

# COVID TESTS MANAGER

Programação em Ambiente Web

Grupo 39

Diogo Costa 8170455 | Luis Marques 8170485 | Luis Teixeira 8110156

## Índice

Histórico Alterações Documento	3
1. Introdução	4
1.1 Contextualização do Documento	4
1.2 Apresentação do caso de estudo	4
1.3 Objetivo	4
1.3 Ferramentas e tecnologias utilizadas	4
2. Definições, Acrónimos e Abreviaturas	5
3. Visão geral do projeto	5
3.1 Objetivo do Produto	5
4. Implementação da API REST	6
4.1 Definição	6
4.2 Definição de tabelas da base de dados	6
4.3 Operações CRUD	8
4.3.1 Users	9
4.3.2 Testes	15
4.4 Documentação	21
5. Conclusão	22
6. GitHub	23
7. Bibliografia	24

## Índice de Figuras

Figura 1 Logo MongoDB	6
Figura 2 Tabela users com Password encriptada	7
Figura 3 Postman	8
Figura 4 registar utilizador (post)	9
Figura 5 Listar utilizadores	10
Figura 6 listar utilizador pelo id (get)	11
Figura 7 atualizar utilizador pelo id (put)	12
Figura 8 remover utilizador pelo id (delete)	13
Figura 9 login (post)	14
Figura 10 Middleware validateuser.js	15
Figura 11 Inserir Teste (post)	16
Figura 12 Listar Teste (get)	17
Figura 13 listar teste pelo id (get)	18
Figura 14 atualizar teste pelo id (put)	19
Figura 15 Remover teste pelo id (delete)	20
Figura 16 Logo swagger	21
Figura 17 Swagger covid tests manager api	21

## Histórico Alterações Documento

Date	Description	Version
02/05/2020	<ul style="list-style-type: none"><li>Contextualização do Documento</li><li>Apresentação do Caso de Estudo</li><li>Definições, Acrónimos e Abreviaturas</li></ul>	1.0
14/05/2020	<ul style="list-style-type: none"><li>Operações CRUD</li></ul>	1.1

## 1. Introdução

---

### 1.1 Contextualização do Documento

Este documento descreve todo o projeto desenvolvido no âmbito da unidade curricular de Programação em Ambiente Web.

### 1.2 Apresentação do caso de estudo

Devido ao panorama atual e à crescente necessidade de dar resposta a um elevado número de pedidos de testes de despiste à COVID.19, é necessário montar um novo centro de análises regional para a realização dos testes. Este documento descreve todo trabalho realizado, durante o desenvolvimento de uma plataforma web que dará suporte ao novo centro de análises.

### 1.3 Objetivo

O trabalho descrito neste documento, tem como objetivo o desenvolvimento de uma plataforma web, para processamento de pedidos de teste de diagnóstico, agendamento de testes e registo de resultados de um centro de testes despiste e imunização à Covid-19. Para isso, será necessário ir de encontro a uma série de requisitos, que serão descritos numa secção posterior deste documento.

### 1.3 Ferramentas e tecnologias utilizadas

No processo de desenvolvimento do projeto, recorreremos às seguintes ferramentas:

- NodeJS e a framework ExpressJS
- Postman
- MongoDB
- Angular
- VSCode
- Git e Github/Gitlab

## 2. Definições, Acrónimos e Abreviaturas

---

*REST* - Representational State Transfer

*API* - Application Programming Interface

## 3. Visão geral do projeto

---

### 3.1 Objetivo do Produto

A aplicação web desenvolvida neste trabalho, foi idealizada para agilizar o processo de pedidos de teste de diagnóstico, agendamento de testes e registo do histórico de cada paciente testado no centro de análises.

No pedido de teste de diagnóstico, o utilizador deverá indicar se foi encaminhado pela linha Saúde24, se pertence a um grupo de risco ou se trabalha em locais de risco.

Após a realização do teste, a ficha do utilizador será alterada para ‘teste realizado’. Quando obtidos os resultados, será possível registar o resultado clínico na ficha do pedido do utilizador anexando um ficheiro (pdf) com os resultados clínicos e adicionando o resultado ao pedido.

Por defeito, todos os pacientes estarão classificados como ‘suspeito’, e consoante os resultados de testes forem inseridos no histórico do paciente, a classificação será atualizada.

## 4. Implementação da API REST

---

### 4.1 Definição

API (Application Program Interface): é um conjunto de funções e procedimentos que permitem a criação de aplicações que acedem a recursos ou dados de um sistema, aplicação ou outro serviço REST

API: definição de apis baseadas no protocolo HTTP universal.

Os métodos HTTP mais usados em serviços REST, são GET, POST, PUT, DELETE e que predefinem o CRUD em HTTP.

Partindo do que foi descrito anteriormente elaboramos então o CRUD da nossa API para responder às nossas necessidades.

### 4.2 Definição de tabelas da base de dados

A primeira abordagem foi delinear as tabelas que necessitávamos para responder ao nosso problema.

Para tal, utilizamos a ferramenta de base de dados abordada na aula, nomeadamente o MongoDB.



FIGURA 1 LOGO MONGODB

Possuímos duas tabelas na nossa base de dados.

A tabela “users” constituída pelos campos: xxxxxxxxx.

