Universidade do Minho
Escola de Engenharia

Mestrado em Engenharia Informática

Ano letivo 2024/2025

Engenharia de Segurança

Cofre Digital - Trabalho Prático #2

Grupo 02

Eduardo Cunha - PG55939
João Rodrigues - PG57880

Índice

1 Introdução

Esta segunda fase do projeto "Cofre Digital" consiste na passagem da simples definição de requisitos e prototipagem inicial para a concretização efetiva de uma solução completa. O principal objetivo consiste na implementação integral de uma WebAPI segura e fiável, capaz de suportar todas as operações de gestão de cofre, desde a autenticação e encriptação dos dados até ao armazenamento e recuperação de informação sensível. Em paralelo, será desenvolvida uma aplicação cliente de linha de comando leve e intuitiva, que permitirá ao utilizador interagir de forma consistente com a API, garantindo uma experiência de utilização segura.

Para além da implementação de código, outro dos objetivos fundamentais desta fase é a criação de documentação. Proceder-se-á, neste relatório, à explicação de fluxos de autenticação, decisões de design adotadas e a estrutura modular dos componentes da aplicação.

Finalmente, pretende-se realizar um levantamento completo das dependências do projeto, através da elaboração de um Software Bill of Materials (SBOM). Este inventário detalhará todas as bibliotecas e componentes utilizados (como FastAPI, Typer, Pydantic, PyJWT, joserfc, entre outros,...) permitindo avaliar riscos de segurança associados, garantir a conformidade com normas e facilitar a identificação de eventuais vulnerabilidades.

2 Arquitetura da solução

A arquitetura da solução consistiu no desenvolvimento de uma API utilizando FastAPI. O cliente foi desenvolvido com recurso ao Typer, de forma a facilitar a interação com a API via linha de comandos.

Relativamente à API, esta tem acesso a uma base de dados PostgreSQL, onde são armazenadas todas as informações relacionadas com o servidor, como ficheiros, utilizadores, permissões, entre outros. A API estabelece também uma ligação a um servidor Redis, que é exclusivamente responsável pela gestão da blacklist de tokens. Esta decisão foi tomada com o objetivo de melhorar a escalabilidade da solução proposta.

No lado do servidor, foram criadas diversas classes para abstrair a lógica e simplificar a manutenção do código. Entre elas:

- AccountManager: Responsável pela gestão de contas, incluindo tentativas de login e verificação de credenciais.
- DBManager: Responsável por todo o acesso à base de dados. Embora funcione de forma semelhante a um ORM, optámos por definir diretamente as *queries* SQL em Python, utilizando a biblioteca *psycopg2*, uma vez que as *queries* eram relativamente simples e, desta forma, conseguimos garantir que o SQL é executado exatamente como esperado.
- IntegrityManager: Responsável por periodicamente verificar a integridade de todos os ficheiros armazenados.
- Logger: Responsável por registar todos os logs das ações realizadas pelos clientes no servidor.

Segue abaixo um esquema da arquitetura:

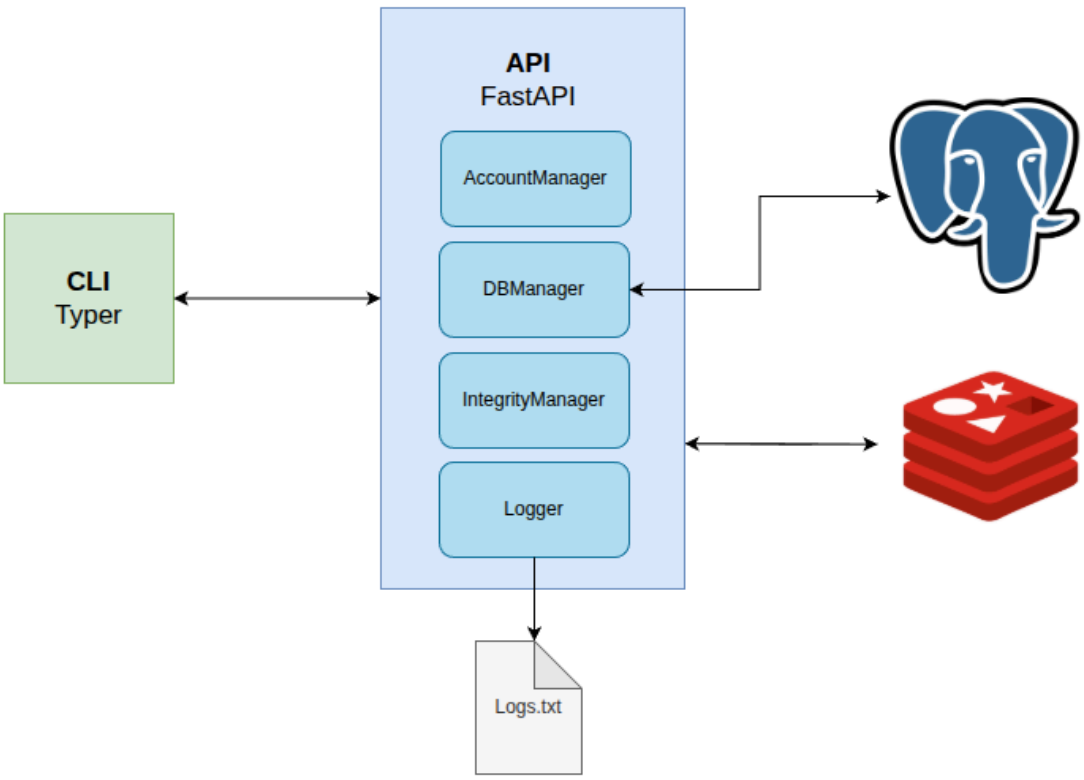

Figura 2: Diagrama não técnico da arquitetura

3 Cliente

O cliente encontra-se implementado em dois ficheiros distintos: o *"cliente.py"*, onde estão definidos os comandos do Typer que fazem os pedidos à API e tratam as respetivas respostas, e o *"run.py"*, que serve como ponto de entrada para o utilizador, limitando-se a invocar as funções definidas no *"cliente.py"*.

