# Xander

Certified
B Corporation™

# ASSESSMENT DOCUMENTATION

API Project

Salvatore Amaddio
02/04/2024

# Contents

# 1.0. INTRODUCTION

The assessment is a back-end-only Python application by Flask that runs on the web. The application interrogates a local MySQL database through API endpoints, which allow data fetching and all CRUD operations. To protect the endpoints from unauthorised access, a JSON Web Token system has been implemented. The session's timeout is 30 minutes.

However, the web token is refreshed at every request to guarantee the use of the application without interruptions. The application also handles error inputs that the User might run into, preventing the application from crashing. In case of human error, the User can remake a request by providing the right inputs.

Requests can be made through the website Postman. However, to perform some requests, the desktop version of the service has to be installed. Screenshots of examples will illustrate the application's functionalities and how human errors are handled.
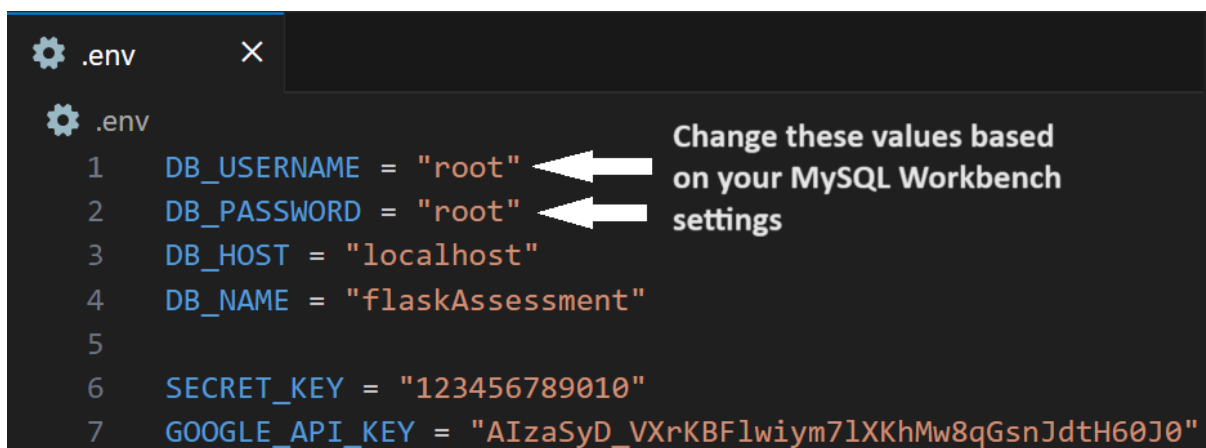
A database is created on the application startup, and some data are uploaded. This process is performed only once, meaning data will not be overwritten. The database has two tables joined by a One to One relationship. The table Student comes with a sample dataset whilst the Address table is empty. An additional table, User, hosts information about users in order to perform login operations. A default user is added with the following credentials:

> **email:** **guest@xandertalent.com**
>
> **password:** Welcome_to_this_assessment01

The project folder also comes with a .env file, which contains sensitive information. This file is the root of the project directory. Before launching the application, two pieces of data used to build the connection string to the database might need to be changed. Please refer to Figure 1 below as an example.

*Figure 1*



```
.env                ×

.env
1    DB_USERNAME = "root"        Change these values based
2    DB_PASSWORD = "root"        on your MySQL Workbench
3    DB_HOST = "localhost"       settings
4    DB_NAME = "flaskAssessment"
5
6    SECRET_KEY = "123456789010"
7    GOOGLE_API_KEY = "AIzaSyD_VXrKBFlwiym7lXKhMw8qGsnJdtH60J0"
```

Finally, the application provides an additional endpoint to interact with Google's Geocoding API. By passing the ***StudentID***, the application can interrogate Google's API using latitude and longitude as parameters. This will return an address, which will be inserted into the address table. Google's API documentation can be viewed and studied at the following link.

