

Editor de Documentos Colaborativo Seguro

Segurança em Engenharia de *Software*

Grupo EMA

André Filipe Fernandes Serra – up201804979

Eduardo Luís Fernandes Roçadas – up202108758

Manuel Ramos Leite Carvalho Neto – up202108744

Mestrado em Segurança Informática

2024/2025

Índice

Introdução	4
<i>Design</i>	5
Requisitos	5
Arquitetura	8
Atores.....	9
<i>Use, Misuse e Abuse Cases</i>	10
<i>Use Cases</i>	11
<i>Misuse Cases</i>	12
<i>Abuse Cases</i>	13
Modelação de Ameaças.....	14
Implementação.....	17
CryptPad	18
Configuração	20
HTTPS	22
Execução	22
Análise	27
Linguagens de Programação	27
<i>Design Patterns</i>	28
Dependências e Bibliotecas.....	31
Mitigações para Vulnerabilidades Comuns	33
Metodologias de Teste de Segurança	35
Ferramentas de Análise de Código	37
CodeQL	37
Semgrep	39
SonarQube Cloud	41
Análise Global.....	43
Lições Aprendidas	44
Conclusão	45

Índice de Figuras

Figura 1 – Diagrama de Classes	7
Figura 2 – Diagrama de Arquitetura	8
Figura 3 – Diagrama de Atores.....	9
Figura 4 – Diagrama de Use Cases	11
Figura 5 – Diagrama de Fluxo de Dados	15
Figura 6 – Página Inicial.....	23
Figura 7 – Registo	23
Figura 8 – Autenticação Multifator.....	24
Figura 9 – Criação de Documento	24
Figura 10 – Documento.....	25
Figura 11 – Partilha de Documento por Link.....	25
Figura 12 – Partilha de Documento com Utilizadores.....	26
Figura 13 – Proprietário de Documento	26
Figura 14 – Arquitetura do CryptPad.....	30
Figura 15 – npm audit	31
Figura 16 – Snyk.....	32
Figura 17 – Dependabot.....	32
Figura 18 – OWASP Zap.....	36
Figura 19 – CodeQL	37
Figura 20 – Semgrep	39
Figura 21 – SonarQube Cloud	41
Figura 22 – SonarQube Cloud (Detalhe)	42

Índice de Tabelas

Tabela 1 – Requisitos de Interface.....	6
Tabela 2 – Requisitos de Autenticação	6
Tabela 3 – Requisitos de Controlo de Acessos.....	7
Tabela 4 – Descrição dos Atores	10
Tabela 5 – Modelo STRIDE.....	14
Tabela 6 – Ameaças à Aplicação Web	15
Tabela 7 – Ameaças à API	16
Tabela 8 – Ameaças à Base de Dados	16
Tabela 9 – Análise Global.....	43

Introdução

No âmbito da Unidade Curricular Segurança em Engenharia de *Software*, propõe-se o desenvolvimento de um projeto que consiste num editor de documentos colaborativo seguro. Nesse sentido, este relatório visa documentar todas as etapas integrantes deste processo de desenvolvimento, dividindo-o em três fases.

Em primeiro lugar, define-se o *design* dos componentes de *software* necessários para a plataforma de edição colaborativa, juntamente com os mecanismos de segurança adequados para satisfazer os requisitos do projeto. Para tal, estabelece-se a arquitetura pretendida para o sistema, os atores intervenientes no mesmo e os *use*, *misuse* e *abuse cases* de maneira a identificar ameaças de segurança e assunções antes de iniciar a implementação propriamente dita.

Em segundo lugar, procede-se à implementação do sistema proposto, instanciando o *design* definido anteriormente. Para o efeito, opta-se pela utilização de um projeto *open-source* já existente, devidamente adaptado, estendido e configurado para satisfazer os requisitos funcionais e de segurança delineados. Com o intuito de a plataforma desenvolvida ser tão segura quanto possível, ponderam-se diferentes opções de projetos *open-source* para a implementação, comparando as suas vantagens e desvantagens de modo a escolher aquele que mais fielmente cumpre o *design* pretendido para o sistema, tanto em termos de segurança, como em termos de funcionalidade.

Por último, demonstra-se que a implementação respeita o *design* e cumpre os requisitos estabelecidos, seguindo um processo de análise aprofundada. Assim, descreve-se a metodologia de análise de segurança empregue para validar a implementação e corroborar a segurança global do sistema, detalhando-se não só os procedimentos seguidos pelo próprio projeto *open-source* já implementado, mas também as análises adicionais efetuadas para o desenvolvimento deste projeto.

Em todos os momentos, procura-se justificar detalhadamente todas as decisões tomadas tanto quanto possível. Para o efeito, utilizam-se algumas ferramentas e técnicas posteriormente introduzidas, que atuam como elementos essenciais para o processo de análise.

Em suma, este relatório pretende documentar todo o processo de desenvolvimento do *software* em causa, destacando, principalmente, o *design* e a análise, mas não descurando a implementação. Deste modo, segue-se uma metodologia enquadrada em *SecDevOps*, colocando a segurança em primeiro lugar.

Design

O sistema a desenvolver enquadra-se no contexto de uma organização de segurança em *software* que pretende alojar uma plataforma de edição colaborativa de documentos. O propósito desta plataforma passa por disponibilizar os seus tutoriais de formação interna sobre segurança aos novos membros da organização, bem como anunciar tutoriais públicos aos utilizadores dos produtos comercializados pela empresa. Portanto, evidentemente, a plataforma deve não só ser segura, mas também colocar a segurança como uma prioridade, de modo a manter a reputação positiva da empresa no mercado.

Assim, o processo de desenvolvimento deve iniciar-se por estabelecer os requisitos necessários ao correto funcionamento da plataforma, que funcionam como ponto de partida para a definição de uma arquitetura adequada e para a clarificação dos atores do sistema, juntamente com os respetivos *use*, *misuse* e *abuse cases*, culminando na explicação das ameaças de segurança contempladas.

Requisitos

A plataforma de edição colaborativa de documentos deve ser desenvolvida sob a forma de uma aplicação *web* moderna, dividida em dois componentes: cliente (*frontend*) e servidor (*backend*).

No *frontend*, a plataforma deve disponibilizar uma página *web* que permite aos seus utilizadores criar, gerir e configurar documentos, de acordo com as suas permissões e papéis na organização. Deste modo, permite-se que outros utilizadores e o público em geral possa consultar e navegar nos conteúdos disponibilizados. O *frontend* deve ser simples, apelativo e fácil de utilizar, sem funcionalidades desnecessárias, garantindo usabilidade e segurança.

No *backend*, deve existir um servidor que exponha *endpoints* de uma *Application Programming Interface* (API) segundo uma arquitetura REST, utilizada pela página *web* anteriormente descrita. Com isto, possibilita-se também que utilizadores mais avançados possam comunicar diretamente com os *endpoints* da API, facilitando e diversificando os meios de utilização da plataforma. O *backend* deve ser robusto, seguro e fácil de colocar em produção.

De acordo com esta especificação de alto-nível da aplicação, podem ser detalhados os requisitos funcionais, divididos em interface, autenticação e controlo de acessos.

Os requisitos relativos à interface apresentam-se na Tabela 1.

ID	Descrição
I1	Os utilizadores devem ser capazes de criar documentos
I2	Os utilizadores devem ser capazes de configurar documentos
I3	Os utilizadores devem ser capazes de editar documentos
I4	Os utilizadores devem ser capazes de eliminar documentos
I5	Um documento deve consistir em múltiplas secções e parágrafos
I6	Os utilizadores devem ser capazes de editar colaborativamente um documento em tempo-real
I7	Os utilizadores devem ser capazes de ver as alterações de outros
I8	Os autores dos documentos devem ter controlo sobre as permissões
I9	Os documentos devem poder ser públicos (visíveis ou editáveis por todos os utilizadores)
I10	A plataforma deve ser suportada por uma REST API no <i>backend</i>

Tabela 1 – Requisitos de Interface

Os requisitos correspondentes à autenticação apresentam-se na Tabela 2.

ID	Descrição
A1	Os utilizadores não anónimos têm de se autenticar para aceder à plataforma
A2	Os utilizadores anónimos só podem aceder a conteúdo público
A3	A plataforma deve ter um método mínimo de autenticação
A4	A plataforma deve comunicar de forma segura (HTTPS)
A5	A plataforma deve encriptar documentos <i>end-to-end</i> ou no servidor

Tabela 2 – Requisitos de Autenticação

Finalmente, os requisitos de controlo de acessos expõem-se na Tabela 3.

ID	Descrição
CA1	O sistema deve suportar <i>Role-Based Access Control</i>
CA2	O sistema deve suportar papéis de proprietário, editor e leitor
CA3	Só os utilizadores com o papel de proprietário devem ser capazes de configurar as permissões do documento ou eliminá-lo
CA4	O sistema deve suportar permissões granulares
CA5	Os utilizadores devem ser capazes de restringir o acesso a todo o documento ou a certas secções ou parágrafos do documento

Tabela 3 – Requisitos de Controlo de Acessos

Com isto, os requisitos do sistema a desenvolver estão delineados.

Complementarmente, com o intuito de esclarecer o domínio do sistema a desenvolver, deve definir-se o modelo conceptual de dados, que procura elencar e caracterizar os elementos significativos do sistema (classes) e a forma como eles se relacionam entre si (associações). Para o efeito, representa-se, na Figura 1, o diagrama de classes do sistema, que inclui as entidades/classes mais relevantes, as associações entre elas e a respetiva multiplicidade.

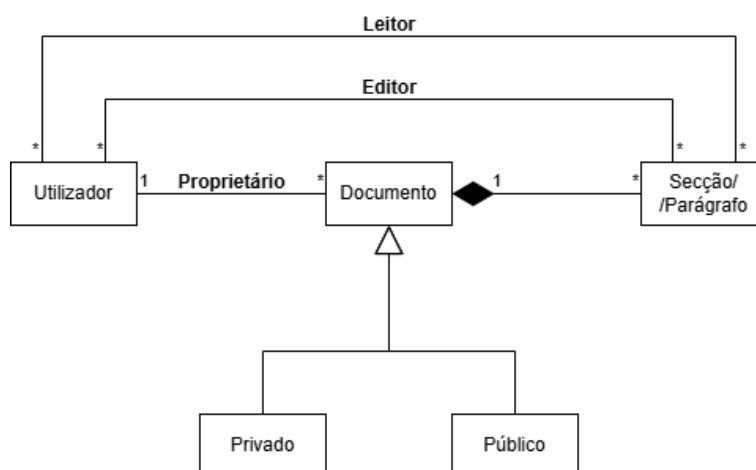


Figura 1 – Diagrama de Classes

Desta forma, o diagrama da Figura 1 destaca que cada documento, constituído por várias secções/parágrafos, só pode ter um proprietário, mas pode ter vários leitores e editores, ao nível das secções. Além disto, tal como requerido, cada documento pode ser considerado público ou privado.

Assim, estabelece-se uma visão de alto-nível do sistema pretendido.

Arquitetura

A arquitetura do sistema deve permitir satisfazer os requisitos estabelecidos, garantindo, simultaneamente, as funcionalidades pretendidas e a segurança necessária da plataforma.

De acordo com os requisitos explicitados, o sistema deve ser dividido em duas partes: cliente e servidor. O cliente deve ser constituído por uma aplicação *web* moderna, sob a forma de um *site* com uma ou várias páginas, enquanto o servidor deve contemplar uma API segundo a arquitetura REST. Além disso, o servidor deve conter, também, uma solução e um mecanismo para armazenamento dos dados, nomeadamente a informação sobre os utilizadores do sistema e os documentos por eles criados.

A comunicação entre o cliente e a API do servidor deve ser efetuada por *HyperText Transfer Protocol Secure* (HTTPS), garantindo segurança contra ataques de *Man-in-the-Middle*, no qual as comunicações podem ser interceptadas por *eavesdroppers*. A par disto, os documentos devem ser armazenados no servidor de forma encriptada, de maneira a permanecerem protegidos/seguros contra eventuais fugas de informação.

Esta definição de alto-nível do sistema representa-se na Figura 2, num diagrama de arquitetura do sistema para produção.

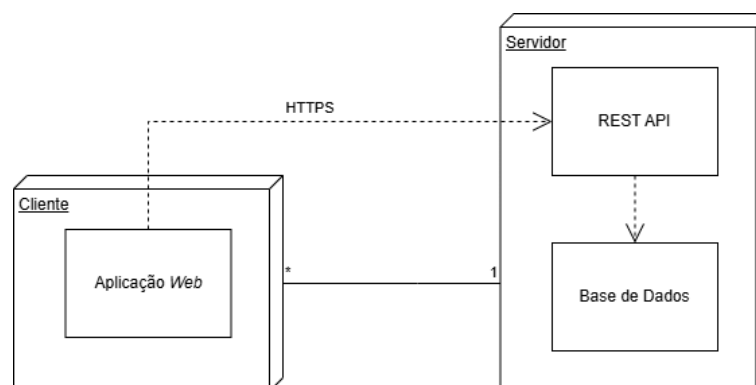


Figura 2 – Diagrama de Arquitetura

Ora, como se evidencia na Figura 1, o sistema deve ser simples, respeitando o princípio *Keep It Simple, Stupid!* (KISS), visto que aumentar a complexidade do sistema aumenta a propensão para a introdução de vulnerabilidades de segurança. Efetivamente, a interação dos clientes com o servidor deve ser via API, sem acesso direto à base de dados e com comunicação por HTTPS. A base de dados do sistema deve conter toda a informação necessária para o seu funcionamento, tanto a nível dos utilizadores (para a sua autenticação), como a nível dos documentos.

Com isto, pretende-se que o sistema ofereça proteção e segurança contra atacantes externos do lado do cliente e/ou utilizadores maliciosos, admitindo que o servidor é considerado seguro.

Atores

No sentido de clarificar as partes envolvidas no sistema, devem mapear-se os atores e os respetivos papéis, funcionalidades e responsabilidades.

Em conformidade com os requisitos descritos, o sistema deve suportar utilizadores não autenticados (anónimos) e autenticados (não anónimos), com diferentes permissões associadas. Dentro dos utilizadores autenticados, um utilizador – no contexto de um dado documento – pode ser considerado leitor, editor ou proprietário do mesmo, o que também lhe confere diferentes tipos de permissões.

O diagrama de atores da Figura 3 representa esta hierarquia de utilizadores.