Este cliente comunica com o servidor através de uma API REST, utilizando HTTPS para garantir comunicações seguras. Foi desenvolvido com o objetivo de oferecer funcionalidades como autenticação de utilizadores, gestão de ficheiros e pastas, e encriptação de ponta a ponta.

O cliente funciona como uma interface de linha de comandos (CLI) simples, recorrendo ao terminal para receber os inputs do utilizador e apresentar os resultados das operações efetuadas.

Para a edição do conteúdo de ficheiros, o cliente integra o utilitário externo *vipe*, que permite ao utilizador editar texto recorrendo ao seu editor padrão. Este método de interação torna a utilização do sistema simples e direta.

A opção por uma interface CLI foi inicialmente sugerida e implementada pela sua simplicidade. No entanto, ao longo do desenvolvimento, considerámos que uma implementação TUI (Text-based User Interface) poderia ter trazido vantagens adicionais, nomeadamente na gestão de tokens de acesso e refresh, que poderiam ser mantidos em memória, bem como na persistência do utilizador atual, facilitando assim os testes com múltiplos utilizadores.

Como optámos por manter a abordagem em CLI, decidimos guardar os tokens num ficheiro cifrado, como será detalhado mais à frente, e para simular diferentes utilizadores durante os testes, recorrermos à utilização de variáveis de ambiente.

3.1 Operações do cliente

O cliente oferece um conjunto abrangente de operações para gerir contas, ficheiros e permissões:

3.1.1 Gestão de conta:

- **Criação de conta (criar-conta):** Regista um novo utilizador no sistema, gerando e armazenando o par de chaves RSA;
- **Autenticação (login, login-google):** Permite o acesso ao sistema através de credenciais ou conta Google;
- **Remoção de conta (remover-conta):** Elimina permanentemente a conta do utilizador e todos os seus ficheiros dos quais é dono;
- **Ativação e desativação de 2FA (ativar_2fa, desativar_2fa):** Configura a autenticação de dois fatores;
- **Encerramento de sessão (logout):** Termina a sessão atual, invalidando os tokens de acesso.

3.1.2 Gestão de ficheiros:

- **Carregamento de ficheiros (upload_ficheiro):** Cifra o ficheiro localmente e envia para o servidor;
- **Leitura de ficheiros (ler_ficheiro):** Obtém, decifra e apresenta o conteúdo do ficheiro;
- **Modificação de ficheiros (modificar_ficheiro):** Permite editar o conteúdo completo de um ficheiro;
- **Adição de conteúdo (adicionar_ao_ficheiro):** Anexa novo conteúdo ao final do ficheiro existente;
- **Remoção de ficheiros (remover_ficheiro):** Elimina o ficheiro do servidor.

3.1.3 Gestão de pastas:

- **Criação de pastas (criar-pasta):** Cria uma nova pasta no sistema, opcionalmente dentro de outra pasta;

- **Listagem do conteúdo (listar-conteudo-pasta):** Mostra os ficheiros e subpastas dentro de uma pasta;
- **Remoção de pastas (remover-pasta):** Elimina uma pasta e todo o seu conteúdo;
- **Visualização do cofre (listar-conteudo-cofre):** Apresenta uma visão geral dos recursos do cofre do utilizador.

3.1.4 Gestão de permissões:

- **Concessão de permissões (conceder-permissao):** Partilha recursos com outros utilizadores, cifrando a chave do ficheiro com a chave pública do destinatário;
- **Remoção de permissões (remover-permissao):** Revoga o acesso previamente concedido a um utilizador;
- **Alteração de níveis (alterar-permissao):** Modifica o nível de acesso que um utilizador tem a um recurso;
- **Listagem de recursos com permissão (listar-recursos-com-permissao):** Mostra todos os recursos a que o utilizador tem acesso.

3.1.5 Operações adicionais:

- **Renovação de tokens (refresh):** Atualiza automaticamente tokens de acesso expirados, caso o *refresh token* seja válido. Esta operação não é acessivel ao utilizador mas é usada aquando o primeiro aviso de *acess token* inválido em algum *request*.
- **Visualização de registos (logs):** Apresenta o registo de logs. Esta operação é apenas válida a *users* considerados administradores do sistema.

Cada operação segue um padrão semelhante: em primeiro lugar, é enviado um pedido ao servidor; em seguida, a resposta é analisada e os resultados são apresentados ao utilizador. Durante a análise da resposta, caso esta indique que o token de acesso expirou, o cliente tenta renová-lo automaticamente e repete o pedido uma única vez.

3.2 Autenticação no cliente

O sistema de autenticação implementado é robusto e assenta em múltiplas camadas de segurança. O cliente recorre a um mecanismo de autenticação baseado em tokens JWT (JSON Web Tokens), que são armazenados em ficheiros localmente de forma segura, utilizando a classe Fernet para cifragem dos dados. São utilizados dois tipos de tokens: o access token, destinado à autenticação de operações de curta duração, e o refresh token, que permite obter novos tokens de acesso quando o anterior expira.

A decisão de implementar também refresh tokens deve-se ao facto de conhecermos as limitações do armazenamento de tokens em ficheiro. Para aumentar a segurança da aplicação, optámos pela utilização de refresh tokens, de forma a garantir que os access tokens tenham uma validade extremamente curta, minimizando assim as janelas de ataque e aumentando a proteção do sistema. Além disso, a chave para cifrar os tokens é armazenada separadamente, com permissões restritas.

Relativamente ao login, o cliente oferece duas opções, com a possibilidade de uma terceira. A primeira é um login normal, utilizando o fluxo *OAuth2 Password Grant*, onde os parâmetros de *username* (que, no nosso caso, corresponde ao e-mail) e *password* são verificados através do *schema OAuth*, sendo validados pela API na base de dados.

A segunda opção é o login via OAuth2 pelo Google, onde, após a autenticação, o utilizador recebe um código que deve inserir na aplicação CLI. Este código permite à API obter os dados do utilizador através de um *callback* na Google e realizar a autenticação da conta.

