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

Descrição gerada automaticamente

Mestrado em Engenharia Informática

Desenvolvimento de Software Seguro

**Relatório Sprint 2**

Made by:

Francisco Oliveira, 1201545

Lourenço Teixeira, 1201399

Tomás Cancela, 1200985

Porto, maio 2024

Índice

[Melhorias 3](#_Toc168869166)

[ FASE 1 3](#_Toc168869167)

[ FASE 2 4](#_Toc168869168)

[Modificações no ASVS 6](#_Toc168869169)

[O que foi implementado 7](#_Toc168869170)

[ Logs 7](#_Toc168869171)

[ Dependency Check 8](#_Toc168869172)

[ Business Logic 8](#_Toc168869173)

[ Cookies 9](#_Toc168869174)

[ SonarQube 9](#_Toc168869175)

[ Atributo “Consumes” 10](#_Toc168869176)

[ Malicious Code 10](#_Toc168869177)

[ SBOM (configuration) 11](#_Toc168869178)

[ Separação de Pipelines 11](#_Toc168869179)

[ *Password Breach Validation* 12](#_Toc168869180)

[ *Strong Passwords* 13](#_Toc168869181)

[ Visualização Temporária de *Passwords* 14](#_Toc168869182)

[ Verificação Captcha no *Login* 14](#_Toc168869183)

[ Esconder *Secrets* em Variáveis de Ambiente 15](#_Toc168869184)

[ HTTPS (13.2.6) 15](#_Toc168869185)

[ Outras Pequenas Melhorias 17](#_Toc168869186)

[Erros/problemas da aplicação 17](#_Toc168869187)

