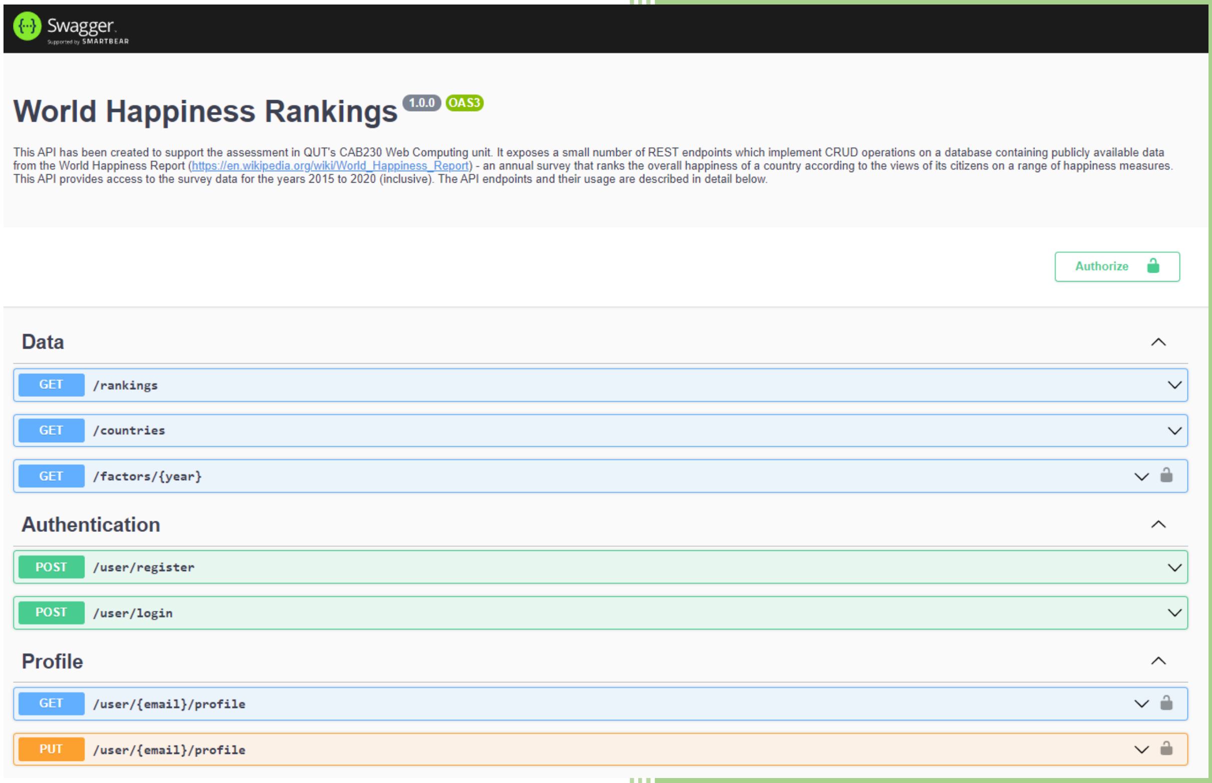
2021

CAB230 REST API – Server Side



World Happiness Rankings 1.0.0 OAS3

This API has been created to support the assessment in QUT's CAB230 Web Computing unit. It exposes a small number of REST endpoints which implement CRUD operations on a database containing publicly available data from the World Happiness Report (https://en.wikipedia.org/wiki/World_Happiness_Report) - an annual survey that ranks the overall happiness of a country according to the views of its citizens on a range of happiness measures. This API provides access to the survey data for the years 2015 to 2020 (inclusive). The API endpoints and their usage are described in detail below.

