Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

Segurança em Engenharia de Software (CC4078)

Semester Project

Secure Learning Platform

Project participants:

António Pedro Pinheiro (201704931)
Manuel Veiga (201905924)
Simão Ribeiro (202309188)

Teacher: Hugo Pacheco & Rui Maranhão

Contents

1 Introdução

Atualmente, na era em que vivemos, sistemas com *software* tornaram-se extremamente importantes nas nossas vidas. Dependemos intensivamente destes sistemas em áreas do nosso quotidiano, como serviços financeiros, telecomunicações, eletrónica, transportação, eletrodomésticos, e muito mais. [6] Uma vez que os sistemas de *software* estão envolvidos em vários aspetos da sociedade, a segurança torna-se uma questão importante e um requisito vital para os mesmos. [8,9] Alguns tópicos generalizados de segurança, como a confidencialidade, a disponibilidade e a integridade, têm de ser preservados para que o *software* seja considerado seguro. [3]

Tradicionalmente, a segurança do software é considerada apenas nas fases posteriores do desenvolvimento do software, incorporando as preocupações de segurança como uma reflexão posterior. [2] Consequentemente, aumenta o risco de introdução de novas vulnerabilidades de segurança em várias fases do ciclo de desenvolvimento do *software*. Seguir o método tradicional de proteção do *software* levou à abordagem "Penetrate and Patch", em que o especialista em segurança tenta avaliar o *software*, separando-o do seu ambiente através da exploração de vulnerabilidades de segurança comuns. Uma penetração bem sucedida leva ao desenvolvimento de correções e à implementação das vulnerabilidades identificadas. A segurança é geralmente tratada como uma caraterística adicional no ciclo do desenvolvimento de *software* e é abordada por profissionais de segurança que utilizam antivírus, segurança de plataformas, *proxies*, *firewalls* e sistemas de prevenção de intrusões. [1, 5]

Os mecanismos de defesa que são acrescentados a um sistema de *software* no final do ciclo de desenvolvimento, como os sistemas de deteção de intrusões e as *firewalls*, muitas vezes não são suficientes e podem levar a retrabalhos dispendiosos. [8] A investigação demonstrou igualmente que estas abordagens suplementares para dar resposta às preocupações relacionadas com a segurança não são suficientes e podem conduzir a um número significativo de alterações, para além de quaisquer consequências intangíveis causadas por uma violação da segurança. [1] Para fazer face a estes retrabalhos e alterações, os desafios de segurança devem ser abordados desde o início do ciclo do desenvolvimento de *software* (ou seja, desde a recolha dos requisitos de *software* até à manutenção do mesmo). [1] Para tal, a engenharia de *software* seguro tornou-se recentemente uma área de investigação muito ativa. Na engenharia de *software*, são utilizadas diferentes fases para desenvolver software. As principais fases da engenharia de *software* são: requisitos de *software*, conceção de *software*, codificação, testes, integração e manutenção. [6] Estas fases estão representadas na Figura 1.

O objetivo da engenharia de *software* seguro consiste em atacar as vulnerabilidades de segurança do *software*, tendo em conta as preocupações de segurança e as abordagens de desenvolvimento no ciclo de vida do *software*. Como tal, a engenharia de *software* seguro é um procedimento que visa atingir objectivos de segurança através da construção, conceção e teste do *software*. A segurança do *software* é diferente da segurança das aplicações, na medida em que a segurança das aplicações consiste em proteger o *software* após o desenvolvimento e a implementação. Inclui, normalmente, vários mecanismos de proteção, como *firewalls*, antivírus e sistemas de deteção de intrusões. [2, 3]

Desta forma, as abordagens de segurança tradicionais resultam, frequentemente, em medidas reativas, deixando as aplicações vulneráveis a ataques. O *DevSecOps* surge como uma solução proativa, integrando a segurança no *pipeline* de *DevOps* desde o início do desenvolvimento. [10]

Ao praticar o príncipio de "shift left" das práticas de segurança, as organizações têm como objetivo abordar as questões de segurança o mais cedo possível, idealmente durante as fases de conceção e desenvolvimento. Esta abordagem proativa ajuda a identificar e mitigar os riscos de

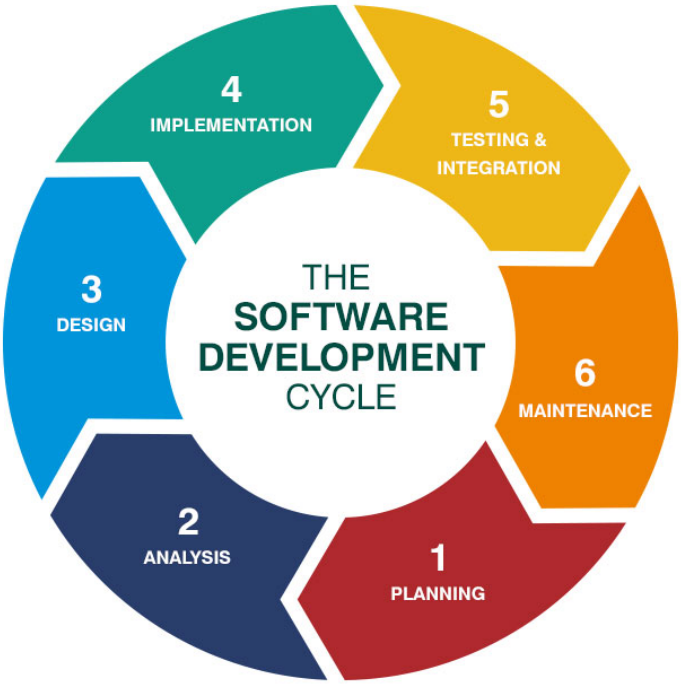


Figure 1: Fases do Desenvolvimento de *Software*

segurança antes que estes aumentem, resultando em implementações de software mais seguras e resilientes. Os principais componentes do *pipeline DevSecOps* incluem a análise de código estático, os testes dinâmicos de segurança de aplicações (DAST), a análise de segurança de contentores, a avaliação de vulnerabilidades, a validação da conformidade e a monitorização da segurança. Estes componentes trabalham em conjunto para identificar, corrigir e prevenir vulnerabilidades de segurança e violações de conformidade em todo o processo de desenvolvimento de *software*. [4]

O presente relatório está organizado da seguinte forma: na secção 1, fazemos uma breve introdução sobre o papel da segurança no desenvolvimento de *software*, os príncipios teórios e o que significa o príncipio de segurança "shift left". Na secção 2, fazemos um levantamento dos requisitos de segurança e funcionais da aplicação a desenvolver, enquanto que na secção 3 procedemos à especificação das características de *design* da aplicação, nomeadamente da sua arquitetura, atores, casos de uso e mau uso, bem como de possíveis ameaças à segurança. Na secção 4, fazemos uma descrição da implementação prática da aplicação e, na secção 5, uma análise abrangente à sua segurança. Por fim, na secção 6, procedemos à conclusão.

2 Requisitos de Segurança/Funcionais

Em termos de requisitos funcionais, dividimos estes em três categorias, relativas à interface, autenticação e controlo de acesso, de acordo com o requisitado para este projeto. [7]

2.1 Interface

- Deve ser possível criar, configurar, editar e apagar cursos;

- Cada curso deve incluir:
  - Uma lista de tópicos;
  - Cada tópico deve possuir:
    - Um estado, que pode estar em rascunho (apenas vísivel para o dono) ou publicado;
    - Os tópicos publicados podem ser visíveis para todos os utilizadores ou para apenas alguns;
  - Um forúm onde os utilizadores registados podem discutir e fazer questões sobre os tópicos:
    - Deve ser possível fazer upload de ficheiros ou imagens;
- Deve haver uma página pessoal para cada estudante com os cursos em que está registado.

