Computação Paralela e Distribuída
2023 / 2024

Licenciatura em Engenharia Informática

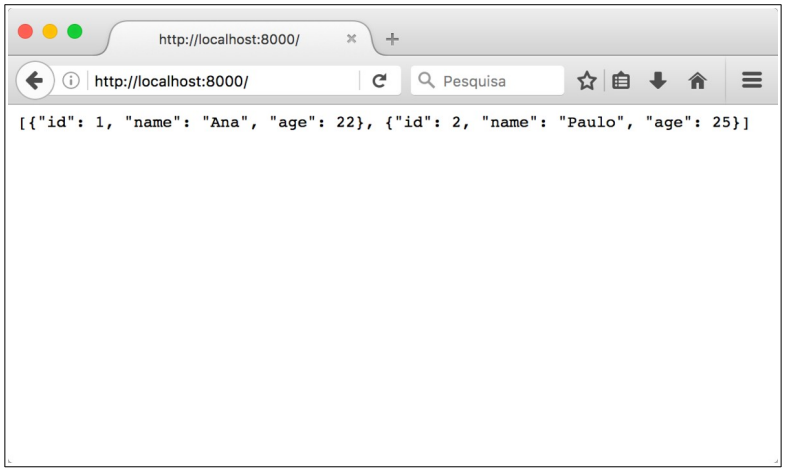
Lab. 09 – Webservices

Introdução

Nesta ficha iremos implementar um pequeno serviço *RESTful* baseado em JSON. O laboratório tem por base o ficheiro **9-webservices-base.zip** que contém a implementação inicial com o seguinte:

- **server.py:** responsável por definir a interacção com os *sockets* de rede, obter os pedidos dos clientes e enviar as respostas de volta.
- **httputils.py:** implementa funções para converter as *strings* de *http request* e *http response* para dicionários de Python, de modo a serem mais fáceis de lidar.
- **app.py:** ficheiro onde a aplicação de *webservices* deverá ser implementada. Contém inicialmente uma lista de utilizadores com identificador, nome e idade, e uma função (*controller*) que será utilizada para implementar os vários serviços disponíveis.

Crie um novo projecto e execute o código do servidor (*server.py*). O servidor estará à escuta no porto 8000 e deverá ver o seguinte conteúdo no browser.



Leia o código dos vários ficheiros, tente compreender como estão estruturados e o seu funcionamento, e verifique através das ferramentas de programador do seu *browser* que está a obter uma resposta do tipo *json*.

Nível 1

Considere a função *controller* no ficheiro *app.py*.

1. Altere a função de modo a enviar a lista de todos os utilizadores se o *url* do pedido for **/users/.** Caso contrário deverá enviar uma resposta *404 Not found*. (Sugestão: obtenha o *url* do pedido em *request[‘url’]*).

```
return {
    'status': '404 Not found',
    'headers': {
        'Content-Type': 'application/json'
    },
    'body': '{"error": "not found"}',
}
```

2. Altere a função de modo a lidar com os números de identificação dos utilizadores. Ou seja, se o *url* do pedido for igual a **/users/1/**, a função deverá retornar como resposta *json* apenas o utilizador cujo *id* é igual a 1. (Sugestão: verifique que converte o *id* passado no *url* para inteiro e procure na lista de utilizadores).

3. Modifique o processo anterior de modo a retornar um “*404 Not found”* caso o utilizador com o *id* passado pelo *url* não exista na lista de utilizadores.

Nível 2

Pode dar-se o caso que o método *controller* comece a ficar demasiado grande com muitos *returns* de dicionários para lidar com os vários casos possíveis. Há-de reparar que os dicionários a retornar são bastante semelhantes, mudando apenas o conteúdo do *body*.

Inclua as seguintes funções no ficheiro *app.py* de modo a centralizar em dois únicos sítios os vários tipos de resposta possíveis:

```
def response_ok(body):
    """Returns 200 Ok"""
    return {
        'status': '200 OK',
        'headers': {
            'Content-Type': 'application/json'
        },
        'body': json.dumps(body)
    }
```

```
def not_found(body):
    """Returns 404 Not Found"""
    return {
        'status': '404 Not Found',
        'headers': {
                'Content-Type': 'application/json'
        },
        'body': json.dumps(body)
    }
```

Altere o conteúdo da função *controller* de modo a utilizar estas funções como resposta. Como resultado, deverá ver uma simplificação do código da função *controller*. Teste novamente a aplicação e verifique se continua a obter os utilizadores como pretendido.