Adicionalmente, o sistema suporta autenticação em dois fatores (2FA), que, caso o utilizador tenha ativado, será obrigatória. Este processo de autenticação é feito através de uma aplicação móvel, o Google Authenticator.

3.2.1 Comunicação Cliente API:

Relativamente à comunicação entre o cliente e a API, esta é realizada através de HTTPS, garantindo que todas as comunicações são verificadas com certificados, cifradas e seguras. É importante destacar que, em ambiente de desenvolvimento, devido ao uso de certificados autoassinados, podem surgir avisos de segurança em algumas situações, como ao abrir a página da FastAPI para testar as rotas ou ao utilizar o fluxo OAuth na web.

3.3 Cifragem no cliente

A cifragem implementada no cliente segue o princípio de cifra de ponta a ponta, onde os dados são cifrados antes de serem enviados ao servidor e apenas o cliente possui as chaves necessárias para os decifrar, garantindo que o servidor não tem acesso a conteúdos sensíveis.
Para tal foi usado o método de cifragem por envelope. O ficheiro é cifrado com uma chave e essa chave é cifrada com uma chave pública (inicialmente do dono e posteriormente daqueles com quem é partilhado)

3.3.1 Gestão de chaves assimétricas:

Para as chaves relativas a um utilizador, é gerado um par de chaves RSA, composto por uma chave pública e uma chave privada. A chave privada é armazenada localmente no dispositivo do utilizador e nunca é enviada para o servidor, garantindo a confidencialidade e segurança da informação sensível. Por outro lado, a chave pública é partilhada com o servidor, permitindo que este possa cifrar dados destinados ao utilizador, que só poderão ser descifrados através da correspondente chave privada.

3.3.2 Cifra híbrida para ficheiros:

Na cifra híbrida aplicada a ficheiros, recorre-se à combinação de criptografia simétrica e assimétrica para garantir tanto a eficiência como a segurança. O conteúdo dos ficheiros é cifrado utilizando o algoritmo simétrico AES-GCM, conhecido pela sua elevada performance e pela capacidade de garantir a confidencialidade e integridade dos dados. Posteriormente, a chave simétrica usada para cifrar o ficheiro é, por sua vez, protegida através de criptografia assimétrica com RSA. Esta chave cifrada pode então ser partilhada de forma segura, já que apenas o destinatário que possuir a correspondente chave privada poderá descifrá-la e, assim, aceder ao conteúdo do ficheiro.

3.3.3 Geração segura de chaves de ficheiros:

Para a geração de chaves únicas dos ficheiros foi utilizado o algoritmo Argon2 para derivação de chaves resistentes a ataques.

3.3.4 Segurança na partilha de ficheiros:

Quando um ficheiro é partilhado, a chave única do ficheiro é cifrada com a chave pública do utilizador que receberá a permissão e posteriormente enviada de volta para armazenamento na base de dados

Esta implementação de cifragem garante que mesmo se o servidor for comprometido, os dados permanecerão protegidos, pois apenas o cliente possui as chaves necessárias para decifrar os conteúdos.

4 API

Esta API RESTful funciona como o backend da aplicação, permitindo aos utilizadores criar contas, autenticar-se, gerir pastas e ficheiros, bem como definir permissões de acesso para outros utilizadores.

Como já foi referido, a API é composta por várias classes que aumentam o nível de abstração e modularidade da aplicação, tornando o código mais organizado e altamente manutenível. As principais classes são as seguintes:

- AccountManager: Responsável pela gestão de contas de utilizadores;
- DBManager: Gere todo o acesso à base de dados;
- PermissionManager: Controla as permissões de acesso aos recursos;
- IntegrityManager: Verifica a integridade dos ficheiros armazenados;
- Logger: Regista todas as atividades realizadas na aplicação;

4.1 Schema e Validators

A API utiliza o modelo Pydantic Request_Schema para validar os dados recebidos nos pedidos do cliente. Esta abordagem assegura que os dados estão corretamente formatados antes de serem processados pela aplicação, funcionando assim como um mecanismo de sanitização de inputs que poderiam, potencialmente, conter tentativas de injeção ou dados malformados.

O esquema implementado inclui diversos validators específicos para diferentes campos, com o objetivo de reforçar a fiabilidade e segurança dos dados recebidos. Entre as validações aplicadas destacam-se:

- Validação de Email: Utiliza uma expressão regular (regex) para garantir que o endereço segue o formato correto, como por exemplo “email@exemplo.com”.
- Validação de Password: Verifica se a palavra-passe tem entre 5 e 32 caracteres.
- Validação de Nome: Verifica se o nome do utilizador segue um formato normal.
- Validação de Tipo de Recurso: Confirma se o tipo de recurso é válido (“pasta” ou “ficheiro”).
- Validação de Nível de Acesso: Garante que o nível de acesso é um dos valores permitidos (“read”, “append”, “write”).
- Validação de Nome de Ficheiro: Certifica-se que o nome do ficheiro tem formato válido e apenas permite ficheiros “.txt”.
- Validação de Código 2FA: Verifica se o código de autenticação de dois fatores tem exatamente 6 dígitos.
- Validação de UUID: Confirma se os identificadores fornecidos seguem o formato UUID válido.
- Validação de Base64: Assegura que campos criptográficos, como chaves e conteúdo cifrado estão devidamente codificados em *Base64*.

4.2 Endpoints

A API disponibiliza diversos endpoints para interação com o sistema. Em complemento ao que já foi descrito na parte do cliente, segue-se uma breve explicação de cada um. É importante destacar que todas as ações realizadas através dos endpoints são registadas nos logs, garantindo rastreabilidade e facilitando a monitorização e auditoria da aplicação.