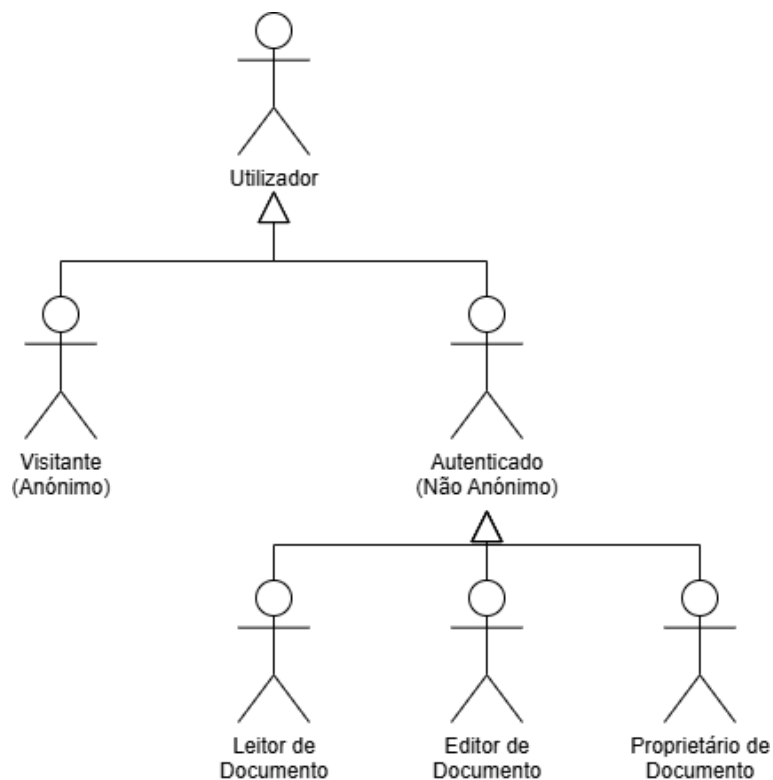


Figura 3 – Diagrama de Atores

O diagrama de atores da Figura 3 explicita a generalização entre atores, a partir de um utilizador genérico inicial.

A Tabela 4 identifica as funções e as permissões de cada ator, servindo como justificação dos diferentes papéis considerados.

Ator	Descrição
Utilizador	Utilizador genérico, só capaz de aceder a conteúdo público, para ver ou editar, conforme as permissões
Visitante (Anónimo)	Utilizador não autenticado, que pode criar uma conta ou entrar numa conta do sistema
Autenticado (Não Anónimo)	Utilizador autenticado, que pode criar documentos
Leitor de Documento	Utilizador autenticado, com o poder de ler o documento privado
Editor de Documento	Utilizador autenticado, com o poder de ler e editar o documento privado
Proprietário de Documento	Utilizador autenticado, com o poder de ler, editar, eliminar e configurar as permissões do documento

Tabela 4 – Descrição dos Atores

Com isto, os atores do sistema estão devidamente definidos.

Use, Misuse e Abuse Cases

A partir dos atores previamente mapeados, é possível passar para a definição de *use*, *misuse* e *abuse cases*.

Enquanto os requisitos descrevem o que o sistema deve ou não fazer e quem interage com o sistema, os *use cases* especificam fluxos de ações que descrevem os cenários principais de utilização da plataforma por parte dos atores. Por sua vez, os *misuse* e *abuse cases* especificam cenários de *use cases* nos quais um ator compromete o sistema, seja de forma não intencional, seja de forma intencional, respetivamente.

Assim, ao questionar as assunções do sistema e modificar o foco para aquilo que cada ator pode fazer e não para o que vai fazer, torna-se possível clarificar os requisitos de segurança do sistema a desenvolver.

Use Cases

O diagrama da Figura 4 ilustra visualmente os *use cases* do sistema, legendados posteriormente.

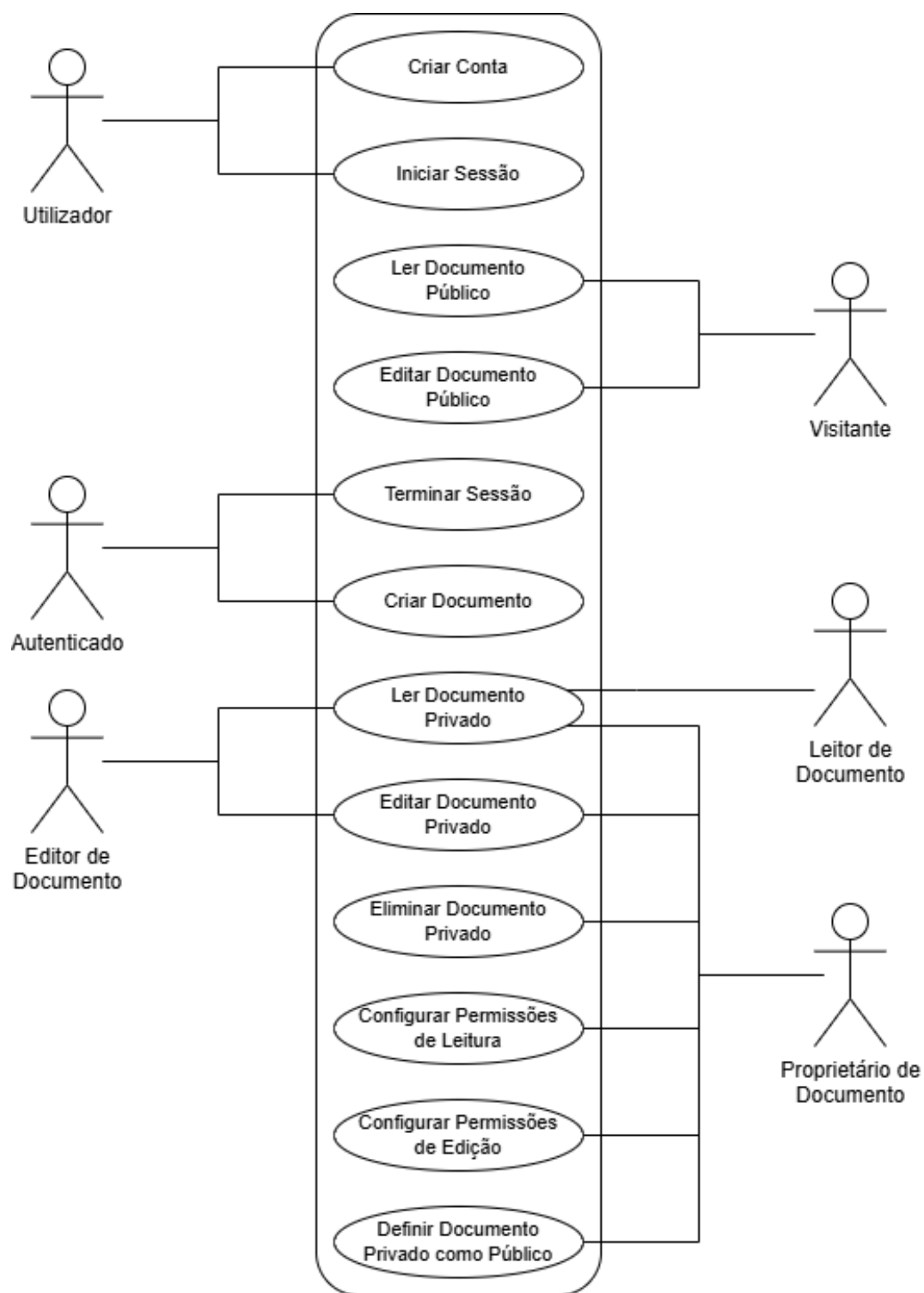


Figura 4 – Diagrama de Use Cases

De acordo com a Figura 4, os *use cases* associados aos atores do sistema são:

1. Um Utilizador cria uma conta na plataforma;
2. Um Utilizador inicia sessão na plataforma;
3. Um Visitante lê um documento público;
4. Um Visitante edita um documento público;
5. Um Utilizador Autenticado termina sessão na plataforma;
6. Um Utilizador Autenticado cria um documento;
7. Um Leitor de Documento lê o documento privado;
8. Um Editor de Documento lê o documento privado;
9. Um Editor de Documento edita o documento privado;
10. Um Proprietário de Documento lê o documento privado;
11. Um Proprietário de Documento edita o documento privado;
12. Um Proprietário de Documento elimina o documento privado;
13. Um Proprietário de Documento configura permissões de leitura do documento privado;
14. Um Proprietário de Documento configura permissões de edição do documento privado;
15. Um Proprietário de Documento define um documento privado como público.

Com isto, estão definidos os usos considerados legítimos para o sistema a desenvolver, de acordo com os requisitos previamente definidos e as funcionalidades pretendidas.

Misuse Cases

Os *misuse cases* – ações contrárias ao expectável no sistema, mas sem intuito malicioso – considerados no sistema proposto são:

1. Um Visitante cria um documento;
2. Um Visitante lê um documento privado;
3. Um Visitante edita um documento privado;
4. Um Visitante elimina um documento privado;
5. Um Visitante configura permissões de leitura de um documento privado;
6. Um Visitante configura permissões de edição de um documento privado;
7. Um Leitor de Documento edita o documento privado;
8. Um Leitor de Documento elimina o documento privado;
9. Um Leitor de Documento configura permissões de leitura do documento privado;
10. Um Leitor de Documento configura permissões de edição do documento privado;

11. Um Editor de Documento elimina o documento privado;
12. Um Editor de Documento configura permissões de leitura do documento privado;
13. Um Editor de Documento configura permissões de edição do documento privado.

Ora, embora estes casos não representem ações com intenção maliciosa, os comportamentos expostos devem ser prevenidos através de mecanismos de segurança.

Abuse Cases

Os *abuse cases* definidos para a plataforma são:

1. Um Utilizador cria uma conta na plataforma em nome de outro Utilizador;
2. Um Utilizador inicia sessão na plataforma em nome de outro Utilizador;
3. Um Visitante acede diretamente a um documento privado;
4. Um Visitante ludibria/engana um Proprietário de Documento para eliminar o documento privado;
5. Um Visitante ludibria/engana um Proprietário de Documento para configurar permissões de leitura do documento privado;
6. Um Visitante ludibria/engana um Proprietário de Documento para configurar permissões de edição do documento privado;
7. Um Visitante ludibria/engana um Proprietário de Documento para definir o documento privado como público;
8. Um Utilizador Autenticado cria um documento em nome de outro Utilizador;
9. Um Utilizador Autenticado edita um documento em nome de outro Utilizador;
10. Um Utilizador Autenticado injeta um *script* num documento;
11. Um Utilizador Autenticado cria demasiados documentos num curto intervalo de tempo, comprometendo a disponibilidade do sistema;
12. Um Utilizador Autenticado edita demasiados documentos num curto intervalo de tempo, comprometendo a disponibilidade do sistema;
13. Um Utilizador Autenticado torna-se o proprietário de um documento não criado por ele.

Deste modo, o comportamento dos utilizadores na plataforma encontra-se devidamente modelado em termos daquilo que deve e não deve ser feito, pelo que permite clarificar os requisitos de segurança necessários.

Modelação de Ameaças

De modo a concluir o processo de *design*, as ameaças ao sistema devem ser modeladas. Para o efeito, recorre-se ao modelo STRIDE, desenvolvido pela Microsoft, que permite clarificar as ameaças com base em seis categorias, como se expõe na Tabela 5.

Ameaça	Propriedade	Definição
<i>Spoofing</i>	Autenticação	Pretender ser alguém que não o próprio
<i>Tampering</i>	Integridade	Modificar informação indevidamente
<i>Repudiation</i>	Não-Repúdio	Refutar/Negar uma ação efetuada
<i>Information Disclosure</i>	Confidencialidade	Obter informação indevidamente
<i>Denial of Service</i>	Disponibilidade	Exaustar os recursos disponíveis
<i>Elevation of Privilege</i>	Autorização	Realizar algo sem autorização

Tabela 5 – Modelo STRIDE

Ora, considerando estas categorias de ameaças, o modelo STRIDE estabelece que o ponto de partida deve ser a definição de um diagrama de fluxo de dados – contendo os processos, as entidades externas que interagem com o sistema, o armazenamento de dados e os fluxos de dados –, juntamente com a definição das fronteiras de confiança. Com isto, mapeiam-se as ameaças em cada categoria STRIDE para cada elemento e relação do modelo

Assim sendo, o diagrama de arquitetura da Figura 2 identifica os componentes do sistema, sendo eles, essencialmente, (1) a aplicação *web*, (2) a API e (3) a base de dados. Para estes componentes, elabora-se o diagrama de fluxo de dados da Figura 5.

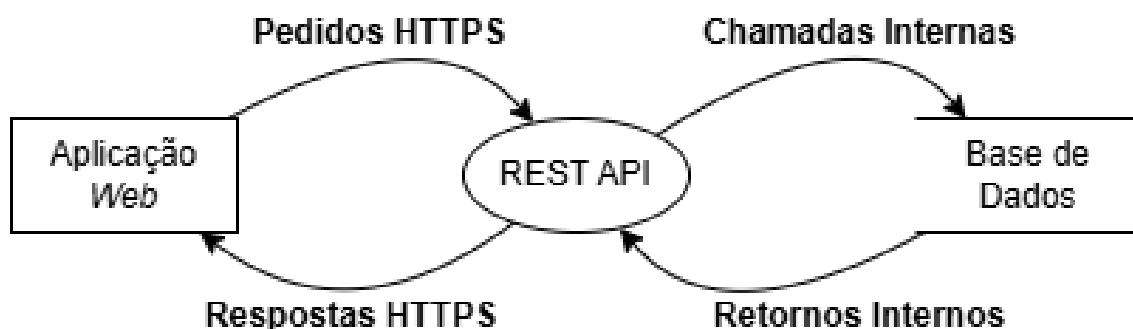


Figura 5 – Diagrama de Fluxo de Dados

Note-se que o diagrama da Figura 5 representa o fluxo de dados de forma simplificada, visto que, em rigor, cada seta deveria ser dividida em duas: a transmissão de conteúdo dos documentos e os dados de autenticação, sendo estes os principais dados tratados e armazenados pelo sistema.

Deste modo, o próximo passo consiste em estabelecer as assunções de segurança e as fronteiras de confiança do sistema. Nesse caso, os dados considerados não confiáveis são aqueles que provêm da aplicação *web* – diretamente, para a API, ou indiretamente, para a base de dados – enquanto os retornos internos e as respostas da API vêm-se como confiáveis. Além disso, admite-se que a comunicação entre o cliente e o servidor, bem como o método de autenticação, são seguros, visto que o objetivo de segurança passa por proteger o sistema contra ataques externos do lado do cliente e/ou utilizadores maliciosos.

Com isto, resta mapear as ameaças pertencentes às categorias STRIDE nos elementos e relações do diagrama. Para tal, apresentam-se as tabelas seguintes.

A Tabela 6 elenca as ameaças para a aplicação *web*.