## Melhorias

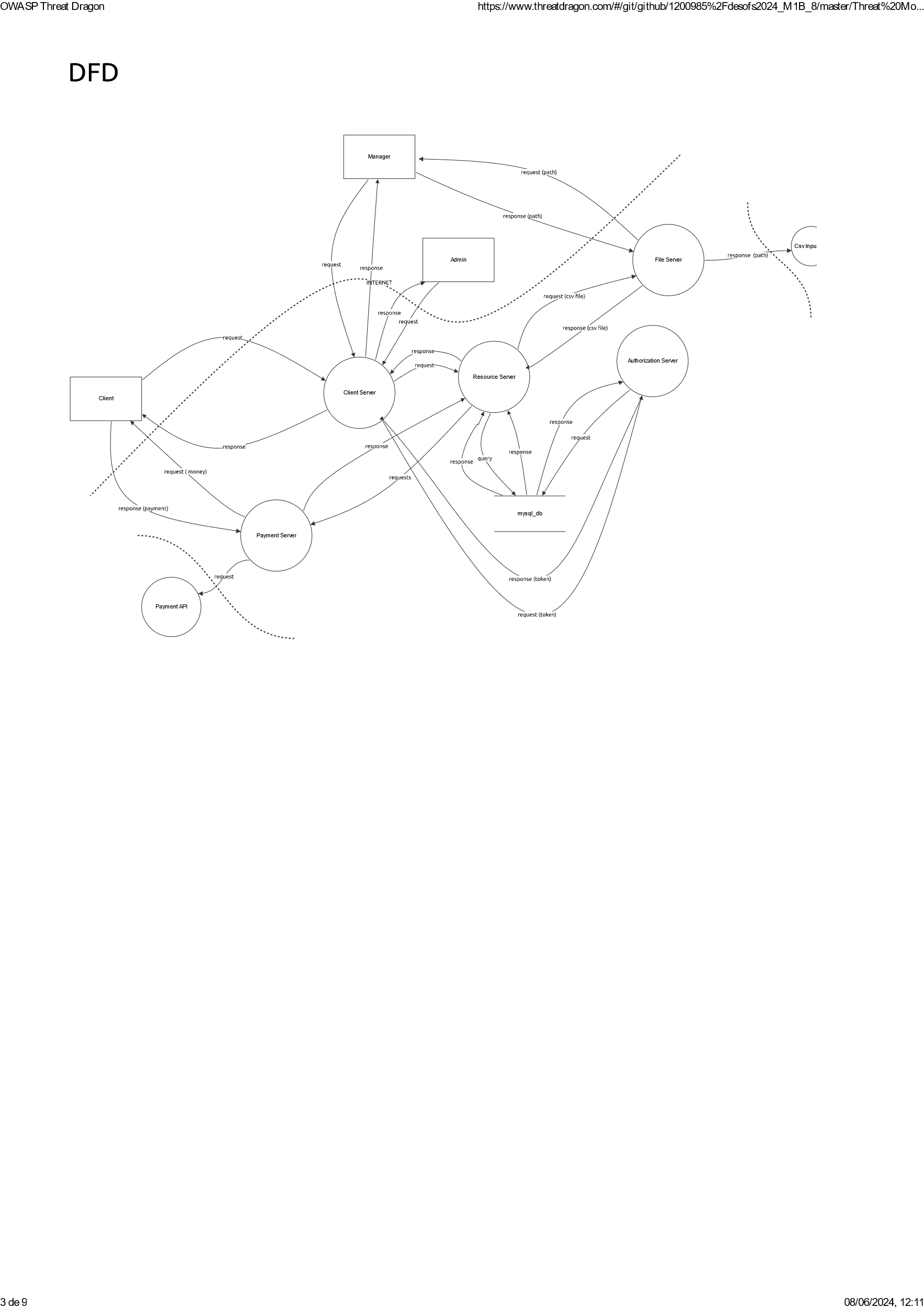
### FASE 1

Em relação à fase 1 mudamos a vista Física de nível 2.

Uma imagem com texto, ecrã, diagrama, captura de ecrã

Descrição gerada automaticamente

Também foi alterado o data flow diagram, para um diagrama da aplicação em geral. Anteriormente estava feito um data flow diagram para cada use case, o que não fazia muito sentido pois a maioria dos elementos repetia-se.



Também foram alterados os abuses cases e os requirements de forma a ficar mais coeso e coerente com a aplicação.

### FASE 2

#### Sprint 1

Num momento inicial deste sprint, foram feitas certas alterações à *pipeline*, nomeadamente:

* **Adicionar *dependency checking***

No *workflow* “develop and rest of branches”, encontramos a *dependency check* implementada. Este processo automatiza a deteção de vulnerabilidades nas dependências do projeto, melhorando a segurança e a qualidade do software.

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

Descrição gerada automaticamente

* **Ativar o Dependabot no GitHub.**

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

Descrição gerada automaticamente

O Dependabot do GitHub é uma ferramenta integrada que ajuda a manter as dependências de um projeto atualizadas automaticamente. Automatiza a gestão de dependências, garantindo que o projeto utilize versões atualizadas e seguras, o que economiza tempo e reduz o risco de vulnerabilidades de segurança.

* **Foi Implementado o SonarQube** **na pipeline**

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

Descrição gerada automaticamente

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

Descrição gerada automaticamente

O SonarQube é uma plataforma de código aberto usada para realizar análises contínuas da qualidade do código em projetos de software. Avalia vários aspetos do código, como bugs, vulnerabilidades, code smells cobertura de testes, duplicação de código...

Neste caso o SonarQube está a ser usado para analisar e avaliar a qualidade do código de diferentes componentes do projeto.

## Modificações no ASVS

Tudo o que está destacado a verde no Excel foi alterado para válido nesta sprint. Para isso, tivemos de implementar algumas coisas que discutiremos no próximo tópico.

Exemplo:

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

Descrição gerada automaticamente

Por outro lado, tudo o que se encontra a castanho foram tópicos que tentamos implementar, mas tornou-se impossível a sua realização devido ao projeto base.

Exemplo:

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

Descrição gerada automaticamente

Já a verde mais claro são tópicos que foram alterados em relação aos sprints anteriores mas que não foi necessária implementação.

Exemplo:



## O que foi implementado

### Logs

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamenteFoi implementado o sistema de logging para garantir que a aplicação não registe outros dados sensíveis. Além disso, o sistema regista eventos relevantes de segurança, incluindo eventos de autenticação bem-sucedidos e falhos, falhas de controle de acesso, falhas de desserialização e falhas de validação de entrada. Cada evento de log inclui as informações necessárias para permitir uma investigação dos eventos. Todas as decisões de autenticação são registadas sem armazenar identificadores de sessão sensíveis ou senhas.

Figura 1 - logback-spring.xml

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Figura 2 - CustomAuthenticationSucessHandler.java

Da checklist do asvs passaram então a válidos os seguintes:

1.7.1;7.1.2; 7.1.4; 7.2.1; 7.2.2

### Dependency Check

Já referido anteriormente a dependency check foi implementada também neste sprint. Isso fez com que os seguintes tópicos da checklist passassem a válidos:

14.2.1

### Business Logic

Para verificar que a aplicação processa fluxos de lógica de negócio com todos os passos processados em tempo realista para humanos, ou seja, que as transações não são submetidas rapidamente demais, foi implementado um sistema de controle de fluxo nas transações (11.1.2).

Por exemplo quando o utilizador dá checkout na order demora no mínimo 2 segundos a responder se foi aceite ou não.

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Para verificar se a aplicação possui limites apropriados para ações ou transações específicas do negócio, foi implementado os limites apropriados para as ações (11.1.3).

