Overview.

This document outlines the design of the restful api and the database persistent layers that will be developed to allow a client to create and view Earthquake data. It also discussed the mechanism by which the service provides an interface that supports the HATEOAS constraint central to the notion of a restful API

Restful API.

The Earthquakes api will consist of 4 main resources as shown below

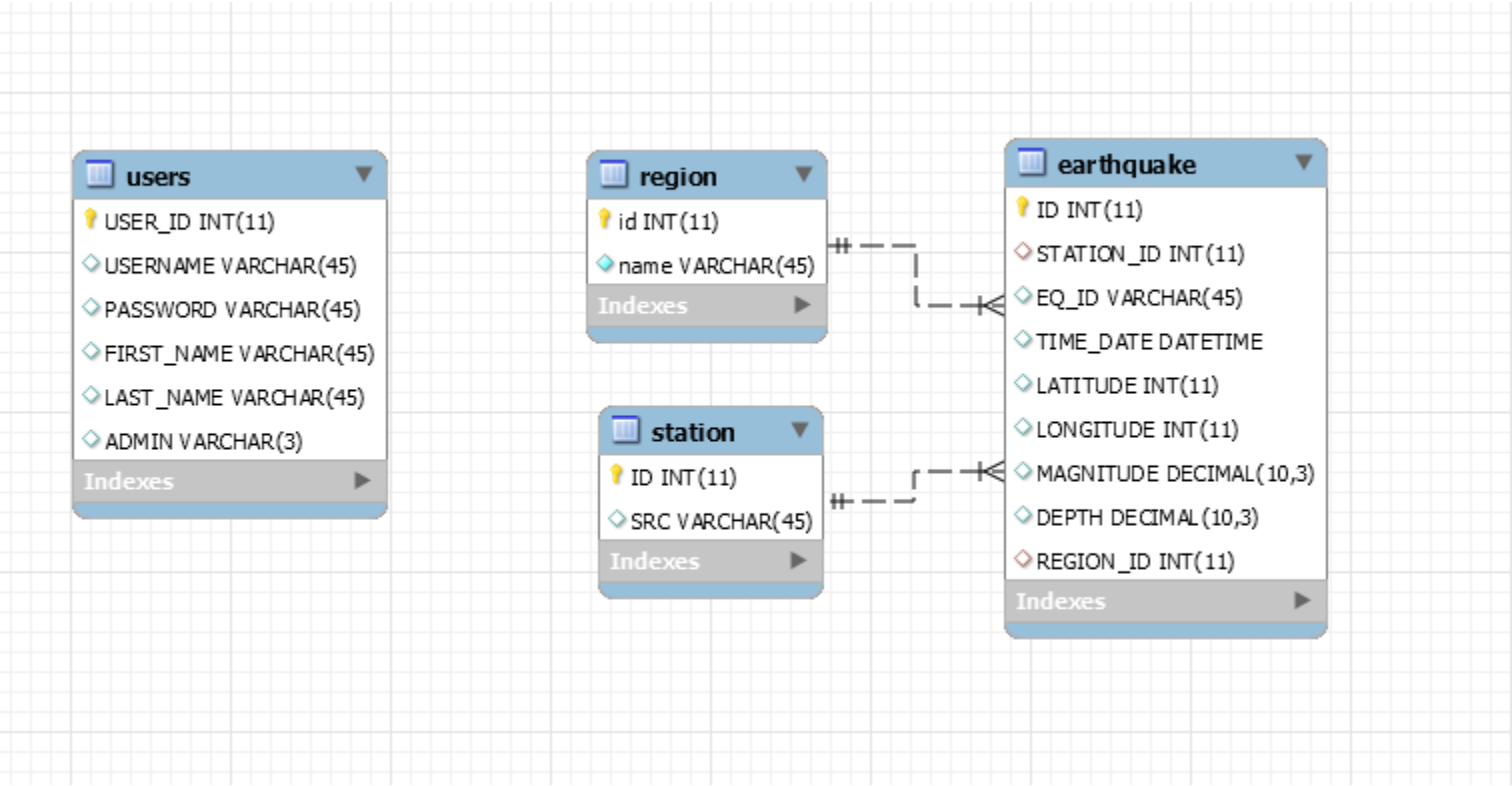
Resource	Description
Earthquakes	Collection Resource
Stations	Collection Resource
Regions	Collection Resource