A tabela “testes” constituída pelos campos:

A password é guardada na base de dados encriptada tornando assim a nossa plataforma mais segura.

Para isso, instalamos o módulo bcryptjs fornecido pelos packages do npm.

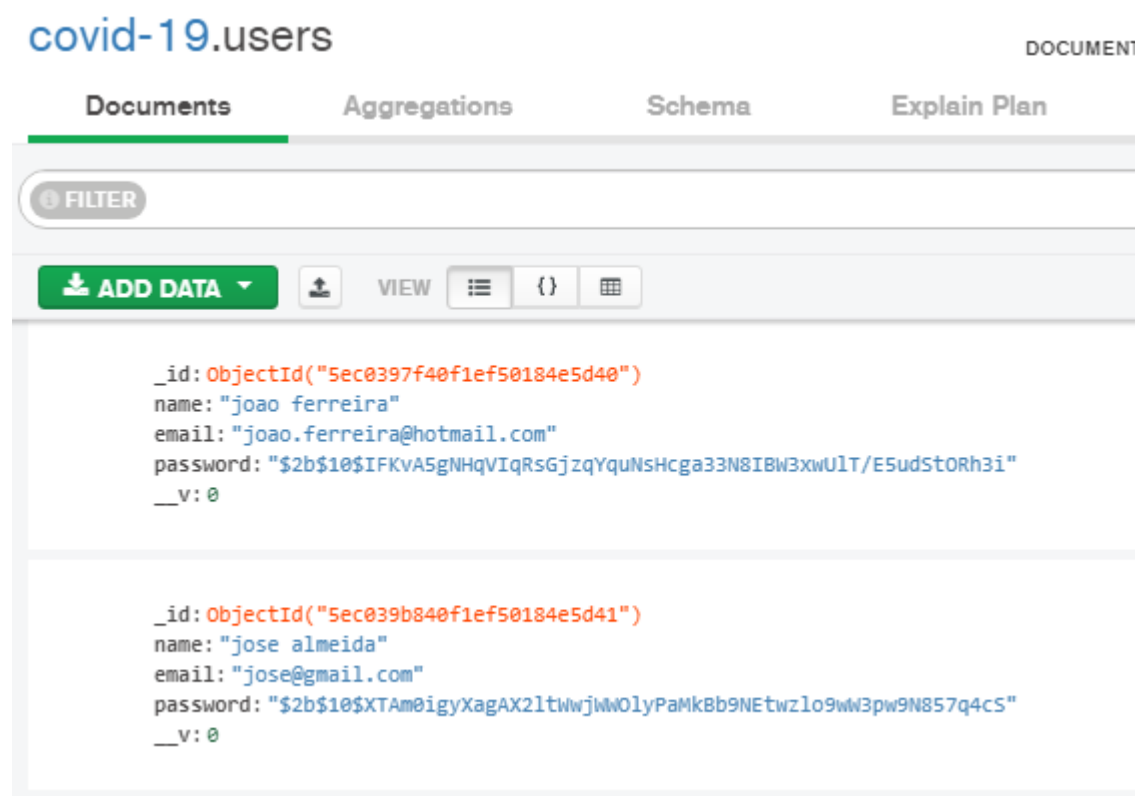


FIGURA 2 TABELA USERS COM PASSWORD ENCRYPTADA

Estas tabelas foram implementadas na pasta "models" sendo essa pasta por padrão onde ficam implementadas as tabelas.



### 4.3 Operações CRUD

Depois de já definidos os “models” elaboramos as operações CRUD ( Create, Read ,Update and Delete) dos “Users” e “Tests”.

Todas estas operações foram inseridas na pasta “controllers” onde se encontram todas as ações do controlador.

Posteriormente, para testar os nossos pedidos REST utilizamos o Postman sendo uma ferramenta bastante vantajosa, que permite:

- Definir parâmetros na área Header e Body dos pedidos HTTP;
- Verificar as respostas do servidor ;
- Criar projetos para teste e validação de APIs.

