REST API Assignment

First we need to create an npm project with:
npm init
Next we need to install knex with:
npm install knex --save
And mysql2 with:
npm install mysql2
For the database part, we need to first login into mysql with:
sudo mysql -u root -p
And create a database called “rest_api” with:
CREATE DATABASE rest_api;

Once we have the database, we download the db.sql file and import the dump file into that database with:
sudo mysql -u root -p rest_api < db.sql

1. (2.5 pt) CREATE: Add an endpoint that allows to insert a city to the city table. Specify city names, country codes, districts, and population counts in a POST request. Your API response should be the created record in the database.

   Here I use the post method in order to handle post requests. Next we get the city, country code, district and population values and then store them in variables. Then we execute the createID function, which is an asynchronous function I created that returns a new ID based on the latest ID in the database. Once the execution is done we use the insert query which inserts our post data + the new ID into the database. Once the insertion query is done we run the getRecord function, which I’ll explain in more detail later, in order to send the inserted data as a response.

```
//For creating data
app.post('/create', function(req, res){
    let city = req.body.city;
    let countryCode = req.body.countryCode;
    let district = req.body.district;
    let population = req.body.population;

    createID().then(function(newID){
        knex('city').insert({
            ID: newID,
            Name: city,
            CountryCode: countryCode,
            District: district,
            Population: population
        }).then(function(response){
            getRecord(city).then(function(record){
                console.log(record);
                res.send(record);
            });
        });
    });
});
```

```
//Creates an ID based on the last record ID
async function createID(){
    let newID = knex('city').orderBy('ID', 'desc').then(function(total){
        return (total[0].ID + 1);
    });
    return newID;
}
```

2. (2.5 pt) READ: Add an endpoint that returns the details of a city (Name, Country, District, Population count) when the name of a city is passed as input to the service. Your API response should be the retrieved record from the database.

Here I used the get method which allows the user to send a get request and specify the city with the city parameter.
Next we get the city value and store it in a variable called “city” which we then use in the getRecord function.
The getRecord function is an asynchronous function that returns all the information of the specified city. I created this function because getting the specified city row is a common task. Once we have the information we send it back as a response.

```
//For reading data
app.get('/read/:city', function(req, res){
    let city = req.params.city;

    getRecord(city).then(function(r){
        res.send(r);
    });
});
```

In the getRecord function we basically do the select query to the city table and return the row where the name is equal to the specified city.

```
//Returns the row of the city we want
async function getRecord(city){
    let result = knex('city').where('Name', city).then(function(data){
        return data;
    });
    return result;
};
```

3. (2.5 pt) UPDATE: Add an endpoint that updates the population count of a city stored in the database. Your API response should be the modified database record.

Here I used the get method which allows the user to send a get request and specify the city with the city parameter and the population with the population parameter.
Next we get the city value and store it in a variable called "city" and the population value which we store in the "population" variable.
Then we use the update query in order to update the population of the specified city.
If the query has been executed we then call the getRecord function in order to send the city information back as a response.

```
//For updating the data
app.get('/updatePopulation/:city/:population', function(req, res){
    let city = req.params.city;
    let population = req.params.population;

    knex('city').update({Population: population}).where('Name', city).then(function(data){
        getRecord(city).then(function(r){
            res.send(r);
        });
    });
});
```

4. (2.5 pt) DELETE: Add an endpoint that deletes a city from the database. Your API response should be a boolean: true if the record was successfully deleted or false otherwise.

Here I used the get method which allows the user to send a get request and specify the city, which we want to delete, with the city parameter.
Next we get the city value and store it in a variable called "city".
Then we use the delete query in order to delete the specified city. If the query has been executed we then check whether the response is a 0 or 1, if it's a 1 we return true, else false.
The boolean is stored in the "result" variable, which we then send back as a response.

```
//For removing the data
app.get('/delete/:city', function(req, res){
    let city = req.params.city;

    knex('city').where('Name', city).del().then(function(data){
        let result = (data == 1)? true: false;
        res.send(result);
    });
});
```