Authorize

Data

GET /rankings

GET /countries

GET /factors/{year}

Authentication

POST /user/register

POST /user/login

Profile

GET /user/{email}/profile

PUT /user/{email}/profile

CAB230 Happiness API – The Server-Side Application

Louis Yanagisawa

N10221221

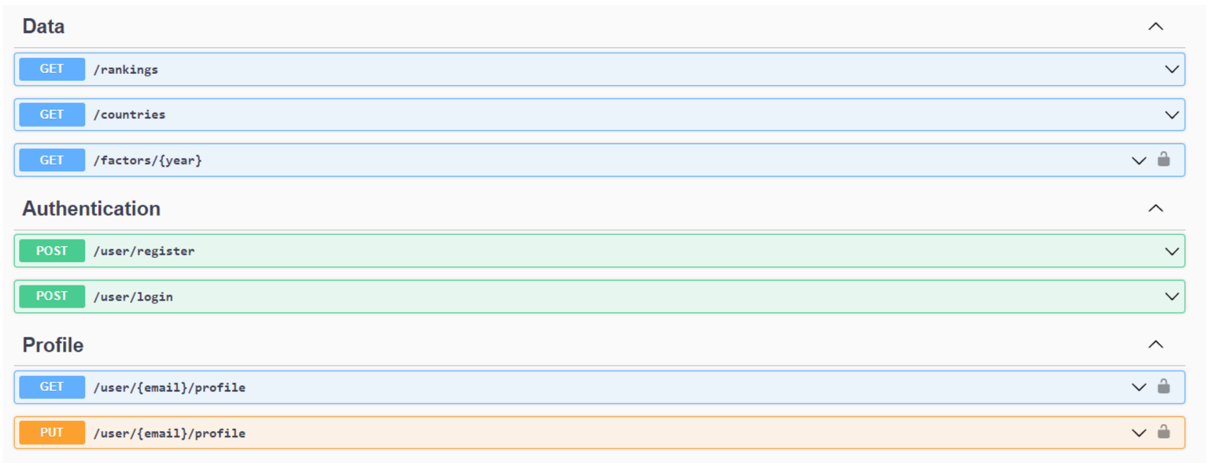
6/11/2021

Contents

Introduction

Purpose & description

The application designed is a happiness database Applicant Programming Interface (API) with the purpose of allowing users to interact with the database. The happiness database contains information about each country, including factors that all lead into an overall happiness score which can be used to rank the countries. This API allows users to retrieve data from the database, including the overall rankings, the total list of countries in the rankings and the factor scores of each country. The API also allows users to register and account and login to access the authorized end points and also allows users to set and retrieve information about the users.



Data

GET /rankings

GET /countries

GET /factors/{year}

Authentication

POST /user/register

POST /user/login

Profile

GET /user/{email}/profile

PUT /user/{email}/profile

Figure 1: Endpoints in the swagger docs

Completeness and Limitations

The tests return full marks, so all the end points are fully functional. All the endpoints return the correct status code, message, and information. Please refer to appendices for the test results from postman and refer the tests file in the submission to see the full test results.

/rankings

Fully Functional

/countries

Fully Functional

/factors/{year}

Fully Functional

/user/register

Fully Functional

/user/login

Fully Functional

/user/{email}/profile

Fully Functional

Modules used

Moment

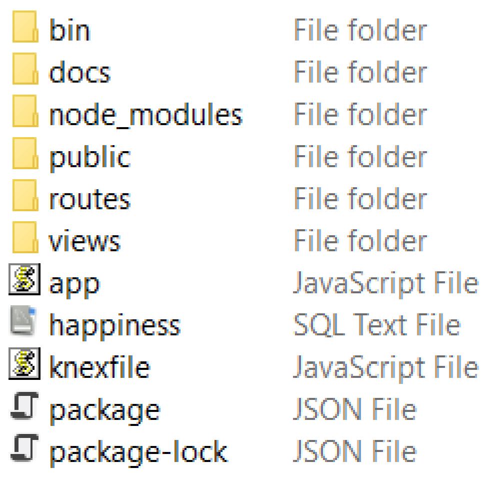
Module to provide valid date checking, especially for invalid leap years

https://momentjs.com/

Technical Description

Architecture

The application has been structured similarly to an Express-Generator app with minor details altered. Firstly, a file knexfile.js was added to the root level to allow for SQL query building and for the application to communicate with the SQL database. The happiness database SQL dump has also been added in the root level for easy access. One folder has been added and that is the docs folder which contains the swagger.JSON file which is responsible for containing the information for the swagger document page.



bin	File folder
docs	File folder
node_modules	File folder
public	File folder
routes	File folder
views	File folder
app	JavaScript File
happiness	SQL Text File
knexfile	JavaScript File
package	JSON File
package-lock	JSON File

Figure 2: App architecture

The routes folder contains the routes index and user. The index file has been organized so it contains all the information needed for all the endpoints that are not /users endpoints. This includes /rankings, /countries, /factor/{year} and also redirects the basic URL with no route to the swagger docs page. As mentioned before the user file contains all the user endpoints such as /user/register, /user/login and /user/{email}/profile.

Security

The app has used many middleware functions to bolster the security features and capabilities of the app. App.js has used knex for SQL query building so raw SQL does not have to be used to ensure injection attacks cannot occur. A knex.js file has been added to connect the app to the SQL database. App.js has also used helmet to secure the HTTP headers returned by the app. Finally, App.js has used morgan to aid in logging and the app uses the morgan logger in dev mode. The morgan logging generates logs automatically every time a request is made.

