Uma imagem com texto, Tipo de letra, logótipo, Gráficos

Descrição gerada automaticamenteUma imagem com Tipo de letra, texto, Gráficos, logótipo

Descrição gerada automaticamente

**ASIST**

**SPRINT 2**

**Realizado por:**

**Diogo Ferreira, 1220829**

**Francisco Osório, 1220846**

**Sérgio Moreira, 1220890**

**Rafael Ferraz, 1221104**

**11/2024**

# Índice

[Índice 2](#_Toc183197282)

[Índice de Figuras 3](#_Toc183197283)

[Índice de Tabelas 4](#_Toc183197284)

[Divisão Tarefas 5](#_Toc183197285)

[User Stories 6](#_Toc183197286)

[User Story 1 6](#_Toc183197287)

[User Story 2 11](#_Toc183197288)

[User Story 3 12](#_Toc183197289)

[User Story 4 14](#_Toc183197290)

[User Story 5 15](#_Toc183197291)

[User Story 6 17](#_Toc183197292)

[User Story 7 18](#_Toc183197293)

[User Story 8 19](#_Toc183197294)

# Índice de Figuras

[Figura 1 - Script deployment\_backend.sh 5](#_Toc183197148)

[Figura 2 - Github Secrets 6](#_Toc183197149)

[Figura 3 - deploy.yml (workflow) 7](#_Toc183197150)

[Figura 4 - Resultado do workflow 8](#_Toc183197151)

[Figura 5 - Ficheiro deployment\_backend.log com data do deploy 9](#_Toc183197152)

[Figura 6 - Ficheiro deployment\_backend\_output.log com output do deploy 9](#_Toc183197153)

[Figura 7 - Teste de uma rota do backend para verificar que ele estava a correr 9](#_Toc183197154)

[Figura 8 - ips.txt 11](#_Toc183197155)

[Figura 9 - Script para as iptables 11](#_Toc183197156)

[Figura 10 - iptables depois de correr o script 11](#_Toc183197157)

[Figura 11- Guardar permanentemente os iptables 12](#_Toc183197158)

[Figura 12 - iptables depois de reiniciar o sistema 12](#_Toc183197159)

[Figura 13 - Criar a pasta partilhada 17](#_Toc183197160)

[Figura 14 - Dar permissão de leitura para ficheiros na pasta partilhada 17](#_Toc183197161)

[Figura 15 - Criar um ficheiro de teste 17](#_Toc183197162)

[Figura 16 - Ficheiro de teste com permissão de leitura para todos 17](#_Toc183197163)

# Índice de Tabelas

[Tabela 1- Divisão das Tarefas 5](#_Toc183197275)

# Divisão Tarefas

Tabela 1- Divisão das Tarefas

|  |  |
| --- | --- |
| User Stories | Aluno |
| 1 | Francisco Osório |
| 2 | Diogo Ferreira |
| 3 | Rafael Ferraz |
| 4 | Sérgio Moreira |
| 5 | Francisco Osório |
| 6 | Diogo Ferreira |
| 7 | Rafael Ferraz |
| 8 | Sérgio Moreira |

# User Stories

## User Story 1

Nesta *user story* era solicitado o *deployment* de um dos módulos do *RFP* numa *VM* do *DEI* e que este seja sistemático, validando de forma agendada com o plano de testes.

Assim, decidiu-se que seria o módulo *backend* a ser *deployment* na *VM* do *DEI*, devido a ser o módulo que contém as funcionalidades principais da aplicação.

Inicialmente criou-se uma máquina *Debian12* nos servidores do *DEI*, e configurou-se a máquina instalando-se o nano para criar *scripts* e instalar o *dotnet*. Posteriormente, criou-se um diretório designado por *“/root/sarm”* através do comando *“mkdir /root/sarm”*. Após termos o diretório criado, criou-se um script, dentro do diretório anterior, que será responsável pelo *deployment* do respetivo módulo, através do comando “*nano* *deployment\_backend.sh*”.