Note que as funções já incluem a conversão de dicionário para *json* pelo que deve passar dicionários directamente como argumentos das funções.

Nível 3

Pretende-se generalizar o *webservice* de modo a permitir adicionar utilizadores usando o método *POST*. Considere o seguinte excerto de código:

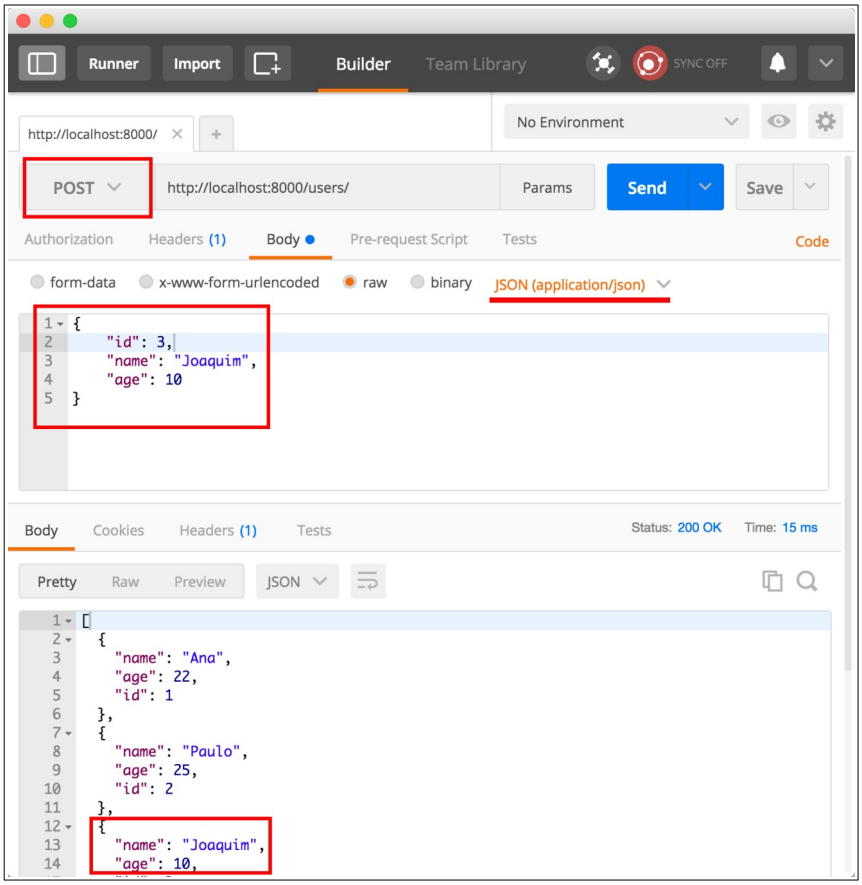
```
def controller(request):

    if request['method'] == 'GET':
        # Handle all gets from users
        ...
    elif request['method'] == 'POST':
        # Handle all posts to users
        ...
    else:
        return not_found({'error': 'Method unavailable'})
```

1. Modifique a função *controller* de modo a continuar a obter utilizadores a partir do *browser* (usando o método *GET*), testando se o *request* tem o método *GET*.

2. Modifique a função *controller* de modo a permitir inserir utilizadores quando o método http usado é o *POST*. Deverá obter o novo utilizador a partir do *body* do *request*, convertê-lo para json (usando a função *json.loads*), e adicioná-lo à lista de utilizadores existente. Por fim, deverá devolver uma resposta *json* com a nova lista de utilizadores.

Considere utilizar a aplicação *Postman*, colocando o método http como *POST* para o *url* **http://localhost:8000/users/**, colocando um utilizador em *json* no *body* da mensagem, e não esquecendo de definir que o *body* é do tipo *raw* e *application/json*.