2.2 Autenticação

- Utilizadores que pretendam não ficar anónimos têm de se autenticar perante a aplicação;
- Implementar autenticação baseada em políticas de palavras-passe fortes (tamanho mínimo, complexidade, etc.);

2.3 Controlos de Acesso

- Apenas utilizadores com permissões especiais podem criar ou apagar cursos;
- Um curso tem um ou mais administradores que podem gerir o seu conteúdo e o registo de estudantes;
- Os cursos podem ser públicos ou privados, apenas estando acessíveis aos estudantes registados nos mesmos.
- Garantir a gestão adequada da sessão de forma a prevenir acesso não autorizado, através de:
  - Identificadores de Sessão Seguros (aleatórios, longos e únicos);
  - Expiração da Validade da Sessão e Mecanismos de Logout.

2.4 Proteção contra Injeção

- Proteger o site relativamente a ataques através de injeção de qualquer tipo de código malicioso (SQL, XSS, ...);
- Implementar validação dos inputs e sanitização a toda a informação, removendo ou escapando qualquer potencial caractére malicioso antes de processar a informação;
- Garantir a parametrização de queries à base de dados;
- Parametrizar devidamente os pedidos HTTP, nomeadamente os cabeçalhos de modo a restringir modificações não desejáveis.

- Garantir upload de ficheiros e imagens seguro, incluindo:
  - Limitar tipos de fichieiros e o seu tamanho;
  - Analisar os ficheiros para procurar malware;
  - Armazenar os ficheiros devidamente com controlo de acesso restrito.

2.5 Encriptação dos Dados

- Utilizar armazenamento seguro para as credenciais dos utilizadores, como fazer as hashes das palavras-passe com algoritmos com elevada segurança.
- Encriptar dados sensíveis do utilizador, como "personal identifiable information" (PII), quer na base de dados, quer em trânsito, entre cliente e servidor.

3 Design do Website

Nesta secção, será abordado o *design* do *website*, incluindo a arquitetura do sistema, os atores envolvidos e os casos de uso/mau uso, bem como as potenciais ameaças à segurança. Esta análise permitirá uma compreensão detalhada de como o sistema foi estruturado para atender aos requisitos funcionais e de segurança definidos.

3.1 Arquitetura do Sistema

A arquitetura do sistema de gestão de cursos foi projetada para ser modular e escalável, assegurando que cada componente possa ser desenvolvido, testado e mantido de forma independente. A arquitetura é baseada numa separação clara entre o *frontend*, *backend* e a base de dados, com comunicação entre os componentes através de *endpoints* REST.

3.1.1 Componentes Principais

- ***Frontend*:** Implementado em React com *Bootstrap*, responsável por todas as interações com o utilizador. O *frontend* envia requisições HTTP para o *backend* e processa as respostas para exibir os dados de forma intuitiva e organizada.
- ***Backend*:** Desenvolvido com FastAPI, gere a lógica da plataforma, autenticação, autorização e manipulação de dados. O *backend* expõe uma API RESTful que é consumida pelo *frontend*.
- **Base de Dados:** Utiliza SQLAlchemy como ORM para interagir com a base de dados SQLite. Armazena todas as informações sobre utilizadores, cursos, tópicos e postagens no fórum.
- **Pasta de *Upload*:** Gere os ficheiros carregados pelos utilizadores, armazenando-os de forma organizada e permitindo o acesso através de URLs geradas.

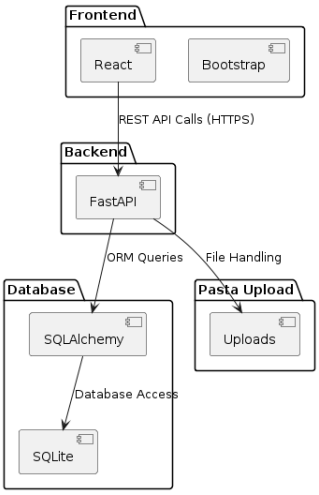


Figure 2: Diagrama de Componentes

3.1.2 Diagrama de Classes

A Figura 3 apesenta o diagrama de classes representativo da implementação feita no âmbito deste projeto.

1. User A classe `User` representa um utilizador da aplicação. Os principais atributos incluem:

- `id`: identificador único do utilizador.
- `username`: nome de utilizador.
- `email`: endereço de email do utilizador.
- `hashed_password`: palavra-passe encriptada do utilizador.
- `role`: papel do utilizador na aplicação, que pode ser "admin", "teacher" ou "student".

2. Course A classe `Course` representa um curso na plataforma. Os principais atributos incluem:

- `id`: Identificador único do curso.
- `title`: Título do curso.
- `description`: Descrição do curso.
- `owner_id`: Identificador do dono do curso.
- `owners`: Lista de utilizadores que são donos do curso.
- `participants`: Lista de utilizadores que participam do curso.

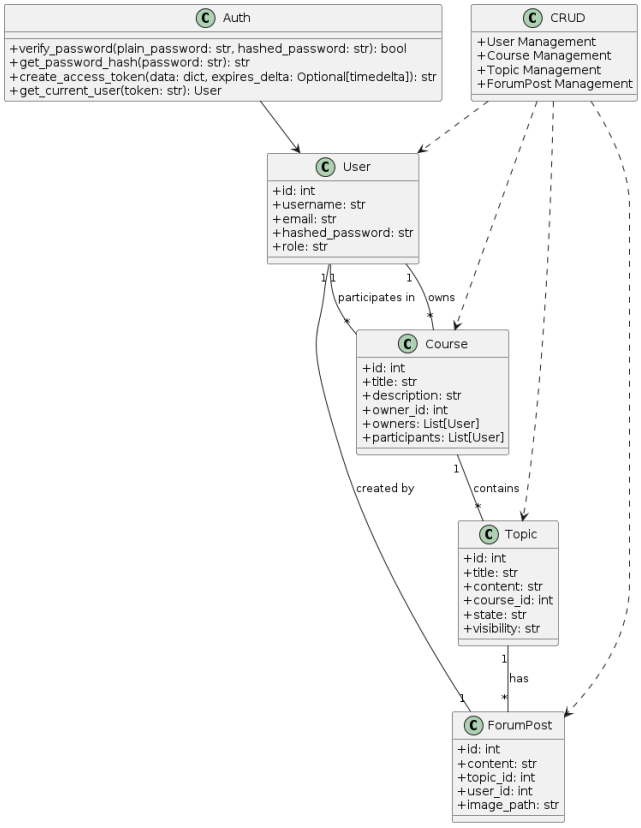


Figure 3: Diagrama de Classes

3. Topic A classe `Topic` representa um tópico dentro de um curso. Os principais atributos incluem:

- `id`: Identificador único do tópico.
- `title`: Título do tópico.
- `content`: Conteúdo do tópico.
- `course_id`: Identificador do curso ao qual o tópico pertence.
- `state`: Estado do tópico (ex. "draft", "published").
- `visibility`: Visibilidade do tópico (ex. "public", "private").

4. ForumPost A classe `ForumPost` representa uma postagem no fórum de um tópico. Os principais atributos incluem:

- `id`: identificador único da postagem.
- `content`: conteúdo da postagem.
- `topic_id`: identificador do tópico ao qual a postagem pertence.
- `user_id`: identificador do utilizador que criou a postagem.
- `image_path`: caminho para a imagem anexada à postagem.

5. Auth A classe `Auth` trata da autenticação e autorização dos utilizadores. Os principais métodos incluem:

- `verify_password`: verifica se a palavra-passe corresponde ao hash armazenado.
- `get_password_hash`: gera um hash para a palavra-passe fornecida.
- `create_access_token`: cria um token de acesso JWT.
- `get_current_user`: obtém o utilizador atual com base no token JWT.

6. CRUD A classe `CRUD` é responsável pelas operações de criação, leitura, atualização e eliminação (Create, Read, Update, Delete) de várias entidades. Está dividida em quatro categorias principais:

- **User Management**: Métodos para gerir utilizadores, como `get_user`, `create_user`, `update_user` e `delete_user`.
- **Course Management**: Métodos para gerir cursos, como `get_course`, `create_course`, `update_course` e `delete_course`.
- **Topic Management**: Métodos para gerir tópicos, como `get_topic`, `create_topic`, `update_topic` e `delete_topic`.
- **ForumPost Management**: Métodos para gerir postagens no fórum, como `get_forum_posts`, `create_forum_post` e `delete_forum_post`.