The application handles the passwords safely and securely. When handling the user's password, it is not saved directly by hashing the password using bcrypt, then saving this hash and creating a token with Jason web token and signing the email, expiry, and the secret key. Furthermore, salt rounds are used to ensure that users with the same password do not receive the same hash value. Bcrypt is also used to compare hashed passwords to further ensure that the passwords are not saved directly at any step where the password is used.

Deployment utilizes https and SSL to ensure the security of the app. A self-signed key and certificate is created for the app and the SSL key is saved secretly on the server and encrypts the content sent to users while the SSL certificate is public to users allowing anyone to decrypt the content received. Also uses more modern and secure HTTPS instead of HTTP.

Although the app implements many security features, they are all basic in their range and implementation. More can be done to improve the security abilities of the app, especially when considering the scope of attack types. The following will be a brief overview of the some of the top threats of the Open Web Application Security Project.

Injection attacks can occur anywhere there is communication with a server and a database, such as a server querying a database, and injection attacks exploit these communications by inputting queries in the request parameter to be able to use these values in a unintended way. The injection security threat is addressed by knex as knex is a SQL query builder and is used protect from injection attacks as it outputs SQL queries and bindings that are run by the database. The knex use in this function is limited and more of its functionalities could have been explored.

Broken authentication is a broad term for vulnerabilities that can be exploited to impersonate legitimate users. This can occur when attackers a database of valid username and password combinations or when session management weakness are exploited. To address this issue of session management, validation tokens are all equipped with expiration timers so the token will be unusable after a period of time to mitigate the chances of a successful attack as the opportunity window of being able to use the token is controlled and reduced. Although currently expiration times are 24 hours so the time could be reduced to reduce the opportunity of attack ever further. Furthermore, authentication has been implemented to check the JWT tokens and to check if the email of the JWT token matches the email that the user requests for some routes. To improve the authentication http cookies can be used in conjunction with JWT token storage.

Sensitive data exposure is a security risk that can arise when the server passes information to be stolen by an attacker. To mitigate the damage of such an attack all passwords on the app are encrypted and their corresponding hash is sent and saved by the database, meaning that if an attack got hold of one it would take an unreasonable amount of time to decrypt it. Furthermore, password comparisons are also done using encryption to ensure that the password itself is next used or saved in its plain form and salt rounds are used to ensure that two or more of the same passwords do not return the same hash.

Broken Access control is an exploitation of access control of a server where attackers can obtain administrative accounts or privileges. One mitigation strategy implemented in the app is the checking of parameters and queries. In most endpoints with the option of parameters and queries each are carefully checked so they are of the correct type and length of what is expected to stop the input of unexpected values. More could be done to ensure parameters and queries are only being used in their intended purpose.

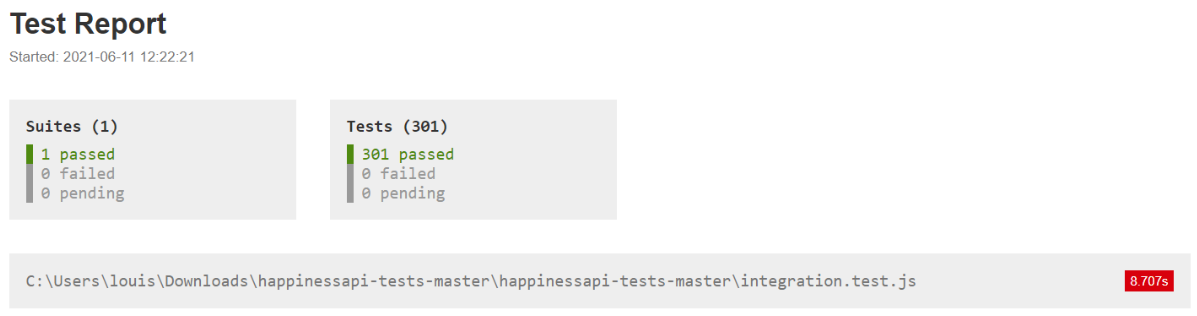
Security Misconfiguration is an attempt to gain authorized access or knowledge of a system through unchecked errors and used and unmonitored assets. The app addresses this threat by not having any unused pages and with sample routes being redirected to the swagger docs to make sure everything route is remembered and monitored.

Cross site Scripting can occur when a hacker sends a script injected version of a real application to victims. These altered applications usually have malicious scripts, and an unsuspecting victim may believe these to be real. To combat this the app uses helmet to set HTTP headers appropriately and set security related HTTP response headers. Although helmet is implemented, only its default modules are used so more modules from its library could be turned on and implemented.