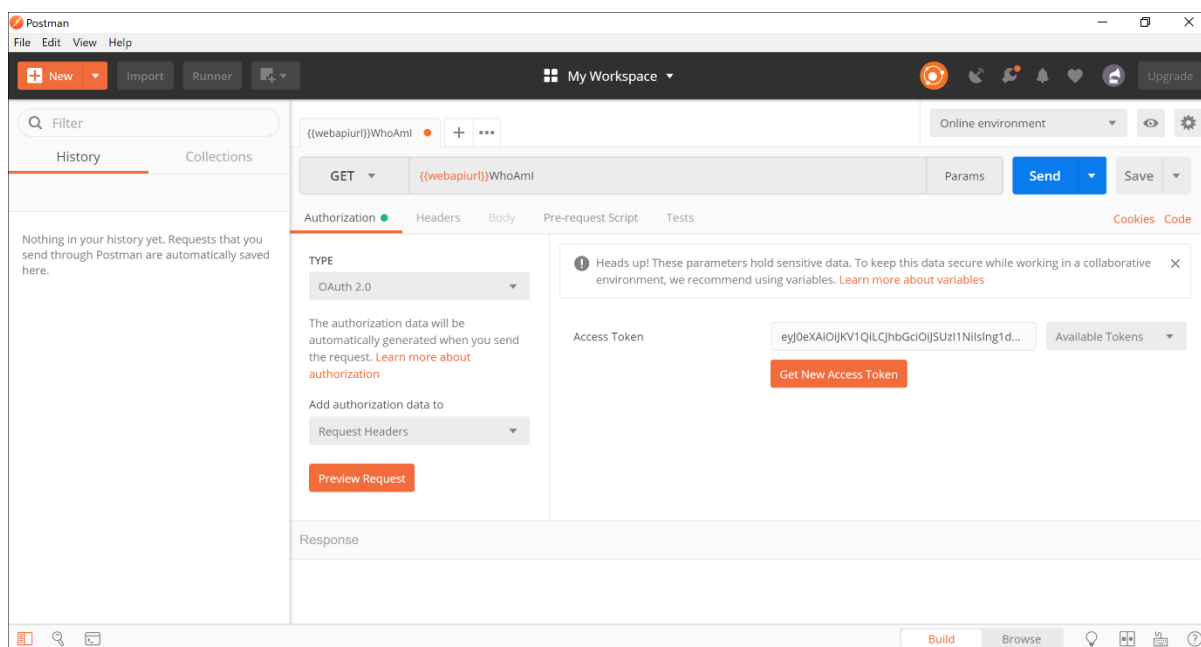


FIGURA 3 POSTMAN

## 4.3.1 Users

### 4.3.1.1 Registar Utilizador (POST)

Neste pedido é feito o registo do utilizador, ou seja, após este pedido o utilizador é inserido na base de dados.

The screenshot displays a REST client interface for a POST request named "User Register". The request is sent to the endpoint `localhost:3000/api/v1/users/register` using the `x-www-form-urlencoded` content type. The request body contains the following data:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> name	joao	
<input checked="" type="checkbox"/> email	joao@hotmail.com	
<input checked="" type="checkbox"/> password	joaozinho123	
Key	Value	Description

The response status is `200 OK` with a time of `205 ms` and a size of `283 B`. The response body is shown in JSON format:

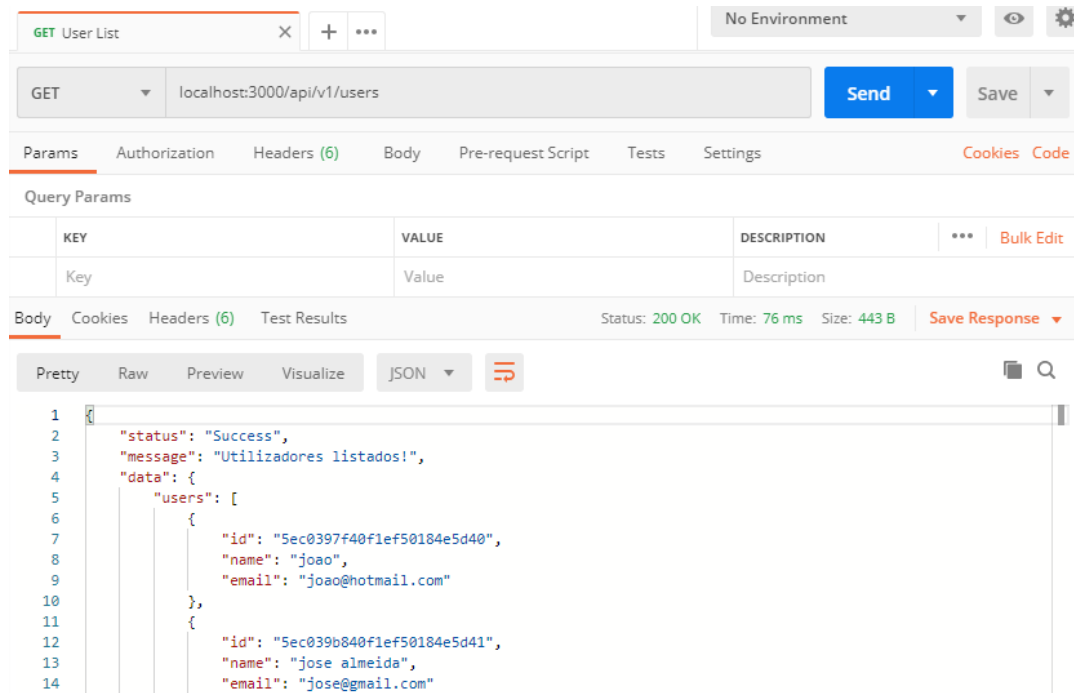
```

1 {
2   "status": "Success",
3   "message": "User added Successfully!!!",
4   "data": null
5 }
```

FIGURA 4 REGISTAR UTILIZADOR (POST)

#### 4.3.1.2 Listar Utilizadores (GET)

Após o pedido POST é possível listar os utilizadores já existentes na base de dados. Se não tiver qualquer utilizador adicionado, este responde com uma lista vazia.