Categoria	Ameaças
<i>Spoofing</i>	Roubar <i>cookies</i> de sessão de outro utilizador
<i>Tampering</i>	Injetar <i>scripts</i> ao editar documentos
<i>Repudiation</i>	Refutar a autoria de ações efetuadas através da interface
<i>Information Disclosure</i>	Aceder a documentos privados ou sem permissões de leitura
<i>Denial of Service</i>	Criar/Editar documentos repetidamente através da interface
<i>Elevation of Privilege</i>	Aceder ao painel de administração sem permissões

Tabela 6 – Ameaças à Aplicação Web

Analogamente, a Tabela 7 evidencia as ameaças consideradas para a REST API.

Categoria	Ameaças
<i>Spoofing</i>	Autenticar-se ou criar/editar documentos como outro utilizador
<i>Tampering</i>	Manipular parâmetros nos pedidos HTTP
<i>Repudiation</i>	Negar a realização de pedidos HTTP a certos <i>endpoints</i>
<i>Information Disclosure</i>	Retornar respostas para pedidos sem permissões
<i>Denial of Service</i>	Enviar muitas chamadas à API num curto intervalo de tempo
<i>Elevation of Privilege</i>	Efetuar ações sobre documentos sem permissões

Tabela 7 – Ameaças à API

De igual modo, a Tabela 8 mostra as ameaças à base de dados.

Categoria	Ameaças
<i>Spoofing</i>	Falsificar a identidade de outro utilizador via <i>SQL Injection</i>
<i>Tampering</i>	Alterar dados de documentos ou utilizadores via <i>SQL Injection</i>
<i>Repudiation</i>	Não Aplicável – o atacante não deve ter acesso à base de dados
<i>Information Disclosure</i>	Expor dados sensíveis
<i>Denial of Service</i>	Sobrecarregar a base de dados com demasiadas <i>queries</i>
<i>Elevation of Privilege</i>	Modificar diretamente papéis ou permissões

Tabela 8 – Ameaças à Base de Dados

Assim sendo, estão devidamente modeladas as ameaças ao sistema considerado, pelo que é possível prosseguir para a implementação, tendo em vista a adoção de mecanismos de segurança específicos para a prevenção ou mitigação das mesmas.

Implementação

Tendo concluído o processo de *design*, deve passar-se à implementação da plataforma de edição colaborativa de modo a cumprir os requisitos funcionais e de segurança propostos.

Para o efeito, pode aproveitar-se um projeto *open-source* e/ou uma *framework* que se enquadre no âmbito e que vá ao encontro do propósito estabelecido. Nesse sentido, comparam-se sete implementações já existentes de editores colaborativos de documentos, descritas em seguida.

1. **Etherpad**¹: um sistema extremamente simples na sua versão mais básica, mas altamente configurável/personalizável através de *plugins* adicionais, desenvolvido maioritariamente em TypeScript e assente numa API que permite gerir documentos e utilizadores;
2. **OnlyOffice Document Server**²: uma aplicação com suporte para leitura e edição de diversos tipos de documentos, com diferentes funcionalidades dependendo da edição, também extensível através de *plugins* e apoiada por uma API no *backend*;
3. **Collabora Online**³: uma plataforma que pode ser executada em diversos tipos de dispositivos – fixos e móveis – e integrada em múltiplos sistemas, com foco na versatilidade dos documentos suportados e nas funcionalidades de edição colaborativa;
4. **CryptPad**⁴: uma solução encriptada ponto-a-ponto, focada na privacidade dos dados e na segurança dos utilizadores, que complementa a possibilidade de personalização/configuração com funcionalidades tanto de edição colaborativa como de segurança, em JavaScript;
5. **HedgeDoc**⁵: um editor colaborativo em tempo-real de notas em *markdown*, desenvolvido em TypeScript, mas extremamente básico;
6. **MUTE**⁶: uma abordagem leve, com bom desempenho e fácil de colocar em produção, mas que já não é mantida há cerca de dois anos;
7. **PeerPad**⁷: um *software* experimental para edição descentralizada de documentos, mas que deixou de ser suportada há mais de cinco anos;

Assim, a partir destas sete potenciais soluções, é possível escolher aquela que mais se adequa ao problema proposto.

¹ <https://github.com/ether/etherpad-lite>

² <https://github.com/ONLYOFFICE/DocumentServer>

³ <https://github.com/CollaboraOnline/online>

⁴ <https://github.com/cryptpad/cryptpad>

⁵ <https://github.com/hedgedoc/hedgedoc>

⁶ <https://github.com/coast-team/mute>

⁷ <https://github.com/peer-base/peer-pad>

Ora, **MUTE** e **PeerPad** já não têm manutenção nem suporte há mais de um ano, pelo que podem estar desatualizados em termos de vulnerabilidades, com falhas de segurança por corrigir e versões de *software* obsoletas, o que os torna desaconselháveis. Por sua vez, o **Etherpad**, **OnlyOffice Document Server** e **Collabora Online** são constantemente atualizados pela comunidade, mas dispõem de demasiadas funcionalidades em relação às pretendidas, o que, particularmente devido à possibilidade de extensão através de *plugins*, faz com que possam estar mais suscetíveis a eventuais falhas de segurança, introduzidas intencionalmente ou não. Em sentido contrário, o **HedgeDoc** é demasiado simples para cumprir os requisitos estabelecidos para o projeto proposto. Como tal, resta apenas o **CryptPad**, que não só cumpre todos os requisitos delineados, mas também se foca na segurança e privacidade dos dados e dos utilizadores, parecendo a solução ideal para o problema em causa.

Assim sendo, opta-se pela exploração do **CryptPad** com maior detalhe e profundidade.

CryptPad

O **CryptPad** é uma plataforma projetada para a edição colaborativa de documentos em tempo-real, com suporte para uma enorme variedade de documentos e funcionalidades, mas com especial foco na privacidade dos dados e na segurança dos utilizadores.

O **CryptPad** divide os utilizadores em dois tipos: Convidados e Autenticados. Enquanto os utilizadores convidados têm acesso limitado à plataforma, com permissões configuráveis no servidor, os utilizadores autenticados têm acesso total. Deste modo, os convidados podem ser limitados à visualização e edição de documentos públicos, bem como impedidos de criar documentos. Esta distinção coincide precisamente com o *design* pretendido para o sistema proposto, pelo que é adequada.

Para autenticação, o **CryptPad** requer a introdução de um nome de utilizador e de uma palavra-passe. Note-se que, contrariamente ao habitual, este nome de utilizador não tem de ser único na plataforma, visto que a identificação unívoca de cada utilizador é efetuada através da chave pública derivada da combinação do seu nome de utilizador e da respetiva palavra-passe. De maneira a aumentar a robustez do método de autenticação, a plataforma permite também a configuração de autenticação multifator (MFA), através de uma aplicação móvel para a geração de códigos, como o *Microsoft Authenticator*, por exemplo. A par disto, existe ainda uma opção para terminar todas as sessões ativas associadas a uma determinada conta, o que pode facilitar a mitigação de eventuais tentativas de acesso indevido bem-sucedidas.

O **CryptPad** suporta diversos tipos de documentos: documentos de texto, apresentações, folhas de cálculo, código, *markdown*, formulários, diagramas, entre muitas outras possibilidades. Contudo, no âmbito do projeto proposto, estes documentos limitam-se, por configuração da instância, apenas a documentos de texto.

De acordo com os requisitos, o acesso aos documentos pode ser limitado à leitura ou edição dos mesmos. Para tal, existem dois níveis de permissões: (1) visualização, isto é, somente leitura, e (2) edição, ou seja, visualização e possibilidade de efetuar alterações ao documento. Existe, ainda, uma terceira permissão, que consiste em atribuir a possibilidade de um utilizador visualizar um documento apenas uma vez, sendo imediatamente destruído após a primeira visualização, mas que vai além dos requisitos do projeto. Além disto, tal como explicitado no *design*, há também o papel do proprietário de cada documento, sendo este o único responsável por gerir o acesso de outros utilizadores ao documento, assim como eliminá-lo. No entanto, apesar de o modelo anteriormente delineado estabelecer que cada documento só poderia ter um proprietário, o **CryptPad** permite que os documentos tenham múltiplos proprietários, mas isto não introduz qualquer falha de segurança, nem altera a modelação de ameaças previamente efetuada.

Relativamente a estes documentos, existem dois principais meios para os partilhar. Em primeiro lugar, pode ser partilhada uma ligação direta para os mesmos, que podem ser protegidos através de uma palavra-passe, para que o acesso só seja concedido mediante a introdução correta desta credencial. Este mecanismo é especialmente útil para editar colaborativamente documentos com utilizadores não autenticados, garantindo, simultaneamente, a devida autenticação e autorização. Em segundo lugar, existe a possibilidade de partilhar documentos com utilizadores da própria plataforma, que, desde que o acesso lhes seja concedido, já não necessitam de introduzir uma palavra-passe. Poderia existir ainda um terceiro meio para a partilha de documentos: a incorporação noutra página *web*, mas esta possibilidade foi desativada na instância configurada, de maneira a remover funcionalidades desnecessárias, reduzindo eventuais vulnerabilidades. O **CryptPad** permite ainda a organização de documentos em pastas, tal como num sistema de ficheiros tradicional, que, por sua vez, também podem ser partilhadas, seguindo os mesmos princípios e mecanismos.

Além de tudo isto, o **CryptPad** fornece ainda diversas funcionalidades úteis, mas fora do âmbito e dos requisitos do sistema pretendido. A título de exemplo, existem *tags*, *templates*, histórico de edições/versões, calendários, equipas, *tickets* de suporte, entre outros, que visam tornar a plataforma uma solução útil e viável na prática.

Assim sendo, considera-se que o **CryptPad** cumpre a totalidade dos requisitos funcionais propostos para o projeto, pelo que é admissível prosseguir para a sua implementação propriamente dita e configuração.

Configuração

O repositório original do **CryptPad** já está preparado para uma instalação simples. No entanto, é possível e recomendável não só personalizar, mas também configurar a instância do **CryptPad** a executar, de maneira a cumprir fielmente os requisitos estabelecidos e aumentar a robustez das garantias de segurança fornecidas.

Nesse sentido, de acordo com a documentação do **CryptPad**, existem dois ficheiros que devem ser criados antes de iniciar o servidor: **config/config.js** e **customize/application_config.js**.

O ficheiro **config/config.js** – copiado de **config/config.example.js** – deve ser alterado para definir as seguintes características/propriedades do servidor:

1. **otpSessionExpiration**: o tempo de vida das sessões dos utilizadores que se autenticam através de uma *One Time Password* (OTP) – definido para 24 horas;
2. **enforceMFA**: a obrigação de todos os utilizadores protegerem a sua conta com autenticação multifator;
3. **logIP**: o registo em *logs* dos endereços IP dos utilizadores que fazem alterações a um documento;
4. **adminKeys**: as chaves públicas dos utilizadores que devem ter permissões para aceder ao painel de administração da plataforma – [admin@localhost:3000/C7qtEwEpJLcIGb0IdzSdYnKYwPisCbhuJ9FH Md8r4Yo=].

Estas configurações simples visam reforçar a segurança do sistema, em particular ao dificultar ataques de acesso indevido (pelas medidas (1) e (2)), ao endereçar as ameaças de não-repúdio (com (3)) e ao definir adequadamente os utilizadores com privilégios administrativos (4).

A autenticação multifator, embora não seja um requisito inicial do projeto a desenvolver, é extremamente importante para garantir a segurança dos utilizadores, sendo, atualmente, essencial em sistemas aplicados em contexto real. Efetivamente, a maior parte dos ataques concretizados contra utilizadores não surge por vulnerabilidades técnicas ou tecnológicas, mas sim pelo roubo de credenciais através de mecanismos de engenharia social, ou com ataques de *phishing*, ou por via de *malware infostealers*, entre outros meios. No entanto, ao exigir um segundo método de autenticação para iniciar sessão, o sistema torna-se resistente a este tipo de ataques, visto que passa a exigir aos utilizadores o recurso a outro fator de autenticação que, neste caso, passa por uma aplicação móvel geradora de códigos de uso único, como o *Microsoft Authenticator*. Assim, esta configuração específica torna-se uma mais-valia da plataforma implementada.

De forma complementar, o ficheiro **customize/application_config.js** – a partir de **www/common/application_config_internal.js** – deve incluir mais algumas configurações, sendo elas:

1. **availablePadTypes:** define as aplicações acessíveis pelos utilizadores, dentro das suportadas pela plataforma – limitado a documentos de texto;
2. **availableLanguages:** explicita as linguagens nas quais a plataforma se encontra disponível – apenas inglês, sem traduções;
3. **surveyURL:** contém o URL para um formulário de *feedback* – desativado;
4. **hostDescription:** armazena a descrição a apresentar na página inicial;
5. **enableTemplates:** ativa a funcionalidade de guardar um documento como *template* – desativada, por não ser necessária;
6. **enableHistory:** ativa a funcionalidade de visualização do histórico de documentos – desativada, por não ser necessária;
7. **loginSalt:** contém o sal para ser utilizado no processo de *hashing* das palavras-passe dos utilizadores da plataforma pelo algoritmo *scrypt* – `+g&~D02Tcw_x_ew0pv03Q+;%Fh9cU3)S,Fw9k@{.Ry,Ar)j_#y,` gerado aleatoriamente;
8. **minimumPasswordLength:** especifica o comprimento mínimo necessário para as palavras-passe dos utilizadores da plataforma – definido para 12;
9. **disableAnonymousStore:** previne os utilizadores anónimos de armazenarem documentos na sua *drive* – configurado, de acordo com os requisitos;
10. **disableAnonymousPadCreation:** impede que os utilizadores anónimos criem documentos, mas permite que possam aceder ou editar documentos já existentes – definido, conforme os requisitos;
11. **disableFeedback:** desativa o formulário de *feedback* e as respetivas configurações – desativado, por não ser necessário.

Assim, a especificação destes parâmetros permite, por um lado, reduzir as funcionalidades da plataforma, de maneira a limitar a superfície de ataque ao não introduzir funcionalidades desnecessárias e, por outro lado, configurar alguns parâmetros de segurança (como o comprimento das palavras-passe e o sal associado). Além disso, também se afina a plataforma para ir ao encontro dos requisitos previamente estabelecidos.

Deste modo, definem-se as principais configurações do **CryptPad** tendo em vista o cumprimento do proposto.

No seguimento destas configurações, a instância do **CryptPad** pode ainda ser personalizada com alguns elementos que a tornem mais visualmente apelativa, enquadrando-a no contexto em que se deve inserir. Para isso, definem-se, através da interface, o nome e o logo da mesma, assim como, nos ficheiros **colortheme.less** e **colortheme-dark.less**, o esquema de cores a utilizar.

HTTPS

Um dos requisitos do projeto exige a comunicação de forma segura, ou seja, a utilização do protocolo HTTPS, em vez de HTTP. Efetivamente, a implementação do **CryptPad** encontra-se preparada para isto, necessitando apenas de algumas configurações adicionais.