POST /criar_conta
O endpoint POST /criar_conta permite a criação de uma nova conta de utilizador. Este requer a submissão de um e-mail, uma palavra-passe e a chave pública do utilizador, previamente gerada na interface CLI antes do envio do pedido. A API verifica se o e-mail fornecido já está associado a outra conta; caso contrário, e estando todos os dados válidos, é criada uma nova entrada correspondente ao utilizador na base de dados.

POST /login
O endpoint POST /login realiza a autenticação do utilizador. Para além de verificar se as credenciais estão corretas, a API assegura que a conta não excedeu o limite de cinco tentativas de login falhadas nos últimos cinco minutos, como medida de segurança contra ataques de força bruta. Caso a autenticação em dois fatores (2FA) esteja ativada para o utilizador, a API comunica essa exigência ao cliente, que por sua vez solicitará ao utilizador o código correspondente e será efetuado o login através de outro endpoint. Quando o processo de autenticação é bem-sucedida, são então gerados e devolvidos os tokens de acesso e de refresh.

POST /login/2fa
O endpoint POST /login/2fa recebe o e-mail do utilizador e o respetivo código de autenticação de dois fatores. Primeiramente, a API verifica se o utilizador tem efetivamente a autenticação em dois fatores ativa. Em seguida, procede à validação do código TOTP fornecido, utilizando a biblioteca pyotp. Caso a verificação seja bem-sucedida, são então gerados e devolvidos os tokens de acesso e de refresh, concluindo o processo de autenticação.

POST /2fa
O endpoint POST /2fa é uma rota protegida, acessível apenas a utilizadores autenticados. A API obtém o utilizador que efetuou o pedido através do access token presente no request. Esta operação ativa a autenticação de dois fatores na conta do utilizador, gerando um *secret* e atualizando o respetivo registo na base de dados para refletir essa ativação. Após a atualização, é gerado e devolvido um código e um URI necessários para a configuração da aplicação de autenticação, o Google Authenticator.

DELETE /2fa
O endpoint DELETE /2fa é uma rota protegida, acessível apenas a utilizadores autenticados. A API identifica o utilizador através do access token incluído no pedido e procede à desativação da autenticação de dois fatores na respetiva conta. Esta ação consiste em atualizar o campo correspondente na base de dados, definindo-o como false, e assim remover a obrigatoriedade do 2FA nos logins futuros.

POST /logout
O endpoint POST /logout é uma rota protegida, acessível apenas a utilizadores autenticados. A API identifica o utilizador através do access token presente no pedido. Ao receber a solicitação, termina a sessão do utilizador, adicionando os tokens de acesso e de refresh à *blacklist* mantida no Redis. Estes tokens são armazenados com um tempo de expiração (TTL) correspondente ao tempo restante da sua validade. Esta abordagem assegura que os tokens não possam ser reutilizados e contribui para uma solução altamente escalável na gestão de sessões.

POST /remover_conta
O endpoint POST /remover_conta é uma rota protegida, acessível apenas a utilizadores autenticados. A API identifica o utilizador através do access token presente no pedido. Este endpoint permite a remoção da conta do utilizador do sistema. Para confirmar a intenção de remoção, são solicitados o e-mail e a palavra-passe do utilizador. A API verifica primeiro se o e-mail autenticado corresponde ao e-mail da conta que se pretende eliminar e, em seguida, valida a palavra-passe associada a essa conta. Caso as validações sejam bem-sucedidas, a conta é eliminada juntamente com todos os ficheiros pertencentes ao utilizador, incluindo os ficheiros do seu cofre e quaisquer permissões que outros utilizadores possam ter sobre esses ficheiros.

GET /cofre
O endpoint GET /cofre é uma rota protegida, acessível apenas a utilizadores autenticados. A API identifica o utilizador através do access token presente no pedido e devolve uma lista com o conteúdo do cofre do utilizador, ou seja, a raiz do seu armazenamento.

POST /pastas
O endpoint POST /pastas é uma rota protegida, acessível apenas a utilizadores autenticados. A API obtém o utilizador através do access token e cria uma nova pasta dentro do cofre do utilizador. Para tal, requer o nome da pasta e, opcionalmente, o ID da pasta pai. Caso o ID da pasta pai não seja fornecido, a pasta é criada diretamente na raiz do cofre.

DELETE /pastas/{pasta_id}
O endpoint DELETE /pastas/{pasta_id} é uma rota protegida, acessível apenas a utilizadores autenticados. A API verifica se a pasta especificada pertence ao cofre do utilizador e, caso afirmativo, procede à remoção da pasta e de todo o seu conteúdo, incluindo subpastas e ficheiros.

Deve-se destacar que é utilizado o ID da pasta em vez do nome da própria pasta. Ficámos com dúvidas sobre se o ID deveria ser utilizado nas rotas, mas, no caso de acessos a ficheiros fora do cofre, é justificável usar o ID em vez do nome ou do *path*. Isto deve-se ao facto de existirem vários cofres e à possibilidade de múltiplos cofres possuírem os mesmos nomes de pastas e ficheiros, o que aumentaria consideravelmente a complexidade da busca. Não consideramos que isso seja necessário, visto que o foco do trabalho é a segurança. Estamos cientes desta situação e a norma estabelecida é que, nas rotas que se aplicam apenas a ficheiros ou pastas do próprio cofre, utiliza-se o nome do ficheiro ou o seu *path*. Já para as rotas que permitem o acesso a ficheiros ou pastas de outros cofres, utiliza-se o ID. Curiosamente, este exemplo é a única exceção, sendo uma pequena gralha dado que o 'delete' só é possível no cofre do utilizador, mas está com o ID da pasta.

GET /pastas/{pasta_id}
O endpoint GET /pastas/{pasta_id} é uma rota protegida, acessível apenas a utilizadores autenticados. A API obtém o utilizador através do access token e verifica se a pasta especificada pertence ao cofre do utilizador. Caso contrário, é verificado se o utilizador tem permissões de acesso àquela pasta. Se as permissões forem válidas, a API retorna o conteúdo da pasta especificada.