The screenshot shows a REST client interface with the following details:

- Request:** GET `localhost:3000/api/v1/users`
- Response Status:** 200 OK, Time: 76 ms, Size: 443 B
- Response Body (JSON):**

```

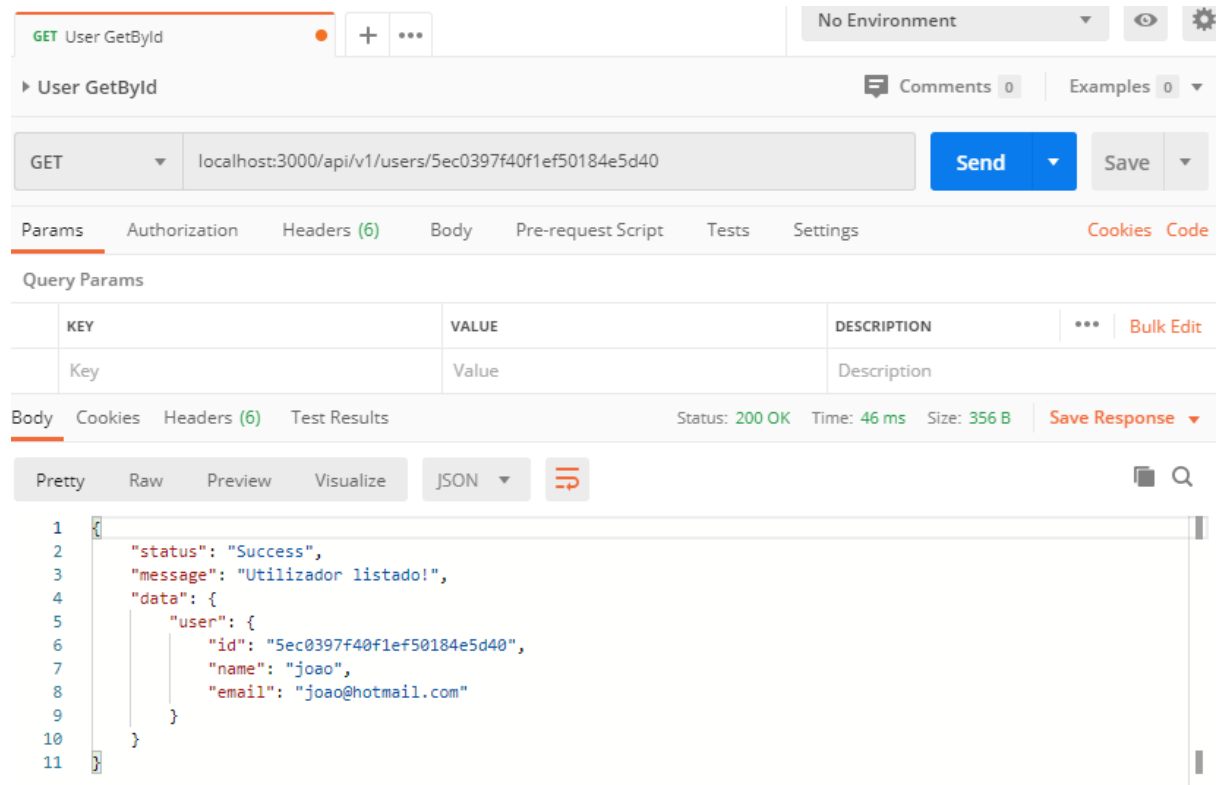
1 {
2   "status": "Success",
3   "message": "Utilizadores listados!",
4   "data": {
5     "users": [
6       {
7         "id": "5ec0397f40f1ef50184e5d40",
8         "name": "joao",
9         "email": "joao@hotmail.com"
10      },
11      {
12        "id": "5ec039b840f1ef50184e5d41",
13        "name": "jose almeida",
14        "email": "jose@gmail.com"

```

FIGURA 5 LISTAR UTILIZADORES

#### 4.3.1.3 Listar Utilizador pelo ID (GET)

É possível listar o utilizador com o respetivo ID com o `Model.findById()`



The screenshot shows a REST client interface with the following details:

- Request Method:** GET
- URL:** localhost:3000/api/v1/users/5ec0397f40f1ef50184e5d40
- Response Status:** 200 OK
- Response Time:** 46 ms
- Response Size:** 356 B
- Response Body (JSON):**

```

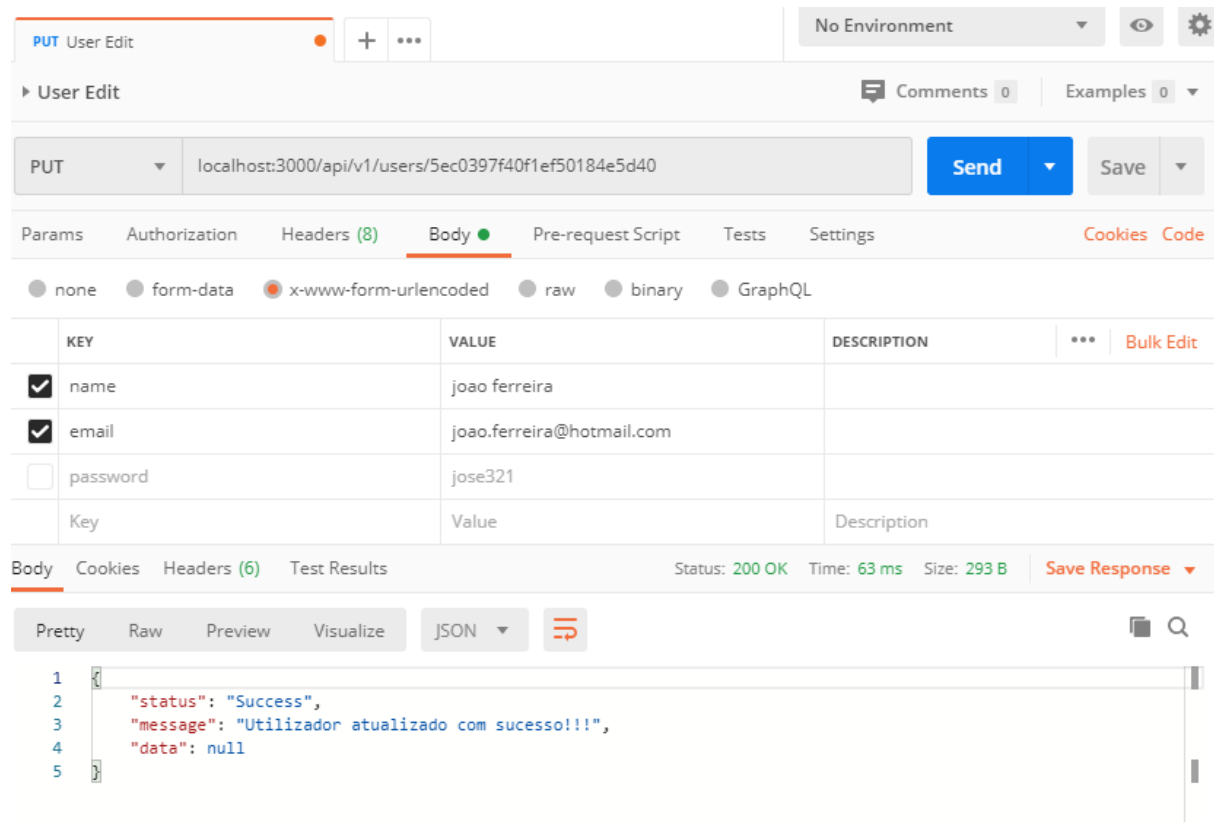
1 {
2   "status": "Success",
3   "message": "Utilizador listado!",
4   "data": {
5     "user": {
6       "id": "5ec0397f40f1ef50184e5d40",
7       "name": "joao",
8       "email": "joao@hotmail.com"
9     }
10  }
11 }

```

FIGURA 6 LISTAR UTILIZADOR PELO ID (GET)

#### 4.3.1.4 Atualizar Utilizador pelo ID (PUT)

Pedido REST que através do `Model.findByIdAndUpdate()` atualiza os campos fornecendo o ID que é gerado pelo Mongo ao ser inserido na base de dados.



The screenshot shows a REST client interface with the following details:

- Request Method:** PUT
- URL:** localhost:3000/api/v1/users/5ec0397f40f1ef50184e5d40
- Body Type:** x-www-form-urlencoded
- Body Content:**

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> name	joao ferreira	
<input checked="" type="checkbox"/> email	joao.ferreira@hotmail.com	
<input type="checkbox"/> password	jose321	
Key	Value	Description
- Status:** 200 OK
- Time:** 63 ms
- Size:** 293 B
- Response Body (JSON):**

```

1 {
2   "status": "Success",
3   "message": "Utilizador atualizado com sucesso!!!",
4   "data": null
5 }
```

FIGURA 7 ATUALIZAR UTILIZADOR PELO ID (PUT)

#### 4.3.1.5 Remover Utilizador pelo ID (DELETE)

Neste pedido é usado o `Model.findByIdAndDelete()` para remover o utilizador da nossa base de dados, sendo o ID fornecido.

The screenshot shows a REST client interface with the following details:

- Request Method:** DELETE
- URL:** localhost:3000/api/v1/users/5ec031c9a83e994d5ebef0cd
- Response Status:** 200 OK
- Response Time:** 43 ms
- Response Size:** 290 B
- Response Body (JSON):**

```
{
  "status": "Success",
  "message": "Utilizador apagado com sucesso!!!",
  "data": null
}
```

FIGURA 8 REMOVER UTILIZADOR PELO ID (DELETE)

#### 4.3.1.6 Login (POST)

Para este pedido utilizamos uma ferramenta lecionada nas sessões desta unidade curricular nomeadamente o JWT.

JSON Web Token (JWT) é um open standard (RFC 7519) que define um método compacto e autocontido para transmitir com segurança informações entre as partes num objeto JSON.



Quando os tokens são assinados usando pares de chaves pública/privada, a assinatura também certifica que a parte que é proprietária da chave privada é aquela que a assinou.

Após o login o token é fornecido sendo este usado para todas operações CRUD dos testes, estando eles numa rota privada onde só depois do registo e posterior login será possível aceder a essa rota.

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:3000/api/v1/users/login
- Status:** 200 OK
- Time:** 187 ms
- Size:** 616 B

The response body is a JSON object:

```

{
  "status": "Success",
  "message": "user found!!!",
  "data": {
    "user": {
      "_id": "5ec0397f40f1ef50184e5d40",
      "name": "joao",
      "email": "joao@hotmail.com",
      "password": "$2b$10$IFKvA5gNHqVIqRsGjzqYquNsHcga33N8IBW3xwU1t/E5udStORh3i",
      "__v": 0
    },
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjV1YzAzOTdmNDBmMwVmNTAxODRlNWQ0MCI6Im1hdCI6MTU4OTY1NjE0OSwiZmxhIjozNTg5NjU5NzQ5fQ.5EKGbQ1vVlrDM2wJedZWJF12-XoAwbtZf5ug7bcrp5HA"
  }
}

```

FIGURA 9 LOGIN (POST)

### 4.3.2 Testes

Como já foi dito anteriormente, esta é uma “private route”, ou seja, só fornecendo o JWT é que é possível realizar as operações CRUD apresentadas posteriormente.

