Plataforma de apoio a centro de testes covid-19

Programação em Ambiente Web

Grupo 39

Diogo Costa 8170455 | Luis Marques 8170485 | Luis Teixeira 8110156

Índice

[1 Introdução 4](#_Toc42898161)

[1.1 Contextualização do Documento 4](#_Toc42898162)

[1.2 Apresentação do caso de estudo 4](#_Toc42898163)

[2 Objetivo 5](#_Toc42898164)

[2.1 Ferramentas e tecnologias utilizadas 5](#_Toc42898165)

[3 Definições, Acrónimos e Abreviaturas 6](#_Toc42898166)

[4 Visão geral do projeto 7](#_Toc42898167)

[4.1 Objetivo do Software 7](#_Toc42898168)

[5 Implementação da API REST 8](#_Toc42898169)

[5.1 Definição 8](#_Toc42898170)

[5.2 Definição de tabelas da base de dados 8](#_Toc42898171)

[5.3 Operações CRUD 9](#_Toc42898172)

[5.4 Users 10](#_Toc42898173)

[5.4.1 Registar Utilizador (POST) 10](#_Toc42898174)

[5.4.2 Listar Utilizadores (GET) 11](#_Toc42898175)

[5.4.3 Listar Utilizadores pelo ID (GET) 12](#_Toc42898176)

[5.4.4 Atualizar Utilizador através do ID (PUT) 13](#_Toc42898177)

[5.4.5 Remover Utilizador através do ID (DELETE) 0](#_Toc42898178)

[5.5 Testes 1](#_Toc42898179)

[5.5.1 Inserir Teste (POST) 2](#_Toc42898180)

[5.5.2 Listar Teste (GET) 3](#_Toc42898181)

[5.5.3 Listar Teste pelo ID (GET) 4](#_Toc42898182)

[5.5.4 Atualizar Teste pelo ID (PUT) 5](#_Toc42898183)

[5.5.5 Remover Teste através de ID (DELETE) 6](#_Toc42898184)

[5.6 Authenticate (POST) 6](#_Toc42898185)

[5.7 Documentação 7](#_Toc42898186)

[6 FrontEnd (Angular) 7](#_Toc42898187)

[6.1 Introdução 7](#_Toc42898188)

[6.2 Estrutura 8](#_Toc42898189)

[6.2.1 Services 8](#_Toc42898190)

[6.2.2 Models 9](#_Toc42898191)

[6.2.3 Components 9](#_Toc42898192)

[7 Conclusão 14](#_Toc42898193)

[8 GitLab 15](#_Toc42898194)

[9 Bibliografia 16](#_Toc42898195)

|  |  |  |
| --- | --- | --- |
| Date | Description | Version |
| 02/05/2020 | * Contextualização do Documento * Apresentação do Caso de Estudo * Definições, Acrónimos e Abreviaturas | 1.0 |
| 02/06/2020 | * Descrição da implementação da REST API e Front End | 1.1 |
| 12/06/2020 | * Conclusões finais | 2.0 |

Histórico Alterações Documento

# Introdução

## Contextualização do Documento

Este documento descreve o processo de implementação do projeto desenvolvido no âmbito da unidade curricular ‘Programação em Ambiente Web’, e fornece uma visão geral sobre o produto final.

## 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.

# 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.

## Ferramentas e tecnologias utilizadas

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

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

# Definições, Acrónimos e Abreviaturas

REST - Representational State Transfer

API - Application Programming Interface

# Visão geral do projeto

## Objetivo do Software

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.

# Implementação da API REST

## 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.

## 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 nas aulas, nomeadamente o MongoDB.

Possuímos duas tabelas na nossa base de dados. A tabela “Test” constituída pelos campos:

* saude24 (boolean)
* risk\_group (boolean)
* risk\_local (boolean)
* information (String)
* user\_state (String)
* test\_state (String)
* test\_result (String)
* date (date)
* pdf
* user\_id (mongoose.Schema.Types.ObjectId)

E a tabela “User” é constituída pelos seguintes campos:

* name (String)
* email (String)
* password (String)
* Role (String)

