Sistemas de Computação na Cloud

Carlos Bruno Machado Martins – 18836

João Ricardo Pinto Azevedo – 18845

Professor

Miguel Lopes

Ano letivo 2022/2023

Mestrado em Engenharia Informática

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e do Ave

Índice

Introdução

O presente relatório tem como objetivo documentar todo o desenvolvimento.

Tema

Este trabalho prático consistiu no desenvolvimento de um serviço de autenticação que servisse de suporte para gestão de acessos e permissões de outros micro serviços.

De modo a que este fosse um serviço de autenticação robusto, os seguintes requisitos foram propostos:

- A autenticação deve expirar
- Deve conter um mecanismo de proteção contra ataques *brute-force*
- As credenciais devem ser armazenadas de forma segura
- Deve conter um serviço de *logs*

Arquitetura aplicacional

A arquitetura da aplicação sofreu uma ligeira alteração tendo em conta a primeira entrega do trabalho prático. Na versão inicial o modelo continha o RabbitMQ, mas decidimos descartá-lo devido a uns problemas que serão explicados no capítulo de constrangimentos.

Apresentamos abaixo a versão final da arquitetura da aplicação que contém dois micro serviços que comunicam entre eles. O serviço de *logs* é acionado sempre que o utilizador efetua um processo relativo ao serviço de autenticação, isto é, sempre que o utilizador efetua login, registo e alteração da palavra-passe é gerado um documento na base de dados relativo à ação que o mesmo realizou.

Além disso, também foi desenvolvido um *frontend* com páginas relativamente às funcionalidades de registo e login.

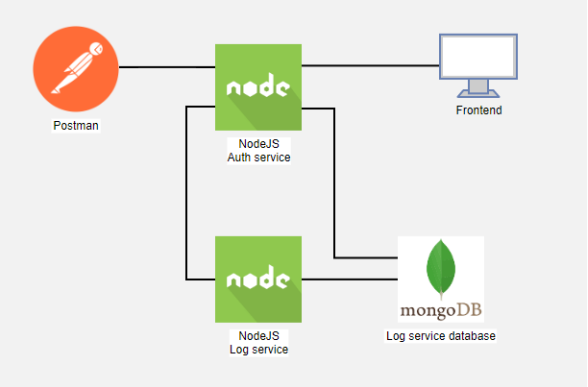

Figura 1 - Diagrama da arquitetura da aplicação

Implementação

Neste capítulo serão apresentados os detalhes técnicos relativos ao desenvolvimento da aplicação.

Base de dados

A base de dados escolhida para este desenvolvimento foi o MongoDB que é uma base de dados não relacional que utiliza uma estrutura de documentos.

Tendo em conta o objetivo da cadeira decidimos implementar esta base de dados não localmente, mas sim na *cloud*. O facto de termos implementado a base de dados na *cloud* trouxe alguns constrangimentos quando tínhamos oportunidade de desenvolver o trabalho prático em aula, porque não era possível aceder à mesma através do código desenvolvido.

O serviço de alojamento da base de dados foi fornecido pelo próprio mongoDB que disponibiliza um cluster de forma gratuita.

Serviço de autenticação

Para o serviço de autenticação foi definido um modelo para a tabela responsável pelo armazenamento de informações de utilizador utilizando a biblioteca mongoose. Este modelo garante que a estrutura de cada documento deve conter *username*, *password*, nome e um email. Além disso, foi definido que tanto o *username* como o email devem ser únicos.

Registo

A funcionalidade de registo recebe todos os parâmetros mencionados anteriormente e, de modo a garantir o armazenamento seguro da palavra-passe, a mesma é encriptada utilizando mecanismos disponibilizados pela biblioteca bcryptjs.

Quando o utilizador é criado com sucesso é então gerado um documento no mongoDB relativo ao registo do mesmo. Se não for gerado com sucesso, dependendo do problema é enviado um código HTTP correspondente ao problema encontrado.

Login

Relativamente à funcionalidade de login é utilizado o reCAPTCHA que é um sistema oferecido pela google e que possibilita a distinção de humanos e máquinas de modo a prevenir ataques de *brute-force.* Com isto, garantimos sempre que o utilizador passe por esta verificação antes de fazer qualquer tipo de pedido.

Com o reCAPTCHA validado são então tratadas as informações enviadas, no corpo do pedido, pelo utilizador. Inicialmente, é feita uma *query* à base de dados de modo a obter o seu *username* e a respetiva palavra-passe, de seguida são comparadas ambas as palavras-passes e caso o resultado seja *true* é gerado um *token* de acesso relativo ao utilizador. Este token tem duração de 1h.