Todas estas operações terão de passar no ficheiro validateuser.js da pasta middleware.

```
You, a few seconds ago | 2 authors (Diogo Costa and others)
1 // private route
2 app.use('/testes', validateUser, testes);
3
4 function validateUser(req, res, next) {
5   jwt.verify(
6     req.headers['x-access-token'],
7     req.app.get('secretKey'),
8     function (err, decoded) {
9       if (err) {
10         res.json({ status: 'error', message: err.message, data: null });
11       } else {
12         // add user id to request
13         req.body.userId = decoded.id;
14         next();
15       }
16     }
17   );
18 }
```

You, a few seconds ago • Uncommitted changes

FIGURA 10 MIDDLEWARE VALIDATEUSER.JS



#### 4.3.2.1 Inserir Teste (POST)

Neste pedido é feito o registo do teste, ou seja, após este pedido o teste é inserido na base de dados.

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:3000/api/v1/tests/
- Environment:** No Environment
- Buttons:** Send, Save
- Table of Headers:**

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> saude24	false	
<input checked="" type="checkbox"/> risk_group	false	
<input checked="" type="checkbox"/> risk_local	true	
- Body Tab:**
  - Method:** POST
  - Status:** 200 OK
  - Time:** 69 ms
  - Size:** 533 B
  - Buttons:** Save Response
  - Response Body (JSON):**

```

1 {
2   "status": "Success",
3   "message": "Teste adicionado com sucesso!",
4   "data": {
5     "user_state": "suspeito",
6     "test_state": "pendente",
7     "test_result": null,
8     "priority": false,
9     "_id": "5ec04e498f9a765780aa4686",
10    "saude24": false,
11    "risk_group": false,
12    "risk_local": true,
13    "userId": "5ec0397f40f1ef50184e5d40",
14    "data": "2020-05-16T20:34:17.601Z",
15    "__v": 0
16  }
17 }
```

FIGURA 11 INSERIR TESTE (POST)

#### 4.3.2.2 Listar Teste (GET)

Após o pedido POST é possível listar os testes feitos por um determinado utilizador já existentes na base de dados. Se não tiver qualquer utilizador adicionado, este responde com uma lista vazia.

