DTT-Houses API

Vladimir Abuhnoaei

1. Introduction

Dear reader, this document will present the contents of this small project. It will be a simple 'readme' type of file and will not have any particular structure, only information about components of the API, such as description or motivation for various decisions.

2. API Components

a. External Components

The only external component used for this project was the MySQL DB Seeder. This helped with filling the database with 150 records of houses, 300 records for rooms, 150 listings, 150 house filters, and 50 users. This library takes use of the popular Faker Library and helps with not filling the database with any number of records, but generates random values for numbers, email addresses, streets, etc.

b. Internal Components

The internal structure was based on the provided user stories and description given by the DTT team in the initial PDF file. The internal structure contains components such as Models, SQL-files, Migrations and Controllers. Where suited, comments have been added to provide insight to the code's logic and purpose.

- Models & SQL – Files

The models contain the blueprints of the objects that will be used to interact with the database and show what variables will they hold. Moreover, each model contains validators for data that should come from the user (such as validators for emails – validating that the input is of email type, validators for fields to not be null or containing only values like ' ', etc.). Rooms, Houses and Users were the three main components required by the client.

Besides required properties written in the initial document, the room component contains one other property necessary for the API to function properly. *Rooms Controller* contains the added 'house_id' property, as every room is part of a house, when a GET Request for a house is called, the house will have appended a 'rooms' array containing all the rooms that have the 'house_id' of the specified house.

Moreover, there are two extra components added, 'Listings' and 'Houses_filter'. The *'Listings'* components, as the name suggests, should represent listings stored in the database. It contains properties such as, 'user_id', 'house_id', 'active', 'post_date' and 'inactive_date'. Each listing

will be created by a user, at the same time as of the creation of a house and rooms. Listings cannot be directly deleted from the database, and when a user ‘deletes’ a listing, it will only remove the house and room data from the database, and will change the listing to inactive, or ‘active = 0’ and will update the ‘inactive_date’ property. The deletion of a listing could be therefore automated in the future, for example a listing will be permanently deleted from the database after 30 days. This is done because we do not want to lose all the data from the database regarding an object when a user requests so. The ‘*Houses_filter’* component contains fields regarding room type count and total room count of a house. This is done to easier filter houses, but it is not such a reliable method, as it is only limited to the current state of the ‘House’ component. It is related to a house, as it has a ‘house_id’ property and can be called when a user wants to filter the results of listings for his own preferences.

SQL-files are created as well for easier database creation on any local machine.

- Controllers

The controllers are the middleman between the database and the visual interface. They contain ‘action’ functions, used for any requests and are responsible for all CRUD operations.

‘ListingsController’ is more specific, as when the GET request is called, it does not only return listing data, but calls a GET action for both rooms and houses controllers. This is done to retrieve the listing’s house data and the house’s room data. Afterwards, the room data is appended to the house data, and the house data is appended to the listing data.

- Migrations

Migrations are used describe the structure of the database at any stage of development.