Nesse sentido, para desenvolvimento e execução locais, o método mais simples para a utilização de HTTPS passa por gerar certificados *Secure Sockets Layer* (SSL) auto-assinados, que devem ser importados para o navegador. Para tal, utiliza-se a ferramenta **mkcert**⁸.

O **mkcert** é uma ferramenta simples cujo propósito consiste em automatizar o processo de construção e assinatura de certificados confiáveis localmente, no ambiente de desenvolvimento e sem configurações complexas. Assim, de acordo com as instruções de execução presentes no repositório do projeto, o **mkcert** começa por criar e instalar uma Autoridade Certificadora (CA) local no sistema operativo e nos navegadores do dispositivo em causa. Posteriormente, esta ferramenta permite gerar um certificado SSL e a respetiva chave privada para o domínio do ambiente de desenvolvimento local – **localhost** –, que são usados para a correta configuração e execução do protocolo HTTPS.

Além disto, deve configurar-se a instância do **CryptPad** para iniciar um servidor HTTPS que utilize este certificado. Para tal, a opção adotada passa por alterar ligeiramente o ficheiro **lib/http-worker.js**, para que os ficheiros gerados anteriormente – com o certificado e a chave associada – sejam utilizados na criação do servidor HTTPS. Além disso, é necessário modificar também o ficheiro **config/config.js**, definindo os parâmetros **httpUnsafeOrigin** e **httpSafeOrigin** para **https://localhost:3000** e **https://localhost:3001**, respetivamente. A necessidade de utilização destes dois domínios explica-se posteriormente, na secção de Análise.

Com isto, consegue-se estabelecer comunicação por HTTPS entre os clientes e o servidor, tal como pretendido.

Execução

O repositório do projeto contém, no ficheiro **README.md**, as instruções de instalação e execução do sistema desenvolvido. Além disso, apresenta-se um vídeo demonstrativo em <https://youtu.be/EoywKTsAIP8>, bem como algumas imagens da interface da plataforma. Note-se que a plataforma deve ser acedida em <https://localhost:3000/>.

⁸ <https://github.com/FiloSottile/mkcert>

A Figura 6 mostra a página inicial da plataforma.

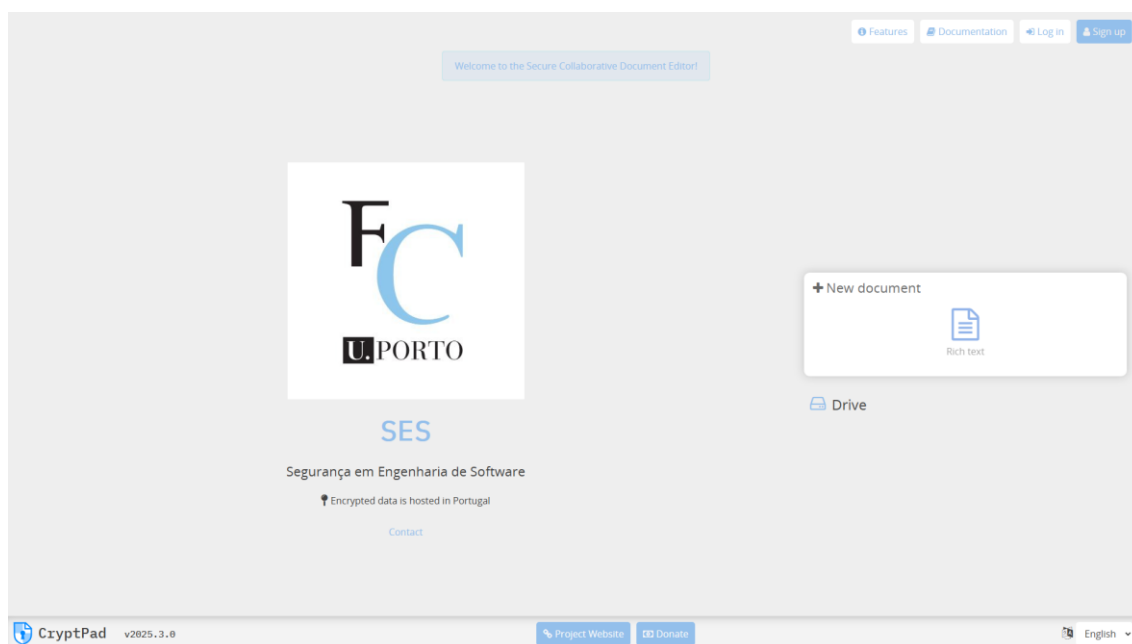


Figura 6 – Página Inicial

A Figura 7 demonstra o formulário de registo na plataforma.

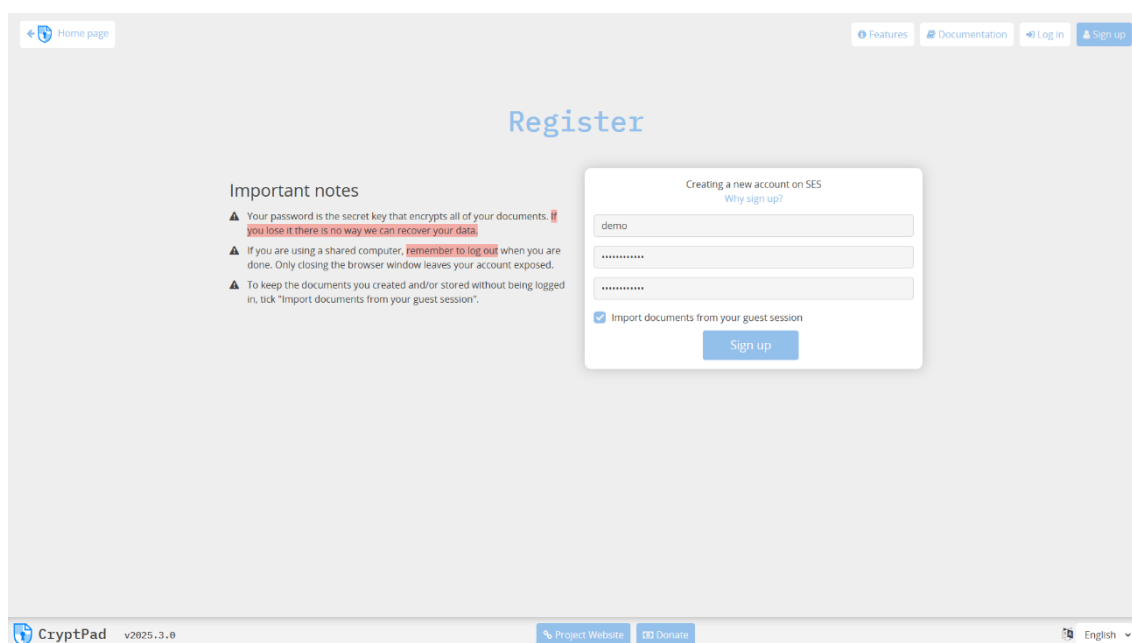


Figura 7 – Registo

A Figura 8 evidencia o processo de configuração da autenticação multifator.



Figura 8 – Autenticação Multifator

A Figura 9 contém a página de criação de um documento.

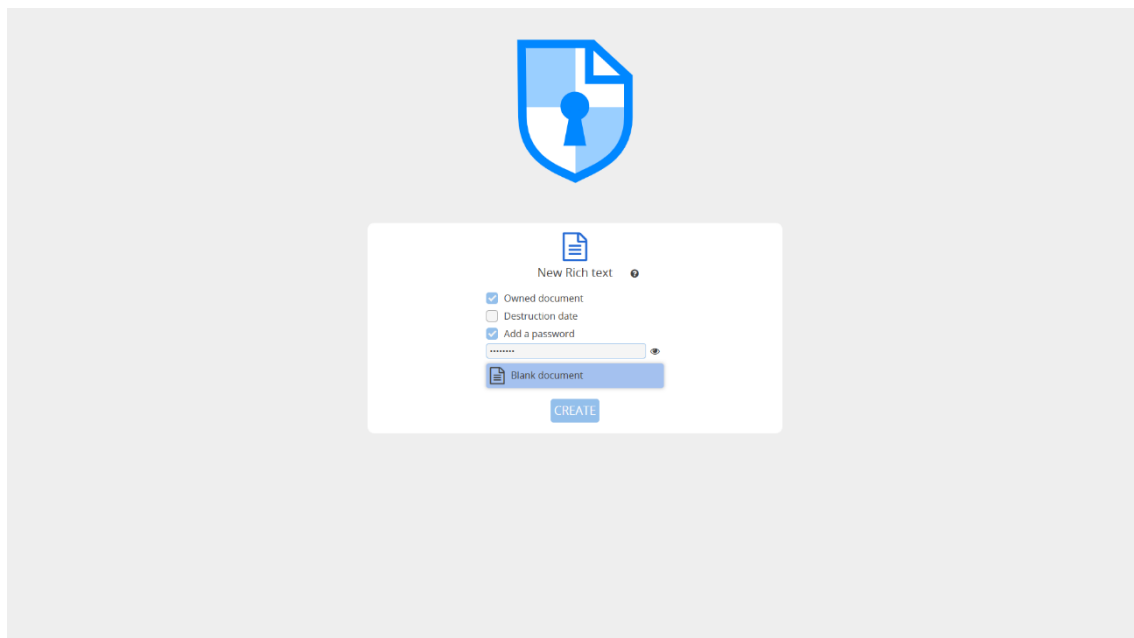


Figura 9 – Criação de Documento

A Figura 10 apresenta o documento aberto para edição.



Figura 10 – Documento

A Figura 11 transmite um dos meios para a partilha de documentos: através de um *link*, com permissões configuráveis.

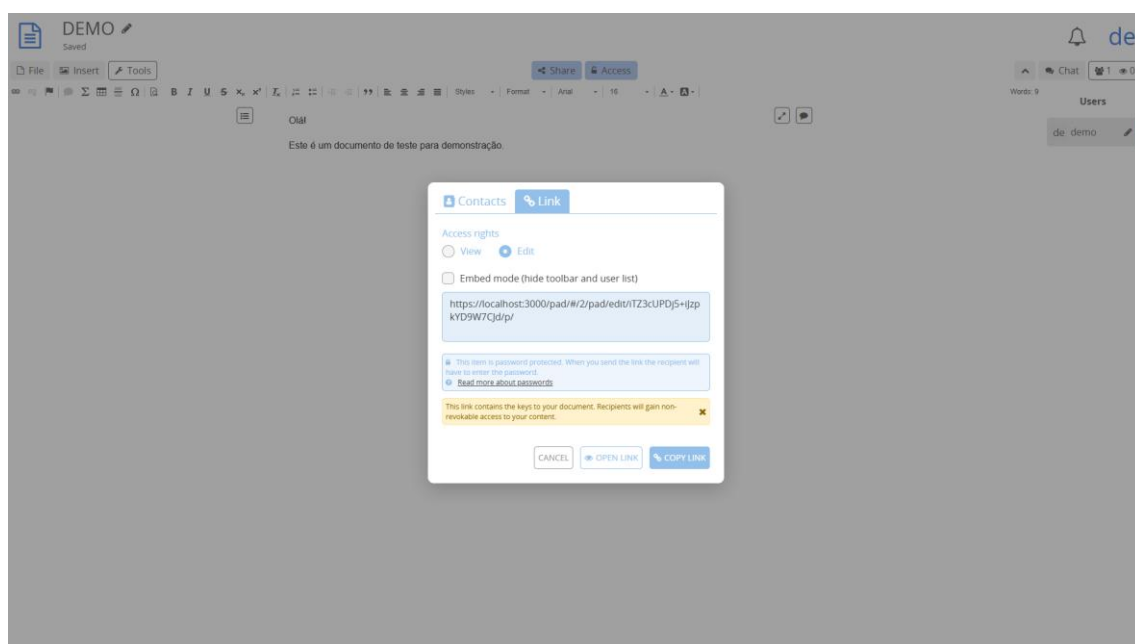


Figura 11 – Partilha de Documento por Link

Analogamente, a Figura 12 traduz o outro meio através do qual é possível partilhar documentos: com determinados utilizadores especificados.

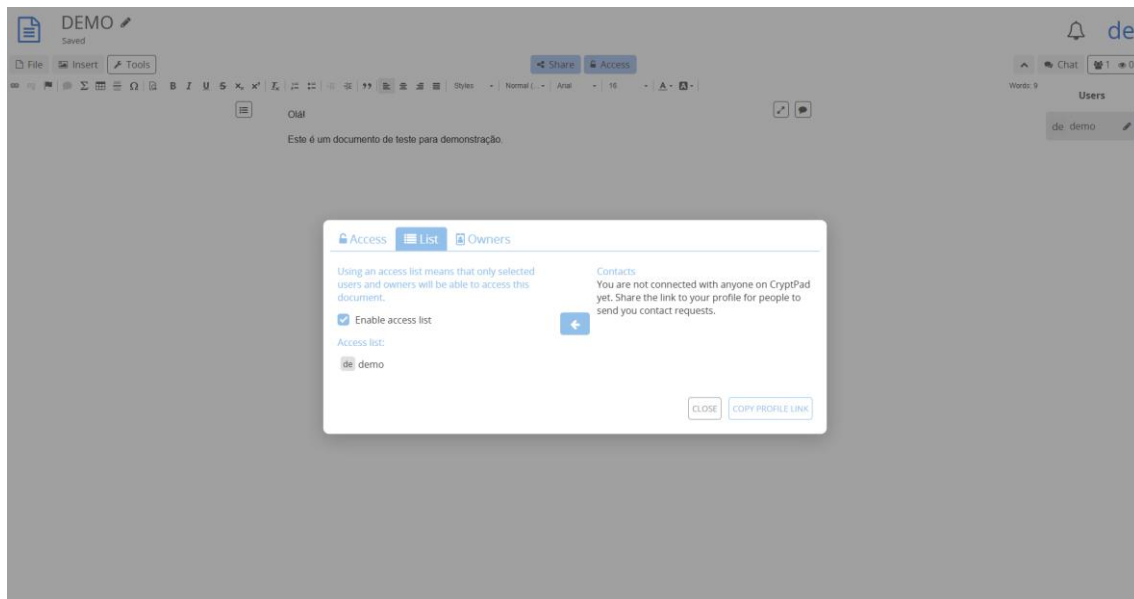


Figura 12 – Partilha de Documento com Utilizadores

Finalmente, a Figura 13 explicita algumas das funcionalidades disponíveis exclusivamente ao proprietário do documento.