Using Components with Known Vulnerabilities is an issue that commonly plagues component heavy developments where the components are misunderstood and forgotten. This negligence can lead to security issues as most components run on similar privilege levels as the app so an opening in a component can easily lead to a comprise of the entire app. One solution to this problem is to have the most recent version of the libraries in the app which the app implements. Although not used in this app but NPM audit and NPM fix can be used to resolve the issues of dependencies.

Insufficient Logging and Monitoring is as it states a security risk based on the negligence of the app. The app has implemented logging through morgan to have increased monitoring and reporting of the app. Morgan works by generating a log automatically every time a request is made. More logging could have been utilized as only morgan's dev mode in its basic form is used which may be lacking to secure insufficient logging and monitoring.

Testing



```
Test Report
Started: 2021-06-11 12:22:21

Suites (1)        Tests (301)
1 passed          301 passed
0 failed          0 failed
0 pending         0 pending

C:\Users\louis\Downloads\happinessapi-tests-master\happinessapi-tests-master\integration.test.js   8.707s
```

Figure 3: Test results

Difficulties / Exclusions / unresolved & persistent errors /

One issue that occurred was the following message: *"Error: Cannot set headers after they are sent to the client".* This error would occur in sections of the app with a then statement which would return a JSON object but could also return a JSON error. This would occur in a then statement as the return function would not exit the function but be used as a promise to be used next in the code. This was an issue as the error handling and result passing both had to occur within the then statement but would crash the app during testing. This issue was resolved when it was realized through logging that when an error was being passed as a promise by the then function, it would contain a status code while a JSON object being promised would not. To resolve this issue, the promise of the then function would be checked and if it had a status code, then it would not continue as it knows an error has been raised. If it did not have a status code, then it would continue as it knew that no errors had been raised and the item promised was a valid JSON object.

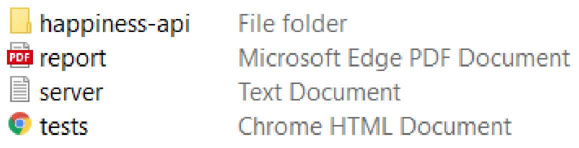
Another issue which arose was in the profile put function After the update, the revised values were not returning. To notify the user of what values have been updated, the parameters and query values that were used to update were returned, not the data returned from the database.

Extensions

Extensions of this API include adding more endpoints to increase functionality, especially to include an account delete function, and to add additional security measures to ensure safe and secure traversal of the app.

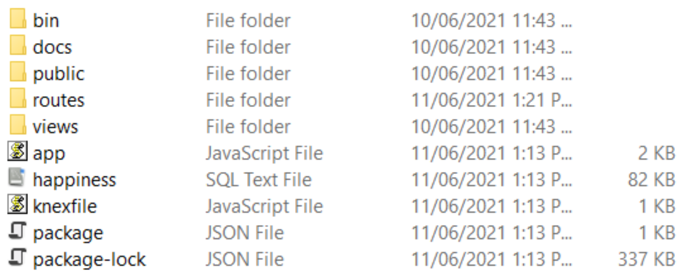
Installation guide

- Download the assign zip file
- Unzip the assign file
- Open the happiness-api folder in a terminal
- Run ‘npm install’ in the happiness-api folder
- Data for the SQL is the SQL Text File ‘happiness’ dump in the happiness-api folder
    - User may need to add a user table with columns
        - Id (INT)(Primary key, not null, unique index, auto increment)
        - Email (VARCHAR(320))(not null, unique index)
        - Hash (VARCHAR(60))(not null)
        - firstName (VARCHAR(60))
        - lastName (VARCHAR(60))
        - dob (VARCHAR(60))
        - address (VARCHAR(60))
- Run ‘npm start’ to start the server
- User needs at least MySQL version 8.0 CE and node v14.16.1 and npm.



happiness-api	File folder
report	Microsoft Edge PDF Document
server	Text Document
tests	Chrome HTML Document