GET /permissoes
O endpoint GET /permissoes é uma rota protegida, acessível apenas a utilizadores autenticados. A API identifica o utilizador através do access token presente no pedido e consulta a base de dados para obter todos os recursos aos quais o utilizador tem permissões de acesso, com base na tabela de permissões. O resultado dessa pesquisa é então devolvido ao cliente.

POST /permissoes
O endpoint POST /permissoes é também uma rota protegida, acessível apenas a utilizadores autenticados. A API identifica o utilizador através do access token e recebe no pedido, o e-mail de outro utilizador a quem será concedida permissão, o tipo de recurso (ficheiro ou pasta), o respetivo ID e o nível de permissão a conceder (read, append ou write). Antes de criar a permissão, a API verifica se o utilizador que está a concedê-la é efetivamente o proprietário do recurso. Caso essa verificação seja positiva, é criada uma nova permissão para permitir o acesso do outro utilizador ao recurso especificado.

PUT /permissoes
O endpoint PUT /permissoes é uma rota protegida que também exige autenticação. A API identifica o utilizador através do access token e recebe os dados necessários para a alteração de permissões: o e--mail do utilizador a quem se pretende alterar o acesso, o tipo e o ID do recurso, bem como o novo nível de permissão. Após verificar se quem está a tentar alterar a permissão é o proprietário do recurso, a API confirma se já existe uma permissão associada. Se existir, procede à sua atualização com o novo nível de acesso.

DELETE /permissoes
O endpoint DELETE /permissoes é uma rota protegida, acessível apenas a utilizadores autenticados.

A API identifica o utilizador através do access token e recebe os dados do utilizador cuja permissão se pretende remover, bem como o ID do recurso em questão. Antes de efetuar a remoção, a API verifica se o utilizador autenticado é o proprietário do recurso. Se essa condição se verificar, a permissão é então removida da base de dados.

GET /logs
O endpoint GET /logs é uma rota protegida com nível de segurança elevado, acessível exclusivamente a utilizadores autenticados com estatuto de administrador. A API identifica o utilizador através do access token, verifica os seus privilégios e, caso seja um administrador, procede à descodificação segura dos registos de atividade (logs). Estes são então enviados ao cliente, protegidos pela cifra do protocolo HTTPS, permitindo assim aos administradores monitorizar o funcionamento e uso do sistema de forma segura.

POST /refresh
O endpoint POST /refresh permite a renovação de um token de acesso. Recebe um refresh token válido e, após verificar a sua autenticidade e validade, gera e devolve ao cliente um novo access token. Esta funcionalidade permite manter sessões ativas de forma segura, mesmo após a expiração do token de acesso.

GET /obter_chave_ficheiro/{ficheiro_id}
Este endpoint permite a obtenção da chave cifrada de um ficheiro específico para o utilizador autenticado. O sistema verifica a autenticação do utilizador através do token de acesso e extrai o email do utilizador. Posteriormente, consulta a base de dados para recuperar a chave do ficheiro correspondente ao ID fornecido, garantindo que apenas utilizadores autorizados possam aceder à chave do ficheiro em questão.

GET /obter_chave_publica/{email_user}
Este endpoint fornece a chave pública de um utilizador específico, identificado pelo seu endereço de email. O sistema requer autenticação através do token de acesso válido. A chave pública é recuperada da base de dados e retornada na resposta.

POST /ficheiros/upload
O endpoint de *upload* permite o carregamento de um novo ficheiro cifrado para o sistema. Requer autenticação através de um token de acesso válido e recebe os dados necessários para o armazenamento seguro do ficheiro, incluindo o nome do ficheiro, o conteúdo cifrado, o vetor de inicialização (IV), a tag de autenticação e a chave cifrada. O sistema também aceita um ID de pasta pai opcional, permitindo a organização hierárquica dos ficheiros. É realizada uma validação inicial para garantir que todos os dados necessários estão presentes. O processo de upload é gerido pelo AccountManager, que trata do armazenamento seguro do ficheiro, e todas as ações são registadas pelo Logger, juntamente com detalhes sobre o sucesso ou falha da operação.

DELETE /ficheiros/{path_ficheiro}/remover
Este endpoint de *delete* permite a remoção de um ficheiro do sistema. Requer autenticação através de um token de acesso válido e o caminho do ficheiro a ser removido. O sistema verifica se o utilizador autenticado tem permissões para remover o ficheiro em questão e, em caso afirmativo, procede à sua remoção. Todas as ações são registadas pelo sistema de logging, bem como detalhes sobre o sucesso ou falha da operação.

GET /ficheiros/{ficheiro_id}/ler
Este endpoint permite a leitura de um ficheiro identificado pelo ficheiro_id. O sistema extrai o email do utilizador através do token de acesso e utiliza o função AccountManager para processar o pedido de leitura. O resultado da operação é registado através do Logger e é devolvida uma resposta indicando

o sucesso ou falha da operação, que incluirá o conteúdo do ficheiro e o necessário para o decifrar em caso de sucesso.

GET /ficheiros/{ficheiro_id}/pedido-modificar
Este endpoint inicia o processo de modificação de um ficheiro, identificado pelo ficheiro_id. O sistema extrai o email do utilizador através do token de acesso e utiliza o AccountManager para processar o pedido. Em caso de sucesso devolve o conteúdo do ficheiro e o necessário para o decifrar.

PUT /ficheiros/{ficheiro_id}/modificar
Este endpoint permite a modificação de um ficheiro existente. Recebe um objeto de dados que deve incluir conteudo_cifrado, iv, tag e recurso_id. O sistema valida se todos os parâmetros necessários estão presentes. Utiliza o AccountManager para processar a modificação, o resultado da operação é registado através do Logger e é devolvida uma resposta indicando o sucesso ou falha da operação.

GET /ficheiros/{ficheiro_id}/pedido-adicionar
Este endpoint inicia o processo de adição de conteúdo a um ficheiro existente, identificado pelo ficheiro_id. O sistema extrai o email do utilizador através do token de acesso e utiliza o AccountManager para processar o pedido. Em caso de sucesso devolve o conteúdo do ficheiro e o necessário para o decifrar.