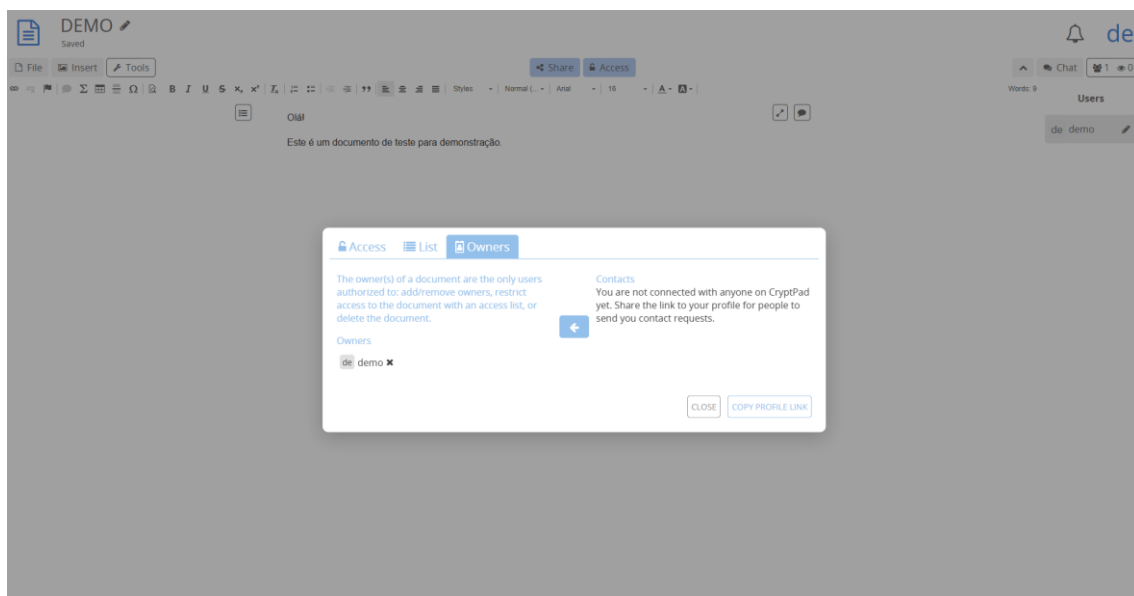


Figura 13 – Proprietário de Documento

Assim, conclui-se uma breve apresentação da solução implementada.

Análise

Por último, com o intuito de demonstrar que a implementação respeita o *design* proposto e satisfaz os requisitos de segurança estabelecidos, efetua-se uma análise profunda e detalhada da mesma, de forma tão abrangente quanto possível.

Linguagens de Programação

Em primeiro lugar, conforme já explicitado, a principal linguagem de programação utilizada pela plataforma é o JavaScript. Esta linguagem é amplamente usada na vasta maioria das aplicações *web*, principalmente, mas não só, para o lado do cliente, assim como – mais recentemente – para a vertente do servidor.

Por um lado, os navegadores *web* têm um mecanismo e um ambiente dedicados para a execução de JavaScript, responsável por correr o código do cliente. Por outro lado, no servidor, o Node.js⁹ é o ambiente de execução JavaScript da aplicação desenvolvida. Assim, o código JavaScript é executado tanto no *frontend* (cliente), como no *backend* (servidor), sendo a principal linguagem de programação na qual assenta a implementação do editor de documentos colaborativo.

O JavaScript é uma linguagem de alto-nível, compilada *just-in-time*, que obedece a um padrão universal para garantir interoperabilidade e compatibilidade entre os diversos sistemas. Globalmente, a linguagem é constituída por tipos dinâmicos e orientada a objetos, mas baseada em protótipos e na qual as funções são de primeira classe. Dada a sua versatilidade, o JavaScript é considerado multiparadigmático, visto que possibilita a adoção de estilos de programação funcional, imperativa e guiada por eventos, por exemplo. A utilização do JavaScript para desenvolvimento *web* prende-se, essencialmente, com o facto de ter APIs próprias para diversas funcionalidades, em particular para manipular o *Document Object Model* (DOM), que é uma representação totalmente orientada a objetos das páginas *web*, sob a forma de uma árvore de nós, permitindo ler e alterar a estrutura, o estilo e o conteúdo das páginas de forma fácil.

Em síntese, as garantias de segurança oferecidas pela utilização de JavaScript podem ser divididas em dois âmbitos: a linguagem propriamente dita e o ambiente de execução.

⁹ <https://nodejs.org/en>

Quanto à linguagem JavaScript propriamente dita, existem algumas propriedades/características que lhe conferem considerável segurança. Em particular, o JavaScript é uma linguagem que gere automaticamente a memória, através de mecanismos de *garbage collection*, o que previne erros comuns associados ao uso indevido de memória, que se poderiam materializar em vulnerabilidades de segurança. Além disto, também não existem apontadores para endereços de memória, nem é permitido o acesso direto a blocos de memória, o que minimiza a possibilidade de ocorrência de erros por parte dos engenheiros de *software*, ao contrário do que sucede nas linguagens de baixo-nível, como o C, por exemplo. Complementarmente, o modelo de execução tradicional de uma única *thread* com um ciclo de eventos para ações assíncronas também contribui para minimizar eventuais *race conditions*, facilitando a gestão de problemas de concorrência.

No entanto, um dos principais pontos fortes da linguagem reside nas medidas de segurança aplicadas pelos ambientes de execução. Em primeiro lugar, destaca-se o conceito de *sandbox* aplicado pelos navegadores, ou seja, a execução de código num ambiente isolado, sem acesso ao sistema de ficheiros, à rede e ao sistema operativo, por exemplo, o que dificulta que *scripts* potencialmente maliciosos tenham impactos diretos no dispositivo do utilizador. Em segundo lugar, o modelo *Same-Origin Policy* (SOP) visa garantir que um *script* correspondente a uma página *web* só pode aceder a dados da mesma origem, isto é, com igual protocolo, domínio e porto. Deste modo, impede-se que um *site* malicioso seja capaz de extrair dados sensíveis de outro *site*, como, por exemplo, os *cookies* de sessão. Além de tudo isto, a definição de *Content Security Policies* (CSP) permite estabelecer os *scripts* que podem ser carregados, de forma a mitigar ataques como *Cross-Site Scripting* (XSS), tal como políticas de *Cross-Origin Resource Sharing* (CORS) e *Sub-Resource Integrity* (SRI) são responsáveis por, respetivamente, controlar o acesso a recursos provenientes de diferentes origens e garantir que os *scripts* carregados para execução são os originais, pelo que não foram alvo de modificações não autorizadas, o que constituiria uma violação da integridade.

Por tudo isto, a linguagem JavaScript surge como uma opção válida não só para o código do cliente, mas também para o código do servidor.

Design Patterns

Do ponto de vista da arquitetura, o **CryptPad** segue um *design* que visa prevenir possíveis ataques e/ou mitigar eventuais impactos que deles resultem. A par disto, o código segue também um conjunto de princípios-chave – elencados de seguida – que pretendem garantir o cumprimento das garantias de segurança pretendidas pelos utilizadores.

Essencialmente, o **CryptPad** recorre a um mecanismo de *sandboxing* para aumentar a robustez do sistema, isolando a interface do utilizador do conteúdo presente em memória. Efetivamente, uma origem (exposta no porto 3000, na implementação local) é utilizada para disponibilizar o acesso da instância aos utilizadores, enquanto uma segunda origem (no porto 3001, neste caso) atua como uma *sandbox*, na qual se aplica um conjunto de *Content Security Policies*. Através disto, consegue-se que a computação sensível – como o processamento de chaves criptográficas – seja efetuada na origem principal, enquanto a interface do utilizador é implementada na origem alternativa, isto é, na *sandbox*. Deste modo, garante-se maior isolamento para o processamento sensível, prevenindo ou dificultando ataques de utilizadores maliciosos. Em síntese, como a parte exposta da estrutura só recebe dados relevantes para o documento atual, mesmo que possa conter alguma vulnerabilidade, nunca será explorável ao ponto de conseguir aceder a informação do utilizador. Este mecanismo é explicado em mais detalhe posteriormente, para a prevenção de ataques de XSS.

Ao nível da base de dados, o **CryptPad** não segue uma abordagem convencional, optando por armazenar os dados no próprio sistema de ficheiros do servidor, em detrimento de uma base de dados estruturada. Na prática, cada documento tem informação armazenada no servidor, numa localização determinada pelo ID do canal de comunicação. Assim, o cliente envia o ID do canal para o servidor de modo a obter informações sobre o documento, bem como receber atualizações efetuadas ao mesmo e/ou modificar o seu conteúdo. Deste modo, cada canal é armazenado no seu próprio ficheiro no servidor e cada alteração ao documento corresponde a uma única linha desse ficheiro. Ora, como o conteúdo de cada documento é encriptado, o servidor não o consegue ler.

Globalmente, o código da plataforma é dividido/estruturado em cinco níveis: dois do lado do servidor e três do lado do cliente.

Do lado do servidor, reside o servidor propriamente dito e os *workers*. O servidor contém o código em execução no processo principal, que gere as conexões e as chamadas internas, enquanto os *workers* lidam com as conexões à base de dados (o sistema de ficheiros) e com os *scripts* que requerem mais recursos computacionais. Assim, o servidor chama um *worker* e lança-o num subprocesso quando recebe determinados comandos dos utilizadores.

Do lado do cliente, os três níveis são o externo, o interno e o *worker*. O nível externo é aquele que contém os dados sensíveis, como as chaves de encriptação dos documentos. O nível interno é constituído pelo *iframe* que contém a interface do utilizador, ou seja, a *sandbox* previamente explicitada. Este *iframe* representa todo o ecrã que é visível aos utilizadores – pelo que nenhum elemento da interface está fora do *iframe* – e só tem acesso aos dados necessários para serem visualizados no ecrã. Finalmente, o *worker* gere a conexão com o servidor e mantém os dados do utilizador em memória, partilhando-os entre os diferentes separadores abertos na mesma instância do **CryptPad**.

A Figura 14 detalha esta arquitetura.

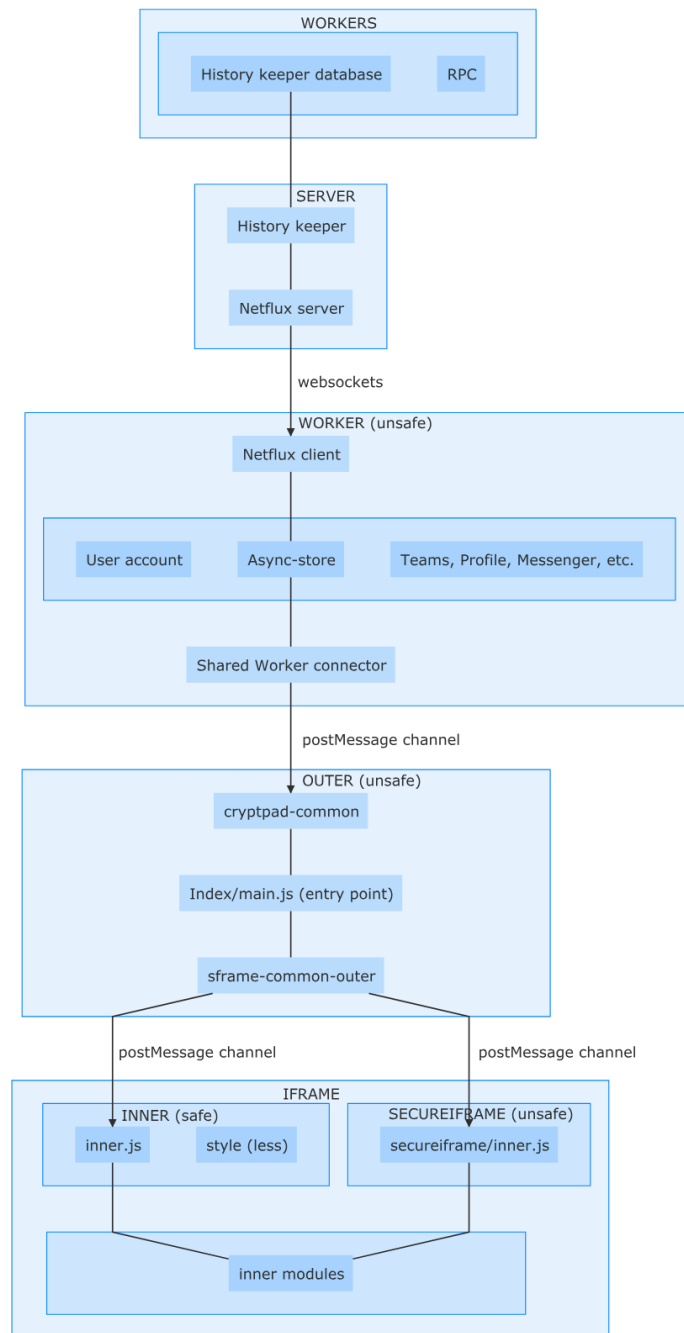


Figura 14 – Arquitetura do CryptPad

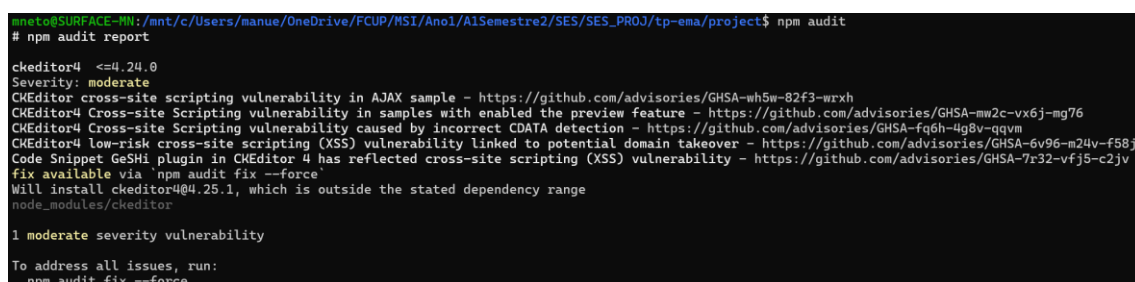
Deste modo, consegue-se o isolamento devido entre os diversos mecanismos que fazem parte do sistema, o que não só facilita o desenvolvimento, mas também garante o respeito pelos princípios basilares de segurança, como a separação de privilégios e o princípio do privilégio mínimo, entre outros.

Dependências e Bibliotecas

O **CryptPad** recorre a várias dependências e bibliotecas externas, que devem ser analisadas em termos de segurança. Para o efeito, dado o paradigma Node.js da aplicação, executa-se o comando **npm audit**.

Aquando da instalação do projeto, o **npm** detetou automaticamente versões de bibliotecas desatualizadas que poderiam introduzir vulnerabilidades de segurança, mas cuja atualização, que corrigia as respetivas vulnerabilidades, não prejudicava o funcionamento correto do sistema. Como tal, essas bibliotecas foram imediatamente atualizadas para a disponibilização do projeto implementado, mitigando/corrigindo as vulnerabilidades existentes.