# 2.0. API DOCUMENTATION

The application defines four routes:

- Student
- Address
- User
- Geocoding

Each allows you to perform a set of requests. Student and Address share a similar code base. To improve code reusability, those two classes inherit from a custom abstract class named **AbstractRoute**. User allows you to perform two POST requests, one for login operations and one for registering a new user. Finally, Geocoding allows a POST request only. Further information about these APIs is provided below.

## 2.1. USER API:

This API can be used through the following link: http://127.0.0.1:8080/api/user/. Depending on your localhost settings, you might need to change http://127.0.0.1:8080/ to your localhost address. The API has two endpoints to perform two POST requests.

### 2.1.1. POST - LOGIN:

By providing an email and password, the User can perform a POST request through http://127.0.0.1:8080/api/user/login. The body to pass in the request should look as follows:

```
{
    "email": "guest@xandertalent.com",
    "password": "Welcome_to_this_assessment01"
}
```

```
1  {
2      "email":"guest@xandertalent.com",
3      "password": "Welcome_to_this_assessment01"
4  }
```

200 OK   286 ms   695 B   Save as example

```
1  {
2      "msg": "Login successful",
3      "status": 201
4  }
```

## 2.1.2. POST - REGISTER:

If you want to add one more User, you can use the
http://127.0.0.1:8080/api/user/register. The body to pass in the request should look as
follows:

```
{
    "email":"salvatore@gmail.com",
    "password": "ciao_salvo"
}
```



http://127.0.0.1:5000/api/student/1

```
1  {
2      "email":"salvatore@gmail.com",
3      "password": "ciao_salvo"
4  }
```

200 OK   218 ms   698 B   Save as example

```
1  {
2      "msg": "Registration Successful",
3      "status": 201
4  }
```

This request will add a new user to the database, as shown below:
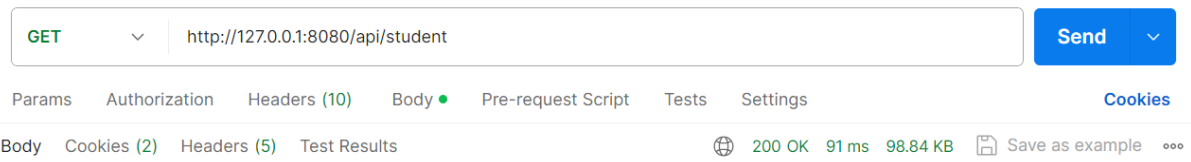
| id | email | password |
|---|---|---|
| 1 | guest@xandertalent.com | scrypt:32768:8:1$7cNhEuKYXsj1IaOf$e8be396141fffca4bdfaa18... |
| 2 | salvatore@gmail.com | scrypt:32768:8:1$uEzFwpMBLUHEiB6c$a784f8e31f67babadc9322... |
| NULL | NULL | NULL |

## 2.2. STUDENT API:

This API can be used through the following link: http://127.0.0.1:8080/api/student. Depending on your localhost settings, you might need to change http://127.0.0.1:8080/ to your localhost address.
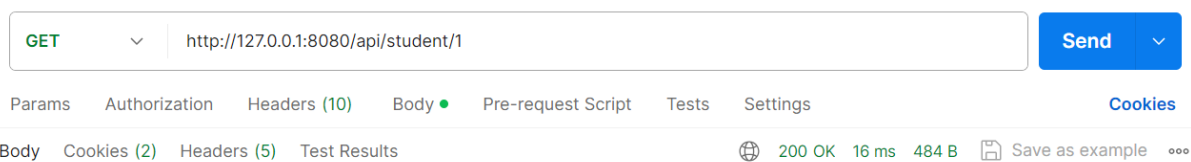
### 2.2.1. GET – ALL RECORDS:

The first request is a GET request where all students in the database can be retrieved.

| GET | ∨ | http://127.0.0.1:8080/api/student | | **Send** ∨ |
|---|---|---|---|---|

Params    Authorization    Headers (10)    Body ●    Pre-request Script    Tests    Settings    **Cookies**

Body    Cookies (2)    Headers (5)    Test Results        🌐  200 OK  91 ms  98.84 KB    💾 Save as example  ooo