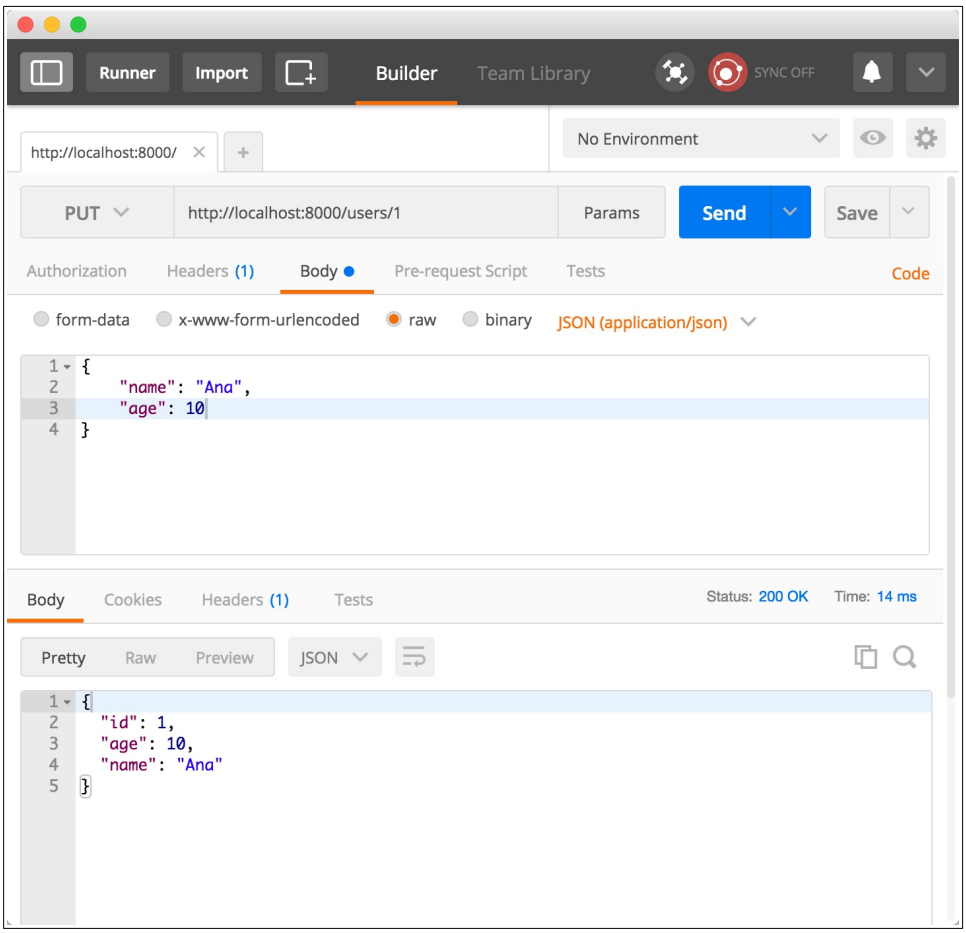


Nível 4

Neste momento é possível listar utilizadores (usando *GET*) e adicionar utilizadores (usando *POST*). No entanto pretende-se melhorar a aplicação de modo a permitir actualizar e remover utilizadores. Considerando novamente a função *controller:*

1. Modifique o código de modo a permitir remover utilizadores. Se o método http for o *DELETE*, e o *url* do *request* for da forma **http://localhost:8000/users/<id>** onde <id> será o identificador do utilizador, o serviço deverá remover o utilizador e retornar a sua informação. (Sugestão: considere utilizar a função *remove* na lista de utilizadores, de modo a remover um utilizador).

2. Modifique o código de modo a permitir alterar a informação de utilizadores usando o método http *PUT*. Utilize *urls* da forma **http://localhost:8000/users/<id>** onde <id> é o identificador do utilizador, e envie a nova informação para o utilizador como *json* no *body* do *request*. Como resultado, deverá retornar a nova informação do utilizador.



Teste as novas funcionalidades do serviço através da aplicação *Postman* e verifique que consegue remover e alterar as informações dos utilizadores. Se necessário, recorra novamente ao *browser* fazendo *GETs* ao url ***/users/*** de modo a obter a lista de todos os utilizadores.

(fim de enunciado)