PUT /ficheiros/{ficheiro_id}/adicionar
Este endpoint de *append* permite adicionar conteúdo a um ficheiro existente. Recebe um objeto de dados que deve incluir conteudo_cifrado, iv, tag e recurso_id. O sistema valida se todos os parâmetros necessários estão presentes. Utiliza o AccountManager para processar a adição de conteúdo. O resultado da operação é registado através do Logger e é devolvida uma resposta indicando o sucesso ou falha da operação.

GET /auth/google/login
Este endpoint gera um URL de autorização para o Google OAuth 2.0. O URL inclui os parâmetros client_id, response_type definido como "code", o redirect_uri, os escopos solicitados (openid, email e profile), o access_type definido como "offline" e o prompt definido como "consent". O endpoint retorna um objeto JSON contendo o URL de autorização no campo "auth_url".

GET /auth/google/callback
Este endpoint recebe um parâmetro opcional "code" via Query String. Retorna uma página HTML com um design que exibe o código de autorização recebido do Google. A página contém estilos CSS incorporados, um layout centrado e um botão para copiar o código para a área de transferência do utilizador. Se nenhum código for encontrado na URL, exibe uma mensagem de erro.

POST /auth/google/token
Este endpoint recebe um código de autorização do Google como parâmetro e faz uma solicitação HTTP POST para o endpoint token do Google OAuth 2.0. Após receber o token de acesso, faz uma solicitação para obter as informações do utilizador usando o endpoint "userinfo" do Google e verifica se o email do utilizador existe na base de dados. Se o utilizador existir, gera tokens de acesso e de atualização. Retorna um objeto JSON contendo o access_token, o refresh_token e o token_type definido como "bearer". Em caso de erro em qualquer etapa, lança uma exceção HTTP com o código de status apropriado.

5 Classes de Abstração

Como já foi referido, foram desenvolvidas diversas classes na API com o objetivo de tornar o código mais legível, modular e, sobretudo, mais fácil de manter. Abaixo segue uma descrição mais detalhada das responsabilidades atribuídas a cada uma delas.

5.1 AccountManager

Tal como o nome indica, esta classe é responsável pela gestão das contas dos utilizadores. A sua principal função consiste na verificação de credenciais durante o processo de login, tarefa para a qual recorre ao DBManager, que assegura o acesso à base de dados. Para além disso, o AccountManager é também responsavel por toda a gestão de contas incluíndo o registo das tentativas de autenticação, contribuindo assim para a proteção contra ataques de força bruta.

Este objeto, em conjunto com o ficheiro auth.py, é responsável por toda a autenticação de utilizadores. Este ficheiro trata da geração de tokens de acesso e de refresh, da renovação de tokens, bem como do armazenamento em Redis para efeitos de blacklist. Importa salientar que, para colocar o sistema em produção, será necessário atualizar o ficheiro de configuração do Redis, de forma a permitir a comunicação exclusivamente com o servidor e a ativar a comunicação cifrada.

Relativamente a ataques de negação de serviço (DoS), a intenção inicial passava por implementar mecanismos de mitigação diretamente nesta classe. No entanto, a própria API conta com um rate limiter que simplifica significativamente essa tarefa, limitando o número de pedidos a seis por minuto por utilizador, o que já oferece uma camada adicional de segurança.

5.2 DBManager

Tal como o nome sugere, esta classe é responsável pela gestão de todas as interações com a base de dados, atuando de forma semelhante a um ORM, embora com consultas SQL escritas manualmente no código, utilizando a biblioteca *psycopg2*.

O DBManager centraliza todos os acessos à base de dados, desde operações simples, como obter o ID de um utilizador a partir do seu email, até consultas mais complexas, como, por exemplo, listar todos os ficheiros e pastas aos quais um determinado utilizador tem acesso.

A utilização do *psycopg2* é adequada, uma vez que esta biblioteca é thread-safe, o que, combinado com as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) da base de dados PostgreSQL, assegura robustez e fiabilidade nas transações.

Importa ainda destacar que, por razões de segurança, as palavras-passe dos utilizadores não são armazenadas em texto simples. Antes de serem inseridas na base de dados, são cifradas com recurso à biblioteca *bcrypt*. Esta operação de cifragem é realizada previamente pelo AccountManager, garantindo que apenas o valor cifrado chega à base de dados.

5.3 IntegrityManager

O IntegrityManager é um componente responsável por verificar e garantir a integridade dos ficheiros armazenados no sistema. Esta classe implementa um sistema de verificação periódica que examina regularmente todos os ficheiros armazenados na base de dados, para detetar qualquer possível corrupção ou alteração não autorizada.

A classe funciona estabelecendo uma comunicação com a base de dados através do DBManager e utiliza um mecanismo de temporizador para executar verificações recorrentes de forma automática. A cada 60 segundos, o sistema recupera a lista completa de identificadores de ficheiros existentes e realiza uma verificação de integridade em cada um deles. Quando um ficheiro corrompido é detetado, o sistema regista esta informação e emite um alerta.

A implementação recorre ao módulo threading do Python para criar um processo de verificação que se executa em segundo plano, sem interferir com outras operações do sistema, mantendo assim uma vigilância constante sobre a integridade dos dados sem comprometer o desempenho geral da aplicação.

5.4 PermissionManager

Tal como o nome indica, esta classe é responsável pela gestão das permissões de acesso a recursos no sistema. Atua como intermediária entre a lógica da aplicação e o DBManager, delegando-lhe operações de leitura e escrita na base de dados relacionadas com permissões.

O PermissionManager permite verificar se um utilizador tem acesso a uma determinada pasta ou ficheiro, validar se é o proprietário de um recurso antes de permitir alterações, criar novas permissões para outros utilizadores, atualizar níveis de acesso, e remover permissões existentes. Esta abstração garante uma gestão de permissões centralizada, consistente e facilmente extensível.

5.5 Logger

A classe Logger é responsável pela gestão dos registos de atividade da aplicação. Todas as ações realizadas pelos utilizadores são registadas neste sistema de logging, sendo os registos armazenados de forma cifrada para garantir a sua confidencialidade.