### 2.2.2. GET – BY STUDENT ID:

Alternatively, you can retrieve student information based on the StudentID.

| GET | ∨ | http://127.0.0.1:8080/api/student/1 | | **Send** ∨ |
|---|---|---|---|---|

Params    Authorization    Headers (10)    Body ●    Pre-request Script    Tests    Settings    **Cookies**

Body    Cookies (2)    Headers (5)    Test Results        🌐  200 OK  16 ms  484 B    💾 Save as example  ooo

### 2.2.2. POST:

If you wish to add a new Student, you can perform a POST request at http://127.0.0.1:8080/api/student by providing the body request as follows:

```
{
        "name":"Salvatore Amaddio",
        "nationality": "Italian",
        "city": "London",
        "lat": "51.582964",
        "long":"-0.086066",
        "gender": "M",
        "age": "30",
        "english_grade": "9",
        "math_grade": "6",
        "sciences_grade": "8",
        "languages_grade": "9"
}
```

The output will be the new record inserted, for double-check purposes, and a message saying New record created. The picture below shows an example of a successful request.

```
  6          "gender": "M",
  7          "languages_grade": "9",
  8          "lat": "51.582964",
  9          "long": "-0.086066",
 10          "math_grade": "6",
 11          "name": "Salvatore Amaddio",
 12          "nationality": "Italian",
 13          "sciences_grade": "8"
 14      },
 15      "msg": "New record created",
 16      "status_code": 201
 17  }
```

### 2.2.3 PUT:
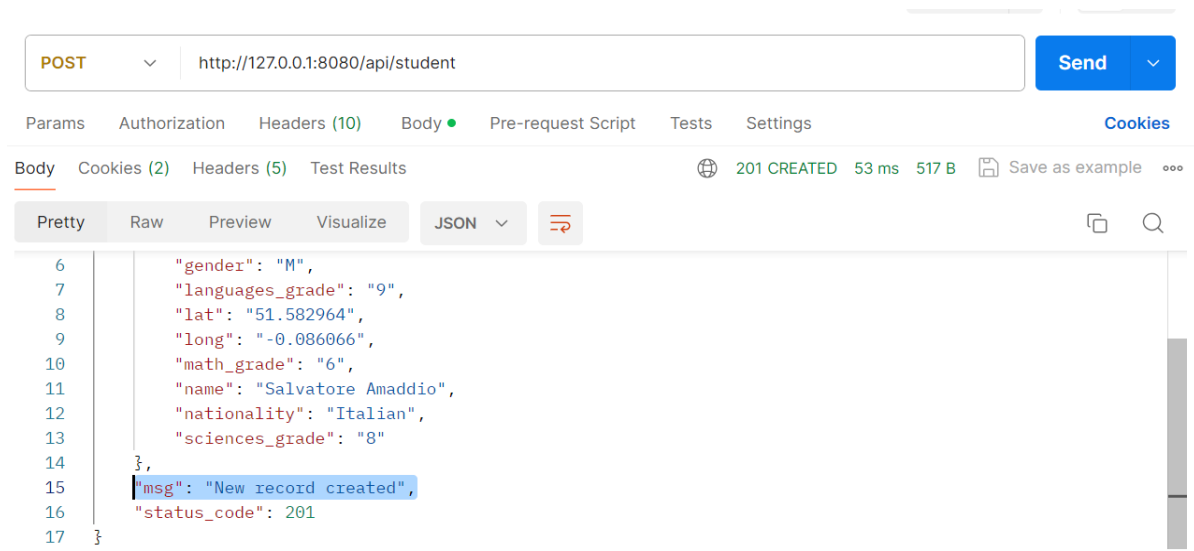
If you wish to amend a record, you can make a PUT request at
http://127.0.0.1:8080/api/student by providing the **StudentID**. You must provide all
required fields for PUT requests as in the POST request. For Example:

```
{
        "name":"Salvatore Amaddio Rivolta",
        "nationality": "Italian",
        "city": "London",
        "lat": "51.582964",
        "long":"-0.086066",
        "gender": "M",
        "age": "30",
        "english_grade": "10",
        "math_grade": "7",
        "sciences_grade": "9",
        "languages_grade": "10"
}
```
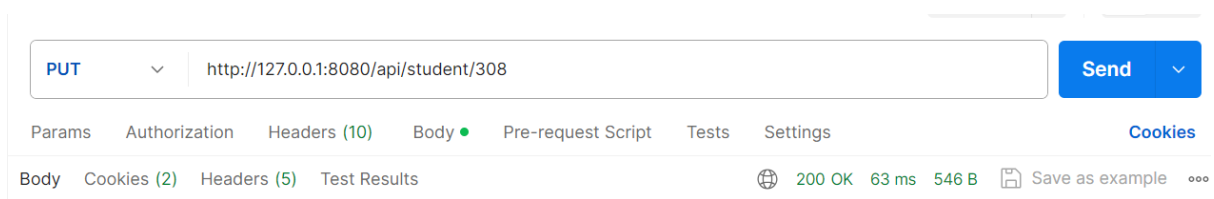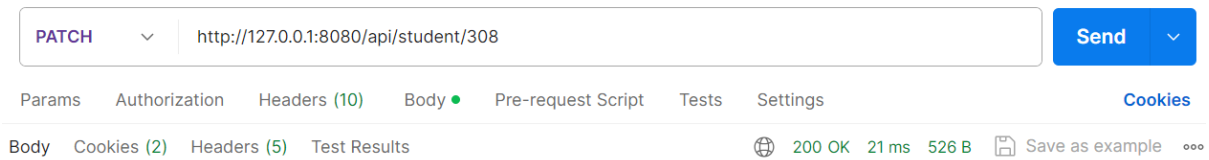
### 2.2.4 PATCH:

If you wish to amend some record attributes, you can make a PATCH request at
http://127.0.0.1:8080/api/student by providing the student ID. Unlike a PUT Request, a
Patch Request does not require all the required fields.

For Example:

```
{
    "name": "Salvo"
}
```
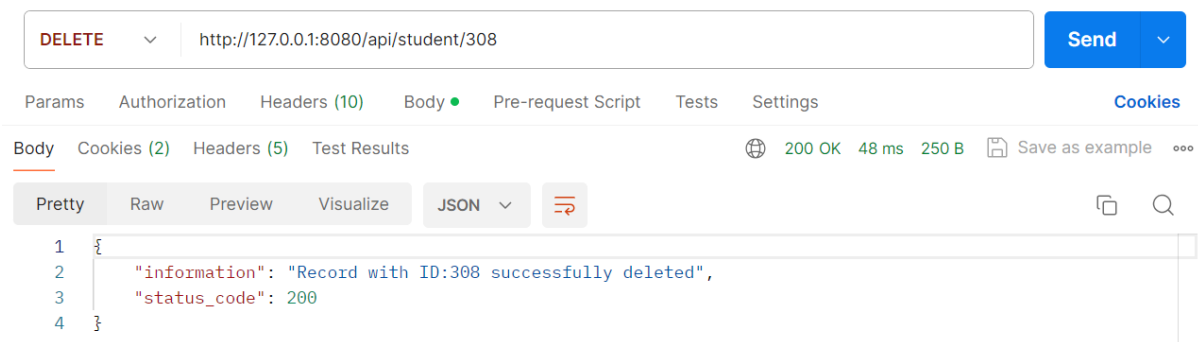
| PATCH ⌄ | http://127.0.0.1:8080/api/student/308 | **Send** ⌄ |
| --- | --- | --- |

Params   Authorization   Headers (10)   Body ●   Pre-request Script   Tests   Settings                                                 **Cookies**

Body   Cookies (2)   Headers (5)   Test Results                          🌐   200 OK   21 ms   526 B   💾 Save as example   ⚬⚬⚬

Please note that the **StudentID** should not appear in the body. The **StudentID** is an autoincremented primary key and should not be altered by the User. Attempting to do so will return a 400 error inviting the User not to include the primary key.