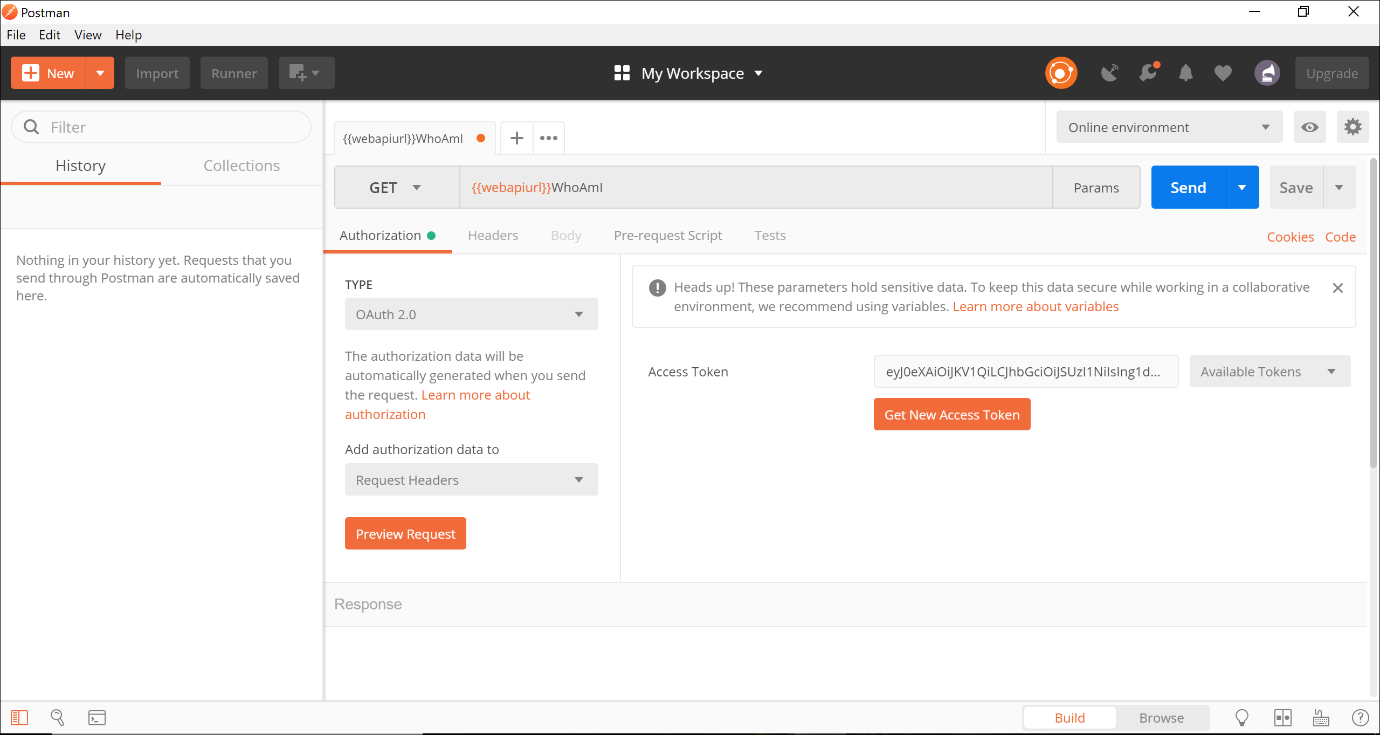
Estas tabelas foram implementadas na pasta “models”, sendo essa pasta por padrão onde ficam implementadas as tabelas.

## Operações CRUD

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

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

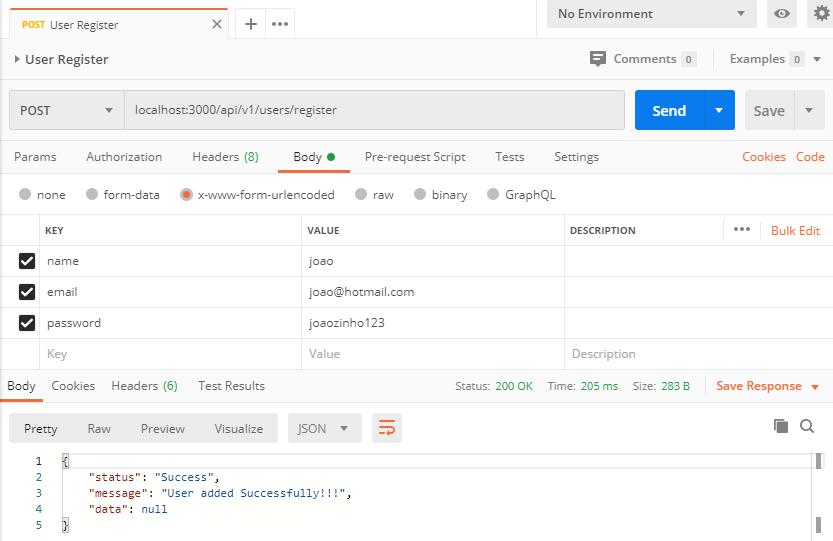
Posteriormente, para testar os nossos pedidos REST utilizamos o Postaman sendo uma ferramenta bastante vantajosa, que permite:

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

## Users

### Registar Utilizador (POST)

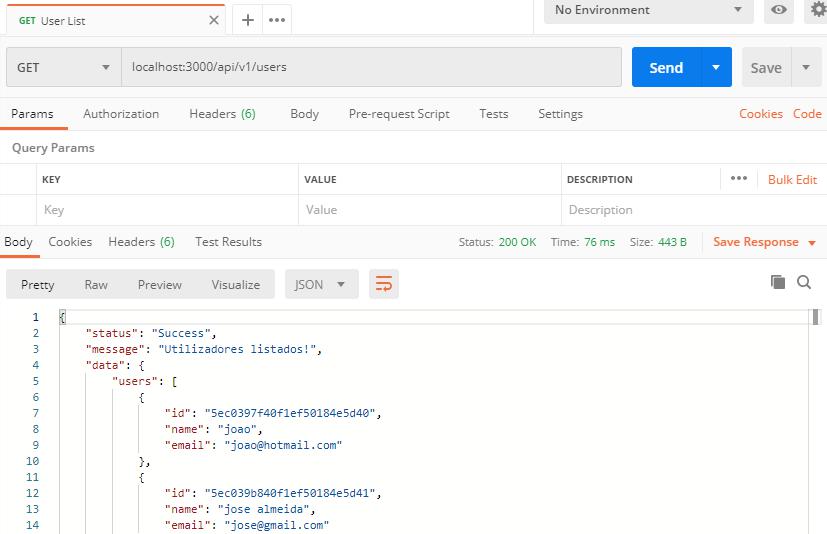
Neste pedido é feito o registo do utilizador, ou seja, após este pedido o utilizador é inserido na base de dados.



9

### Listar Utilizadores (GET)

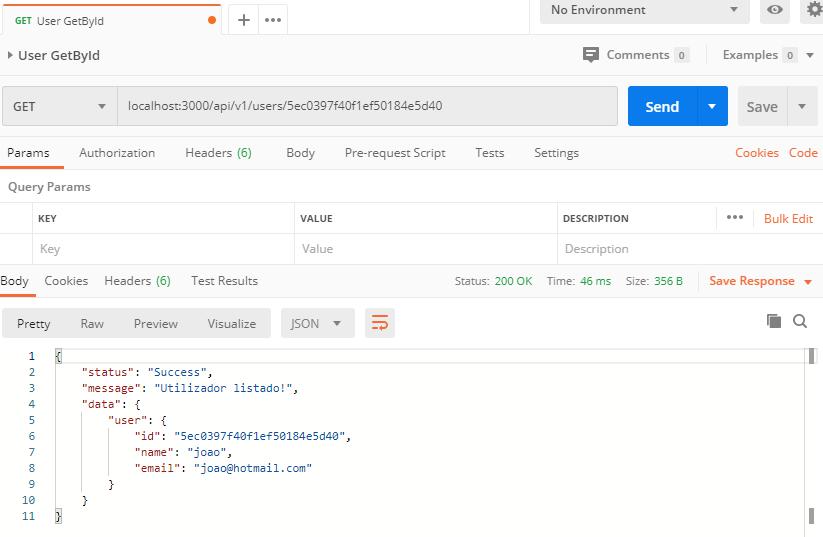
Após o pedido POST é possível listar os utilizadores já existentes na base de dados. Se não tiver qualquer utilizador adicionado, este responde com uma lista vazia.



10

### Listar Utilizadores pelo ID (GET)

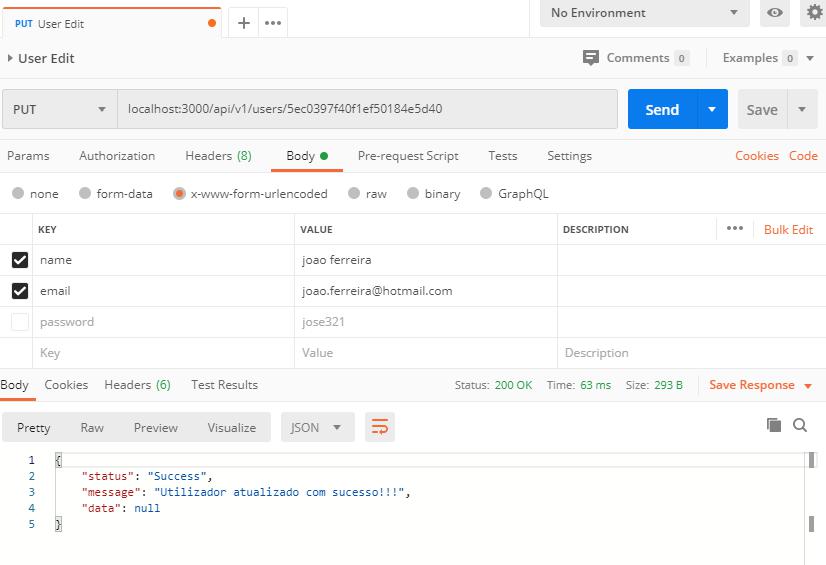
É possível listar o utilizador com o respetivo ID com o Model.findById()