Uma imagem com texto, eletrónica, captura de ecrã, software

Descrição gerada automaticamente

Figura 1 - Script deployment\_backend.sh

No *script* anterior, começa-se por verificar se existe algum processo a utilizar a porta 5001 (outro *deploy* anterior, por exemplo) através do comando “*fuser*”, e caso exista através do comando “*kill -9”* terminamos esse processo para libertar a porta para o novo *deploy*. Após isso entra-se no diretório que contém o *backend* (*“~sarm/dddnetcore”),* faz-se o *build* do módulo através do comando “*dotnet build”,* e corre-se os testes com o comando “*dotnet test*”. De seguida, redireciona-se o tráfego *TCP* da porta 2224 para a porta 5001, através do *iptables*, para ser possível aceder sem *vpn* á aplicação. Para ser possível visualizar quando são feitos *deploys* cria-se um diretório para os *logs* caso ele ainda não exista através do comando “*mkdir -p*", e guardando no ficheiro “*deployment\_backend.log”* a data do *deploy*. Depois pelo comando “*nohup dotnet run --urls "https://0.0.0.0:5001" > ~/sarm/logs/deployment\_backend\_output.log 2>&1 &*”, executa-se o *backend* em segundo plano na porta 5001, redirecionando toda a informação da consola para o ficheiro “*deployment\_backend\_output.log”.* Por fim, através do *“exit 0”* terminamos o script, deixando o *backend* a correr em segundo plano.

Posteriormente, executou-se o comando “*chmod +x deployment\_backend.sh*” para ser possível executar o script.

Depois de já temos o script que irá dar *deploy*, é preciso passar o conteúdo do módulo *backend* para a *VM* e para isso antes de tudo configurou-se umas *GitHub Secrets*, que será explicado mais á frente o intuito de cada uma.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 2 - Github Secrets

De seguida, criou-se um *workflow* designado “*deploy.yml*” (*Github Actions*) que irá automatizar o processo de *deployment* do *backend* na máquina virtual, e será acionado todos os sábados às 2 da manhã.

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Figura 3 - deploy.yml (workflow)

Explicando o *workflow* anterior, inicialmente utilizou-se o evento *schedule* com a expressão “*0 2 \* \* 6*” definindo assim que o workflow será apenas executado aos sábados às 2h da manhã. Após isso através “*actions/checkout@v3”* clona-se o repositório *GitHub* em um ambiente de execução, para ser possível aceder aos ficheiros do *backend*. Depois disso, configura-se um ficheiro *.env* para o *backend*, entrando-se no diretório *“dddnetcore”*, que é onde se encontra o *backend*. Esse ficheiro irá guardar as variáveis de ambiente necessárias para o funcionamento do *backend*, como configuração do *auth0*, da base de dados *MySQL* e do email *SMTP*. Na configuração deste ficheiro, foram utilizadas algumas das *secrets* que foram criadas anteriormente, como *“Auth0\_ClientId”, “Auth0\_ClientSecret”, “Database\_Password”, “Smtp\_From\_Email”, “SMTP\_PASSWORD”*, uma vez que consideradas são informações sensíveis. A próxima etapa refere-se a dar *upload* do *backend* para a *VM* utilizando a ação “*appleboy/scp-action@v0.1.5”* (utiliza protocolo *SCP*), definindo -se o *source* como “*dddnetcore*”, pois é onde se encontra o código do *backend* e o *target* como “~/sarm”, que foi o diretório criado inicialmente, sendo que o acesso à máquina é realizado via *SSH*, utilizando o *host, username, key*, da máquina que foram definidas nas *secrets* também *(“SERVER\_IP”, “SERVER\_USER”, “ SSH\_PRIVATE\_KEY”*). O último passo do *workflow* é executar o script que foi criado anteriormente, utilizando a mesma ação anterior (protocolo *SCP*), tendo acesso também via *SSH*, realizando assim o *deploy*.