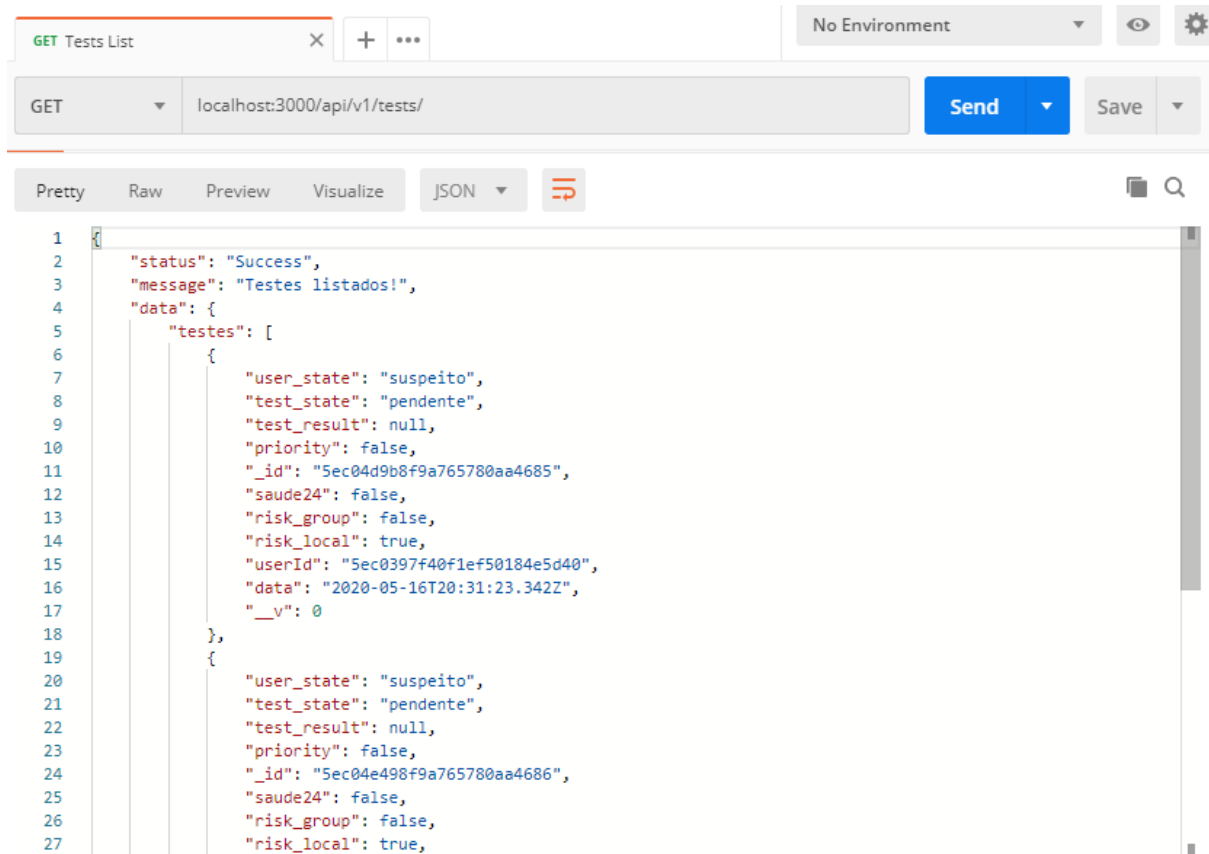
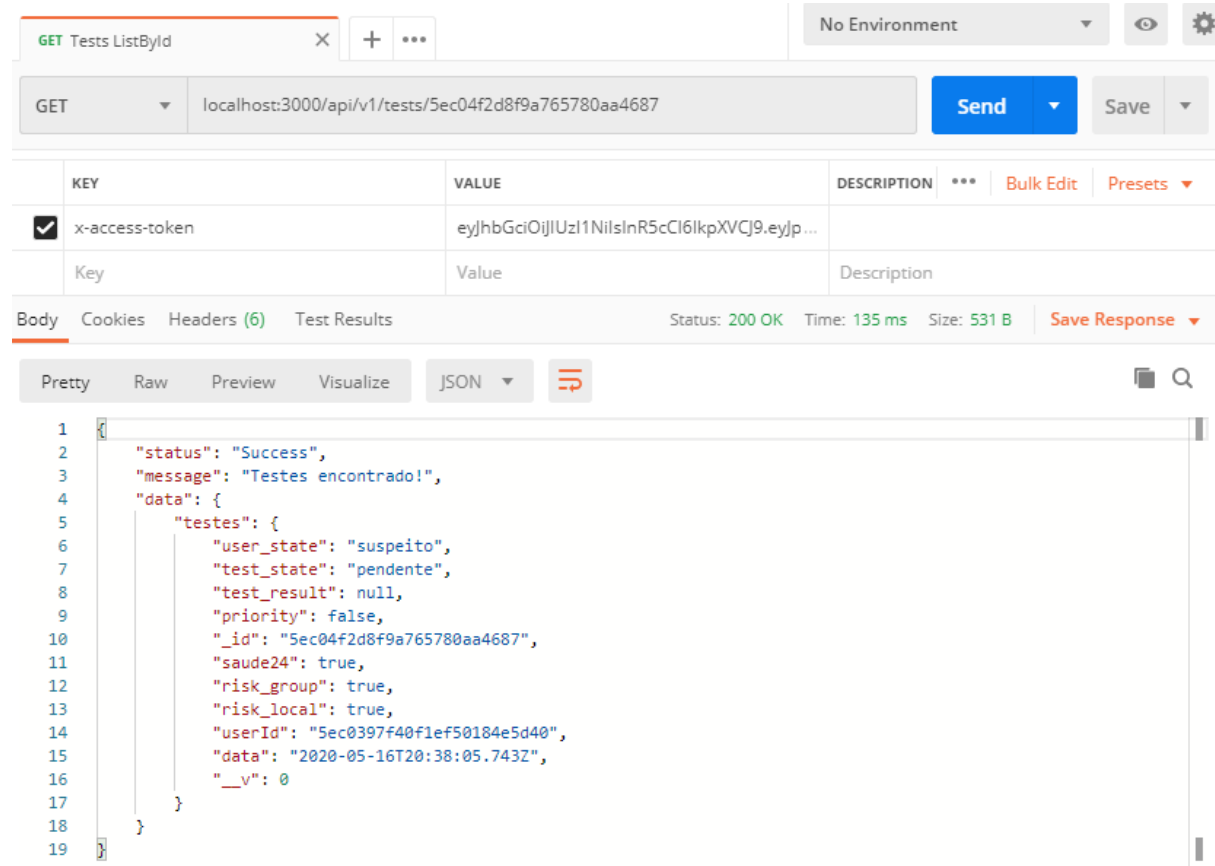


FIGURA 12 LISTAR TESTE (GET)

#### 4.3.2.3 Listar Teste pelo ID (GET)

É possível listar o teste com o respetivo ID com o Model.findById()



The screenshot shows a REST client interface with a GET request to `localhost:3000/api/v1/tests/5ec04f2d8f9a765780aa4687`. The response is a JSON object with the following structure:

```

{
  "status": "Success",
  "message": "Testes encontrado!",
  "data": {
    "testes": {
      "user_state": "suspeito",
      "test_state": "pendente",
      "test_result": null,
      "priority": false,
      "_id": "5ec04f2d8f9a765780aa4687",
      "saude24": true,
      "risk_group": true,
      "risk_local": true,
      "userId": "5ec0397f40f1ef50184e5d40",
      "data": "2020-05-16T20:38:05.743Z",
      "__v": 0
    }
  }
}

```

FIGURA 13 LISTAR TESTE PELO ID (GET)

#### 4.3.2.4 Atualizar Teste pelo ID (PUT)

Pedido REST que através do Model.findByIdAndUpdate() atualiza os campos fornecendo o ID que é gerado pelo Mongo ao ser inserido na base de dados.