11

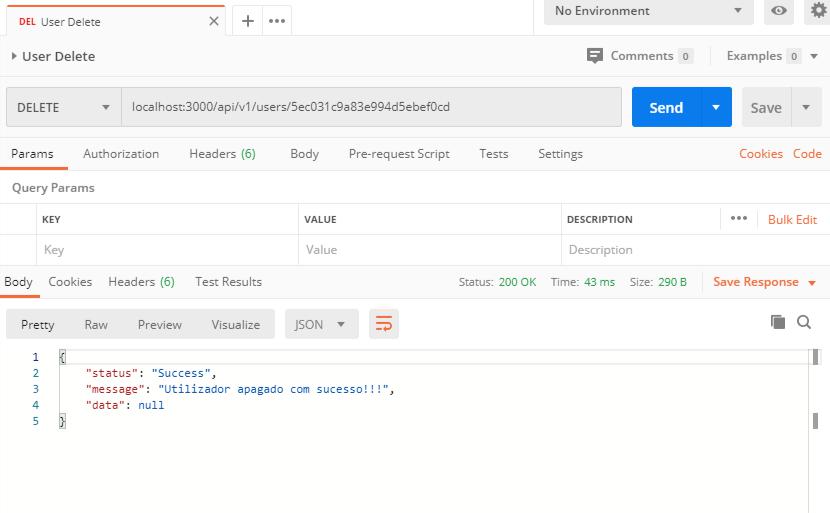
### Atualizar Utilizador através do ID (PUT)

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.



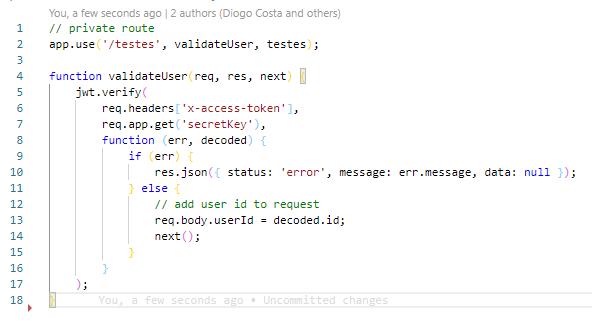
### Remover Utilizador através do ID (DELETE)

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



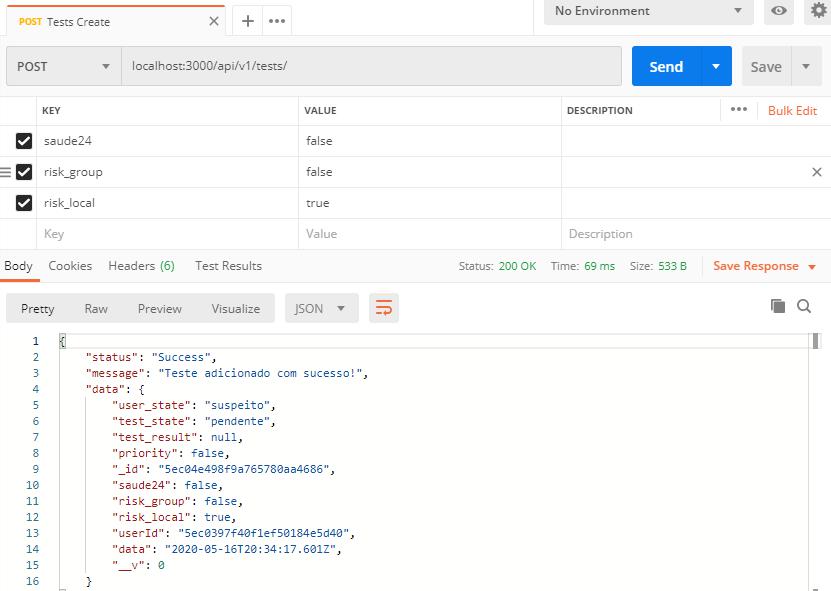
## Testes

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.