Relações Entre as Classes

- **Associação**:
  - Um `User` pode ser dono de muitos `Course` (`owns`).
  - Um `User` pode participar em muitos `Course` (`participates in`).
  - Um `Course` contém muitos `Topic`.
  - Um `Topic` tem muitas `ForumPost`.
  - Um `ForumPost` é criado por um `User`.
- **Dependência**:
  - A classe `CRUD` depende das classes `User`, `Course`, `Topic` e `ForumPost` para realizar operações de gestão.
  - A classe `Auth` depende da classe `User` para realizar operações de autenticação e autorização.

3.2 Atores e Casos de Uso/Mau Uso

Esta secção descreve os atores que interagem com o sistema, os casos de uso principais que definem as interações funcionais esperadas, e os casos de mau uso que representam potenciais tentativas de exploração ou utilização inadequada do sistema.

Atores Principais:

- **Administrador:** Responsável pela criação e gestão de utilizadores e cursos. Possui permissões para realizar todas as operações no sistema.
- **Professor:** Pode criar e gerir cursos, tópicos e fóruns. Adiciona e remove participantes nos seus cursos.
- **Estudante**: Participa nos cursos, visualiza tópicos e postagens no fórum, e pode enviar perguntas e respostas nos fóruns.

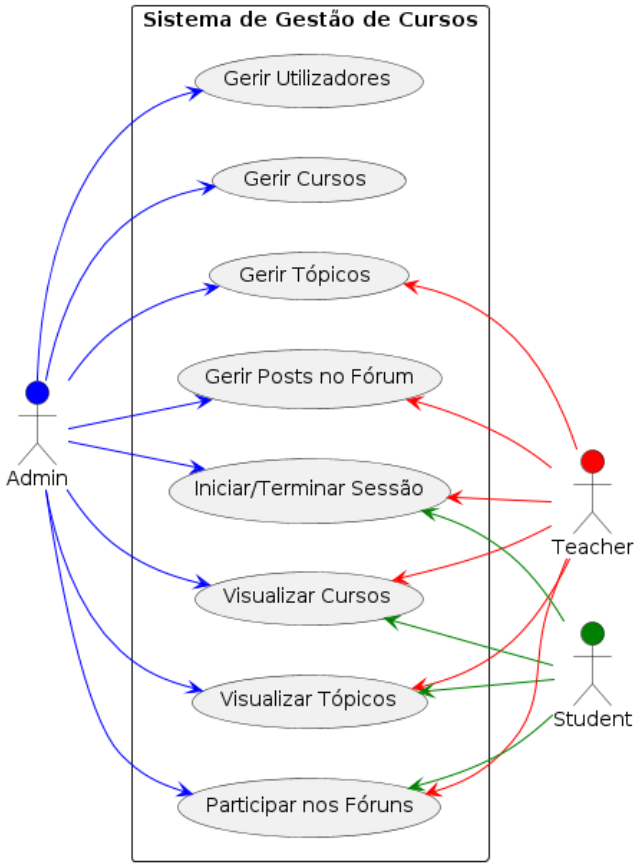


Figure 4: Diagrama de Casos de Uso

Relativamente a casos de uso da aplicação, fica aqui uma listagem abrangente dos mesmos:

- <u>Casos de uso do administrador:</u>
  - **Criar curso:** Um administrador cria um novo curso.
  - **Eliminar curso:** Um administrador elimina um curso existente.
- <u>Casos de uso do proprietário do curso:</u>
  - **Configurar curso:** O proprietário de um curso configura os detalhes do curso (por exemplo, título, descrição).
  - **Editar curso:** O proprietário de um curso edita o conteúdo e os tópicos do curso.

  - **Eliminar curso:** O proprietário de uma disciplina elimina uma disciplina de que é proprietário.
  - **Inscrever alunos:** O proprietário de um curso inscreve alunos no curso.
  - **Configurar fórum:** O proprietário de um curso define as configurações do fórum, incluindo níveis de acesso e permissões.

- **Casos de uso de alunos matriculados:**
  - **Visualizar curso:** Um aluno inscrito visualiza o curso e seus tópicos.
  - **Participar do fórum:** Um aluno inscrito faz perguntas e discute tópicos no fórum.
  - **Carregar ficheiros:** Um aluno inscrito carrega ficheiros para o fórum.

Por fim, no que diz respeito a casos de mau uso, ou seja, uso abusivo ou malicioso, fica aqui uma lista de alguns casos:

- **Casos de Uso Indevido do Administrador:**
  - **Exclusão Não Autorizada de Curso:** Um atacante obtém privilégios de administrador e exclui cursos maliciosamente.
  - **Escalonamento de Privilégios:** Um atacante explora uma vulnerabilidade para escalar privilégios ao nível de administrador.

- **Casos de Uso Indevido do Proprietário do Curso:**
  - **Modificação de Conteúdo Não Autorizada:** Um atacante se passa por um proprietário do curso e modifica ou exclui o conteúdo do curso.
  - **Gestão Indevida de Matrículas:** Um atacante adiciona ou remove alunos dos cursos sem autorização.

- **Casos de Uso Indevido do Aluno Matriculado:**
  - **Acesso Não Autorizado a Tópicos Privados:** Um aluno matriculado acessa tópicos que não está autorizado a visualizar.
  - **Abuso no Fórum:** Um aluno matriculado publica conteúdo inapropriado ou malicioso no fórum.

- **Casos de Uso Indevido do Sistema:**
  - **Injeção de SQL:** Um atacante explora uma vulnerabilidade no sistema para executar comandos SQL não autorizados.
  - **Script de Sites Cruzados (XSS):** Um atacante injeta scripts maliciosos em páginas web visualizadas por outros utilizadores.
  - **Vazamento de Dados:** Um atacante obtém acesso não autorizado a dados sensíveis dos utilizadores por meio de uma vulnerabilidade.
  - **Negação de Serviço (DoS):** Um atacante sobrecarrega o sistema com solicitações, tornando-o indisponível para utilizadores legítimos.

3.3 Ameaças à Segurança

- **Ataques de Injeção:**
  - **Injeção de SQL:** Um atacante executa código SQL arbitrário na base de dados, explorando vulnerabilidades em *inputs*.
  - **Injeção de Comandos:** Um atacante injeta e executa comandos do sistema arbitrários.
- **Cross-Site Scripting (XSS):**
  - **XSS Armazenado:** Scripts maliciosos são armazenados no servidor (por exemplo, em postagens de fórum) e executados no navegador de qualquer utilizador que visualize o conteúdo afetado.
  - **XSS Refletido:** Scripts maliciosos são refletidos de um servidor *web* e executados imediatamente, geralmente através de um URL criado.
- **Cross-Site Request Forgery (CSRF):**
  - Um atacante engana um utilizador para realizar ações que não pretendia (por exemplo, enviar um formulário ou fazer uma compra) explorando a sessão autenticada do utilizador.
- **Autenticação e Gestão de Sessão não funcionais:**
  - ***Hijacking* de Sessão:** Um atacante assume o controlo de uma sessão de utilizador, roubando *tokens* de sessão.
  - **Preenchimento de Credenciais:** Atacantes usam combinações de nome de utilizador/senha roubadas de outros vazamentos para obter acesso não autorizado.
- **Controlo de Acesso Indevido:**
  - **Escalonamento de Privilégios:** Um atacante obtém acesso de nível superior ao pretendido (por exemplo, um aluno obtém privilégios de administrador).
  - **Acesso Não Autorizado:** Um atacante consegue o acesso a recursos que não está autorizado (por exemplo, visualizar conteúdo privado do curso).
- **Configuração de Segurança Incorreta:**
  - **Credenciais Padrão:** Uso de nomes de utilizador e senhas padrão para contas de administrador.
  - **Funcionalidades Desnecessárias:** Habilitação de funcionalidades ou serviços desnecessários que podem ser explorados.
  - **Tratamento Inadequado de Erros:** Mensagens de erro detalhadas que fornecem informações úteis para atacantes.
- **Exposição de Dados Sensíveis:**
  - **Armazenamento Não Encriptado:** Dados sensíveis armazenados em texto simples.
  - **Transmissão Não Segura:** Dados sensíveis transmitidos sem criptografia, por exemplo, via HTTP em vez de HTTPS.