## 2.2.5 DELETE:

If you wish to delete a record, you can make a DELETE request at http://127.0.0.1:8080/api/student by providing the **StudentID**.

| DELETE ⌄ | http://127.0.0.1:8080/api/student/308 | **Send** ⌄ |
| --- | --- | --- |

Params   Authorization   Headers (10)   Body ●   Pre-request Script   Tests   Settings                                                 **Cookies**

Body   Cookies (2)   Headers (5)   Test Results                          🌐   200 OK   48 ms   250 B   💾 Save as example   ⚬⚬⚬

Pretty   Raw   Preview   Visualize   JSON ⌄   ⇥

```
1  {
2      "information": "Record with ID:308 successfully deleted",
3      "status_code": 200
4  }
```

## 2.3. GOOGLE GEOCODING API:

This API can be used through the following link: http://127.0.0.1:8080/api/geocoding by passing the **StudentID**. Depending on your localhost settings, you might need to change http://127.0.0.1:8080/ to your localhost address.

## 2.3.1. POST:

To find an address, you can perform a POST request by passing the **StudentID**. For Example:

```json
{
    "data": "11, None, Yarmouth Road, Toronto, Ontario, Canada - M6G 1W7",
    "msg": "New address created",
    "status_code": 201
}
```

In case the Google API fails to retrieve address information, the following message will be displayed instead:



```json
{
    "data": {
        "lat": "34.39",
        "long": "-118.54",
        "student": "StudentID: 2; Name: Joshua Lonaker; Nationality: United States of America"
    },
    "msg": "Google returned zero results. Sorry :(",
    "status_code": 200
}
```

**IMPORTANT:** If you attempt to use an ID that does not exist, the following message will be displayed:



```json
{
    "information": "Record not found, try with a different ID",
    "status_code": 404
}
```

The address table is empty by default. However, this endpoint will insert data into the Address table:

| id | student_id | number | house_name | road | city | state | country | zipcode |
|----|-----------|--------|-----------|------|------|-------|---------|---------|
| 1 | 307 | 11 | NULL | Yarmouth Road | Toronto | Ontario | Canada | M6G 1W7 |
| 2 | 300 | 1311 | NULL | South Freeway | Fort Worth | Texas | United States | 76104 |
| 3 | 3 | 917 | NULL | Harrison Street | Oakland | California | United States | 94607 |
| 4 | 4 | 21062 | NULL | Gary Drive | Castro Valley | California | United States | 94546 |
| 5 | 5 | 844 | NULL | Rua Sebastião H… | SÃ£o JosÃ© dos Campos | São Paulo | Brazil | 12210-200 |
| 6 | 6 | 180 | NULL | North Illinois Street | Indianapolis | Indiana | United States | 46204 |
| 7 | 7 | 45 | NULL | 东纬路 | Shenyang | Liao Ning Sheng | China | 110069 |
| 8 | 8 | 99 | NULL | Rua Cesário Verde | SÃ£o Paulo | São Paulo | Brazil | 02882-120 |
| 9 | 9 | 1003 | NULL | Ronquillo Street | Manila | Metro Manila | Philippines | 1008 |
| 10 | 10 | 17 | NULL | Hızır Külhanı Sokak | Istanbul | İstanbul | Türkiye | 34134 |
| 11 | 11 | 1315 | NULL | Commerce Street | Dallas | Texas | United States | 75202 |

## 2.4. ADDRESS API:

This API can be used through the following link: http://127.0.0.1:8080/api/address. Depending on your localhost settings, you might need to change http://127.0.0.1:8080/ to your localhost address.