Por exemplo, só é possível fazer no máximo 5 compras por hora.

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

### Cookies

Para verificar se os tokens de sessão baseados em cookies possuem os atributos 'Secure' e 'SameSite' corretamente configurados implementou -se:

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

Descrição gerada automaticamente

1. Atributo 'Secure':

Objetivo: Garantir que os cookies de sessão sejam transmitidos apenas por conexões HTTPS, protegendo-os contra intercetação em conexões não seguras.

1. Atributo 'SameSite':

Objetivo: Proteger contra ataques de Cross-Site Request Forgery (CSRF) limitando a capacidade dos cookies de serem enviados junto com solicitações cross-site.

Da checklist do asvs passaram então a válidos os seguintes:

3.4.1; 3.4.3

### Uma imagem com texto, captura de ecrã, software, Página web Descrição gerada automaticamenteSonarQube

Neste projeto, o SonarQube é uma ferramenta essencial utilizada para garantir a qualidade do código e a segurança das aplicações através da análise contínua de diversos componentes, como o resource server, o authorization server e o cliente Angular.

1. Resource Server

O resource server utiliza o JaCoCo, uma biblioteca Java que gera relatórios de cobertura de testes. Durante o processo de build, o JaCoCo recolhe dados sobre os testes executados e cria um relatório detalhado. Este relatório é posteriormente usado pelo SonarQube para apresentar a cobertura de código de forma visual e detalhada.

1. Cliente Angular

O Angular utiliza o Karma, um test runner que, ao executar os testes, gera um relatório de cobertura de código. Este relatório é gerado pelo script definido no package.json sob o comando test:nowatch, que cria uma pasta chamada coverage contendo um arquivo lcov.info. O SonarQube, durante a análise do projeto Angular, importa o arquivo lcov.info para apresentar a cobertura de código dos testes no dashboard do SonarQube.

Além de analisar a cobertura de código, o SonarQube também realiza diversas verificações automáticas em relação a questões de qualidade e segurança do código como: deteção de issues, hotspots, relatórios…

Da checklist do ASVS passaram então a válidos os seguintes tópicos:

10.1.1 ; 14.1.5

### Atributo “Consumes”

Para verificar que os serviços REST verificam explicitamente se o Content-Type recebido é o esperado pode-se utilizar a anotação @PostMapping ou @RequestMapping com o atributo consumes. Isso garante que o endpoint só aceitará solicitações com o tipo de conteúdo específico, fornecendo uma camada adicional de segurança e validação de entrada.

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

Descrição gerada automaticamente

### Malicious Code

Para verificar se a ferramenta de análise de código está em uso para detetar código potencialmente malicioso, foi utilizado o SonarQube juntamente com o AWS. (10.1.1)

Para verificar se a aplicação e as bibliotecas de terceiros não contêm código malicioso, como ataques salami, bypasses lógicos ou bombas lógicas, e garantir outras medidas de segurança, foi analisado o código manualmente e podemos concluir que não existe código malicioso, e que todo o código foi escrito com o propósito que é suposto ter. Para além disso verificamos que o código é carregado de repositórios confiáveis. (10.2.5; 10.2.6; 10.3.2)

### SBOM (configuration)

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Para verificar que uma SBOM é mantida para todas as bibliotecas de terceiros em uso, utilizamos o CycloneDX. (14.2.5)

O CycloneDX é uma especificação para a criação de SBOMs, que documenta todas as bibliotecas de terceiros utilizadas na aplicação, incluindo as versões, origens e quaisquer vulnerabilidades conhecidas.

Ao gerar o SBOM, utilizamos ferramentas de análise para monitorizar continuamente as bibliotecas documentadas, verificando atualizações e possíveis vulnerabilidades. Isso permite identificar e mitigar riscos associados ao uso de software de terceiros de maneira proativa.

### Separação de Pipelines

Foram feitas 2 pipelines, uma para a *branch “develop”* e outra para a *branch “master*” (*production*).

A diferença entre elas é que na pipeline da procution é feito o publish das imagens no dockerHub.

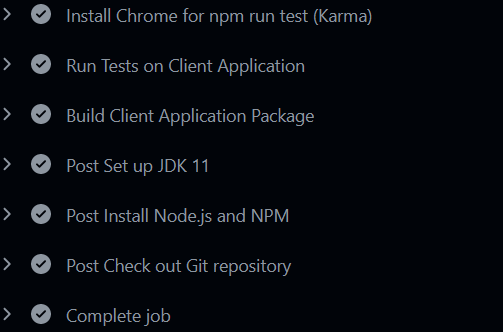


Figura 3 - Pipeline Develop



Figura 4 - Pipeline Production

#### Quando é que a pipeline dá trigger:

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

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

### *Password Breach Validation*

Para melhorar a segurança da criação de *passwords*, implementámos uma funcionalidade que verifica se uma palavra-passe recém-criada se encontra entre as palavras-passe violadas, listadas no “Have I Been Pwned” (HIBP). Desta forma, evita-se a criação e armazenamento de *passwords* anteriormente expostas em violações de dados, reduzindo o risco de acesso não autorizado.