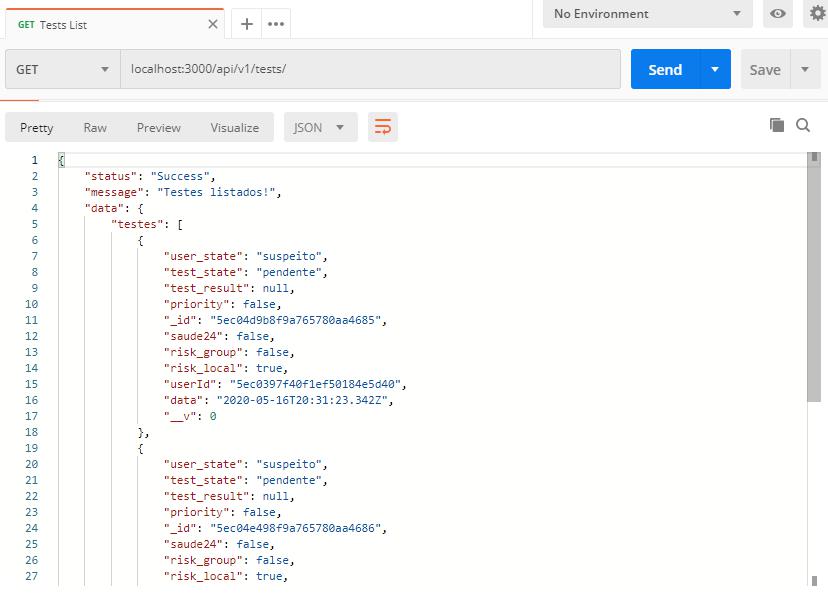
### Inserir Teste (POST)

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



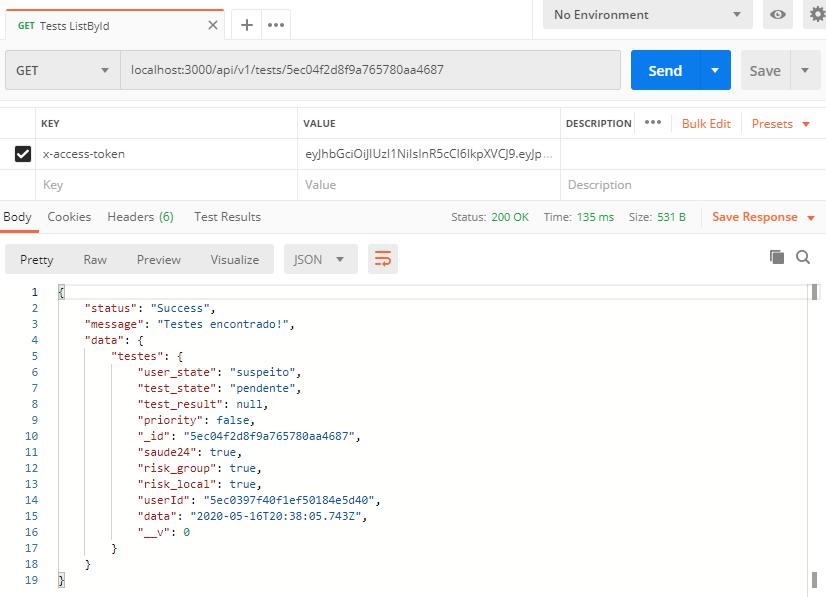
### Listar Teste (GET)

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.



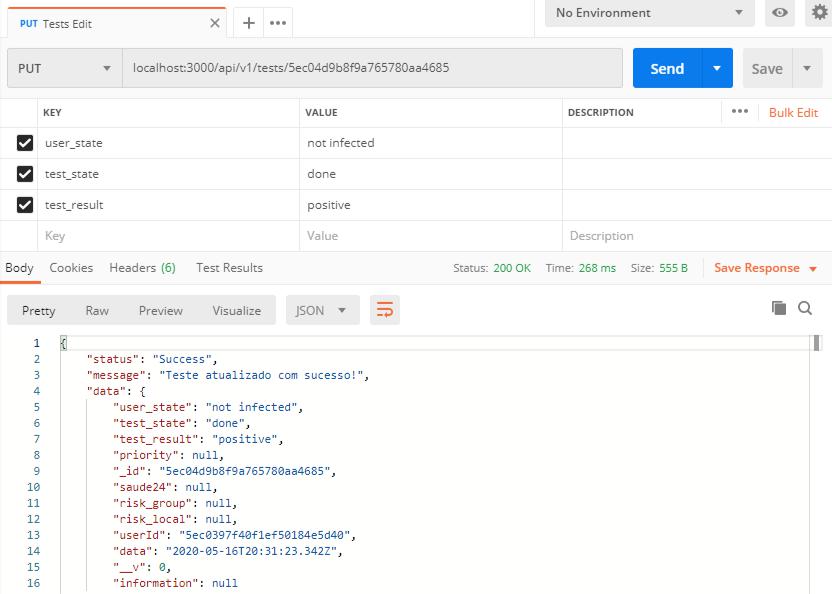
### Listar Teste pelo ID (GET)