PUT Tests Edit

PUT localhost:3000/api/v1/tests/5ec04d9b8f9a765780aa4685

Send Save

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> user_state	not infected	
<input checked="" type="checkbox"/> test_state	done	
<input checked="" type="checkbox"/> test_result	positive	
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 268 ms Size: 555 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "status": "Success",
3   "message": "Teste atualizado com sucesso!",
4   "data": {
5     "user_state": "not infected",
6     "test_state": "done",
7     "test_result": "positive",
8     "priority": null,
9     "_id": "5ec04d9b8f9a765780aa4685",
10    "saude24": null,
11    "risk_group": null,
12    "risk_local": null,
13    "userId": "5ec0397f40f1ef50184e5d40",
14    "data": "2020-05-16T20:31:23.342Z",
15    "__v": 0,
16    "information": null
  }
}
```

FIGURA 14 ATUALIZAR TESTE PELO ID (PUT)

#### 4.3.2.5 Remover Teste pelo ID (DELETE)

Neste pedido é usado o `Model.findByIdAndDelete()` para remover o teste da nossa base de dados, sendo o ID fornecido.

The screenshot shows a REST client interface with the following details:

- Request Method:** DELETE
- URL:** localhost:3000/api/v1/tests/5ec04f2d8f9a765780aa4687
- Headers:**

KEY	VALUE	DESCRIPTION
x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJp...	
Key	Value	Description
- Response:**

```

1 {
2   "status": "Success",
3   "message": "Teste apagado com sucesso!",
4   "data": null
5 }
```
- Status:** 200 OK, Time: 116 ms, Size: 283 B

FIGURA 15 REMOVER TESTE PELO ID (DELETE)

## 4.4 Documentação

Para a documentação dos nossos pedidos REST utilizamos o SWAGGER.



FIGURA 16 LOGO SWAGGER

Sendo uma ferramenta de documentação abordada nas aulas, foi para nós, bastante vantajoso pois facilitou todos os testes aos nossos pedidos.

Na imagem em baixo podemos ver todos os pedidos REST dos “Users” sendo que em cada um é possível enviar/receber os respetivos parâmetros.

Em todos os pedidos PUT e POST que é necessário inserir dados no body neste momento ainda não está funcional pelo que só poderá se testado na aplicação POSTMAN.

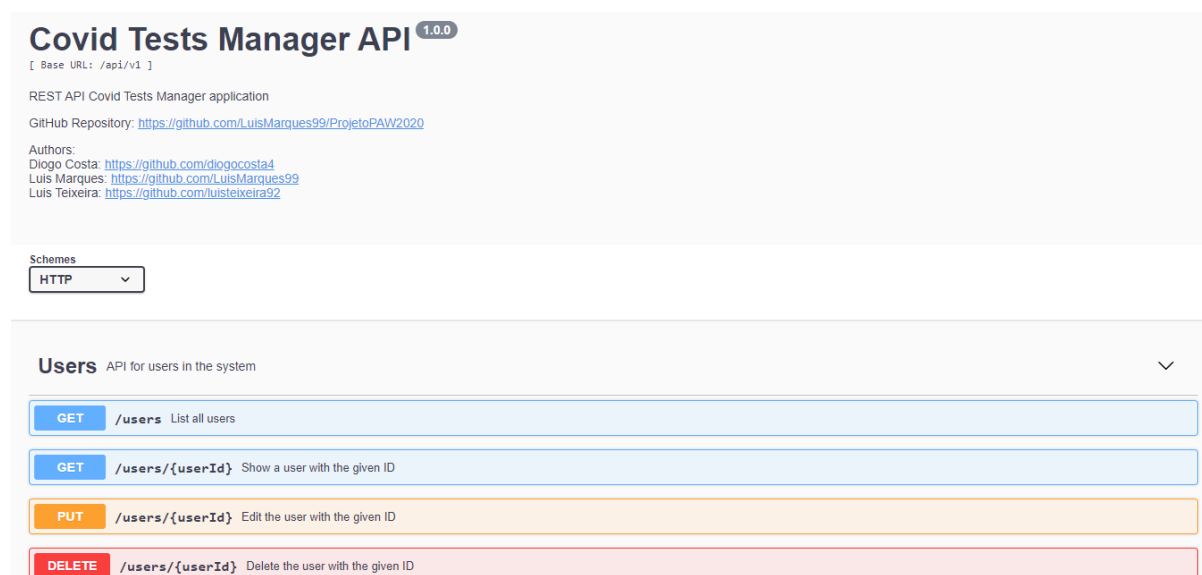


FIGURA 17 SWAGGER COVID TESTS MANAGER API

## 5. Conclusão

---

Ao realizar este trabalho foi nos permitido consolidar a matéria lecionada nesta unidade curricular.

Cumprimos todos os objetivos inicialmente delineados para este primeiro Milestone sendo que implementamos tudo o que nos foi proposto.

## 6. GitHub

---

<https://github.com/LuisMarques99/Covid-Tests-Manager>



## 7. Bibliografia

---

- <https://swagger.io/docs/>
- <https://www.npmjs.com/>
- <https://www.mongodb.com/>