Por fim, é gerado o *log* correspondente ao login feito pelo utilizador e apresentado o código de HTTP de sucesso. Se não, é enviado o código de erro correspondente.

Alterar palavra-passe

Tal como o título diz esta funcionalidade é responsável pela alteração da palavra-passe do utilizador.

Para esta funcionalidade foi desenvolvido um *middleware* que vai atuar em todas as rotas que passem em "/protect". Este middleware verifica se no cabeçalho foi fornecido um *token* de acesso e é utilizada a função *verify* disponibilizada pela biblioteca Jsonwebtoken de modo a obter o *id* e o *username* dono do token.

De seguida, é então encriptada a nova palavra-passe recebida no body e guardada no documento correspondente ao utilizador.

Por fim, é gerado um *log* e é enviado um código de sucesso caso a palavra-passe seja alterada com sucesso. Se não, é enviado o código de erro correspondente.

Envio de email

Esta funcionalidade foi a última a ser desenvolvida no serviço de autenticação.

O objetivo desta funcionalidade é o envio de um email ao utilizador com o conteúdo a ser uma página web onde seria possível alterar a sua palavra-passe.

Serviço de Logs

O serviço de logs tem como finalidade fazer o *tracking* dos utilizadores, isto é, quando foram realizados inícios de sessão, registos e alterações à palavra-passe. Este serviço utiliza duas tabelas sendo que uma é responsável pela identificação dos códigos e a outra é responsável pelo armazenamento de todos os *logs*. Esta separação tem como objetivo reduzir o armazenamento necessário.

Docker

Para o desenvolvimento deste trabalho tínhamos como requisito a utilização de Docker para *containerizar* a aplicação e o DockerHub para armazenar a imagem da aplicação. A primeira etapa foi a elaboração de um *dockerfile* para cada um dos micro serviços como mostram as figuras 2 e 3.

```
FROM node:lts-alpine
ENV NODE_ENV=production
ENV ACCESS_TOKEN_SECRET=e573eef32ee554ad4e05682c8acc62117670780e6d21545dd137
WORKDIR /usr/src/app
COPY ["package.json", "package-lock.json*", "npm-shrinkwrap.json*", "./"]
RUN npm install --production --silent && mv node_modules ../
COPY . .
EXPOSE 3000
RUN chown -R node /usr/src/app
USER node
CMD ["node", "index.js"]
```

Figura 2 - Dockerfile auth service

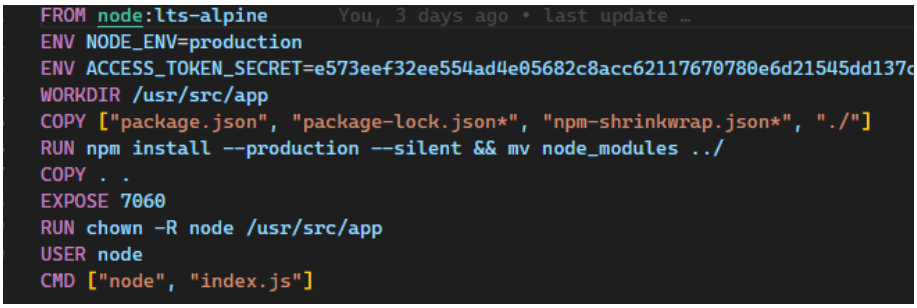

Figura 3 - Dockerfile log service

Após a criação das imagens com os comandos:

- `docker build –t authservice:v1 .`
- `docker build –t logservice:v1 .`

Código 1 - Comandos responsáveis pela criação da imagem dos micro serviços

e respetivos containers com os comandos:

- `docker run –name authservice –d –p 7070:7070 authservice:v1`
- `docker run –name logservice –d –p 7060:7060 logservice:v1`

Código 2 - Comandos responsáveis pela inicialização dos respetivos containers

foi também criada uma network onde fossem alocados os dois containers permitindo, assim, a comunicação entre os mesmos.

- `docker network create sscloud`
- `docker network connect sscloud authservice`
- `docker network connect sscloud logservice`

Código 3 - Comandos responsáveis pela criação da network e integração dos containers na mesma network

Após a *containerização d*os micro serviços, as imagens destes foram colocadas num repositório. Para a realização deste processo foram necessários os seguintes comandos:

- `docker tag authservice:v1 jrazevedo/authservice:authservice_v1`
- `docker tag logservice:v1 jrazevedo/logservice:logservice_v1`

Código 4 - Comandos responsáveis pela atribuição de uma tag aos containers

Por fim, para finalizar o processo de carregamento dos containeres para um repositório devem ser executados os seguintes comandos.