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



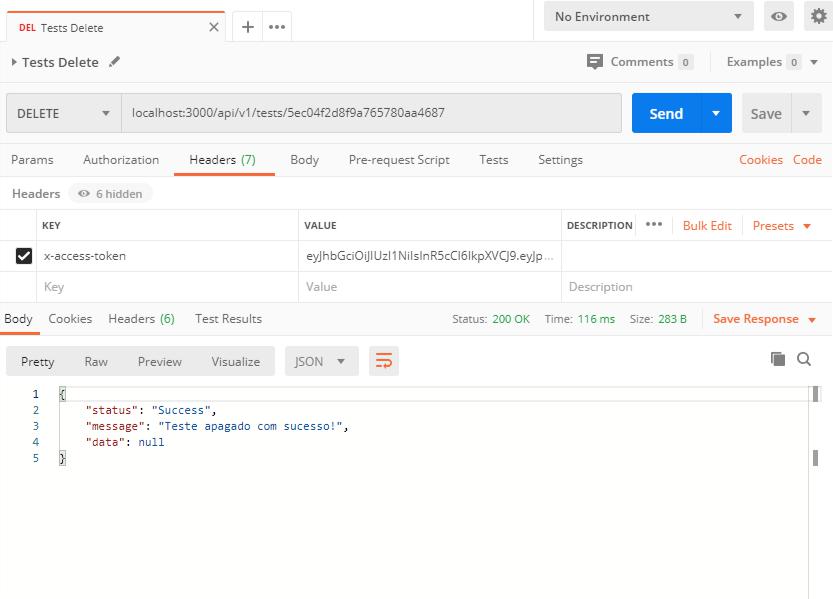
### Atualizar Teste pelo ID (PUT)

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.

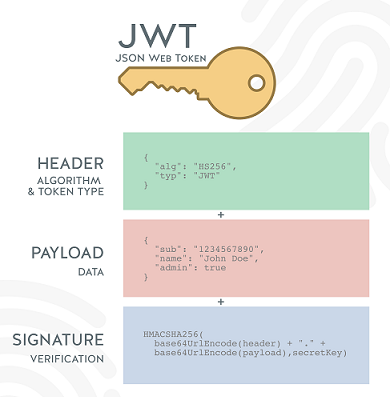


### Remover Teste através de ID (DELETE)

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



## Authenticate (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.

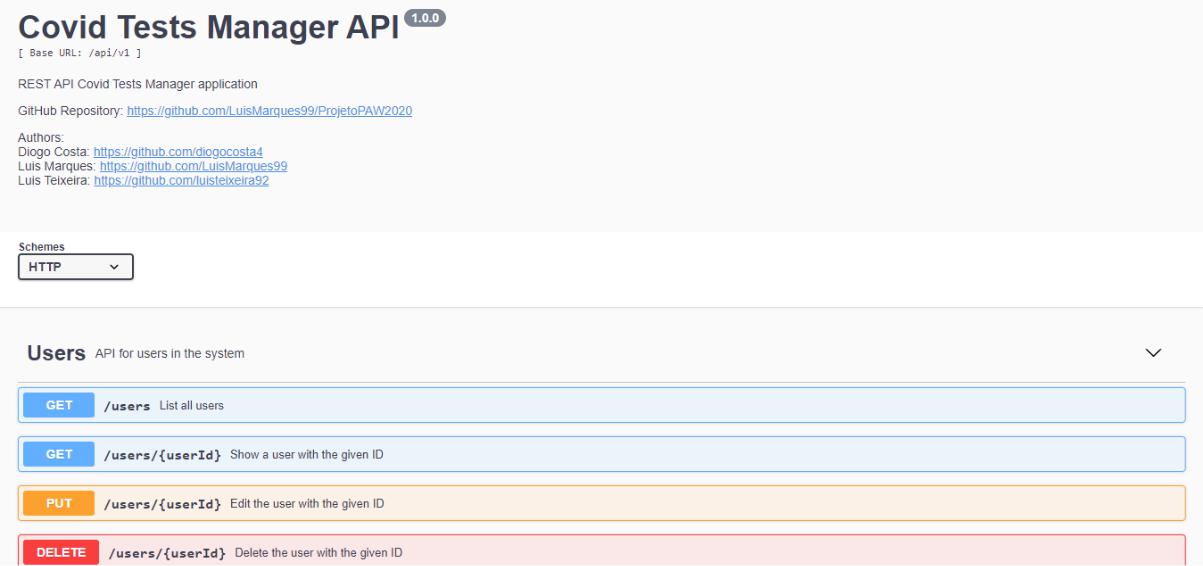
Quando os tokens são assinados usando pares de chaves pública/privada, a assinatura também certifica que a parte que é proprietária da chave privada é aquela que a assinou.