Para testar, definiu-se um *schedule* diferente, para uma data e hora e obteve-se os seguintes resultados:

Uma imagem com captura de ecrã, texto, software, Software de multimédia

Descrição gerada automaticamente

Figura 4 - Resultado do workflow

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 5 - Ficheiro deployment\_backend.log com data do deploy

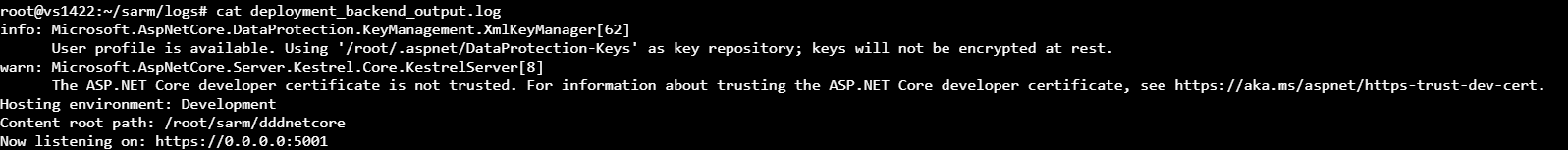


Figura 6 - Ficheiro deployment\_backend\_output.log com output do deploy

Uma imagem com captura de ecrã, Software de multimédia, texto, software

Descrição gerada automaticamente

Figura 7 - Teste de uma rota do backend para verificar que ele estava a correr

## User Story 2

## User Story 3

Para esta *user story*, temos de fazer com que seja possível definir os clientes do *DEI*, simplesmente mudando um ficheiro de texto.

Para isso, começamos por criar o tal ficheiro de texto, que vai conter os endereços permitidos. Fazemos isso com o comando “*nano ips.txt*”. Dentro do ficheiro colocamos o IP da máquina (linha 1) e o IP da rede interna do *DEI*, que os clientes usariam (linha 2):

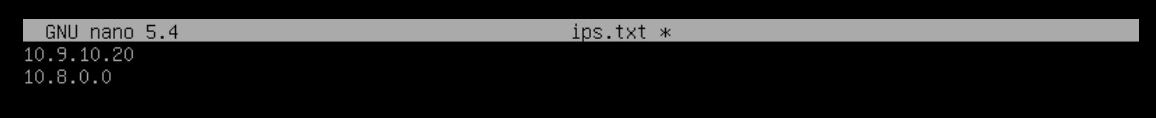


Figura 8 - ips.txt

Em seguida, criamos um pequeno script que permite acesso aos clientes presentes no “*ips*.*txt*”, usamos “*nano script\_us6-4-3.sh*” para criar o ficheiro, que contem um ciclo *for* para cada *ip* encontrado em “*ips.txt”*, adiciona uma regra no *iptables* que permite tráfego TCP na porta 22, que corresponde a *SSH. Finalmente, é adicionada uma regra que bloqueia o trafico de outras ligações*:

*Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente*

Figura 9 - Script para as iptables

Não nos podemos esquecer de dar permissão de execução ao ficheiro com o comando “*chmod +x script\_us6-4-3.sh*”. Agora, corremos o script com “*./script\_us6-4-3.sh*”. Podemos verificar que o script fez o pretendido:

Uma imagem com texto, captura de ecrã, Tipo de letra, preto

Descrição gerada automaticamente

Figura 10 - iptables depois de correr o script

Por último, devemos guardar as mudanças de modo a se manterem mesmo após reiniciar o sistema. Para isso, corremos “*sudo netfilter-persistent save*”

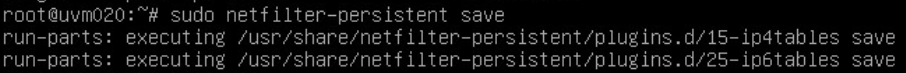


Figura 11- Guardar permanentemente os iptables

Como podemos ver abaixo, após reiniciar o sistema as configurações ainda estão presentes:

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 12 - iptables depois de reiniciar o sistema

## User Story 4

## User Story 5

Nesta user story era pedido que se defini-se o *MBCO (Minimum Business Continuity Objective)* a propor aos stakeholders.

O *Mininum Business Continuity Objective (MBCO)* especifica o nível mínimo de operacionalidade que deve ser mantida durante uma disrupção na infraestrutura.

A aplicação que está a ser desenvolvida encontra-se divida em vários módulos: o *backend*, o *frontend*, incluindo dentro o *3d visualization*, e o *planning*. Sendo que cada módulo tem uma função distinta e consequentemente maior importância, é necessário ter em atenção ao *Maximum Tolerable Downtime (MTD)* e ao *Maximum Tolerable Period of Disruption (MTPD)* para cada um.