- `docker push jrazevedo/authservice_v1 .`
- `docker push jrazevedo/logservice_v1 .`

Código 5 - Comandos responsáveis pelo push do container para o repositório

Constrangimentos

RabbitMQ

A implementação dos micro serviços contou com alguns constrangimentos, entre eles a implementação do RabbitMQ num container.

Numa primeira fase foi feita a implementação sem a utilização do Docker para *containerizar* a aplicação. Através do RabbitMQ a comunicação entre os diferentes micro serviços era efetuada com sucesso, ou seja, o micro serviço de autenticação publicava no RabbitMQ. Como este é assíncrono, o micro serviço de logs acabava por consumir os dados do Rabbit sem qualquer tipo de erro.

No entanto após a containerização dos mesmos, a comunicação entre os elementos deixou de existir e, posto isto, o RabbitMQ foi excluído do produto final.

Acesso à base de dados

Para este projeto utilizamos uma base de dados na *cloud* (Mongo Atlas) que levou à necessidade da utilização de um Hotspot, uma vez que através da rede do IPCA (Instituto Politécnico do Cávado e do Ave) não era possível aceder à mesma. Ainda tentámos utilizar uma VPN no lugar do hotspot, mas a ligação acabava por cair após um certo período de tempo.

Testes unitários

Foram realizados um conjunto de testes unitários de modo a testar todas as funcionalidades oferecidas pela aplicação desenvolvida.

Na Figura 4 mostra um exemplo do login na aplicação como resultado temos o *token* caso o *user* tenha efetuado o registo.

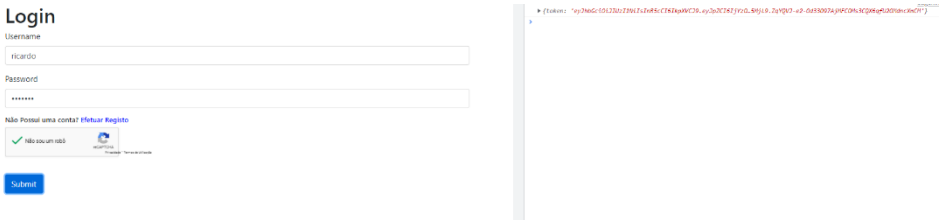

Figura 4 - Página de login e respetivo token obtido

Na Figura 5 temos um exemplo do registo na aplicação para que mais tarde possa utilizar os serviços da mesma.

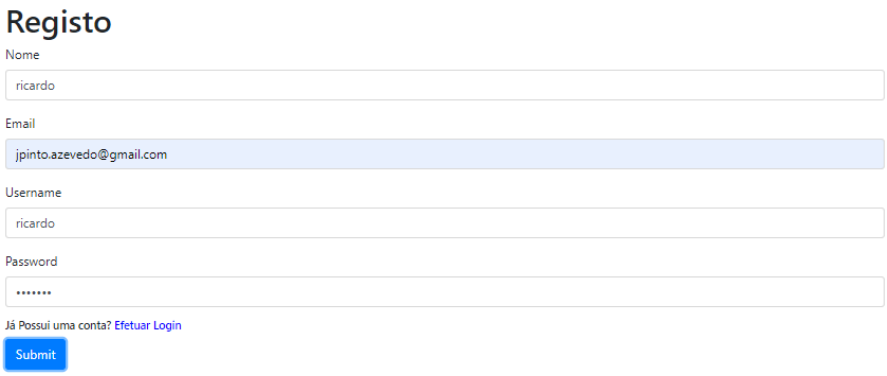

Figura 5 - Página de registo

Na Figura 6 temos um exemplo de um *endpoint* para alterar a password que tem como parâmetro a nova password.

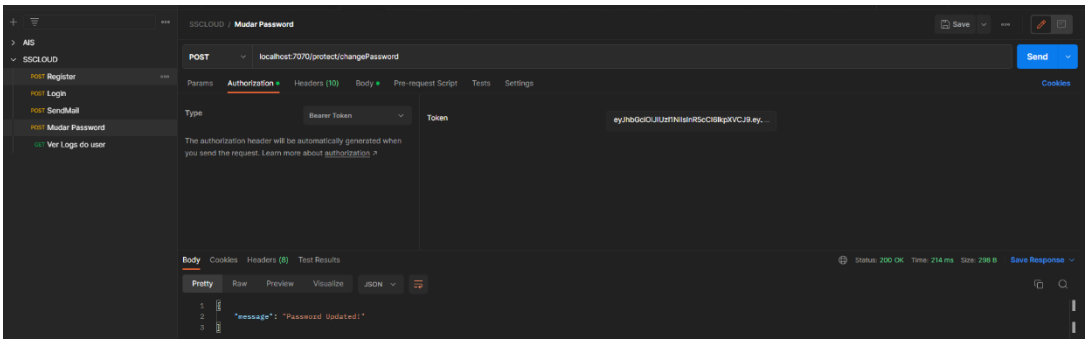

Figura 6 - Rota para alteração da palavra-passe

Na Figura 7 temos um exemplo de um *endpoint* para o envio de um email que tem como parâmetros o utilizador para o qual deve o email ser enviado, o título do assunto e o respetivo conteúdo.