## Documentação

 Para a documentação dos nossos pedidos REST utilizamos o SWAGGER, que foi a ferramenta de comentação abordada nas aulas, da qual tiramos partido para facilitar a execução dos 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.



# FrontEnd (Angular)

## Introdução

Para a parte do cliente utilizamos a Framework Angular.Angular é uma plataforma e framework para construção da interface de aplicações usando HTML, CSS e TypeScrit, criada pelos desenvolvedores da Google.

Dentre os principais, podemos destacar os componentes, templates, pastas, roteamento, módulos, serviços, injeção de dependências e ferramentas de infraestrutura que automatizam tarefas, como a execução de testes unitários de uma aplicação.

## Estrutura

### Services

Para dar resposta aos nossos pedidos HTTP feitos ao servidor criamos serviços em Angular nomeadamente:

* **Serviço para os Users (users.service.ts)**

Serviço para responder aos pedidos READ, UPDATE e DELETE dos utilizadores

* **Serviço para os Testes (tests.service.ts)**

Serviço para responder aos pedidos CREATE, READ, UPDATE e DELETE dos testes

* **Serviço para Autenticação (auth.service.ts)**

Serviço para responder aos pedidos CREATE, READ dos utilizadores nomeadamente o registar e fazer login do utilizador.

* **Serviço para o Token (token-interceptor.service.ts)**

Serviço para o token pois precisamos de colocar o esquema de autenticação HTTP: Bearer Authentication.

### Models

* **Model para o User**

Classe com todos os campos do utilizador e seus respetivos tipos.