No entanto, mesmo após estas atualizações, o **npm** continua a detetar uma vulnerabilidade de severidade moderada numa biblioteca, como se evidencia na Figura 15.



```
manu@SURFACE-MH: /mnt/c/Users/manue/OneDrive/FCUP/MS1/Ano1/A1Semestre2/SES/SES_PROJ/tp-ema/project$ npm audit
# npm audit report

ckeditor4 <=4.24.0
Severity: moderate
CKEditor cross-site scripting vulnerability in AJAX sample - https://github.com/advisories/GHSA-wh5w-82f3-wrxh
CKEditor4 Cross-site Scripting vulnerability in samples with enabled the preview feature - https://github.com/advisories/GHSA-mw2c-vx6j-mg76
CKEditor4 Cross-site Scripting vulnerability caused by incorrect CDATA detection - https://github.com/advisories/GHSA-fq6h-4g8v-qgvm
CKEditor4 low-risk cross-site scripting (XSS) vulnerability linked to potential domain takeover - https://github.com/advisories/GHSA-6v96-m24v-f58j
Code Snippet GeSHi plugin in CKEditor 4 has reflected cross-site scripting (XSS) vulnerability - https://github.com/advisories/GHSA-7r32-vfj5-c2jv
fix available via `npm audit fix --force`
Will install ckeditor4@4.25.1, which is outside the stated dependency range
node_modules/ckeditor

1 moderate severity vulnerability

To address all issues, run:
npm audit fix --force
```

Figura 15 – npm audit

Apesar de o **npm** sugerir a execução do comando **npm audit fix --force** para a atualização desta biblioteca e a correção da respetiva vulnerabilidade, nenhuma tentativa de o executar é bem-sucedida, pelo que a vulnerabilidade permanece. Contudo, tendo em conta que esta vulnerabilidade é de *Cross-Site Scripting* e o **CryptPad** já implementa diversos mecanismos para a prevenção de ataques deste tipo – como a *sandbox* descrita anteriormente e outras medidas a detalhar posteriormente –, considera-se admissível a existência desta dependência, ainda que desatualizada.

Como análise complementar, executa-se também a ferramenta Snyk¹⁰, para detetar potenciais vulnerabilidades nas dependências e bibliotecas do projeto. Esta plataforma exige autenticação prévia do utilizador e os resultados apresentam-se na Figura 16.

¹⁰ <https://snyk.io/>


```

PS C:\Users\manue\OneDrive\FCUP\MSI\Ano1\1Semestre2\SES\SES_PROJ\tp-ema\project> snyk test

Testing C:\Users\manue\OneDrive\FCUP\MSI\Ano1\1Semestre2\SES\SES_PROJ\tp-ema\project...

Tested 330 dependencies for known issues, found 1 issue, 1 vulnerable path.

Issues with no direct upgrade or patch:
X Regular Expression Denial of Service (ReDoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JS-MATHJAX-6210173] in mathjax@3.0.5
  introduced by mathjax@3.0.5
  No upgrade or patch available

Organization: manelneto
Package manager: npm
Target file: package-lock.json
Project name: cryptpad
Open source: no
Project path: C:\Users\manue\OneDrive\FCUP\MSI\Ano1\1Semestre2\SES\SES_PROJ\tp-ema\project
Licenses: enabled

Tip: Detected multiple supported manifests (34), use --all-projects to scan all of them at once.

```

Figura 16 – Snyk

Ora, o Snyk deteta, igualmente, uma vulnerabilidade de severidade média, mas diferente da anterior, desta vez de *Denial of Service* (DoS) através de expressões regulares. Esta vulnerabilidade explora expressões regulares eventualmente mal construídas de modo a causar o consumo elevado de recursos do processador, o que pode bloquear a aplicação em execução no servidor perante determinados *inputs* maliciosos. Contudo, não existem atualizações ou *patches* disponíveis para mitigar esta vulnerabilidade, pelo que, de momento, não se toma qualquer medida.

Além destas duas ferramentas, recorre-se, também, ao Dependabot¹¹ do GitHub para auxiliar na identificação de dependências desatualizadas e vulneráveis no código da plataforma desenvolvida. Assim, esta ferramenta apresenta os resultados da Figura 17.

☐

🔔

5 Open

✓

0 Closed

Package

Ecosystem

Manifest

Severity

Sort

☐

🔔

CKEditor4 low-risk cross-site scripting (XSS) vulnerability linked to potential domain takeover

Moderate

Direct

#5 opened 2 hours ago • Detected in ckeditor4 (npm) • project/package-lock.json

☐

🔔

Code Snippet GeSHi plugin in CKEditor 4 has reflected cross-site scripting (XSS) vulnerability

Moderate

Direct

#4 opened 2 hours ago • Detected in ckeditor4 (npm) • project/package-lock.json

☐

🔔

CKEditor cross-site scripting vulnerability in AJAX sample

Moderate

Direct

#3 opened 2 hours ago • Detected in ckeditor4 (npm) • project/package-lock.json

☐

🔔

CKEditor4 Cross-site Scripting vulnerability in samples with enabled the preview feature

Moderate

Direct

#2 opened 2 hours ago • Detected in ckeditor4 (npm) • project/package-lock.json

☐

🔔

CKEditor4 Cross-site Scripting vulnerability caused by incorrect CDATA detection

Moderate

Direct

#1 opened 2 hours ago • Detected in ckeditor4 (npm) • project/package-lock.json

Figura 17 – Dependabot

¹¹ <https://github.com/dependabot>

Ora, o Dependabot alerta para a existência de cinco vulnerabilidades de XSS associadas a uma dependência, CKEditor, com severidade moderada. Estas vulnerabilidades afetam as versões inferiores a 4.25.0 do CKEditor, pelo que, para as resolver, basta atualizar a dependência para a versão 4.25.0, no mínimo.

De forma automática, é possível solicitar a atualização desta dependência para uma versão que resolve ou mitiga as vulnerabilidades em questão, através do Dependabot. No entanto, ao tentar efetuá-lo, a própria ferramenta indica que a dependência alegadamente vulnerável já se encontra atualizada, pelo que não é necessário (nem possível) atualizá-la novamente. Por isso, estes cinco alertas não representam, na verdade, potenciais falhas de segurança na aplicação desenvolvida, mas sim um erro no próprio Dependabot que, erradamente, devido a algum eventual conflito interno, considera esta versão do CKEditor desatualizada, quando isso não sucede.

Assim sendo, os cinco alertas do Dependabot podem ser fechados no GitHub, por serem imprecisos ou incorretos, logo, falsos positivos.

Com isto, a análise de dependências e bibliotecas dá-se por concluída, tendo-se identificado versões desatualizadas e os seus potenciais impactos para a segurança do projeto, mitigando-os, sempre que possível, através das respetivas atualizações.

Mitigações para Vulnerabilidades Comuns

De modo a complementar e fortalecer a análise de segurança, recorre-se ao OWASP *Top 10* no sentido de identificar as mitigações implementadas para as vulnerabilidades mais comuns. Esta lista – desenvolvida pelo *Open Web Application Security Project* (OWASP) – é uma referência globalmente reconhecida para identificar e priorizar os riscos de segurança mais críticos em aplicações *web*.

A vulnerabilidade mais comum na última avaliação do OWASP (2021), é *Broken Access Control*, que consiste na incorreta implementação de controlo de acessos a recursos. Em particular, esta categoria inclui casos de *Cross-Site Request Forgery* (CSRF) e *Insecure Direct Object Reference* (IDOR), entre outros. Objetivamente, o sistema implementado contém várias medidas que previnem estes ataques. Em particular, a arquitetura demonstrada segue o princípio do privilégio mínimo e respeita totalmente os papéis dos utilizadores autenticados perante os documentos, utiliza *tokens* aleatórios em cada formulário submetido para prevenir ataques de CSRF e tem configurações ao nível do CORS (já explicitadas) para impedir o acesso a recursos de origens não autorizadas. Portanto, estas vulnerabilidades consideram-se endereçadas.

A segunda categoria mais prevalente no OWASP *Top 10* são as falhas criptográficas. Ora, no que concerne à criptografia utilizada, o **CryptPad** recorre a diversos algoritmos, dependendo do caso em questão. Os documentos e os dados da conta do utilizador são encriptados através do algoritmo de encriptação simétrica autenticada ChaCha20-Poly1305, sendo que a chave é derivada da *hash* contida no URL do documento. Como a encriptação é simétrica, todos os utilizadores capazes de ler/desencriptar um documento também o poderiam editar, encriptando as edições e enviando-as. Para permitir a distinção entre acesso só de leitura e acesso com permissões de edição, os editores devem assinar as edições com uma chave privada – usando o algoritmo de assinatura em curvas elípticas Ed25519 – que está associada à chave pública do documento, enviada para o servidor aquando da criação do mesmo. Deste modo, só edições com assinaturas válidas são consideradas, enquanto quaisquer outras tentativas de edição são ignoradas. Existem ainda outros mecanismos de encriptação presentes, por exemplo, para o *chat* entre utilizadores e para o sistema de notificações, mas não fazem parte do âmbito nem dos requisitos do projeto, pelo que não se exploram.

Em terceiro lugar, surgem as vulnerabilidades de injeção, que podem ser divididas, de forma simplista, em *SQL Injection* e *Cross-Site Scripting*. Ora, como o **CryptPad** não utiliza qualquer base de dados SQL, mas apenas o sistema de ficheiros nativo, o primeiro caso não se aplica. Sobre o segundo caso, efetivamente, enquanto editor de documentos colaborativo, uma das principais preocupações do **CryptPad** consiste em impedir que atores maliciosos sejam capazes de enviar código para ser executado no navegador de outro utilizador, concretizando um ataque de XSS. Para isto, a plataforma utiliza três principais mecanismos.

Em primeiro lugar, o **CryptPad** utiliza *sanitizers*, isto é, ferramentas que limpam/sanitizam o conteúdo que é colocado na plataforma por uns utilizadores para ser mostrado para outros. Na prática, estas ferramentas removem todo e qualquer texto que seja capaz de executar código JavaScript, como atributos HTML **onclick="..."** ou tags **<script>**, por exemplo.

Em segundo lugar, o servidor adiciona outras proteções ao acrescentar regras de *Content Security Policies* para os utilizadores. Estas regras permitem que o navegador seja capaz de distinguir entre aquilo que é capaz de fazer de forma segura e aquilo que é potencialmente perigoso, pelo que deve ser bloqueado. Em particular, as CSPs definidas especificam três regras: (1) impedir o carregamento de código a partir de *sites* externos, como *Content Delivery Networks* (CDNs), ou seja, garantir que todos os ficheiros JavaScript carregados estão alojados no domínio associado à instância do **CryptPad** em execução, (2) não executar JavaScript *inline*, isto é, contido em atributos HTML, como o **onclick="..."** visto anteriormente e (3) nunca avaliar uma *string* como código através da função **eval()**. Com estas proteções, dificultam-se os ataques de XSS.

Por último, tal como já explicado, a arquitetura do **CryptPad** prevê a utilização de um sistema de *sandbox* para proteger os utilizadores da plataforma de atores maliciosos. Desta forma, a interface *web* é criada dentro de um *iframe* aberto numa origem HTTP diferente da do separador do navegador, que é aquela que contém os dados da conta do utilizador com sessão iniciada. Assim, como toda a interface colaborativa pertence a um *iframe*, se ocorrer injeção de código por parte de um utilizador, esse código só pode ser executado ao nível desse *iframe*. Com isto, o facto de a *sandbox* e a página propriamente dita provirem de origens diferentes protegem as contas dos utilizadores ao limitar os atores maliciosos a somente serem capazes de obter os dados a partir do documento ao qual já têm acesso.

Assim, as três categorias mais comuns do OWASP *Top 10* estão endereçadas pela plataforma implementada. A quarta categoria – *Insecure Design* – é tratada por todo o processo de modelação de ameaças e pela arquitetura seguida, enquanto a que ocupa a quinta posição – *Security Misconfiguration* – também se considera abordada, visto que todas configurações disponíveis foram revistas e definidas no âmbito deste projeto.

Finalmente, ainda que não se enquadre diretamente nas categorias desta lista, um ataque relativamente comum aos sistemas consiste em testar múltiplas combinações de nome de utilizador e palavra-passe, por meios de força-bruta. O **CryptPad** também mitiga isto. Ora, relativamente às palavras-passe, o sistema utiliza o nome de utilizador e a palavra-passe propriamente dita como argumentos fornecidos à função de derivação de chaves *scrypt*, que computa um resultado equivalente a uma *hash*, mas de forma extremamente dispendiosa em termos de recursos computacionais. Deste modo, a computação extremamente lenta inviabiliza quaisquer ataques de força-bruta, visto que cada computação leva vários segundos a executar. Além disso, com este mecanismo consegue-se que o nome e a palavra-passe dos utilizadores nunca sejam armazenados no servidor, o que também reduz o impacto de eventuais fugas de informação que possam ocorrer.

Por tudo isto, as principais vulnerabilidades de segurança estão resolvidas ou mitigadas pela plataforma implementada.

Metodologias de Teste de Segurança

No sentido de integrar/incorporar metodologias de teste de segurança no desenvolvimento da aplicação, definiu-se uma *pipeline* de *Continuous Integration & Continuous Delivery/Deployment* (CI/CD) sob a forma de uma GitHub Action no repositório do projeto, para realizar um *scan* à plataforma desenvolvida usando a ferramenta OWASP Zap.

O ficheiro que contém o *workflow* responsável por executar a GitHub Action com o *scan* da ferramenta OWASP Zap é o `.github/workflows/security.yml` e os respetivos resultados apresentam-se na Figura 18.

