For this assignment, you must develop a REST API for a fictitious local realtor (LIT *Realty) as well as* writing a brief report (see task 5).

The starter code for this assignment contains five Entity classes which in effect define the schema for your DB.

```
35
36      @Id
37      @GeneratedValue(strategy =GenerationType.AUTO)
38      private Long id;
39      private String street;
40      private String city;
41      private Integer listing_Num;
42      private Integer style_Id;
43      private Integer type_Id;
44      private Integer bedrooms;
45      private Float bathrooms;
46      private Integer squarefeet;
47      private String ber_Rating;
48      @Lob
49      private String description;
50      private String lotsize;
51      private Short garagesize;
52      private Integer garage_Id;
53      private Integer agent_Id;
54      private String photo;
55      private Double price;
56      @Temporal(TemporalType.DATE)
57      private Date date_Added;
58  }
```

```
22  public class Styles implements Serializable {
23
24      @Id
25      @GeneratedValue(strategy =GenerationType.AUTO)
26      private Integer style_Id;
27      private String p_Style;
28
29  }
```

```
24
25      @Id
26      @GeneratedValue(strategy = GenerationType.IDENTITY)
27      private Integer type_Id;
28      private String p_Type;
29
30
31  }
```

```
27  public class Garagetypes implements Serializable {
28
29      @Id
30      @GeneratedValue(strategy =GenerationType.AUTO)
31      private Long garage_Id;
32      private String g_Type;
33  }
```

```java
public class Agents implements Serializable {


    @Id
    @GeneratedValue(strategy =GenerationType.AUTO)
    private Long agent_Id;
    private String name;
    private String phone;
    private String fax;
    private String email;
    private String username;
    private String password;

}
```

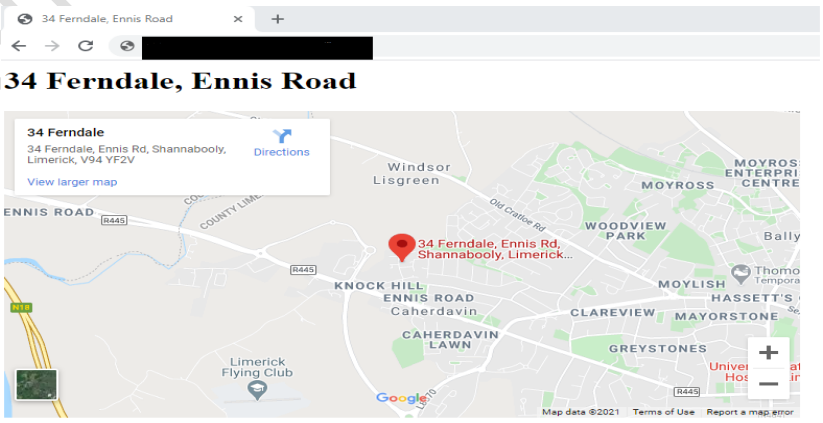
There is also a large number of images and a DB script for this assignment.

1. You must expose **each of the tables in the *realty* database with a suitable API** and you have a free hand regarding the functionality your API offers. However, the following is **required** functionality for the **properties** table where you must support the GET ᵗ, POST, PUT and DELETE HTTP methods. All responses from these HTTP methods should be in JSON.
    - ᵗ GET an individual property based on id.
    - ᵗ GET all properties (appropriate use of pagination is required).

    Both GET requests should adhere to HATEOAS principles:
    1. The GET an individual property should include a link to all properties.
    2. The GET all properties should include two links. One “self” link and another link that when followed will display the street, city, price and agents email address.

    You should aim to be **creative** and **innovative** around the API for the other tables and should look to avoid rehashing the functionality you develop for the properties table. Feel free to use 3rd party API’s/libraries to help you with this task as well as varying the request/response types the methods consume/produce.

2. Your API should return, for a specified property, a (Google) map with the address of the property plotted on it. The address of the property must also be displayed on the *html page* that is returned from the web service. For example:



3. Your API should return, for a specified property, a QR code, that when scanned (by your phone for example) will attempt to add a contact to your phone. The returned *image* stores contact data for the agent responsible for selling the specified property. For example, Sue Roberts is responsible for selling property #18, the following QR code is returned when this property is requested.

If you scan the above code, it will attempt to add a contact to your phone for "Sue Roberts".

I used the ZXing (pronounced Zebra Crossing) library to produce the above QR code. There are other libraries available and some even sit on top of ZXing.

4. Your API should return, for a specified property, an *image*. As part of the request, the client should be able to specify if they require a *full* or *thumbnail* image.

You must adhere to best practice when developing your REST API and it must be intuitive to use.

5. Along with your code, you must also upload a report to Moodle. This report (including a title page) is to be divided in three parts.
   - **5.1 Documentation**. You must document your API as you see fit (possibly using some 3rd party software).
   - **5.2 Self-Evaluation**. Critically appraise the strengths and weakness of the API you have developed (circa 500 words).
   - **5.3 Benchmarking**. Measure the benefits that the development of an API brings to a development project (circa 300 words).

Marks Breakdown:

Task	**Marks (sum to 300)**
1 Core API Development.	150
2 Property plotted on a Map.	20
3 QR Code containing a contact for the selling agent.	25
4 Image returned for a specified property.	15
5 API Documentation + Report.	90

Note:

1. This assignment is to be developed using **Spring Boot**.

2. You must use **MAVEN** to manage your projects dependencies.
3. You must use **the H2 Database.**
4. You must use suitable **bean validation** where necessary.
5. You must use **Tomcat** as your server/container.
6. Use appropriate measures to ensure that all **errors are handled** gracefully.
7. Your report is to be submitted in **PDF** format.
8. You are required to **commit regularly** to your repository on GitHub classroom.