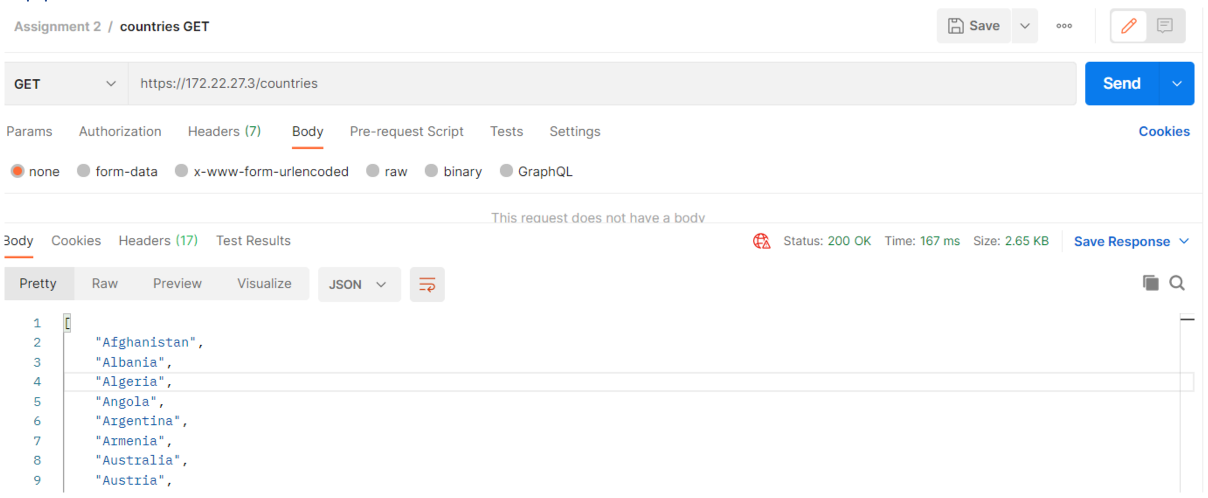
Figure 4: Assign zip file



bin	File folder	10/06/2021 11:43 ...	
docs	File folder	10/06/2021 11:43 ...	
public	File folder	10/06/2021 11:43 ...	
routes	File folder	11/06/2021 1:21 P...	
views	File folder	10/06/2021 11:43 ...	
app	JavaScript File	11/06/2021 1:13 P...	2 KB
happiness	SQL Text File	11/06/2021 1:13 P...	82 KB
knexfile	JavaScript File	11/06/2021 1:13 P...	1 KB
package	JSON File	11/06/2021 1:13 P...	1 KB
package-lock	JSON File	11/06/2021 1:13 P...	337 KB

Figure 5: happiness-api file

Appendix



GET https://172.22.27.3/countries

[✓ none] form-data x-www-form-urlencoded raw binary GraphQL

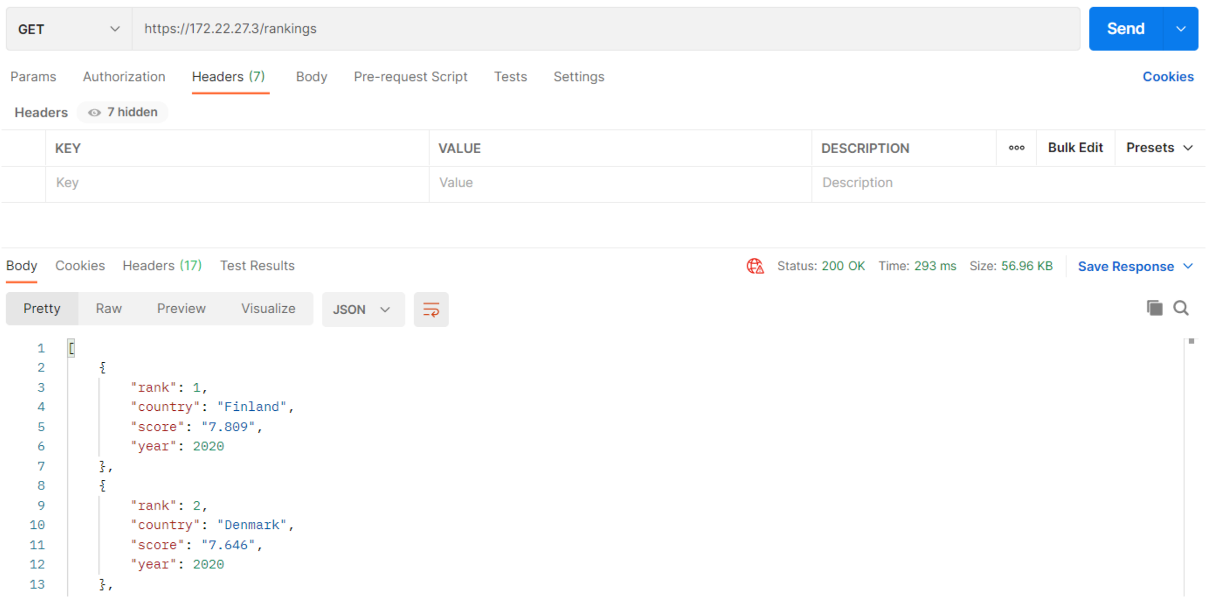
This request does not have a body

Body Cookies Headers (17) Test Results Status: 200 OK Time: 167 ms Size: 2.65 KB

JSON

```
[
    "Afghanistan",
    "Albania",
    "Algeria",
    "Angola",
    "Argentina",
    "Armenia",
    "Australia",
    "Austria",
```