Em caso de pedido por parte de um administrador autorizado, o Logger trata da descodificação dos registos e procede ao seu envio de forma segura. Este mecanismo garante tanto a rastreabilidade das ações como a integridade e segurança da informação registada.

Contudo, reconhecemos algumas limitações desta abordagem, nomeadamente ao nível da segurança e, sobretudo, da escalabilidade. Sendo o ficheiro de registo apenas de escrita contínua (append-only), consideramo-lo uma solução segura. No entanto, estamos cientes da falta de atenção à questão da escalabilidade: idealmente, os registos deveriam ser repartidos por múltiplos ficheiros (por exemplo um por dia) para facilitar a pesquisa e análise dos logs.

6 Requisitos Cumpridos

6.1 Requisitos Funcionais

RF	Categoria	Descrição
RF1	Gestão de Utilizadores	O sistema deve permitir o registo de novos utilizadores
RF2	Gestão de Utilizadores	Cada utilizador deve ser identificado por um ID único e um endereço de email único
RF3	Gestão de Utilizadores	O sistema deve armazenar as credenciais de acesso definidas durante o registo
RF4	Gestão de Utilizadores	O sistema deve permitir a autenticação de utilizadores existentes
RF5	Gestão de Dados	O sistema deve permitir o armazenamento de dois tipos de recursos: ficheiros e pastas
RF6	Gestão de Dados	O sistema deve permitir a adição de ficheiros no cofre digital
RF7	Gestão de Dados	O sistema deve permitir a criação de pastas, utilizadas para armazenar e organizar recursos
RF8	Gestão de Dados	Cada recurso deve ter um proprietário com controlo total sobre ele
RF9	Gestão de Dados	O sistema deve permitir a leitura de ficheiros a utilizadores com permissão
RF10	Gestão de Dados	O sistema deve permitir a modificação de ficheiros a utilizadores com permissão

RF11	Gestão de Dados	O sistema deve permitir listar o conteúdo de pastas a utilizadores com acesso
RF12	Gestão de Dados	O sistema deve permitir adicionar recursos a pastas, com permissões adequadas
RF13	Controlo de Acesso	O sistema deve implementar três níveis de permissões: leitura, extensão e escrita
RF14	Controlo de Acesso	O proprietário pode atribuir permissões de acesso a outros utilizadores
RF15	Controlo de Acesso	O proprietário pode modificar as permissões de outros utilizadores
RF16	Controlo de Acesso	O proprietário pode remover permissões de outros utilizadores
RF17	Interface de Utilizador	O sistema deve fornecer uma aplicação cliente (CLI) para interação com o servidor
RF18	Interface de Utilizador	A CLI deve suportar comandos para todas as operações disponíveis
RF19	Interface de Utilizador	O sistema deve fornecer feedback claro após operações, com mensagens apropriadas

6.2 Requisitos de Segurança

RS	Categoria	Descrição
RS1	Autenticação	O sistema deve verificar as credenciais de autenticação do utilizador
RS2	Autenticação	As credenciais devem ser transmitidas de forma segura
RS3	Autenticação	Mecanismos contra força bruta e acessos suspeitos devem ser implementados
RS4	Proteção de Dados	O conteúdo dos ficheiros deve ser cifrado no cliente
RS5	Proteção de Dados	O servidor não deve ter acesso ao conteúdo dos ficheiros
RS6	Proteção de Dados	O servidor deve verificar a integridade dos dados armazenados
RS7	Controlo de Acesso	O servidor deve aplicar a política de controlo de acesso a cada recurso
RS8	Controlo de Acesso	O sistema deve validar permissões antes de qualquer operação
RS9	Controlo de Acesso	Mecanismos de deteção de modificações não autorizadas devem ser implementados
RS10	Auditoria	O sistema deve registar todas as operações realizadas sobre recursos
RS11	Auditoria	Os registos devem incluir utilizador, timestamp, operação e recurso (quando aplicável)
RS12	Auditoria	Os registos de auditoria devem ser protegidos contra modificações não autorizadas
RS14	Gestão de Chaves	O sistema deve gerir chaves criptográficas de forma segura
RS15	Gestão de Chaves	O sistema deve permitir partilha segura de chaves entre utilizadores autorizados
RS16	Gestão de Chaves	Deve ser possível rodar chaves para mitigar comprometimentos
RS17	Comunicação Segura	A comunicação entre cliente e servidor deve ser cifrada e íntegra

Os requisitos aqui apresentados correspondem integralmente aos requisitos definidos na fase inicial do projeto, o que reflete uma taxa de cumprimento de 100%. Todos os objetivos funcionais e de segurança

foram implementados com sucesso, demonstrando não só a viabilidade técnica da solução proposta, como também o rigor e o cuidado aplicados ao longo do desenvolvimento.

7 Análise do Software Bill of Materials (SBOM)

7.1 Visão geral da stack tecnológica

A stack tecnológica principal é composta por:

- **Framework Web:** FastAPI (0.115.12), uma framework assíncrona de alto desempenho para desenvolvimento rápido de APIs
- **Servidor ASGI:** Uvicorn (0.34.2), implementação ASGI que permite a execução eficiente de aplicações web assíncronas
- **Validação de Dados:** Pydantic (2.11.4), biblioteca para validação de dados, serialização e documentação automática
- **Base de Dados:** Suporte a PostgreSQL através da *driver* psycopg2-binary (2.9.10)
- **Cache:** Redis (6.0.0), como solução de armazenamento de dados em memória
- **Segurança:** Implementações de criptografia robusta com cryptography (44.0.3), bcrypt (4.3.0) e python-jose (3.4.0)

Esta análise da stack demonstra uma escolha moderna para o desenvolvimento de aplicações web baseadas em Python, com foco em desempenho e segurança, justificando positivamente as nossas escolhas de ferramentas e bibliotecas

7.2 Componentes críticos

Nesta fase da análise são apresentados os componentes e as suas funções na nossa implementação.

