Web API development with OpenAPI – Exercise

M2M Communications

Dimitrios J. Vergados

June 30, 2023

Presentation outline

- Introduction
- API definition
- Create and test the server stub
- Create the client
- Implementing the prime factorization algorithm
- Bonus: Add proper exception handling

Introduction

- In this exercise we will implement a client-server application in python, using
  - OpenAPI specification for the API
  - and the swagger codegen tool for creating the client and server stubs.
- Then we will implement the client and server logic

- Assignment:
  - Create a web API server that accepts an array of numbers, and returns the prime factorization of each of the numbers sumbitted.
  - Also create a client application to test the API
- For example, the API may be accessible through a POST request, with the body having the following format:

```
{
    "input_numbers": [ 1, 4, 6, 10, 20]
}
```

  where each number is an integer smaller than `1000`.

- The response could be a JSON object with the following format:

```
{
    "result": [
        {
            "input_number": 1,
            "prime_factors": []
        },{
            "input_number": 4,
            "prime_factors": [2, 2]
        },{
            "input_number": 6,
            "prime_factors": [2, 3]
        },{
            "input_number": 10,
            "prime_factors": [2, 5]
        },{
            "input_number": 20,
            "prime_factors": [2, 2, 5]
        }
    ]
}
```

- The test program should
  - accept a list of command line arguments,
  - send an API call with these numbers,
  - and then print the prime factorization that is returned for each number.
- In case of error the API client should print error messages explaining what the problem is

API definition

- Use the swagger editor tool available at https://editor.swagger.io to write the OpenAPI document, using OpenAPI 3.0

- Use the previous presentation, as well as any other online resources to define the API that was specified above.

- Set the `servers` object to the following:

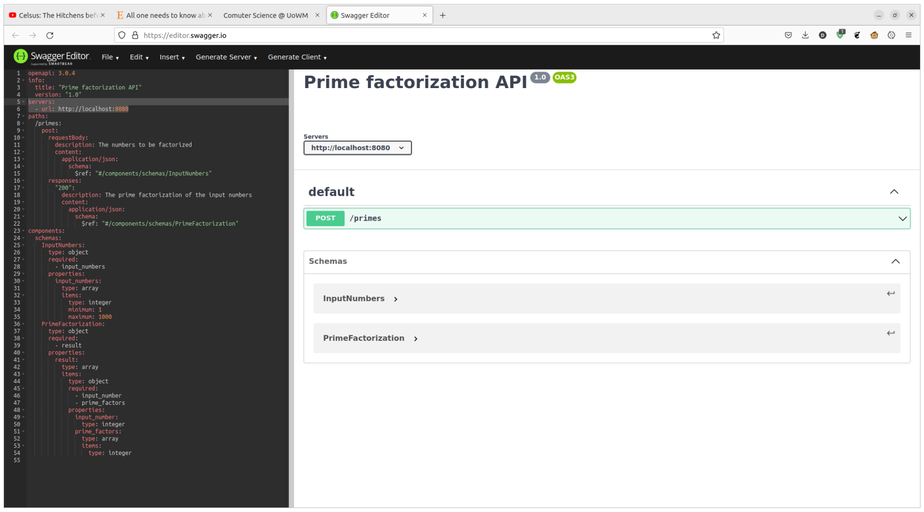
  ```
  servers:
    - url: http://localhost:8080
  ```

- Add an object definition in `components.schemas` for the request body, and for the result

- Add a path (e.g. `/primes`) for the API endpoint

Generate server and client

- Make sure that there are no errors or warnings on the top of the swagger editor preview window



```yaml
openapi: 3.0.4
info:
  title: "Prime factorization API"
  version: "1.0"
servers:
  - url: http://localhost:8080
paths:
  /primes:
    post:
      requestBody:
        description: The numbers to be factorized
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/InputNumbers"
      responses:
        "200":
          description: The prime factorization of the input numbers
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/PrimeFactorization"
components:
  schemas:
    InputNumbers:
      type: object
      required:
        - input_numbers
      properties:
        input_numbers:
          type: array
          items:
            type: integer
            minimum: 1
            maximum: 1000
    PrimeFactorization:
      type: object
      required:
        - result
      properties:
        result:
          type: array
          items:
            type: object
            required:
              - input_number
              - prime_factors
            properties:
              input_number:
                type: integer
              prime_factors:
                type: array
                items:
                  type: integer
```