Appendix 1: countries endpoint



GET https://172.22.27.3/rankings

Headers 7 hidden

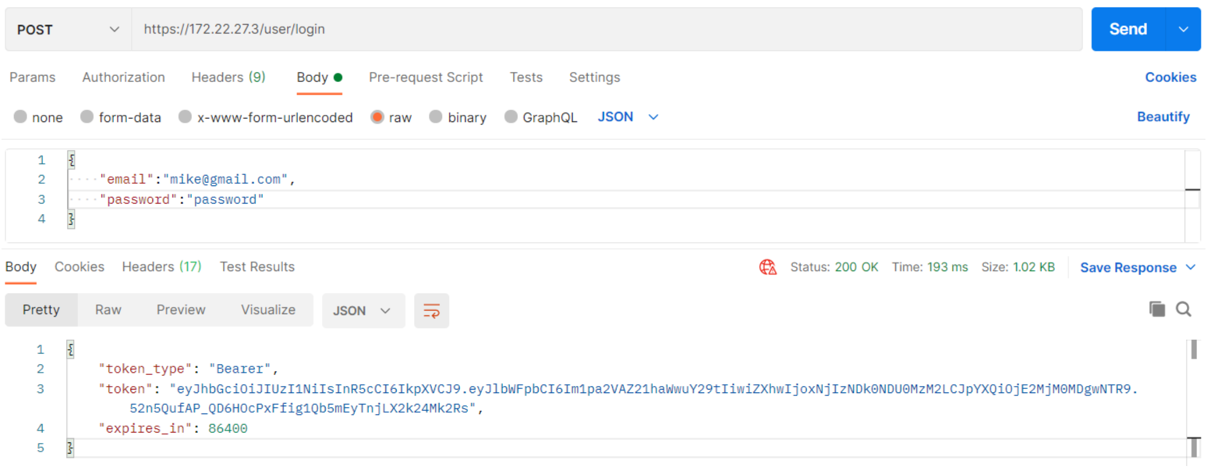
KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (17) Test Results Status: 200 OK Time: 293 ms Size: 56.96 KB

JSON

```
[
    {
        "rank": 1,
        "country": "Finland",
        "score": "7.809",
        "year": 2020
    },
    {
        "rank": 2,
        "country": "Denmark",
        "score": "7.646",
        "year": 2020
    },
```

Appendix 2: rankings endpoint



POST https://172.22.27.3/user/login

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

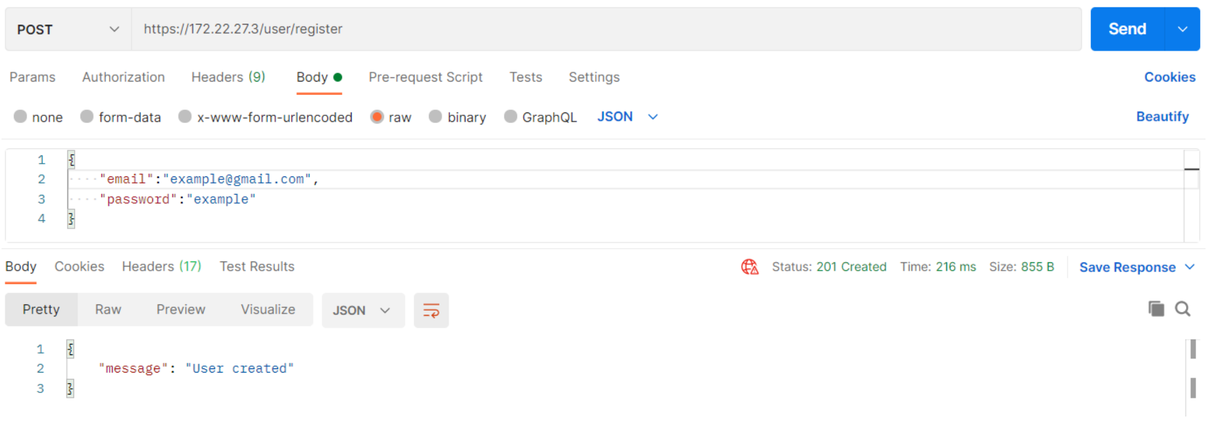
```
{
    "email":"mike@gmail.com",
    "password":"password"
}
```