  - **Uploads de Ficheiros Inseguros:** Armazenamento de arquivos enviados de maneira insegura.

- **Referências Diretas Inseguras a Objetos:**
  - Um atacante obtém acesso a dados não autorizados manipulando referências a objetos (por exemplo, alterando um ID de curso na URL para aceder a outro curso).

- **Negação de Serviço (DoS) e Negação de Serviço Distribuída (DDoS):**
  - Sobrecarregar a aplicação com pedidos excessivos, tornando-a indisponível para utilizadores legítimos.

- **Ataques Man-in-the-Middle (MitM):**
  - Interceptação e alteração de comunicações entre o cliente e o servidor sem o conhecimento de ambas as partes.

- **Registo e Monitorização Insuficientes:**
  - **Falta de Logs de Auditoria:** Nenhum registo das ações dos utilizadores ou eventos de segurança, dificultando a resposta a incidentes e a análise forense.
  - **Falha em Detectar e Responder a Ataques:** Detecção lenta ou inexistente e resposta a violações de segurança.

- **Engenharia Social:**
  - ***Phishing*:** Enganar utilizadores para que divulguem credenciais ou outras informações sensíveis.
  - ***Spear Phishing*:** Ataques de *phishing* direcionados contra indivíduos específicos, como administradores ou proprietários de cursos.

- **APIs Inseguras:**
  - **Acesso Não Autorizado:** APIs mal protegidas, permitindo acesso não autorizado aos dados do backend.
  - **Exposição Excessiva de Dados:** APIs expõem mais dados do que o necessário.

- ***Malware*:**
  - **Uploads de Ficheiros:** Upload de ficheiros maliciosos disfarçados como conteúdo legítimo para explorar vulnerabilidades nos sistemas do servidor ou do cliente.

- **Validação de *Inputs* Insuficiente:**
  - ***Buffer Overflow*:** Validação de *inputs* inadequada, culminando num *buffer overflow*.
  - **Diretório dos Ficheiros:** Validação inadequada de *paths* de ficheiros, permitindo acesso a ficheiros e pastas não autorizados no servidor.

- **Vulnerabilidades em Componentes de Terceiros:**
  - **Bibliotecas Desatualizadas:** Uso de bibliotecas ou *frameworks* desatualizados com vulnerabilidades conhecidas.
  - **Ataques a terceiros:** Componentes de terceiros comprometidos comprometem a segurança da aplicação.

4 Implementação

4.1 Componentes do Sistema

O sistema de gestão de cursos desenvolvido visa proporcionar uma plataforma robusta e eficiente para a criação, gestão e participação em cursos *online*. Este sistema é composto por diversos componentes que interagem de forma coesa para oferecer uma experiência de utilizador intuitiva e funcional. Os principais componentes do sistema incluem o *frontend*, implementado com React e Bootstrap, o *backend*, desenvolvido com FastAPI, a base de dados gerida por SQLAlchemy com SQLite, e uma pasta de *uploads* para a gestão de ficheiros. A seguir, descrevemos cada um destes componentes em detalhe, bem como as interfaces que os interligam.

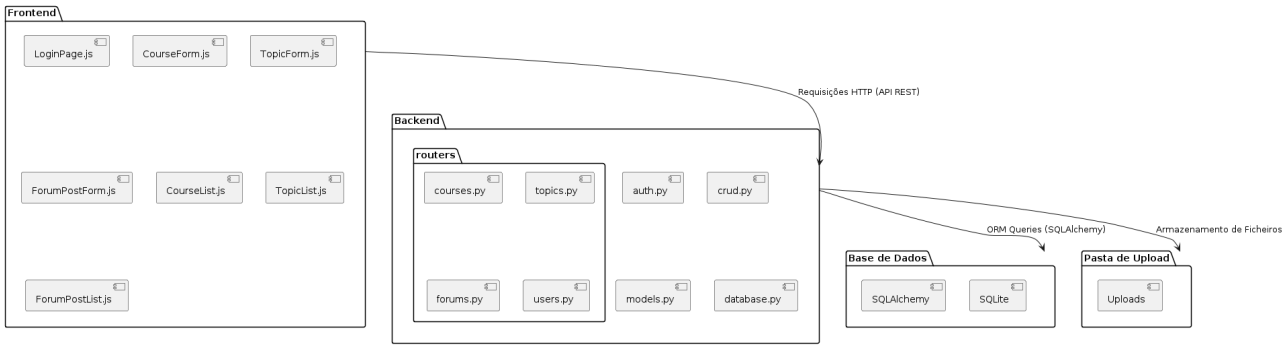


Figure 5: Diagrama de Componentes com Ficheiros

4.1.1 Frontend (React, Bootstrap)

O frontend do sistema é implementado utilizando a biblioteca React, que facilita a construção de interfaces de utilizador interativas e dinâmicas. O Bootstrap é utilizado para estilizar a interface, proporcionando um *design* responsivo e visualmente atraente.

Principais Funções:

- **Interação com o Utilizador**: permite que os utilizadores interajam com o sistema através de formulários, botões e outras componentes de interface.
- **Comunicação com o *Backend***: envia pedidos HTTP para o *backend* para realizar operações como *login*, criação de cursos, publicação de tópicos, etc.
- **Exibição de Dados**: recebe dados do *backend* e exibe-os na interface de utilizador.

Componentes Principais:

- `LoginPage.js`: gere o processo de autenticação dos utilizadores.
- `CourseForm.js`: permite a criação e edição de cursos.
- `TopicForm.js`: permite a criação e edição de tópicos.
- `ForumPostForm.js`: permite a criação de perguntas e respostas nos fóruns.

- `CourseList.js`: exibe a lista de cursos disponíveis.
- `TopicList.js`: exibe a lista de tópicos num curso.
- `ForumPostList.js`: exibe as postagens no fórum.

4.1.2 Backend (FastAPI)

O *backend* é implementado utilizando a *framework* FastAPI, conhecida pela sua eficiência e facilidade de uso na construção de APIs RESTful. O FastAPI é utilizado para gerir a lógica da plataforma, autenticação, autorização e interações com a base de dados.

Principais Funções:

- **Gestão de Autenticação e Autorização**: verifica credenciais dos utilizadores e gera *tokens* JWT para sessões autenticadas.
- **Manipulação de Dados**: realiza operações CRUD (*Create, Read, Update, Delete*) para cursos, tópicos, e postagens no fórum.
- **Comunicação com a Base de Dados**: interage com a base de dados através do SQLAlchemy.

Componentes Principais:

- `auth.py`: gere a autenticação e geração de *tokens* JWT.
- `crud.py`: contém as operações CRUD para os diferentes modelos de dados.
- `models.py`: define os modelos de dados utilizados no sistema.
- `routers`:
  - `auth.py`: roteamento para *endpoints* de autenticação.
  - `courses.py`: roteamento para operações relacionadas a cursos.
  - `topics.py`: roteamento para operações relacionadas a tópicos.
  - `forums.py`: roteamento para operações relacionadas a postagens no fórum.
  - `users.py`: roteamento para operações relacionadas a utilizadores.

4.1.3 Base de Dados (SQLAlchemy / SQLite)

Principais Funções:

- **Armazenamento de Dados**: Armazena dados persistentes como informações de utilizadores, cursos, tópicos e postagens no fórum.
- **Consultas e Atualizações**: Permite consultas eficientes e atualizações dos dados através do SQLAlchemy.

Componentes Principais:

- `models.py`: define as tabelas e os relacionamentos na base de dados.
- `database.py`: configura a ligação com a base de dados e inicializa a base de dados.

4.1.4 Pasta de Upload

Principais Funções:

- **Armazenamento de Ficheiros**: Armazena os ficheiros carregados pelos utilizadores de forma organizada.
- **Acesso aos Ficheiros**: Permite que os ficheiros sejam acedidos e exibidos nas postagens dos fóruns.