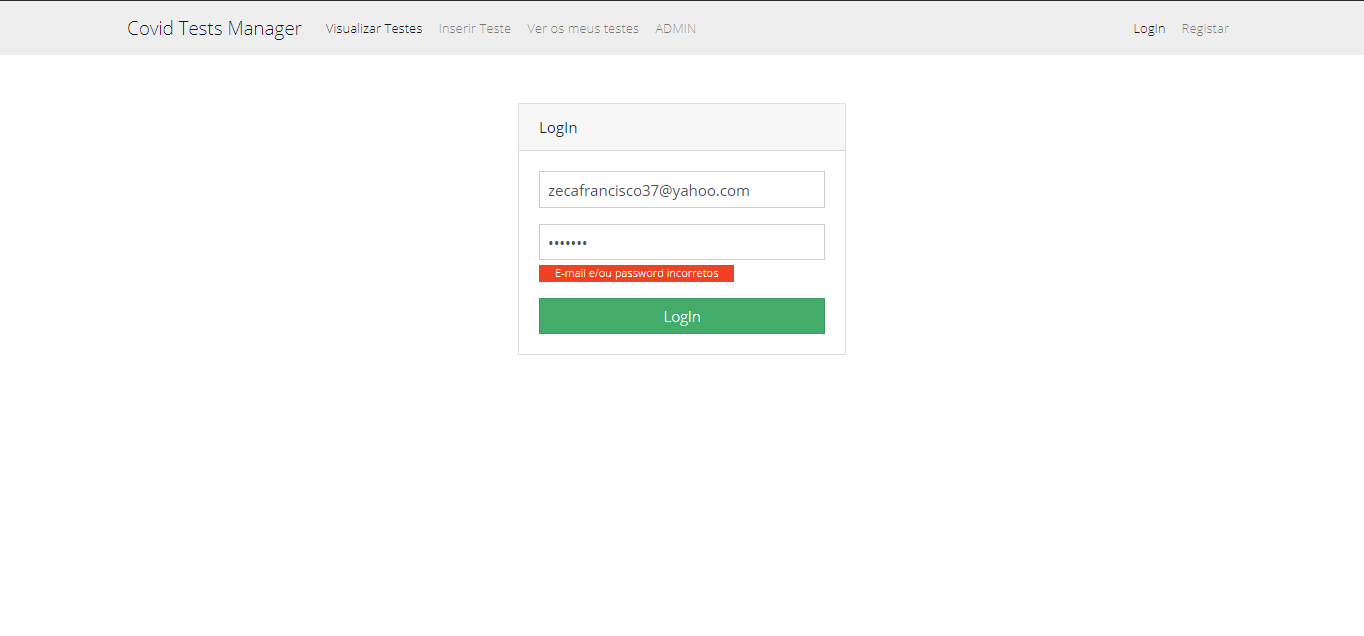
* **Model para o Teste**

Classe com todos os campos do teste e seus respetivos tipos.

### Components

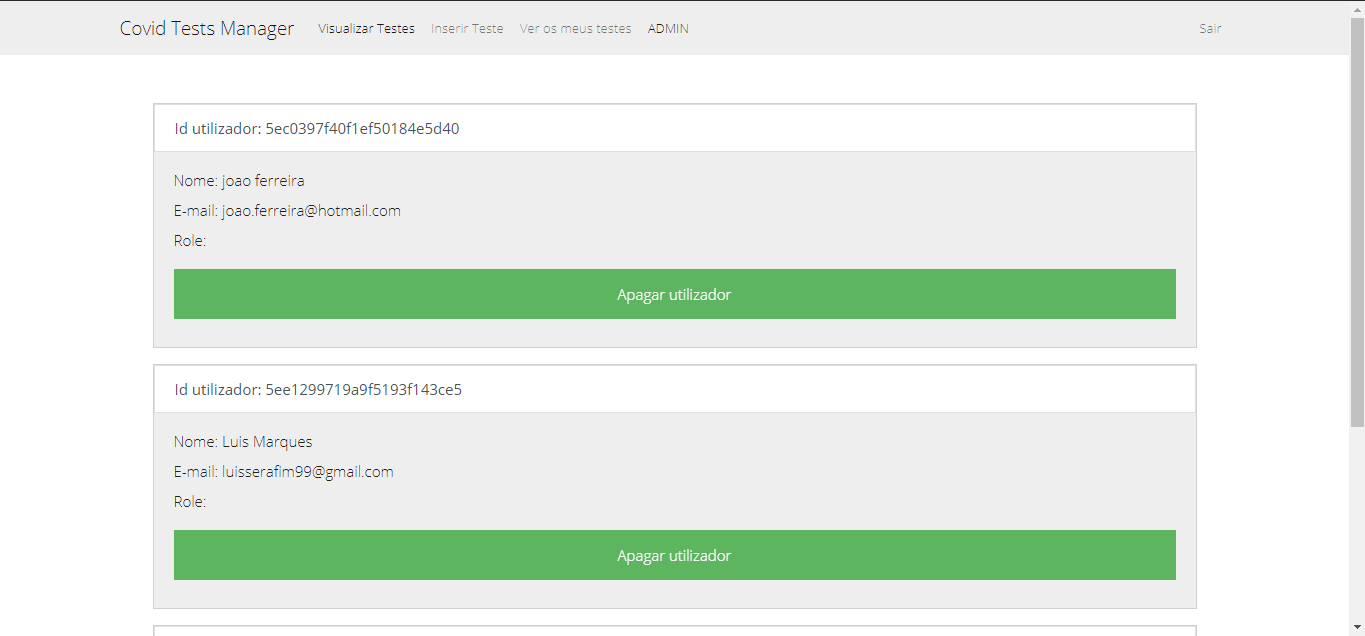
Componentes que contêm um ficheiro TypeScript, responsável por uma parte da lógica do projeto.

#### Sign in (Login)



#### Sign-up (Registo de utilizador)

#### User Management

Componente de gestão de utilizadores, não conseguimos adaptar todos os roles que idealizamos por isso todos os utilizadores do sistema são considerados ADMIN, mesmo não estando registados.

#### Listar Testes

#### Criar Teste

#### Listar Testes de Utilizador

#### Apagar Utilizador

#### Detalhes do Teste

#### Editar Teste

# Conclusão

Ao realizar este trabalho foi nos permitido consolidar toda a matéria lecionada ao longo do semestre na unidade curricular.

Encontramos algumas dificuldades no processo de desenvolvimento, mas no final conseguimos implementar a grande maioria dos requisitos propostos para o trabalho.

# Git

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

# Bibliografia

* <https://angular.io/cli/generate>
* <https://www.mongodb.com/>
* <https://getbootstrap.com/>
* https://nodejs.org/en/