Body Cookies Headers (17) Test Results Status: 200 OK Time: 193 ms Size: 1.02 KB

JSON

```
{
    "token_type": "Bearer",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Im1pa2VAZ21haWwuY29tIiwiZXhwIjoxNjIzNDk0NDU0MzM2LCJpYXQiOjE2MjM0MDgwNTR9.
        52n5QufAP_QD6HOcPxFfig1Qb5mEyTnjLX2k24Mk2Rs",
    "expires_in": 86400
}
```

Appendix 3: login endpoint



POST https://172.22.27.3/user/register

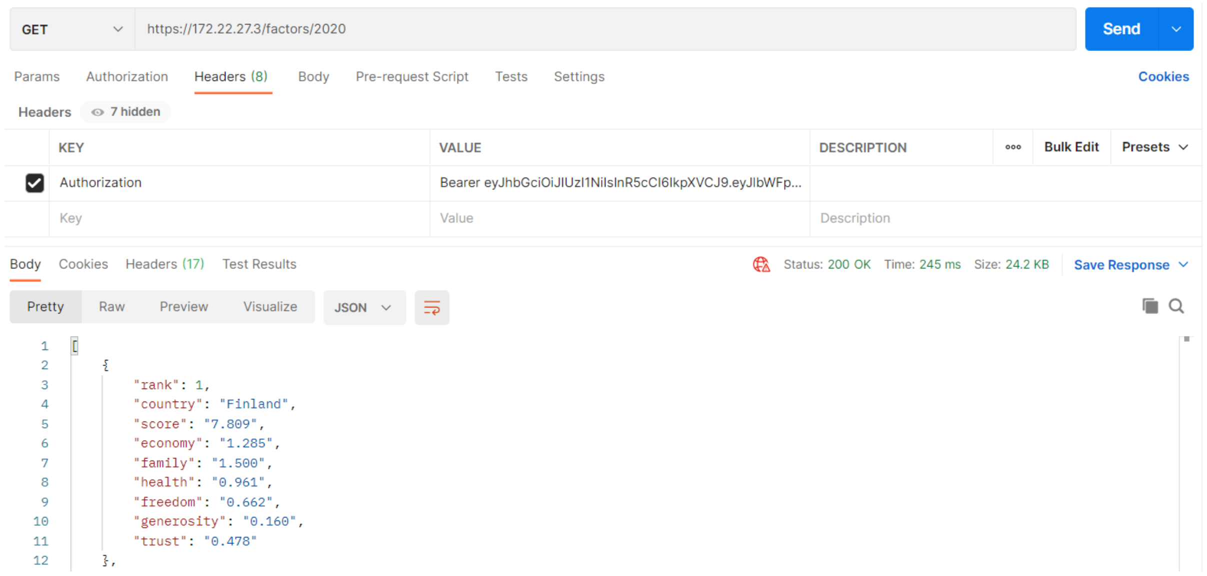
```
{
    "email":"example@gmail.com",
    "password":"example"
}
```

Status: 201 Created Time: 216 ms Size: 855 B

Body: Pretty JSON

```
{
    "message": "User created"
}
```

Appendix 4: register endpoint



GET https://172.22.27.3/factors/2020

Headers (8)

Headers 7 hidden

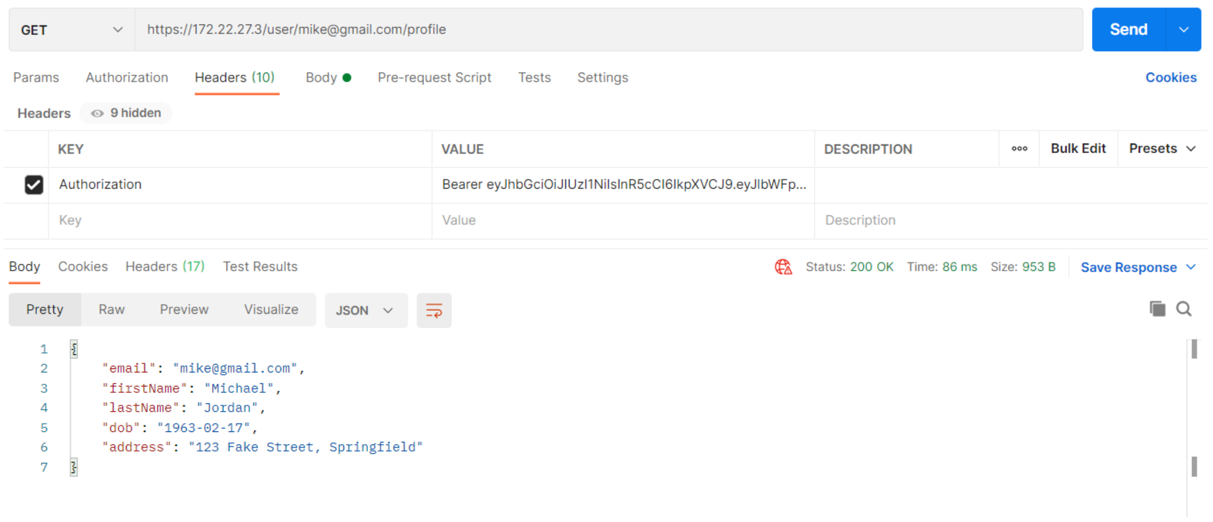
	KEY	VALUE	DESCRIPTION
✓	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFp...	
	Key	Value	Description