Componentes Principais:

- **Configuração de Upload no *Backend***: Configura o FastAPI para aceitar e armazenar ficheiros carregados.
- **Integração com *Frontend***: Permite que os utilizadores carreguem ficheiros através da interface do *frontend*, que são, então, enviados e armazenados pelo *backend*.

4.2 Interfaces do Sistema (Endpoints REST)

As interfaces entre os diferentes componentes do sistema são estabelecidas através de *endpoints* REST, que permitem a comunicação entre o *frontend* e o *backend*. Estes *endpoints* são responsáveis por receber requisições, processar dados e retornar respostas apropriadas.

Principais Endpoints:

- **Autenticação:**
  - `POST /login`: verifica as credenciais do utilizador e retorna um *token* JWT.
  - `POST /logout`: encerra a sessão do utilizador.
- **Utilizadores:**
  - `GET /users/me`: retorna os detalhes do utilizador autenticado.
  - `POST /users`: cria um utilizador.
  - `PUT /users/{id}`: atualiza um utilizador existente.
  - `DELETE /users/{id}`: remove um utilizador.
- **Cursos:**
  - `GET /courses`: lista todos os cursos.
  - `POST /courses`: cria um curso.
  - `PUT /courses/{id}`: atualiza um curso existente.
  - `DELETE /courses/{id}`: remove um curso.

- **Tópicos:**
  - `GET /courses/{course_id}/topics`: lista todos os tópicos de um curso.
  - `POST /courses/{course_id}/topics`: Cria um tópico.
  - `PUT /topics/{id}`: atualiza um tópico existente.
  - `DELETE /topics/{id}`: remove um tópico.
- **Postagens no Fórum:**
  - `GET /topics/{topic_id}/forum_posts`: lista todas as postagens de um tópico.
  - `POST /topics/{topic_id}/forum_posts`: Cria uma postagem.
  - `DELETE /forum_posts/{id}`: remove uma postagem.

Os componentes do sistema, desde o *frontend* até a base de dados, com as interfaces estabelecidas através de *endpoints* REST, formam a base de um sistema de gestão de cursos eficiente e robusto. Cada componente desempenha um papel essencial na funcionalidade do sistema, assegurando uma experiência de utilizador intuitiva e eficiente. A metodologia adotada garante que todas as partes do sistema estão bem integradas e independentes para atingir os objetivos do projeto.

4.3 Fluxo de Autenticação

A autenticação é um dos processos fundamentais em qualquer sistema de gestão de cursos, garantindo que apenas utilizadores autorizados tenham acesso às funcionalidades do sistema. O diagrama de sequência a seguir ilustra o fluxo detalhado do processo de autenticação no sistema, desde a submissão do formulário de *login* até a verificação e armazenamento do *token* JWT.

Descrição do Diagrama de Sequência: Autenticação

1. Submissão do Formulário de *Login*: O utilizador insere o nome de utilizador e a senha e submete o formulário de *login* através da interface fornecida pelo `LoginPage.js`.
2. Hash da *Password*: O `LoginPage.js` aplica um hash à senha utilizando o algoritmo SHA-256 antes de enviar os dados para o *backend*. Esta medida é adotada para assegurar que a senha nunca é transmitida em texto claro, adicionando uma camada adicional de segurança.
3. Envio dos Dados de *Login*: Os dados de *login*, compostos pelo nome de utilizador e a senha hashed, são enviados ao *backend*, especificamente ao *endpoint* gerido pelo FastAPI.
4. Autenticação do Utilizador:
   - O FastAPI chama a função `authenticate_user()` definida em `auth.py` para iniciar o processo de autenticação.
   - `authenticate_user()` por sua vez chama `get_user_by_username()` em `crud.py` para obter os dados do utilizador.
5. Consulta à Base de Dados:

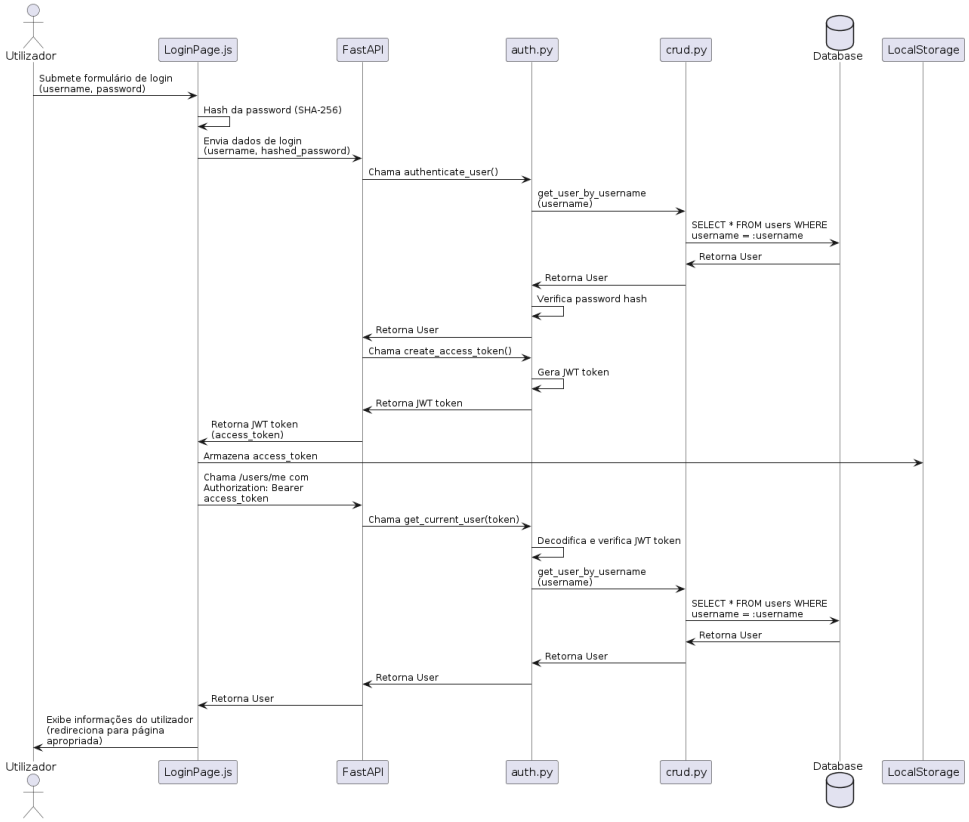


Figure 6: Diagrama de Sequência: Autenticação

- `crud.py` realiza uma consulta na base de dados para procurar um utilizador com o nome de utilizador fornecido.
- A base de dados retorna os dados do utilizador encontrado, que são enviados de volta através de `crud.py` para `auth.py`.

6. Verificação da *Password*:
   - `auth.py` compara a senha hashed fornecida com a senha armazenada na base de dados.
   - Se as senhas coincidirem, o utilizador é autenticado com sucesso.
7. Geração de *Token* JWT:
   - `auth.py` chama `create_access_token()` para gerar um *token* JWT (JSON Web Token).
   - O *token* JWT é gerado e retornado para o FastAPI.
8. Retorno do *Token* JWT: O FastAPI envia o *token* JWT de volta ao `LoginPage.js`.
9. Armazenamento do *Token* JWT: O `LoginPage.js` armazena o *token* JWT no Local-Storage para utilizá-lo em requisições futuras, garantindo que o utilizador não tenha que reautenticar-se a cada requisição.
10. Verificação do *Token* JWT:

- O `LoginPage.js` chama o endpoint `/users/me` com o *token* JWT incluído no cabeçalho de autorização.
- O FastAPI chama `get_current_user(token)` em `auth.py` para descodificar e verificar o *token* JWT.
- `auth.py` decodifica o *token* e verifica a sua validade, depois chama `get_user_by_username()` novamente para obter os dados atualizados do utilizador.

11. Retorno das Informações do Utilizador:
    - Após verificar o *token*, `auth.py` retorna os dados do utilizador para o FastAPI, que por sua vez os envia de volta ao `LoginPage.js`.
    - O `LoginPage.js` exibe as informações do utilizador e redireciona para a página apropriada com base nas permissões e no tipo de utilizador (Administrador, Professor, Estudante).