Este processo envolve o envio de um prefixo de *hash* SHA-1 da *password* para a API disponibilizada pelo HIBP. A API responde a este *request* com uma lista de possíveis sufixos, e verifica-se se o sufixo da *password* faz parte da lista. Este mecanismo evita que a *hash* da *password* seja enviada na sua totalidade à API, aumentando a segurança envolvida neste processo. (2.1.7)

### *Strong Passwords*

Para ajudar aos utilizadores a terem uma password considerada forte foram implementadas algumas restrições a novas *passwords*: (2.1.8)

* Pelo menos:
  + 12 carateres
  + uma letra maiúscula
  + uma letra minúscula
  + um dígito numérico
  + Uma imagem com texto, captura de ecrã, Tipo de letra, número

    Descrição gerada automaticamenteum carater especial.

### Visualização Temporária de *Passwords*

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

Descrição gerada automaticamenteForam acrescentados botões aos *inputs* de *password* para visualização temporária do conteúdo. Quando o utilizador carrega no botão de visualização, o tipo de *input* passa de *password* para *text*, até ao momento em que o mesmo soltar o botão. (2.1.12)

### Verificação Captcha no *Login*

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

Descrição gerada automaticamenteDe forma a impedir ataques de *account lockout* e de *brute force*, foi implementada uma verificação *captcha* ao *login* do utilizador. A ferramenta escolhida foi o “Google reCAPTCHA v2”, que permite, de forma facilitada, incluir uma destas verificações no código *html* da aplicação. Além da inclusão no *html*, o valor de resposta da API disponibilizada pela Google é enviada para o *authorization server*, que depois faz um novo *request* a esta API para verificar que a resposta é valida, consoante o *secret* associado ao domínio da aplicação. (2.2.1)

### Esconder *Secrets* em Variáveis de Ambiente

Anteriormente, todas a informações sensíveis (*secrets*) estavam totalmente expostas no código, o que representava uma falha de segurança grave. Assim, foram criados ficheiros de ambiente, e armazenou-se todos os *secrets* nos mesmos. (2.10.4)

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

Descrição gerada automaticamenteFicheiro .env *authorization server*:

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

Descrição gerada automaticamenteFicheiro .env *resource server*:

### HTTPS (13.2.6)

As aplicações estavam a utilizar o protocolo HTTP, no qual os dados não são encriptados. Com isto, alguém mal-intencionado poderia roubar os dados transmitidos entre as aplicações, sendo uma falha de segurança gravíssima. Tendo isto em conta, foi implementado o HTTPS, a versão segura do HTTP que utiliza TLS/SSL para a encriptação dos dados. Este foi implementado tanto no front-end como no back-end, como se pode observar nas seguintes figuras:

Uma imagem com texto, captura de ecrã, software, Sistema operativo

Descrição gerada automaticamente

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

Descrição gerada automaticamente

No frontend foram criados os ficheiros chave e certificado através do comando ```openssl req -new -x509 -newkey rsa:2048 -sha256 -nodes -keyout localhost.key -days 3560 -out localhost.crt```

Para tal, foi instalado o openSSL através do winget.

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

Descrição gerada automaticamente

Após isso, foi só necessário associar estes ficheiros ao serve do angular, através das opções sslKey e sslCert. Ao executar a aplicação, foi necessário passar o parâmetro –ssl a true.

No back-end, foi necessário gerar um self signed certificate através do seguinte comando: ```keytool -genkeypair -alias resource\_server -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore resource\_server.p12 -validity 3650 ```

Depois é feita uma referência a este ficheiro através do application.yml, onde se obteve a seguinte configuração:

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

Descrição gerada automaticamente

### Outras Pequenas Melhorias

Um dos requisitos de segurança que não estava contemplado anteriormente correspondia à gestão de sessão de login, mais propriamente ao tempo de validade de uma única sessão de utilizador. Foi alterado o tempo de expiração de 365 dias (valor anterior) para apenas 30 dias, segundo o Nível 1 da norma OWASP CWE613-7.2. (3.3.2)

## Erros/problemas da aplicação

A aplicação estava muito desatualizada e continha alguns métodos *deprecated*,o que dificultou bastante a implementação de certas funcionalidades e melhorias.