7.2.1 Componentes Core da Aplicação

O FastAPI (versão 0.115.12) constitui o coração da aplicação, sendo responsável pelo roteamento das requisições, validação de dados, documentação automática da API e gestão de dependências. A sua arquitetura assíncrona permite lidar alta concorrência.
A base sobre a qual o FastAPI opera é o Starlette (versão 0.46.2), uma framework ASGI de baixo nível que gere eficientemente requisições HTTP e WebSockets.
Para garantir a integridade e consistência dos dados que entram e saem da aplicação, é utilizado o Pydantic (versão 2.11.4). Este componente fornece validação de dados, bem como serialização e desserialização, além de permitir a geração automática de esquemas JSON.

7.2.2 Componentes de Segurança

O Python-jose (3.4.0) implementa JWT (JSON Web Tokens) para autenticação e autorização, sendo essencial para proteger os endpoints da API. A biblioteca Cryptography (44.0.3) fornece primitivas criptográficas para operações seguras. O Argon2-cffi (23.1.0) é uma implementação do algoritmo Argon2 para chaves, considerado das melhores opções em segurança.

7.2.3 Componentes de Persistência e Estado

Temos o Psycopg2-binary (2.9.10), um driver PostgreSQL otimizado para Python, que gere conexões com a base de dados relacional. Utilizámos também o Redis (6.0.0) para armazenamento chave-valor em memória.

7.2.4 Componentes de Infraestrutura

Foi utilizado o Uvicorn (0.34.2), sendo este um servidor ASGI de alto desempenho responsável pela execução da aplicação. Por sua vez, o Python-dotenv (1.1.0) gere o funcionamento com variáveis de ambiente, sendo crítico para a configuração segura da aplicação..

7.3 Análise de licenças e compliance

A análise do SBOM revela um panorama de licenciamento predominantemente permissivo. Nesta fase iremos apresentar a distribuição das licenças:

7.3.1 Distribuição de Licenças

- **MIT:** 19 pacotes (31,7%)
- **Apache-2.0:** 8 pacotes (13,3%)
- **BSD (todas as variantes):** 7 pacotes (11,7%)
- **Não especificadas (N/A):** 22 pacotes (36,7%)
- **Outras:** 4 pacotes (6,7%)

7.3.2 Considerações de Compliance

- **Licenças Permissivas:** A maioria das bibliotecas usa licenças MIT, Apache 2.0 e BSD, que são permissivas e geralmente compatíveis entre si, oferecendo baixo risco legal para uso comercial.
- **Licenças Não Especificadas:** Uma parcela significativa dos pacotes (36,7%) não tem licença especificada no SBOM.
- **Licenças Duais:** Alguns pacotes como cryptography (Apache-2.0 OR BSD-3-Clause) e sniffio (MIT OR Apache-2.0) oferecem escolha entre duas licenças, fornecendo flexibilidade legal.

7.4 Postura de segurança e gestão de dependências

7.4.1 Análise de Segurança

- **Vulnerabilidades:** Nenhuma vulnerabilidade foi reportada no SBOM, o que é um indicador positivo para a postura de segurança da aplicação.
- **Bibliotecas Criptográficas:** As bibliotecas relacionadas à segurança (cryptography, bcrypt, python-jose, argon2-cffi) estão em versões recentes.
- **Bibliotecas Core:** Componentes críticos como FastAPI, Pydantic e Starlette estão em versões recentes, o que é ideal para acompanhar as atualizações de segurança.

7.4.2 Gestão de Dependências

- **Complexidade:** O projeto inclui aproximadamente 60 dependências diretas, resultando numa árvore de dependências moderadamente complexa.
- **Dependências Transitivas:** Várias bibliotecas possuem cadeias de dependências longas, como o caso do FastAPI → Starlette → AnyIO → SNIffio, o que aumenta a superfície de ataque potencial.
- **Versões:** O SBOM mostra versões específicas para cada dependência, o que facilita a reprodutibilidade do ambiente e o rastreamento de atualizações necessárias.
- **Dependências Críticas:** Alguns componentes (como typing_extensions) são amplamente utilizados no ecossistema, tornando-os pontos críticos para monitoramento contínuo.

8 Conclusão

Neste trabalho abordámos o desenvolvimento de uma aplicação como foco na sua seguraça para gestão e partilha de ficheiros, composta por uma API RESTful desenvolvida em FastAPI e por um cliente em linha de comandos implementado com Typer. Explorámos múltiplas vertentes técnicas, desde a autenticação com tokens JWT e 2FA, ao armazenamento seguro de dados e tokens, até à implementação de permissões granulares de acesso a recursos.

Cumprimos todos os objetivos a que nos propusemos, tendo implementado todos os requisitos funcionais e de segurança definidos na proposta inicial. Para além da criação de um sistema funcional e seguro, desenvolvemos uma aplicação modular, escalável e facilmente extensível, recorrendo a boas práticas de abstração com classes bem definidas, como o AccountManager, DBManager, PermissionManager e Logger, que contribuem para uma arquitetura clara e de fácil manutenção. Adicionalmente, foi assegurada a segurança das comunicações através de HTTPS, com especial cuidado no tratamento dos tokens e na proteção dos dados dos utilizadores, incluindo a implementação de um servidor com zero knowledge e criptografia de ponta a ponta.

Ainda assim, identificámos oportunidades de melhoria futura, nomeadamente ao nível do sistema de registos de *logs*, cuja evolução poderá reforçar a segurança e a escalabilidade, assim como a eficiência da análise.

Este trabalho foi particularmente importante para consolidarmos conhecimentos em áreas fundamentais do desenvolvimento de software moderno, como segurança, autenticação, criptografia, boas práticas de desenvolvimento backend e integração de ferramentas externas. Para além disso, permitiu-nos aplicar conceitos teóricos em cenários práticos, reforçando a importância da organização do código, da validação rigorosa dos dados e da preocupação constante com a experiência e segurança do utilizador.