4.4 Criação de Curso

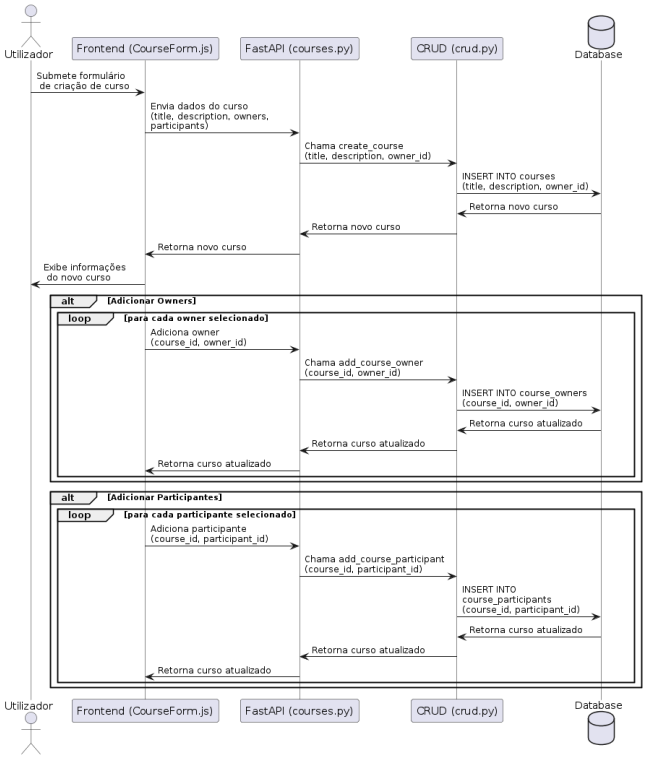


Figure 7: Diagrama de Sequência: Criação de Curso

A criação de cursos é uma funcionalidade essencial no sistema de gestão de cursos, permitindo que administradores e professores criem cursos, adicionem proprietários e participantes. O diagrama de sequência a seguir ilustra o fluxo detalhado do processo de criação de um curso, desde a submissão do formulário de criação até a adição de proprietários e participantes.

Descrição do Diagrama de Sequência: Criação de Curso

1. Submissão do Formulário de Criação de Curso: O utilizador preenche o formulário de criação de curso no frontend (`CourseForm.js`) e submete os dados do curso, incluindo o título, a descrição, os proprietários e os participantes.

2. Envio dos Dados do Curso: O frontend envia os dados do curso para o *backend*, especificamente ao endpoint gerido pelo FastAPI (`courses.py`).

3. Criação do Curso na Base de Dados:
   - O FastAPI chama a função `create_course()` definida em `crud.py`, que insere um novo curso na base de dados.
   - A função `create_course()` executa um comando `INSERT` na tabela de cursos, adicionando o título, a descrição e o ID do proprietário inicial.
   - A base de dados retorna o novo curso criado para `crud.py`, que por sua vez retorna esta informação para o FastAPI.

4. Retorno do Novo Curso: O FastAPI retorna a informação do novo curso criado para o frontend (`CourseForm.js`), que exibe os detalhes do curso para o utilizador.

5. Adição de Proprietários:
   - Se foram selecionados proprietários adicionais, o frontend envia requisições para adicionar cada proprietário ao curso.
   - Para cada proprietário, o FastAPI chama a função `add_course_owner()` em `crud.py`, que insere um novo registo na tabela `course_owners`.
   - A base de dados retorna o curso atualizado após cada inserção, e esta informação é retornada ao frontend.

6. Adição de Participantes:
   - Similarmente, se foram selecionados participantes, o frontend envia requisições para adicionar cada participante ao curso.
   - Para cada participante, o FastAPI chama a função `add_course_participant()` em `crud.py`, que insere um novo registo na tabela `course_participants`.
   - A base de dados retorna o curso atualizado após cada inserção, e esta informação é retornada ao frontend.

4.5 Publicação de Tópico

A publicação de um tópico é uma funcionalidade chave que permite aos professores disponibilizarem conteúdos para os alunos de forma organizada. O diagrama de sequência a seguir ilustra o fluxo detalhado do processo de publicação de um tópico, desde a ação do utilizador até à atualização do estado do tópico na base de dados.

Descrição do Diagrama de Sequência: Publicação de Tópico

1. Ação do Utilizador: O utilizador (professor) acede à página do tópico no frontend (`TopicPage.js`) e clica no botão "Publicar" para mudar o estado do tópico de rascunho para publicado.

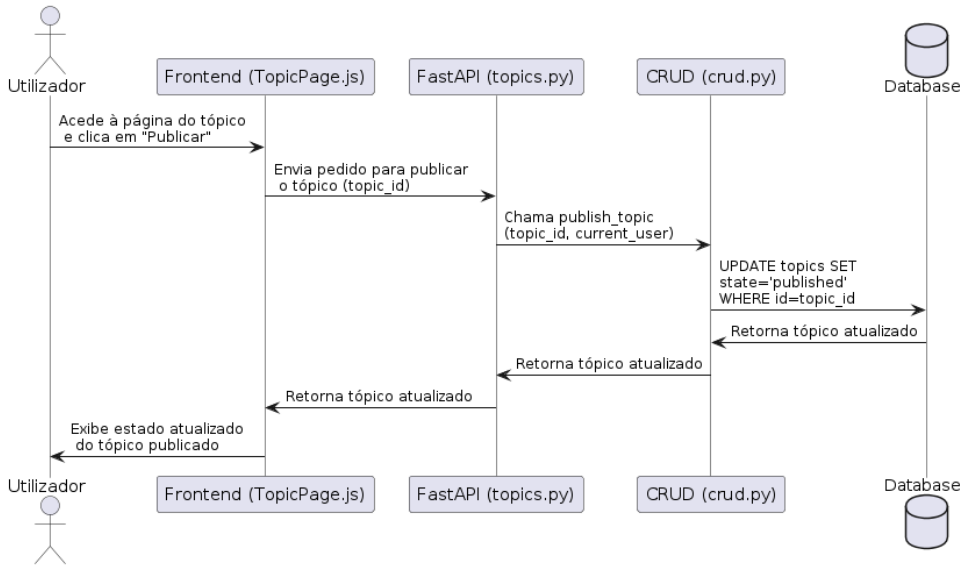


Figure 8: Diagrama de Sequência: Publicação de Tópico

2. Envio do Pedido de Publicação: O frontend (`TopicPage.js`) envia um pedido ao backend para publicar o tópico, incluindo o `topic_id` no pedido.

3. Chamada à Função de Publicação: O backend (`FastAPI` em `topics.py`) recebe o pedido e chama a função `publish_topic()` definida em `crud.py`, passando o `topic_id` e o utilizador atual (`current_user`) como parâmetros.

4. Atualização do Estado do Tópico:
   - A função `publish_topic()` em `crud.py` executa um comando `UPDATE` na base de dados para alterar o estado do tópico para 'published'.
   - A base de dados atualiza o registo do tópico e retorna o tópico atualizado para `crud.py`.

5. Retorno do Tópico Atualizado: `crud.py` retorna o tópico atualizado para o `FastAPI` (`topics.py`), que por sua vez envia esta informação de volta para o frontend.

6. Exibição do Estado Atualizado: O frontend (`TopicPage.js`) recebe o tópico atualizado e exibe o novo estado do tópico (publicado) para o utilizador.

4.6 Envio de Pergunta no Fórum

O envio de perguntas no fórum permite que os utilizadores (estudantes e professores) interajam e discutam tópicos específicos de forma colaborativa. O diagrama de sequência a seguir ilustra o fluxo detalhado do processo de envio de uma pergunta no fórum, desde a submissão pelo utilizador até a exibição da nova pergunta no frontend.

Descrição do Diagrama de Sequência: Envio de Pergunta no Fórum

1. Submissão da Pergunta pelo Utilizador: O utilizador (estudante ou professor) preenche o formulário de envio de pergunta no fórum no frontend (`ForumPostForm.js`) e submete a pergunta.