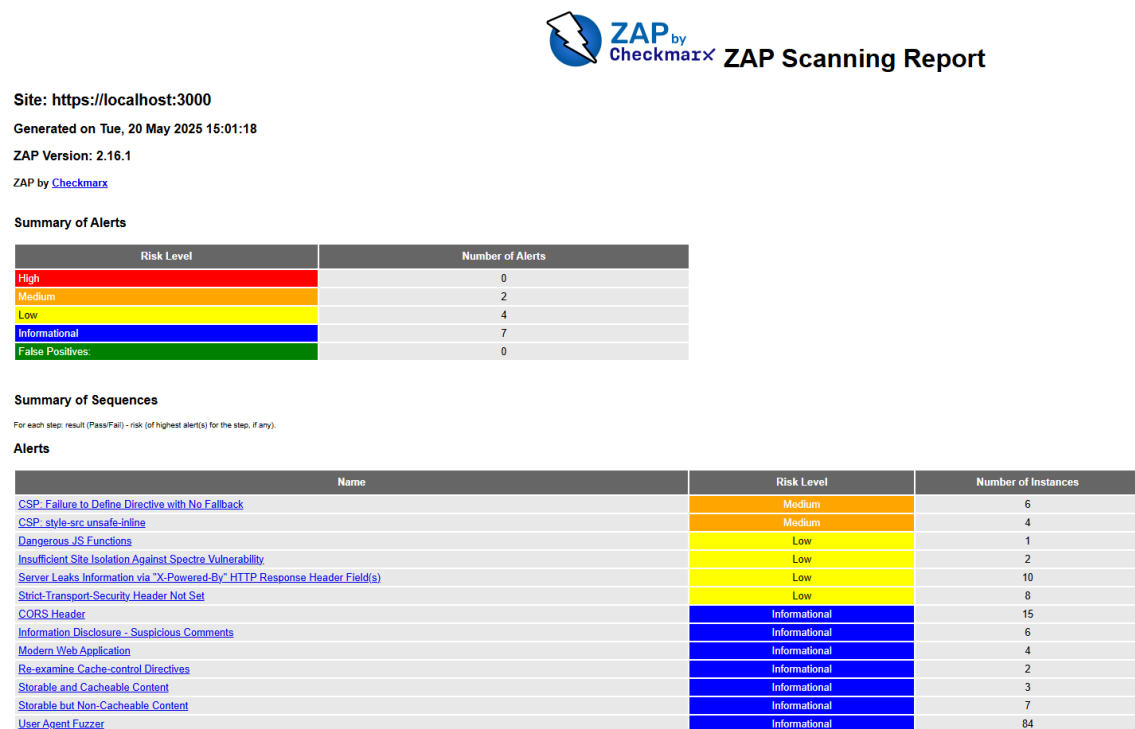


Figura 18 – OWASP Zap

Efetivamente, como demonstra a Figura 18, o *scan* automatizado do OWASP Zap levanta 13 alertas sobre a segurança da aplicação, dos quais dois têm risco médio, quatro têm risco baixo e sete são meramente informativos.

Em primeiro lugar, destaca-se o facto de não ser lançado nenhum alerta com severidade elevada, o que é um ponto positivo a realçar. Além disso, são apenas dois os alertas com severidade média e quatro os de baixa gravidade. Em detalhe, os alertas médios devem-se a algumas *Content Security Policies* que poderiam ter definições mais restritivas, nomeadamente no que toca à diretiva **form-action**, por exemplo. Além disto, os alertas baixos identificam alguns cabeçalhos de segurança em falta – como *Strict-Transport-Security* e *Cross-Origin-Opener-Policy* –, bem como a presença do cabeçalho *X-Powered-By* com o valor *Express* (que revela alguma informação sobre o servidor) e o uso da função `eval()`, que pode permitir a execução de comandos maliciosos, se o *input* não for corretamente validado. Portanto, embora estas situações não sejam críticas, poderiam e deveriam ser corrigidas, sem prejudicar a implementação. Os alertas informacionais não têm particular relevância, cingindo-se a comentários no código, ao tempo definido para a retenção de conteúdos em cache, entre outros.

Assim, o OWASP Zap testa e valida a segurança da aplicação desenvolvida de forma contínua, sempre que o repositório remoto é atualizado. Esta configuração é uma boa-prática que deve ser implementada quando possível, de modo a integrar ferramentas de testes de segurança no desenvolvimento.

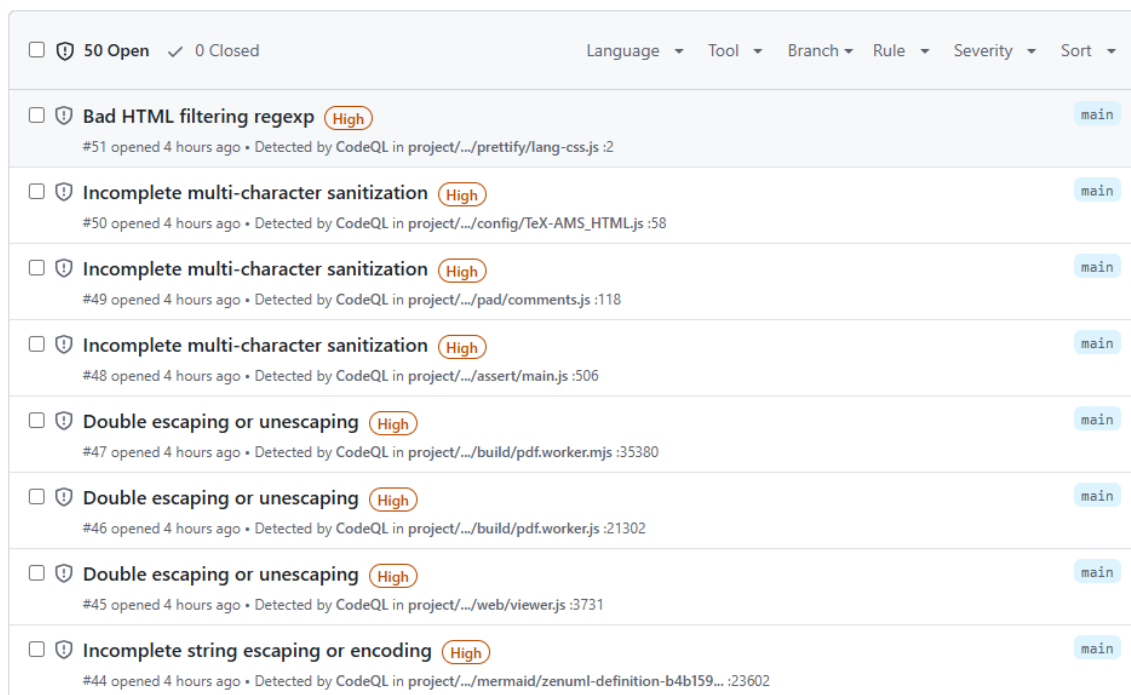
Ferramentas de Análise de Código

Finalmente, para aumentar a robustez da plataforma em termos de segurança, executam-se três ferramentas de análise de código, integrando-as no processo de desenvolvimento: CodeQL, Semgrep e SonarQube Cloud.

CodeQL

O CodeQL¹² é um mecanismo desenvolvido pelo GitHub de análise semântica de código que permite encontrar falhas de segurança de forma semelhante à realização de *queries* SQL. Assim, esta ferramenta pode ser integrada no processo de CI/CD e executada no próprio repositório remoto.

Um excerto dos resultados da análise do CodeQL encontra-se na Figura 19.



<input type="checkbox"/>	<input type="checkbox"/> 50 Open	<input checked="" type="checkbox"/> 0 Closed	Language ▾	Tool ▾	Branch ▾	Rule ▾	Severity ▾	Sort ▾
<input type="checkbox"/>	<input type="checkbox"/> Bad HTML filtering regexp	High						main
#51 opened 4 hours ago • Detected by CodeQL in project/.../prettify/lang-css.js :2								
<input type="checkbox"/>	<input type="checkbox"/> Incomplete multi-character sanitization	High						main
#50 opened 4 hours ago • Detected by CodeQL in project/.../config/TeX-AMS_HTML.js :58								
<input type="checkbox"/>	<input type="checkbox"/> Incomplete multi-character sanitization	High						main
#49 opened 4 hours ago • Detected by CodeQL in project/.../pad/comments.js :118								
<input type="checkbox"/>	<input type="checkbox"/> Incomplete multi-character sanitization	High						main
#48 opened 4 hours ago • Detected by CodeQL in project/.../assert/main.js :506								
<input type="checkbox"/>	<input type="checkbox"/> Double escaping or unescaping	High						main
#47 opened 4 hours ago • Detected by CodeQL in project/.../build/pdf.worker.mjs :35380								
<input type="checkbox"/>	<input type="checkbox"/> Double escaping or unescaping	High						main
#46 opened 4 hours ago • Detected by CodeQL in project/.../build/pdf.worker.js :21302								
<input type="checkbox"/>	<input type="checkbox"/> Double escaping or unescaping	High						main
#45 opened 4 hours ago • Detected by CodeQL in project/.../web/viewer.js :3731								
<input type="checkbox"/>	<input type="checkbox"/> Incomplete string escaping or encoding	High						main
#44 opened 4 hours ago • Detected by CodeQL in project/.../mermaid/zenuml-definition-b4b159... :23602								

Figura 19 – CodeQL

¹² <https://codeql.github.com/>

Ora, como se evidencia na Figura 19, o CodeQL deteta 50 potenciais vulnerabilidades, que se dividem nos seguintes tipos e ocorrências:

1. **Bad HTML filtering regexp (1):** definição incorreta de expressões regulares para filtrar conteúdo HTML;
2. **Incomplete multi-character sanitization (3):** sanitização que não lida de forma adequada com alguns caracteres especiais;
3. **Double escaping or unescaping (3):** *escape* ou *unescape* de dados duas vezes consecutivas, indevidamente;
4. **Incomplete string escaping or encoding (6):** *escape* ou codificação de *inputs* potencialmente incompleto;
5. **Unvalidated dynamic method call (2):** chamada a função cujo nome é definido de forma dinâmica e não validado;
6. **Missing rate limiting (1):** falta de limitação do número de pedidos admissível por intervalo de tempo;
7. **Regular expression injection (1):** utilização de *input* controlado pelo utilizador numa expressão regular, sem validação;
8. **Client-side cross-site scripting (1):** potencial execução de *scripts* provenientes do cliente;
9. **Uncontrolled data used in path expression (2):** utilização de dados não validados para construir caminhos para ficheiros ou diretórios;
10. **Inefficient regular expression (6):** expressão regular ineficiente, que pode prejudicar o desempenho da aplicação;
11. **Polynomial regular expression used on uncontrolled data (2):** aplicação de expressão regular com complexidade polinomial a *input* do utilizador, que pode permitir ataques de *Regular Expression Denial of Service* (REDoS);
12. **Cross-window communication with unrestricted target origin (3):** comunicação entre janelas sem verificação da origem;
13. **Overly permissive regular expression range (1):** expressão regular demasiado permissiva;
14. **DOM text reinterpreted as HTML (14):** inserção direta de conteúdo no DOM para ser interpretado como HTML;
15. **Prototype-polluting function (4):** funções sem proteção contra ataques de *prototype pollution*.

Efetivamente, o CodeQL deteta várias potenciais vulnerabilidades que, sendo verdadeiros positivos, podem significar falhas reais de segurança na aplicação. Note-se que algumas das falhas identificadas – como é o caso do potencial ataque de REDoS – já tinham sido sinalizadas anteriormente por outras ferramentas, o que demonstra a importância de realizar uma análise abrangente. Todavia, a vasta maioria destas vulnerabilidades, se devidamente exploradas, só permitem concretizar ataques de injeção (particularmente, XSS) o que, pelo mecanismo de *sandboxing* já explicado, tem um impacto extremamente reduzido.

Semgrep

O Semgrep¹³ é uma plataforma que atua como ferramenta de análise estática de código (SAST) para identificar potenciais vulnerabilidades, essencialmente assente em regras predefinidas que alertam para eventuais práticas incorretas. Tal como anteriormente, também o Semgrep é capaz de analisar diretamente o repositório do projeto.

A Figura 20 apresenta um exemplo dos resultados obtidos pela análise do Semgrep à aplicação.

The screenshot displays the Semgrep search results interface. At the top, it indicates '173 matching findings' and provides options to sort by highest severity, analyze, or triage. The results are categorized into four sections, each with a title, a brief description, a severity level, and a list of affected files.

- insecure-document-method** (Security, Low): User controlled data in methods like `innerHTML`, `outerHTML` or `document.write` is an anti-pattern that can lead to XSS vulnerabilities. Affected files include `project/customize.dist/loading.js:16`, `project/customize.dist/loading.js:100`, `project/customize.dist/loading.js:102`, `project/customize.dist/pages.js:18`, and `project/customize.dist/pre-loading.js:13`.
- detect-insecure-websocket** (Security, Low): Insecure WebSocket Detected. WebSocket Secure (wss) should be used for all WebSocket connections. Affected files include `project/lib/defaults.js:13` and `project/lib/defaults.js:32`.
- open-redirect-pathname** (Pro, Security, Medium): The application builds a URL using user-controlled input which can lead to an open redirect vulnerability. An attacker can manipulate the URL and redirect users to an arbitrary domain. Open redirect vulnerabilities can lead to issues such as Cross-site scripting (XSS) or redirecting to a malicious domain for activities such as phishing to [Show more](#). Affected files include `project/www/common/sframe-common-outer.js:1559` and `project/www/main.js:29`.
- open-redirect** (Pro, Security, High): The application builds a URL using user-controlled input which can lead to an open redirect vulnerability. An attacker can manipulate the URL and redirect users to an arbitrary domain. Open redirect vulnerabilities can lead to issues such as Cross-site scripting (XSS) or redirecting to a malicious domain for activities such as phishing to [Show more](#).

Figura 20 – Semgrep

Como se verifica na Figura 20, o Semgrep não só elenca as ocorrências identificadas no código, juntamente com a respetiva localização e severidade, mas também lhes atribui um grau de confiança, essencial para facilitar a distinção entre verdadeiros positivos e falsos positivos, no que concerne às vulnerabilidades detetadas.

¹³ <https://semgrep.dev/>

Efetivamente, ao analisar o código-fonte da aplicação, o Semgrep deteta 173 instâncias de potenciais vulnerabilidades, que se dividem em 17 categorias, expostas de seguida, juntamente com o respetivo número de ocorrências:

1. **insecure-document-method (39)**: utilização de dados controlados pelo utilizador em métodos como `innerHTML` e `outerHTML` para injeção no DOM;
2. **detect-insecure-websocket (2)**: utilização de um *WebSocket* inseguro, em vez de *WebSocket Secure* (WSS);
3. **open-redirect-pathname (2)**: construção de um URL usando *input* do utilizador, que pode originar uma vulnerabilidade de *open redirect*;
4. **open-redirect (1)**: idêntico ao anterior;
5. **xss (1)**: utilização de *input* não confiável na geração de uma página *web*;
6. **path-join-resolve-traversal (50)**: deteção de possível *input* do utilizador numa função `path.join` ou `path.resolve`, que pode levar a um ataque de travessia de caminho;
7. **detect-non-literal-regexp (18)**: expressão regular que pode permitir um ataque de *Regular Expression Denial of Service*;
8. **missing-integrity (18)**: falta da *tag integrity* num recurso, que impede a validação da integridade e origem do mesmo;
9. **wildcard-postmessage-configuration (12)**: definição demasiado permissiva (com *wildcard*) da origem ou do destino de uma chamada à API `window.postMessage()`, que pode divulgar informação indevidamente;
10. **insufficient-postmessage-origin-validation (10)**: falta de validação da origem pela API `addEventListener`, que pode permitir ataques de XSS;
11. **prototype-pollution-loop (10)**: possibilidade de modificar atributos de um protótipo de um objeto, para concretizar ataques de *prototype polluting*;
12. **plaintext-http-link (2)**: *link* para um URL em HTTP, em vez de HTTPS;
13. **raw-html-concat (1)**: concatenação de dados controlados pelo utilizador numa *string* HTML;
14. **using-http-server (1)**: utilização de um servidor HTTP, em vez de HTTPS;
15. **insecure-object-assign (1)**: utilização de dados controlados pelo utilizador em `Object.assign()`, que pode levar a uma vulnerabilidade de atribuição em massa;
16. **unsafe-formatstring (3)**: concatenação de uma *string* com uma variável numa chamada a `console.log()`, passível de manipulação do *output*;
17. **express-check-csurf-middleware-usage (2)**: ausência de deteção de um *middleware* para CSRF na aplicação.