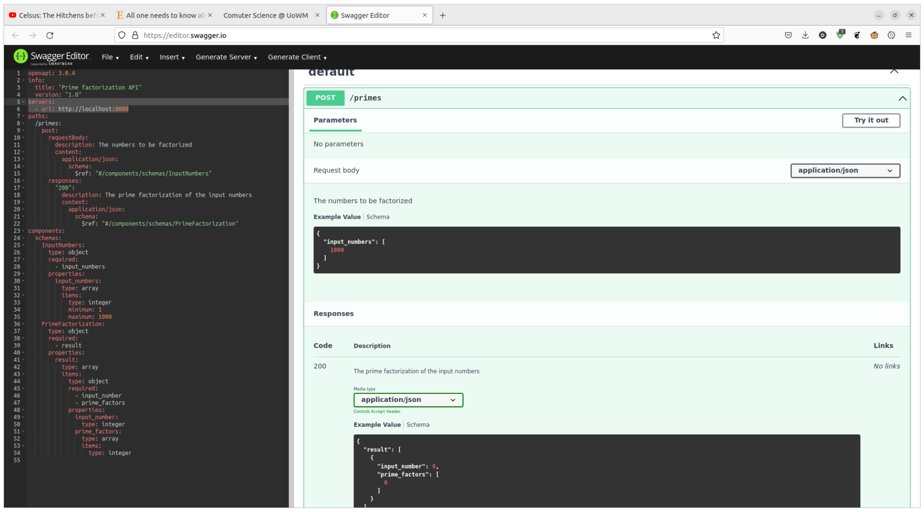
Figure 1: swagger edit screenshot

- To generate the server:
  - From the swagger editor, select `Generate Server`–>`python-flask`
- To generate the client:
  - From the swagger editor, select `Generate Client` –> `python`
- The browser will download the generated zip files in the `Downloads` directory. Move these files to a directory of your choice, that will be the working directory, e.g. in a directory with your name on the Desktop.

Create and test the server stub

Setting up the environment

- Extract the server files form file `python-flask-server-generated.zip` in a subdirectory of your working directory e.g. to `Desktop/myname/server`
- Open a terminal window and enter the directory
- Create a virtual environment for the server with



```yaml
openapi: 3.0.4
info:
  title: "Prime factorization API"
  version: "1.0"
servers:
  - url: http://localhost:8080
paths:
  /primes:
    post:
      requestBody:
        description: The numbers to be factorized
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/InputNumbers"
      responses:
        "200":
          description: The prime factorization of the input numbers
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/PrimeFactorization"
components:
  schemas:
    InputNumbers:
      type: object
      required:
        - input_numbers
      properties:
        input_numbers:
          type: array
          items:
            type: integer
            minimum: 1
            maximum: 1000
    PrimeFactorization:
      type: object
      required:
        - result
      properties:
        result:
          type: array
          items:
            type: object
            required:
              - input_number
              - prime_factors
            properties:
              input_number:
                type: integer
              prime_factors:
                type: array
                items:
                  type: integer

```

No parameters

Request body

The numbers to be factorized

```
{
  "input_numbers": [
    1000
  ]
}
```

Code	Description	Links
200	The prime factorization of the input numbers	*No links*

```
{
  "result": [
    {
      "input_number": 0,
      "prime_factors": [
        0
      ]
    }
```

Figure 2: swagger edit screenshot

```
python3 -m venv testenv
```

- load the virtual environment with

```
source testenv/bin/activate
```

Note: On windows the instructions for creating and activating the environment are different, see https://programwithus.com/learn/python/pip-virtualenv-windows

---

- Fix requriements.txt file to set specific versions for connexion and connexion[swagger-ui]: Replace lines

```
connexion >= 2.6.0
connexion[swagger-ui] >= 2.6.0
```

with

```
connexion == 2.14.2
connexion[swagger-ui] == 2.14.2
```

- Then install the server dependencies with

```
pip install -r requirements.txt
```

Run the server

- To test the server, with the virtual environment activated, run

```
FLASK_DEBUG=development python -m swagger_server
```

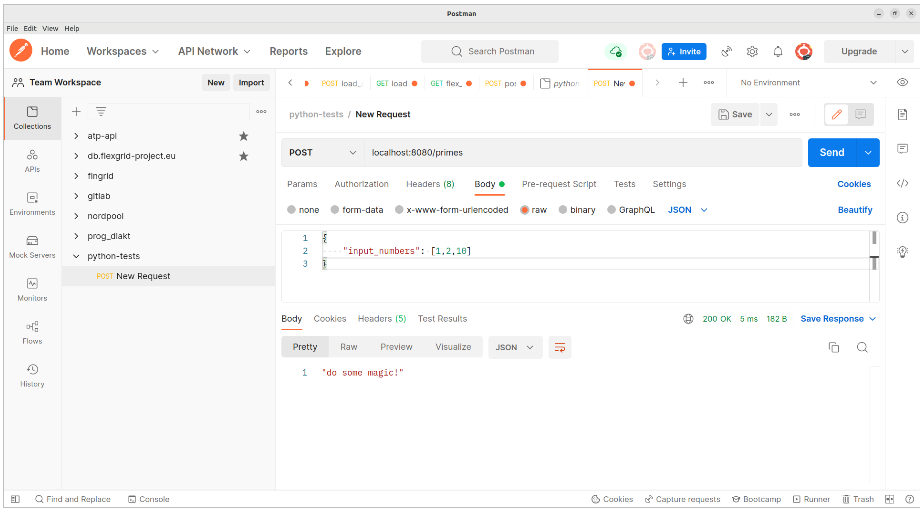
- You should see the following line (among others)

  ```
  Running on http://127.0.0.1:8080
  ```