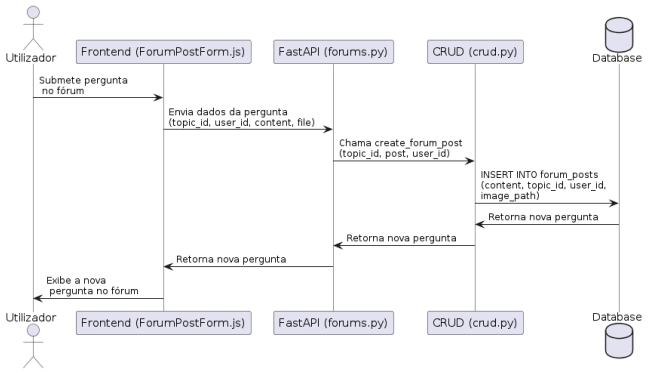


Figure 9: Diagrama de Sequência: Envio de Pergunta no Fórum

2. Envio dos Dados da Pergunta: O frontend (`ForumPostForm.js`) envia os dados da pergunta para o backend, incluindo `topic_id`, `user_id`, `content`, e `file` (se houver).

3. Chamada à Função de Criação de Postagem: O backend (`FastAPI` em `forums.py`) recebe os dados e chama a função `create_forum_post()` definida em `crud.py`, passando os dados da pergunta como parâmetros.

4. Inserção da Pergunta na Base de Dados:
   - A função `create_forum_post()` em `crud.py` executa um comando `INSERT` na tabela de `forum_posts`, adicionando o conteúdo da pergunta, `topic_id`, `user_id`, e `image_path` (se houver).
   - A base de dados insere o novo registo e retorna a pergunta criada para `crud.py`.

5. Retorno da Nova Pergunta: `crud.py` retorna a nova pergunta para o `FastAPI` (`forums.py`), que por sua vez envia esta informação de volta para o frontend.

6. Exibição da Nova Pergunta: O frontend (`ForumPostForm.js`) recebe a nova pergunta e exibe-a no fórum, tornando-a visível para todos os utilizadores que têm acesso ao tópico.

4.7 Upload de Ficheiro/Imagem

O upload de ficheiros ou imagens é uma funcionalidade essencial para permitir que os utilizadores anexem documentos ou imagens às suas postagens no fórum. O diagrama de sequência a seguir ilustra o fluxo detalhado do processo de upload de um ficheiro ou imagem, desde a submissão pelo utilizador até a exibição da URL do ficheiro no frontend.

Descrição do Diagrama de Sequência: Upload de Ficheiro/Imagem

1. Submissão do Upload pelo Utilizador: O utilizador seleciona um ficheiro ou imagem e submete o formulário de upload no frontend (`Frontend`).

2. Envio do Ficheiro/Imagem: O frontend envia o ficheiro ou imagem para o backend (`FastAPI`), utilizando uma requisição HTTP multipart/form-data.

3. Guardar Ficheiro/Imagem:

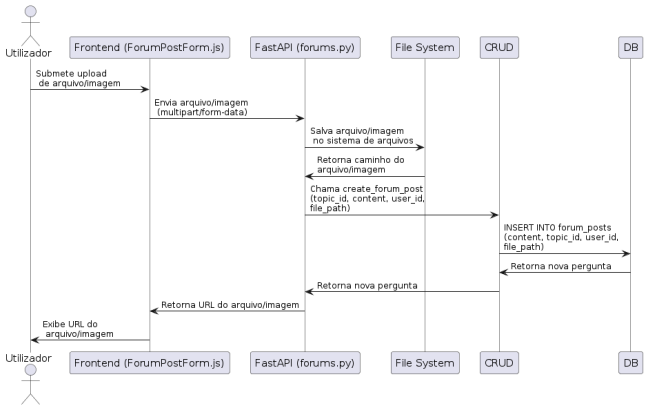


Figure 10: Diagrama de Sequência: Post Forum e Upload de Arquivo/Imagem

- O backend (`FastAPI`) recebe o ficheiro ou imagem e salva-o no sistema de arquivos (`File System`).
- O ficheiro é armazenado numa pasta dedicada a uploads, com um nome de ficheiro único para evitar conflitos.

4. Retorno do Caminho do Ficheiro/Imagem: O sistema de arquivos (`File System`) retorna o caminho onde o ficheiro ou imagem foi salvo ao backend.
5. Retorno da URL do Ficheiro/Imagem: O backend (`FastAPI`) constrói a URL de acesso ao ficheiro ou imagem com base no caminho retornado e envia esta URL de volta para o frontend.
6. Exibição da URL do Ficheiro/Imagem: O frontend recebe a URL do ficheiro ou imagem e a exibe ao utilizador, permitindo que o ficheiro ou imagem carregado seja visualizado ou acedido.

5 Análise de Segurança

A segurança é um aspecto crítico em qualquer sistema de gestão de cursos. Para assegurar a integridade e a confidencialidade dos dados, diversas medidas de segurança foram implementadas. A seguir, são descritas as principais estratégias e análises realizadas para fortalecer a segurança do sistema.

5.1 Injeção

Para prevenir ataques de injeção, várias técnicas foram implementadas:

- Restrição da Extensão dos Ficheiros: Apenas ficheiros com extensões específicas são permitidos para upload no fórum. Isto ajuda a evitar a injeção de ficheiros maliciosos que possam comprometer o sistema.
- Expressões Regulares para Prevenir XSS: Foram utilizadas expressões regulares para sanitizar entradas nos comentários do fórum, evitando ataques de Cross-Site Scripting (XSS). O padrão utilizado é `r'[<>]'`, que remove caractéres que poderiam ser usados para injetar scripts maliciosos.

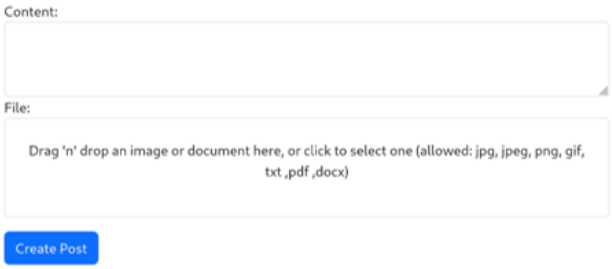


Figure 11: Restrição da Extensão dos Ficheiros

5.2 Análises de Vulnerabilidade

Diversas ferramentas de análise de vulnerabilidade, tanto estáticas quanto dinâmicas, foram utilizadas para identificar e mitigar potenciais falhas de segurança no sistema.

- Análise Dinâmica com Wappalyzer e Nuclei:
  - A ferramenta Wappalyzer foi utilizado para identificar tecnologias e potenciais vulnerabilidades nas camadas de aplicação. Vestindo o "fato de pen-tester", este passo vai ajudar em fases posteriores do *pen-test*.
  - O Nuclei foi empregado para fazer um *scan* do sistema à procura de vulnerabilidades conhecidas, permitindo a correção proativa de problemas. Tentamos corre genericamente, bem como com templates. Conseguimos receber informação sobre vulnerabilidades nos *headers* que estamos a enviar.