### 2.4.1. GET:

The first request is a GET request where all addresses in the database can be retrieved.
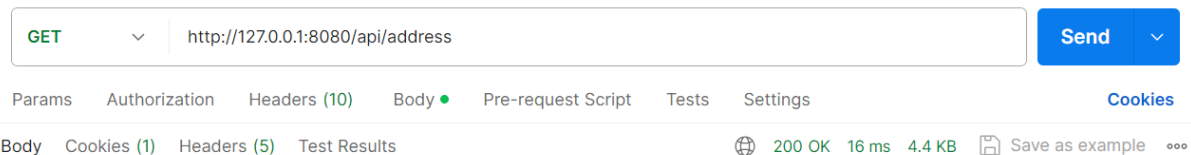


### 2.4.2. GET – BY ADDRESS ID:

Alternatively, you can retrieve address information based on the AddressID.



```json
{
    "information": {
        "city": "Toronto",
        "country": "Canada",
        "house_name": null,
        "id": 1,
        "number": "11",
        "road": "Yarmouth Road",
        "state": "Ontario",
        "student_id": 307,
        "zipcode": "M6G 1W7"
    },
```

### 2.4.3. POST:

If you wish to add a new Address, you can perform a POST request at http://127.0.0.1:8080/api/address by providing the body request as follows:

```json
{
    "student_id": "1",
    "number": "79",
    "house_name": "The Grove",
    "road": "North Grove",
    "city": "London",
    "state": "England",
    "country": "United Kingdom",
    "zipcode": "N15 5QS"
}
```

Please note that the Address table requires a Foreign Key, which is a valid **StudentID**. If you provide a non-existing **StudentID** value, the following error message will be displayed:



## 2.4.3 PUT:

If you wish to amend a record, you can make a PUT request at http://127.0.0.1:8080/api/address by providing the **AddressID**. You must provide all required fields for PUT requests as in the POST request. For Example:

```
{
    "student_id": "1",
    "number": "89",
    "house_name": "The Grove",
    "road": "North Grove",
    "city": "London",
    "state": "England",
    "country": "United Kingdom",
    "zipcode": "N15 5QS"
}
```

### 2.4.4 PATCH:

If you wish to amend some record attributes, you can make a PATCH request at http://127.0.0.1:8080/api/address by providing the *AddressID*. Unlike a PUT Request, a Patch Request does not require all the required fields. For Example:

```
{
    "number": "9"
}
```

Please note that the *AddressID* should not appear in the body. The *AddressID* is an autoincremented primary key and should not be altered by the User. Attempting to do so will return a 400 error inviting the User not to include the primary key.

### 2.4.5 DELETE:

If you wish to delete a record, you can make a DELETE request at http://127.0.0.1:8080/api/address by providing the *AddressID*.
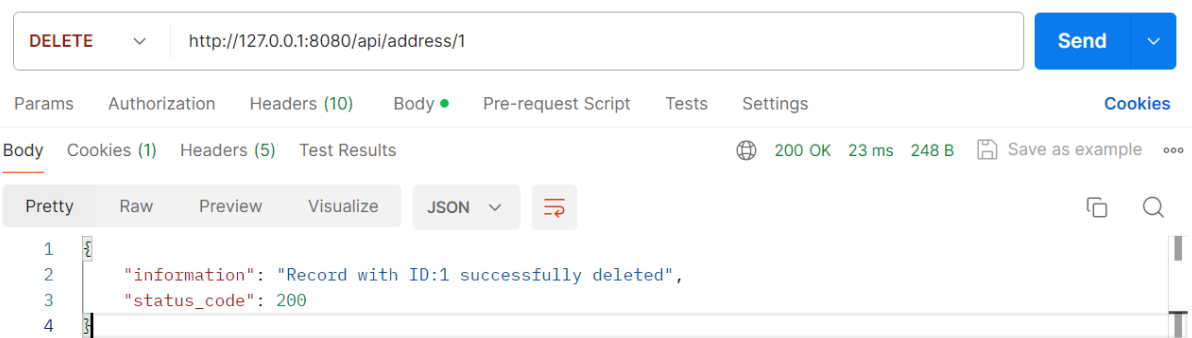


## 3.0. ERROR HANDLING:

The application handles a series of possible human error inputs, such as spelling errors or missing essential inputs.

## 3.1. Example:

The following message will be displayed if the User attempts to send a PUT request or any other request without satisfying the necessary requirements. It will also appear if the User misspells any of the required inputs.