required: true},
    numberOfBedrooms: {type: Number, required: true},
    numberOfBathrooms: {type: Number, required: true},
    sizeInSquareMeters: {type: String, required: true},
    price: {type: Number, required: true},
    address: {
        street: {type: String, required: true},
        city: {type: String, required: true},
        state: {type: String, required: true},
        latitude: {type: Number, required: true},
        longitude: {type: Number, required: true},
    },
    imagesUrls: [{type: String, required: true}],
    contactInfo: {
        name: {type: String, required: true},
        email: {type: String, required: true},
        phone: {type: String, required: true},
    },
    available: {type: Boolean, required: true},
    createdAt: {type: Date, default: Date.now},
    level: {type: String, required: false},
    paymentOptions: [{type: String, enum: [...Object.values(PaymentOptionsEnum)]}],
    furnished: {type: Boolean, required: false},
    type: {type: String, required: true, enum: [...Object.values(ApartmentTypeEnum)]},
    amenities: [AmenityModel.schema],
    listingType: {type: String, required: true, enum: [...Object.values(ListingTypeEnum)]},
    rentPaymentFrequency: {type: String, required: false, enum: [...Object.values(RentPaymentFrequencyEnum)]},
});


export default mongoose.model( name: 'apartment', apartmentSchema);
```

Enums

```
export enum ApartmentTypeEnum {
    APARTMENT = 'Apartment',
    HOUSE = 'House',
    DUPLEX = 'Duplex',
    ROOM = 'Room',
    STUDIO = 'Studio',
    VILLA = 'Villa'
};
```

```
export enum ListingTypeEnum {
    RENT = 'Rent',
    SALE = 'Sale',
}
```

```
export enum PaymentOptionsEnum {
    CASH = 'Cash',
    VISA = 'Visa',
    MORTGAGE = 'Mortgage'
};
```

```
export enum RentPaymentFrequencyEnum {
    DAILY = 'Daily',
    WEEKLY = 'Weekly',
    MONTHLY = 'Monthly',
    YEARLY = 'Yearly'
};
```

Amenities Endpoints

Endpoint 1

POST: /amenity/create					
	Name	**Data Type**	**Optional**	**Default**	**Accepted Values**
Input	name	string	False	-	-
Output	data	AmenityModel	-	-	-

This endpoint creates an amenity and returns the created amenity from the amenity model of mongo listed above.

This amenity with be later used when creating an apartment

Endpoint 2

PUT: /amenity/edit					
	Name	**Data Type**	**Optional**	**Default**	**Accepted Values**
Input	id	string	False	-	-
Input	name	string	False		
Output	data	AmenityModel	-	-	-

This endpoint edits an amenity and returns the edited amenity from the amenity model of mongo listed above.

In case the amenity is used in an apartment it goes and edits the amenity in the apartment as well.

Endpoint 3

DELETE: /amenity/delete					
	Name	**Data Type**	**Optional**	**Default**	**Accepted Values**
Input	id	string	False	-	-
Output	-	-	-	-	-

This endpoint deletes an amenity.

In case the amenity is used in an apartment it goes and deletes the amenity in the apartment as well.

Endpoint 4

GET: /amenity/list					
	Name	**Data Type**	**Optional**	**Default**	**Accepted Values**
Input	-	-	-	-	-
Output	data	AmenityModel[]	-	-	-

This endpoint gets all amenities in the database.

Apartment Endpoints

Endpoint 1

GET: /apartment/list					
	Name	**Data Type**	**Optional**	**Default**	**Accepted Values**
Input	-	-	-	-	-
Output	data	ApartmentListModel[]	-	-	-

This endpoint gets all apartments in the database containing only the needed attributes.

AddressModel	
street	string
city	string
state	string

ApartmentListModel	
title	string
description	string
numberOfBedrooms	number
numberOfBathrooms	number
sizeInSquareMeters	number
price	number
imagesUrls	string[]
address	AddressModel
listingType	ListingTypeEnum (listed above)
rentPaymentFrequency	RentPaymentFrequencyEnum (listed above)

Endpoint 2

GET: /apartment/details					
	Name	**Data Type**	**Optional**	**Default**	**Accepted Values**
Input	id	string	False	-	-
Output	data	ApartmentModel (listed above)	-	-	-

This endpoint gets apartment details using id in the database containing all the attributes.

Endpoint 3

POST: /apartment/create					
	Name	**Data Type**	**Optional**	**Default**	**Accepted Values**
Input	unnamed	CreateApartmentModel	-	-	-
Output	data	ApartmentModel (listed above)	-	-	-

This endpoint creates an apartment and returns the created apartment.

AddressModel	
street	string
city	string
state	string

ContactInfoModel	
phone	string
name	string
email	string

CreateApartmentModel	
title	string
description	string
numberOfBedrooms	number
numberOfBathrooms	number
sizeInSquareMeters	string
price	number
address	AddressModel
imagesUrls	string[]
contactInfo	ContactInfoModel
available	boolean
createdAt	Date
level	string
paymentOptions	PaymentOptionsEnum (listed above)
furnished	boolean
type	ApartmentTypeEnum (listed above)
amenitiesIds	string[]
listingType	ListingTypeEnum (listed above)
rentPaymentFrequency	RentPaymentFrequencyEnum (listed above)