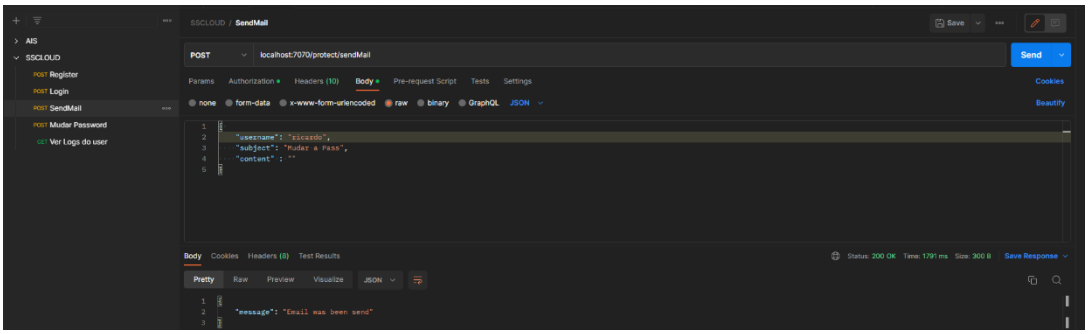

Figura 7 - Rota para envio de email

Na Figura 8 temos um exemplo de um *endpoint* para visualizar os *logs* de todos os utilizadores, isto é visualizar quantos novos registos foram inseridos, logins efetuados, entre outros.

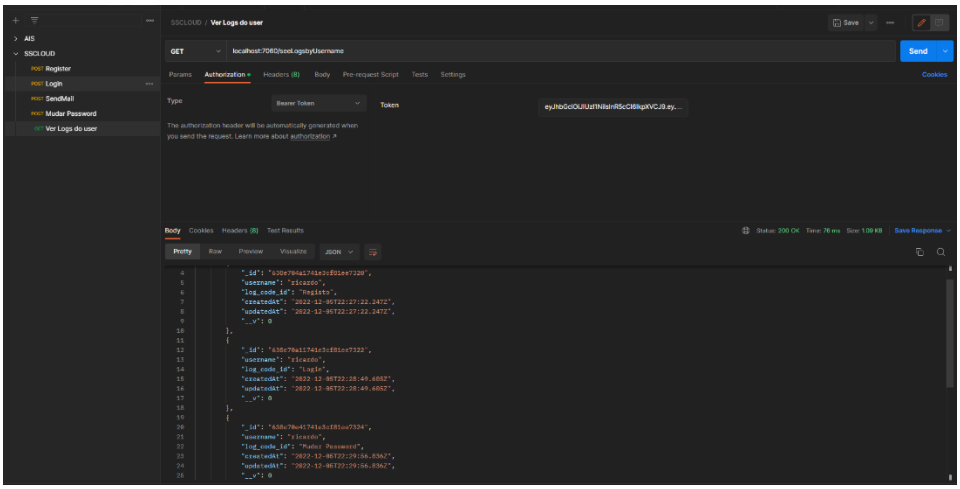

Figura 8 - Rota para visualização de logs

Conclusão

Os objetivos propostos para o desenvolvimento da aplicação podem considerar-se cumpridos, visto foram implementados todos os requisitos obrigatórios. Através do desenvolvimento deste projeto pudemos arrecadar experiência e consolidar conhecimentos da cadeira de sistemas de computação da Cloud.

Alguns pontos a melhorar seriam a implementação do RabbitMQ num container e posterior comunicação com os restantes containers assim como o *deployment* do container num cluster de Kubernetes.

Bibliografia

Docker:

https://docs.docker.com/get-started/

Docker Hub:

https://docs.docker.com/docker-hub/

RabbitMQ:

https://www.rabbitmq.com/documentation.html

NodeMailer:

https://www.npmjs.com/package/nodemailer

Bcrypt:

https://www.npmjs.com/package/bcrypt

Json Web Token:

https://jwt.io/introduction

Docker Compose:

https://docs.docker.com/compose/

MongoDB:

https://www.mongodb.com/docs/