MAD 1 LA WEEK 6

1. Resource

- A `Resource` in Flask-RESTful represents an endpoint in the REST API. Each class you define as a `Resource` can handle various HTTP methods (like GET, POST, PUT, DELETE) that correspond to routes.

```
from flask import Flask
from flask_restful import Api, Resource

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return {"message": "Hello, world!"}

# Adding the resource to the API
api.add_resource(HelloWorld, "/")
```

2. fields

- `fields` is a Flask-RESTful module that defines how data should be structured in the output. This is especially useful for serializing objects.Here, `resource_fields` specifies that our JSON response should contain an integer `id`, a string `name`, and an integer `age`.

```
from flask_restful import fields, marshal_with

resource_fields = {
    "id": fields.Integer,
    "name": fields.String,
    "age": fields.Integer
}
```

3. marshal_with

- **Definition:** `marshal_with` is a decorator that formats the returned data according to the structure defined in `fields`.`marshal_with` automatically applies the structure defined in `resource_fields` to the response, making sure it includes only the fields we specified.

```
class User(Resource):
    @marshal_with(resource_fields)
    def get(self):
        user = {"id": 1, "name": "Alice", "age": 25}
        return user
```

4. reqparse

- **Definition:** `reqparse` is a Flask-RESTful module used to parse and validate request data. It's helpful for managing incoming data from clients (such as form data or JSON).`reqparse` defines expected arguments and their types. In this example, if `name` is missing from the request, it will return an error with the message "Name cannot be blank!"

```
from flask_restful import reqparse

parser = reqparse.RequestParser()

parser.add_argument("name", type=str, required=True, help="Name cannot be
blank!")

parser.add_argument("age", type=int, help="Age of the user")

class User(Resource):

    def post(self):

        args = parser.parse_args()
```

```
        return {"message": "User created", "data": args}
```

5. HTTPException:

- `HTTPException` helps you create HTTP responses with specific status codes, like `404 Not Found` or `400 Bad Request`, along with a custom message.