Apesar de todas estas potenciais falhas de segurança no código, apenas quatro ocorrências (dos tipos **open-redirect**, **xss** e **plaintext-http-link**) correspondem a um nível de confiança elevado e todas elas são de severidade média. Como tal, isto não se afigura, provavelmente, um risco crítico de segurança.

Aliás, considerando as ocorrências com confiança média, surgem apenas 15 potenciais vulnerabilidades de três tipos (**open-redirect-pathname**, **wildcard-postmessage-configuration** e **raw-html-concat**), todos de severidade média. Portanto, mais uma vez, nenhum destes casos impacta de forma séria a segurança aplicacional.

Consequentemente, a maior parte (154) das falhas de segurança identificadas pelo Semgrep têm associado um nível de confiança baixo, pelo que não se analisam de forma mais aprofundada.

Deste modo, demonstra-se a importância de executar ferramenta de análise estática e *scanners* de vulnerabilidades para detetar potenciais falhas de segurança no código-fonte.

SonarQube Cloud

Por último, o SonarQube Cloud¹⁴ é um analisador estático automatizado que procede a verificações contínuas de segurança e qualidade sobre o código-fonte. Esta ferramenta também pode ser integrada no processo de desenvolvimento.

A Figura 21 contém os resultados gerais da aplicação do SonarQube Cloud ao repositório remoto que aloja o projeto.

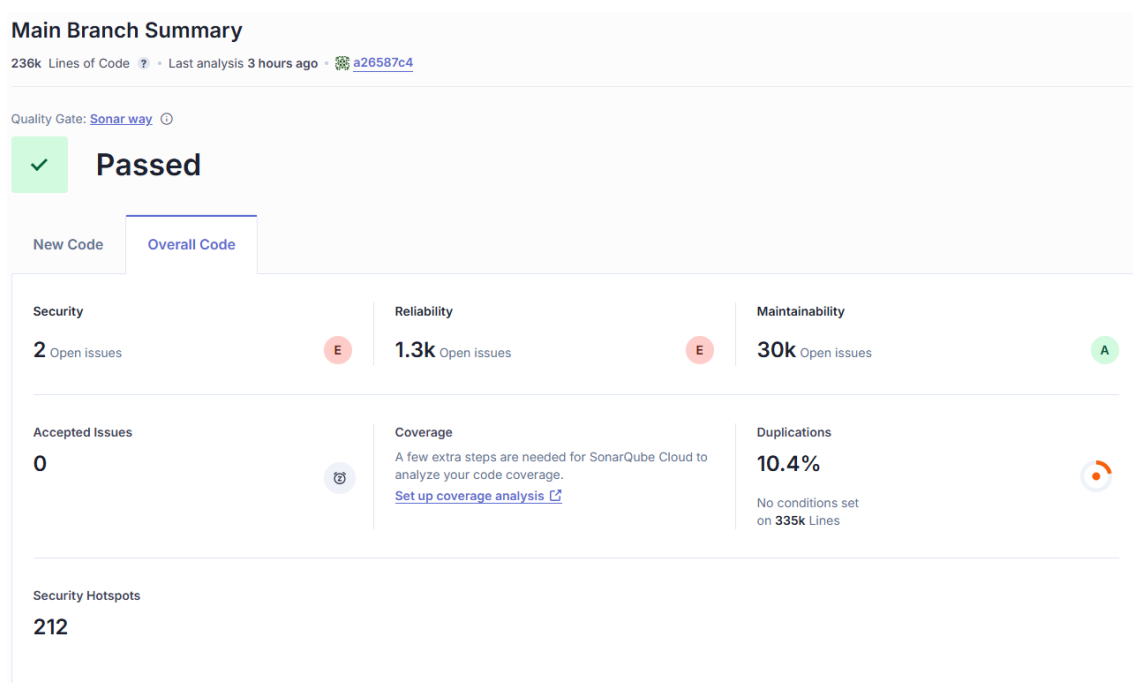


Figura 21 – SonarQube Cloud

¹⁴ <https://sonarcloud.io/>

Na sua visão geral, o SonarQube Cloud identifica apenas dois problemas de segurança, apesar de levantar inúmeras questões de fiabilidade – consistência, intencionalidade, adaptabilidade e responsabilidade – e manutenção do código, mas que se encontram fora do âmbito deste projeto. A Figura 22 detalha os alertas de segurança identificados.

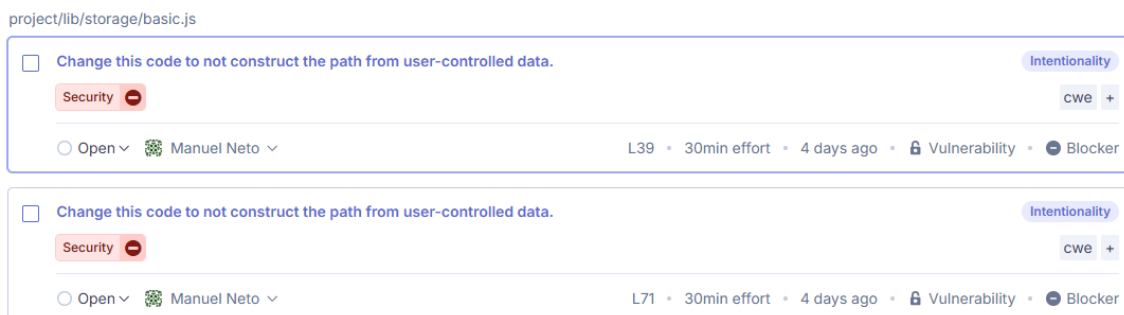


Figura 22 – SonarQube Cloud (Detalhe)

Efetivamente, ambos os alertas lançados pela plataforma têm severidade crítica e prendem-se com a construção de caminhos a partir de dados controlados pelo utilizador. Na prática, estes dois casos devem-se, em concreto, ao processo de *taint analysis* efetuado pelo SonarQube Cloud, que identifica que o *input* do utilizador proveniente do ficheiro **lib/http-worker.js** (*source*) flui até ao ficheiro **lib/storage/basic.js** (*sink*) para ler ou eliminar, respetivamente, um ficheiro do sistema de ficheiros, referente ao processo de autenticação. Pelo meio, o *input* passa por todo o mecanismo de desafio-resposta associado ao protocolo de autenticação.

Assim sendo, apesar da utilidade da deteção do SonarQube Cloud, torna-se complexo compreender o código na sua plenitude ao ponto de ser capaz de o modificar para corrigir esta potencial vulnerabilidade, ou de compreender se, no fluxo de dados, existe algum processo de sanitização/validação implementado e adequado, que invalide a deteção automática.

Por tudo isto, realça-se, uma vez mais, a importância da utilização de ferramentas de análise de código num processo de desenvolvimento que se quer seguro. Tal como é possível verificar, estas ferramentas não só aumentam a segurança do código em geral, mas também o fazem de forma automatizada, diminuindo os potenciais impactos sobre os engenheiros de *software* e facilitando o seu trabalho.

Análise Global

Considerando as três ferramentas de análise de código aplicadas ao processo de desenvolvimento, é possível comparar o seu desempenho através de uma análise global.

Nesse sentido, a Tabela 9 consolida os resultados obtidos para as diferentes ferramentas, agrupando as descobertas por níveis/categorias de severidade.

Severidade	CodeQL	Semgrep	SonarQube Cloud
Crítica	0	0	2
Elevada	28	41	0
Média	22	127	0
Baixa	0	5	0
Total	50	173	2

Tabela 9 – Análise Global

Efetivamente, tal como demonstra a Tabela 9, o número de vulnerabilidades detetas pelas diferentes ferramentas é bastante díspar. Por um lado, a deteção de uma maior quantidade de vulnerabilidades pode dever-se a capacidades de análise mais avançada, abrangente e aprofundada, mas, por outro lado, isto também pode ser resultado de uma análise menos precisa ou exata, na qual a maioria dos resultados poderão ser falsos positivos ou deteções com baixa confiança associada.

Assim sendo, tal como no princípio de defesa em profundidade, também na utilização de plataformas/ferramentas de análise de código é importante diversificar os mecanismos utilizados e as técnicas adotadas, de maneira a obter os melhores resultados possíveis. Para isso, tal como foi feito, é essencial recorrer a diversas soluções tecnológicas e analisar os resultados não só de forma individual, mas também de forma consolidada.

Deste modo, é possível concluir um processo de análise abrangente à solução implementada – **CryptPad** –, não deixando de realçar que, embora a própria plataforma não divulgue os meios que implementados para garantir a segurança do código-fonte, tem uma política de segurança associada que incentiva a divulgação de vulnerabilidades de forma responsável, o que auxilia na rápida deteção e resolução/mitigação de potenciais falhas de segurança.

Com isto, dá-se por concluída a análise ao sistema implementado.

Lições Aprendidas

Depois de serem seguidas, de forma sequencial, as três fases do processo de desenvolvimento – *design*, implementação e análise –, é possível refletir sobre o mesmo como um todo. De facto, sendo este um ato contínuo e iterativo, é natural que os resultados de fases posteriores suscitem a revisitação de etapas anteriores.

Em primeiro lugar, observa-se que existem vulnerabilidades resultantes da implementação, que não se encontravam contempladas no processo de *design*. Em particular, a possibilidade – inerente à arquitetura do **CryptPad** – de dois utilizadores se registarem com o mesmo nome materializa-se numa eventual ameaça que não foi considerada previamente. Efetivamente, este facto permite que um utilizador, até de forma indevida, se autentique na conta de outro utilizador, ao enganar-se na sua própria palavra-passe, constituindo este fluxo de execução um *misuse case*. No entanto, na prática, este possível ataque acaba por ser prevenido ao forçar a autenticação multifator para todos os utilizadores, o que realça a importância deste mecanismo complementar de autenticação.

Em segundo lugar, esta obrigação de MFA acaba por, em sentido inverso, diminuir a flexibilidade do sistema, trazendo dificuldades acrescidas ao processo de análise. Concretamente, a integração com a ferramenta de *scan* automático de vulnerabilidades OWASP Zap fica limitada à realização de ações por utilizadores não autenticados, visto que não tem a capacidade de completar o desafio associado ao segundo fator de autenticação. A par disto, tentou-se também guiar o *scan* da ferramenta OWASP Zap através de um ficheiro HAR com capturas do tráfego associado ao uso da plataforma, mas constatou-se que a versão utilizada desta ferramenta não suporta esta configuração, dado que realiza o processo de *spidering* autonomamente para percorrer a plataforma.

Em terceiro lugar, embora o processo de análise tenha sido bem-sucedido, pode salientar-se a importância de transpor a análise de bibliotecas e dependências para o momento de implementação, de maneira a reduzir, desde logo, eventuais vulnerabilidades. Assim, reforça-se a importância de ter a segurança em consideração em todas as fases do desenvolvimento.

Por último, comparando a arquitetura do sistema implementado com a arquitetura proposta para o mesmo, observa-se que a primeira é extremamente mais detalhada e concreta do que a segunda. Ora, apesar de isto ser algo natural como resultado de um processo de desenvolvimento, pretende-se que a *gap* inevitável entre o *design* e a implementação seja tão reduzida quanto possível.

Em síntese, apesar de todo o processo de desenvolvimento – desde o *design* até à análise, passando pela implementação – ter sido globalmente bem-sucedido, isto não invalida que tenham surgido aspetos a melhorar em eventuais desenvolvimentos futuros semelhantes, englobados em cada uma das três fases.

Conclusão

Finalmente, cumpre refletir sobre todo o processo de desenvolvimento desenrolado e versado neste relatório, dividido em três principais etapas.

Em primeiro lugar, tratou-se do *design* do projeto a desenvolver, de acordo com o enunciado previamente estabelecido. Nesse sentido, procurou-se que o sistema satisfizesse determinados requisitos e seguisse a arquitetura estabelecida. A par disto, definiram-se também os atores intervenientes no sistema e as suas respetivas funções/responsabilidades, bem como os *use*, *misuse* e *abuse cases* correspondentes. Por fim, modelaram-se as ameaças do sistema segundo a metodologia/framework STRIDE.

Em segundo lugar, na etapa central do projeto, implementou-se o editor de documentos colaborativo idealizado. Para isto, aproveitou-se a implementação já existente do **CryptPad**, por ser aquela que, das opções disponíveis e ponderadas, mais fielmente cumpria e respeitava os requisitos anteriormente delineados. Esta implementação foi ainda devidamente configurada, seguindo um processo de *hardening* que visa reforçar a segurança do projeto.

Em terceiro e último lugar, procedeu-se à análise do sistema como um todo em termos da sua segurança. Para isso, teceram-se considerações sobre a linguagem de programação JavaScript, procedidas pela observação dos *design patterns* e da arquitetura do **CryptPad**, realçando-se a encriptação ponto-a-ponto e a filosofia *zero-knowledge* do servidor sobre os utilizadores, garantindo a confidencialidade e a privacidade como prioridades. Complementarmente, investigaram-se as dependências e bibliotecas do sistema quanto à existência de potenciais vulnerabilidades, através do recurso a ferramentas como **npm audit**, **Snyk** e **Dependabot**. Posteriormente, consideram-se as mitigações colocadas em prática para as vulnerabilidades mais comum – extraídas do **OWASP Top 10** –, destacando-se o mecanismo de *sandboxing* explicitado. Finalmente, seguiu-se uma metodologia de testes de segurança integrando a ferramenta **OWASP Zap** numa **GitHub Action**, juntamente com três ferramentas de análise de código: **CodeQL**, **Semgrep** e **SonarQube Cloud**.

Assim, o desenvolvimento de um editor de documentos colaborativo através do **CryptPad** demonstrou a viabilidade e a importância de adotar princípios de *Security by Design*, considerando a segurança desde as fases iniciais do projeto, numa mentalidade de *Shift-Left Security* no processo de *SecDevOps*.

Em suma, este trabalho cumpriu não só os seus objetivos funcionais, mas também serviu como uma demonstração prática de que a segurança e a usabilidade podem coexistir quando as decisões de *design* são orientadas por uma mentalidade fortemente focada na segurança. Deste modo, o projeto considera-se bem-sucedido.