Status: 200 OK Time: 245 ms Size: 24.2 KB

Body: Pretty JSON

```
[
    {
        "rank": 1,
        "country": "Finland",
        "score": "7.809",
        "economy": "1.285",
        "family": "1.500",
        "health": "0.961",
        "freedom": "0.662",
        "generosity": "0.160",
        "trust": "0.478"
    },
```

Appendix 5: factors endpoint



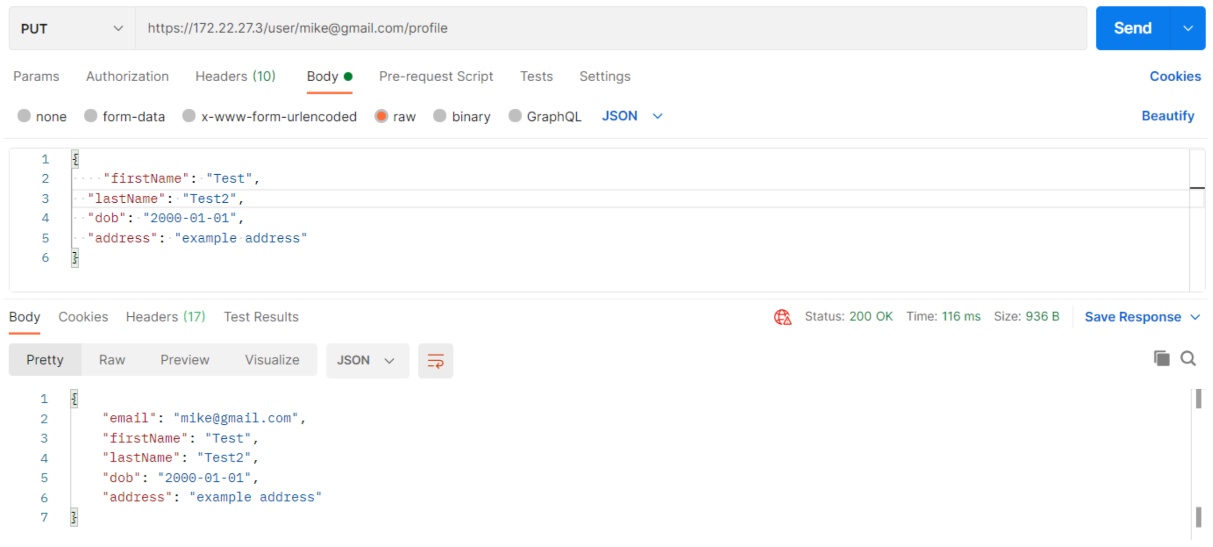
GET https://172.22.27.3/user/mike@gmail.com/profile

KEY	VALUE	DESCRIPTION
✓ Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFp...	

Status: 200 OK Time: 86 ms Size: 953 B

```
{
    "email": "mike@gmail.com",
    "firstName": "Michael",
    "lastName": "Jordan",
    "dob": "1963-02-17",
    "address": "123 Fake Street, Springfield"
}
```

Appendix 6: profile GET endpoint



PUT https://172.22.27.3/user/mike@gmail.com/profile

```
{
    "firstName": "Test",
  "lastName": "Test2",
  "dob": "2000-01-01",
  "address": "example address"
}
```

Status: 200 OK Time: 116 ms Size: 936 B

```
{
    "email": "mike@gmail.com",
    "firstName": "Test",
    "lastName": "Test2",
    "dob": "2000-01-01",
    "address": "example address"
}
```

Appendix 7: profile PUT endpoint