A restful api will be developed to provide the following end-points to these resources

Method	Endpoint	Input	Success	Error	Description	Security
GET	/earthquakes	Body:Empty	Status 200 Body Earthquake List	Status 500	Retrieves all available Earthquakes	NONE
GET	/earthquakes/ {earthquakeid}	Body Empty	Status 200 Body Individual Earthquake data	Status 500 or 404	Retrieves individual Earthquake data for the supplied id	NONE
POST	/earthquakes	Body: Earthquake data to create	Status 200 Body Newly created Earhquake Id	Status 500 or 401 or 404	Creates new Earthquake entry	Authentic ated user only
GET	/stations	Body:Empty	Status 200 Body Station List	Status 500	Retrieves all available Stations	NONE
GET	//regions	Body:Empty	Status 200 Body Region List	Status 500	Retrieves all available Regions	NONE
All Other	All Other	N/A	N/A	Status 400	Not available	Forbidden

Persistence Layer.

The diagram below shows the database design. There will be 4 main tables as shown. An earthquake is linked to a region and a station



Hateoas

Resources will be passed in the responses via a media type of *application/json* which will contain the server provided links to dynamically discover available actions to access the resources it needs. A generic object – see class *ResponseDTO* in the code solution – will be returned for each GET request and will provide 3 fields, the DATA which is the resource itself, the URL which is the link used to retrieve the resource, and LINKS which provides other resources that are available from this point.

```
public class ResponseDTO<T> {

    private String url;
    private T data;
    private List<Link> links = new ArrayList<~>();
    public String getUrl() { return url; }

    public void setUrl(Link link) { this.url = link.getHref(); }

    public T getData() { return data; }

    public void setData(T data) { this.data = data; }

    public List<Link> getLinks() { return links; }

    public void addLink(Link link) { this.links.add(link); }

}
```

The generic nature of this class allows the Response to be used for all resources (Earthquake, Station and Region) – e.g. *ResponseDTO<EarthquakeDTO>* is returned in the response to a GET request for an individual Earthquake.

Below is an image of the response to GET request on */earthquakes* – note the 3 resources (earthquakes) have been passed in the main *data* property, and each has it's own URL


```
{
  "url": "http://localhost:8080/earthquakes",
  "data": {
    "count": 3,
    "earthquakes": [
      {
        "url": "http://localhost:8080/earthquakes/1",
        "data": {
          "src": "NZ",
          "eqid": "EQ-ID1",
          "timedate": "2012-09-17 08:47:53",
          "lat": 60,
          "lon": 88,
          "magnitude": 3.66,
          "depth": 25.789,
          "region": "Nicobar Islands, India region"
        },
        "links": [
          {
            "href": "http://localhost:8080/earthquakes"
          }
        ]
      },
      {"url": "http://localhost:8080/earthquakes/2"...},
      {"url": "http://localhost:8080/earthquakes/5"...}
    ]
  },
  "links": []
}
```

Security

Two options have been considered for the security, which is only being applied to the POST method which creates a new earthquake

1. **Basic Authentication**

   The client must provide a user/password which the earthquake service accepts. This has been implemented with spring's @Configuration and @EnableWebSecurity annotations and an implementation of o*rg.springframework.security.core.userdetails.UserDetailsService* - see class *EarthquakeUserDetailsService*

2. **OAUTH2**

   This has not been completed but can be achieved by building on the basic authentication users and providing an Authorisation service which provides tokens for an authenticated user to then access the resource(s). The Spring Boot framework has Configuration capability which supports OAUTH2 .