  indicating that the server is listening on localhost on port 8080

Test the server with Postman

- Open postman, and create a request that matches the one you defined in your OpenAPI file
- Post a valid object to the API
- You should see a result like:



POST localhost:8080/primes

Body ● raw JSON

```
{
    "input_numbers": [1,2,10]
}
```

Body Pretty JSON 200 OK 5 ms 182 B

```
"do some magic!"
```

Figure 3: postman

Add placeholder response

- The server now is returning a string that is not a valid response object.
- This means that the client will throw an exception if it tried to connect to the server
- To fix this, in the server files open the controller that is responsible for the API call, under `swagger_server/controllers/`

- Find the line with

```
    return 'do some magic!'
```

- And replace it with a call that returns a valid object (for testing purposes)
- For example if the response object is named `PrimeFactorization`, the line should become:

```
return PrimeFactorization.from_dict({
    "result": [
        {
            "input_number": 1,
            "prime_factors": []
        },{
            "input_number": 4,
            "prime_factors": [2, 3]
        },{
            "input_number": 6,
            "prime_factors": [2, 3]
        },{
            "input_number": 10,
            "prime_factors": [2, 5]
        },{
            "input_number": 20,
            "prime_factors": [2, 2, 5]
        }
    ]
})
```

---

1. Don't forget to import the relevant model classes
2. Don't forget to restart your server before testing

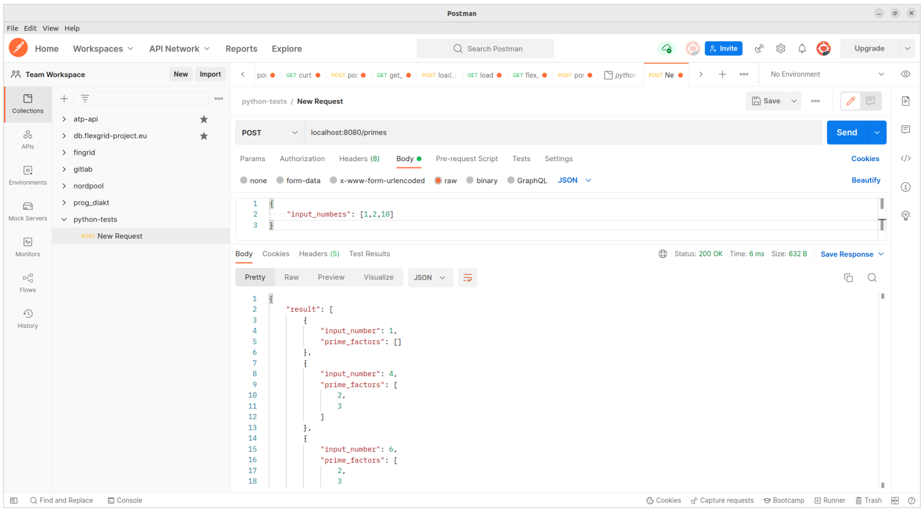
- Now the response in Postman should look like this:

We will do a proper implementation of the prime factor generating code in a bit, for now lets write the client that will connect to the API.

Create the client

Setup the environment

- Extract the client files form file `python-client-generated.zip` in a subdirectory of your working directory e.g. to `Desktop/myname/client`
- Open a terminal window and enter the directory



python-tests / New Request

POST localhost:8080/primes

Body: ○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
    "input_numbers": [1,2,10]
}
```

Body Cookies Headers (5) Test Results — Status: 200 OK Time: 6 ms Size: 632 B

```
{
    "result": [
        {
            "input_number": 1,
            "prime_factors": []
        },
        {
            "input_number": 4,
            "prime_factors": [
                2,
                3
            ]
        },
        {
            "input_number": 6,
            "prime_factors": [
                2,
                3
```

Figure 4: postman with output

- Create a virtual environment for the client with

  `python3 -m venv testenv`

- load the virtual environment with

  `source testenv/bin/activate`

- Then install the server dependencies with

  `pip install -r requirements.txt`

Note: On windows the instructions for creating and activating the environment are different, see https://programwithus.com/learn/python/pip-virtualenv-windows

---

Creating a client

- Open the `README.md` file and copy the example from the section `## Getting Started` into a new file, in the same directory, e.g. in `my_client.py`.
- Activate the environment
- Ensure that the server is running
- Run the file `my_client.py` with

  `python my_client.py`

- You should see an error message like:

  ```
  Traceback (most recent call last):
    File "/home/djvergad/lessons/swagger_excersize/solution/python-client-generated/clien
      api_instance = swagger_client.DefaultApi(swagger_client.ApiClient(configuration))
  NameError: name 'configuration' is not defined
  ```

- To fix the previous error message delete the undefined variable by replacing

  ```
  api_instance = swagger_client.DefaultApi(swagger_client.ApiClient(configuration))
  ```

  with

  ```
  api_instance = swagger_client.DefaultApi(swagger_client.ApiClient())
  ```

- Run the client again, and now you should get the following error

  ```
  python client.py
  Traceback (most recent call last):
    File "/home/djvergad/lessons/swagger_excersize/solution/python-client-generated/clien
      body = swagger_client.InputNumbers() # InputNumbers | The numbers to be factorized
    File "/home/djvergad/lessons/swagger_excersize/solution/python-client-generated/swagg
      self.input_numbers = input_numbers
    File "/home/djvergad/lessons/swagger_excersize/solution/python-client-generated/swagg
      raise ValueError("Invalid value for `input_numbers`, must not be `None`")  # noqa:
  ValueError: Invalid value for `input_numbers`, must not be `None`
  ```

- The reason for the error is the object we are transmitting to the server doesn't have the required property `input_numbers`

- Replace the line

  ```
  body = swagger_client.InputNumbers() # InputNumbers | The numbers to be factorized (opt
  ```

  with

  ```
  body = swagger_client.InputNumbers([30, 45]) # InputNumbers | The numbers to be factori
  ```

- Now the variable should not be None, and the API server should respond with the default object we created earlier:

  ```
  python client.py
  {'result': [{'input_number': 1, 'prime_factors': []},
              {'input_number': 4, 'prime_factors': [2, 3]},
              {'input_number': 6, 'prime_factors': [2, 3]},
              {'input_number': 10, 'prime_factors': [2, 5]},
              {'input_number': 20, 'prime_factors': [2, 2, 5]}]}
  ```

Read command line arguments

- In order to read the command line arguments, we first should import the `sys` library, by placing

  ```
  import sys
  ```

  near the top of the file

- Next, we will replace the command that prints the `InputNumbers` object, so that instead of static values, it will accept the numbers from the command line.

- Replace the line:

  ```
  body = swagger_client.InputNumbers([30, 45]) # InputNumbers | The numbers to be factor
  ```

  with

  ```
  body = swagger_client.InputNumbers([int(n) for n in sys.argv[1:]]) # InputNumbers | The
  pprint(body)
  ```

- Explanation:
  - `sys.argv` is the list of the command line arguments
  - with `sys.argv[1:]` we slice the list to get all the arguments from the second to the last. We exclude the first argument, because that is the name of our program.
  - Next, `int(n)` converts each argument from a string to an integer

---

Print error messages more clearly

- In order for any error messages to be displayed more clearly, add the following statement in the last line of the file, in the `except` block:

  ```
      print(json.loads(e.body)['detail'])
  ```

  while also adding `import json` in header.

The client is now ready!

Next, we will implement the prime factorization algorithm in the server, so that the server returns the correct values.

Implementing the prime factorization algorithm

Create a file with the implementation

- Create a file named `factorization.py` in directory under `swagger_server/adapters`.
- The file will implement a function for prime factorization:

```python
def prime_factors(n):
    i = 2
    factors = []

    # Add here an implementation of the prime factorization algorithm

    return factors
```

Modify the controller to call the adapter for each input value

- Replace the body of the `primes_post` function with the following:

```python
if connexion.request.is_json:
    body = InputNumbers.from_dict(connexion.request.get_json())  # noqa: E501

    print(body.input_numbers)
return PrimeFactorization.from_dict({
    "result": [{
        "input_number": n,
        "prime_factors": prime_factors(n)
    } for n in body.input_numbers]
})
```

- Don't forget to import the prime factorization method from the adapter. In the header, add:

```python
from swagger_server.adapters.factorization import prime_factors
```

Now it should be working

- Test that the client receives the correct values from the server

Bonus: Add proper exception handling

- Hints:
  - Modify the response object in the openapi definition to include an optional field for an error message. The result field should also become optional.
  - Use the `abort` function from the `flask` module: `from flask import abort`

  - Use a `try except` block in the controller method

Thank you for the attention

Questions?

dvergados@uowm.gr