Começando pelo módulo de *backend*, este é o responsável por processar e armazenar os dados, permitindo a gestão de pacientes, staffs, tipos de cirurgias, pedidos de cirurgias e marcação destas, sendo assim considerado o core da nossa aplicação. Desta forma, foi definido que o *MTD* para este módulo seria de 1 hora, e após esse tempo este módulo apresentará funcionalidades mínimas como gestão de pacientes, pedidos de cirurgias e suas marcações até ao *MTPD,* que foi definido para 12 horas.

Passando agora para o módulo de *frontend*, este é o que permite aos utilizadores interagirem com o sistema e realizarem funcionalidades como gestão de pacientes, staffs, tipos de cirurgias, pedidos de cirurgias e respetivas marcações, funcionalidades baseadas no módulo *backend*. Sendo assim, o *frontend* é considerado um módulo essencial para a continuidade das operações da aplicação. Assim, como no *backend*, definiu-se também um *MTD* de 1 hora. Após esse tempo, funcionalidades como gestão de pacientes, pedidos de cirurgias e suas marcações, estarão disponíveis, podendo estar outras ainda indisponíveis. Para recuperar todas as funcionalidades foi definido um período também de 12 horas, ou seja, para o *MTPD*.

O módulo *3d visualization* encontra-se integrado no *frontend*, e este permite aos utilizadores uma visão em tempo real do hospital, sendo possível ver os quartos e as respetivas operações que se encontram a decorrer. Apesar deste módulo ser útil, é secundário em relação aos módulos anteriores. Desta forma, foi definido um *MTD* de 6 horas, sendo que após este já deve ser possível os utilizadores verem um layout simplificado com o status de ocupação dos quartos, ainda com poucos detalhes. Já o *MTPD* foi definido para 48 horas, permitindo que este seja restaurado integralmente dentro desse prazo.

Por fim, o módulo *planning* é o que é responsável por gerar o agendamento das cirurgias e otimizar estes horários de acordo com diferentes critérios e disponibilidade de profissionais médicos e quartos. Este módulo também tem a sua importância, como a otimização, apesar disso, pode também ser considerado secundários, pois é possível realizar agendamentos manuais temporariamente. Assim, determinou-se que o tempo de inatividade deste módulo será de 3 horas, ou seja, o *MTD*. Após o *MTD* e até o *MTPD* a sua funcionalidade mínima, permitirá o agendamento básico de cirurgias, deixando otimização automática desativada temporariamente, sendo que o definiu-se o *MTPD* para 24 horas.

Concluindo, garante-se que as funcionalidades críticas da aplicação, continuem disponíveis, com tolerâncias razoáveis para restabelecimento de cada módulo em caso de disrupção, minimizando assim os impactos.

## User Story 6

## User Story 7

Nesta US era-nos pedido que se define uma pasta pública para todos os utilizadores registados no sistema, de modo a poderem ler o que lá está.

Primeiro, precisamos de criar a tal pasta, que depois será configurada como pública. Para isto usamos o comando “*mkdir*”:



Figura 13 - Criar a pasta partilhada

Em seguida, precisamos de mudar as permissões da pasta, o objetivo é que todos tenham permissão de leitura, para isso, temos de usar o comando “*chmod -R a+r /shared”*:



Figura 14 - Dar permissão de leitura para ficheiros na pasta partilhada

O comando “*chmod*” é usado para mudar as permissões de acesso, leitura ou execução. “*-R*” muda as permissões recursivamente (a tudo o que se encontra dentro do diretório), “*a+r*” indica que todos (a) devem receber (+) permissão de leitura (r), e depois metemos a nossa pasta. Podemos rapidamente ver se as permissões estão a funcionar corretamente criando um ficheiro de texto e correndo o comando “*ls -l*”:



Figura 15 - Criar um ficheiro de teste

Uma imagem com texto, Tipo de letra, captura de ecrã, preto

Descrição gerada automaticamente

Figura 16 - Ficheiro de teste com permissão de leitura para todos

O ficheiro criado tem permissões de leitura e escrita para o dono, e apenas de leitura para o grupo e para todos, tornando isto uma pasta que todos podem aceder.

## User Story 8