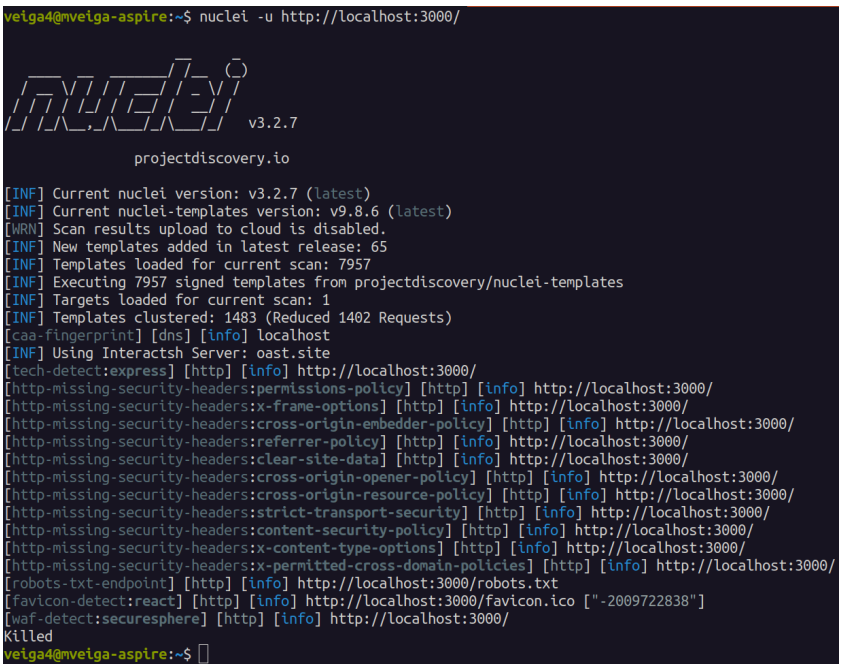


Figure 12: Análise da Ferramenta Nuclei

- Análise Estática com Node's Packet Vulnerability Search: foi utilizado para identificar vulnerabilidades nas dependências de pacotes Node.js, garantindo que todas as bibliotecas

utilizadas estejam livres de falhas conhecidas. Foram identificadas 8 vulnerabilidades em pacotes:

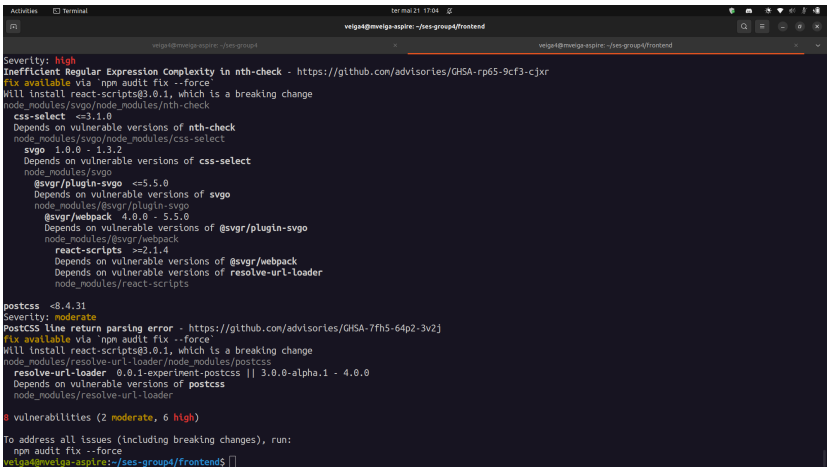


Figure 13: Análise do Node's Packet Vulnerability antes de fazer *patch*

Complementando a informação obtido também com o Nuclei, de forma a corrigir estas falhas, fizemos *patch*, ou seja, fizemos *override* da versão dos pacotes vulneráveis no ficheiro "packages.json" para versões onde as vulnerabilidades já tenham sido resolvidas.

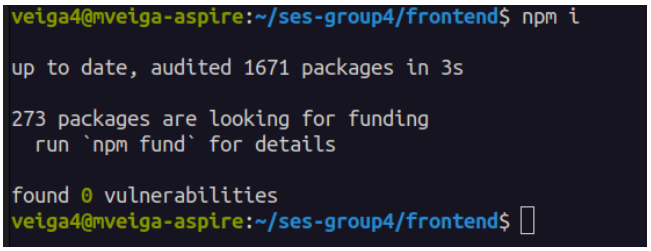


Figure 14: Análise do Node's Packet Vulnerability depois de fazer *patch*

- Análise Estática com Sonar Cloud: Sonar Cloud forneceu uma análise detalhada do código-fonte, não identificando nenhuma vulnerabilidade visto que apresentou um "Security Rating" de A, em todas as páginas e ficheiros analisados.

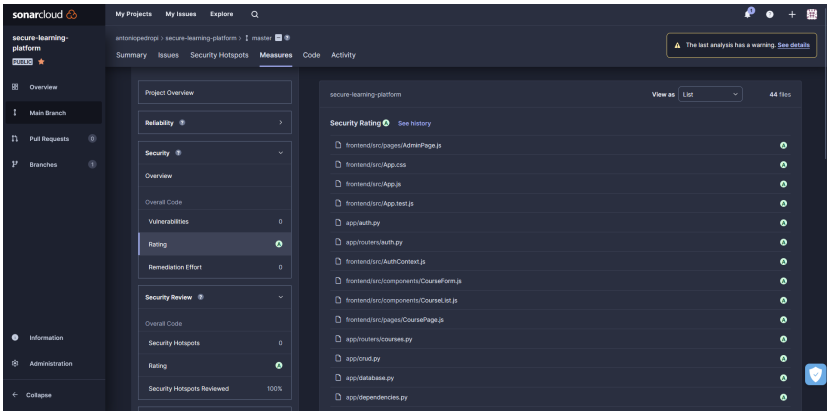
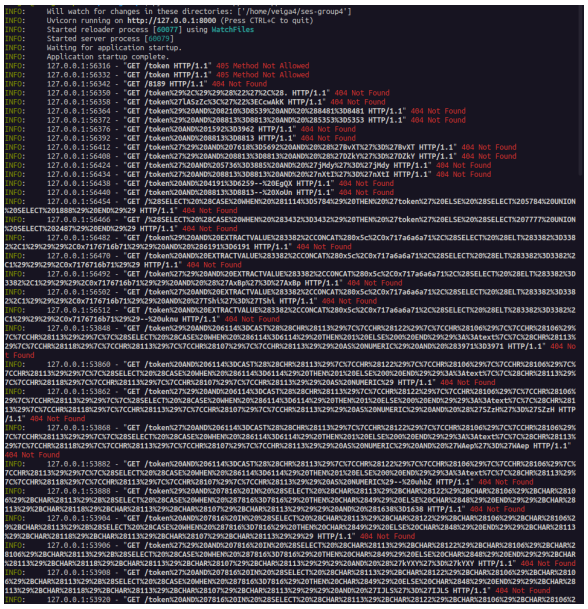
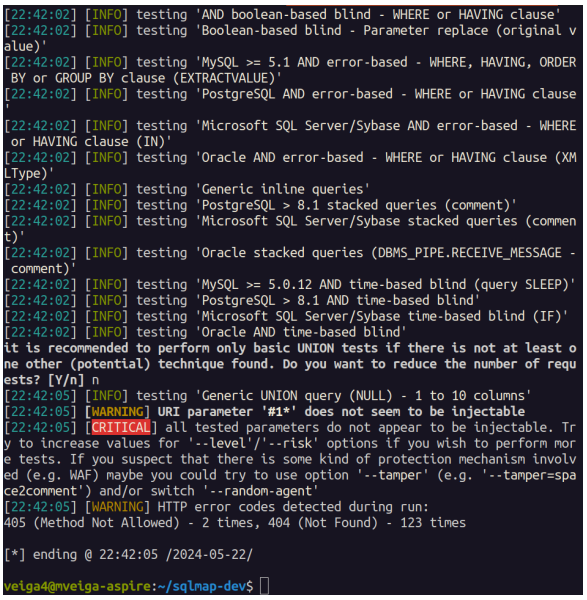


Figure 15: Análise do Sonar Cloud

- Análise Dinâmica com SQLMap: SQLMap foi utilizado para testar a aplicação contra ataques de SQL Injection. A ferramenta automatizou o processo de detecção e exploração de vulnerabilidades de injeção SQL, ajudando a garantir que as consultas ao banco de dados são seguras. Os resultados foram positivos, no teste que efetuamos para a página de *login*, demonstrando que os parâmetros foram bem parametrizados.



(a) Queries SQL na tentativa de fazer injeção



(b) Resultados globais

Figure 16: Análises SQLMap

- Análise Dinâmica com ZAP: O OWASP ZAP (Zed Attack Proxy) foi utilizado para realizar testes de penetração dinâmicos na aplicação web. A análise do SQLMap para testes de injeção não é completa, devido à existência de vulnerabilidades de XSS e a possibilidade de injeção de código SQL não só na página de *login*, mas também nos tópicos. Como tal, utilizamos esta ferramenta para testar estas possibilidades. Os resultados foram positivos, revelando que conseguimos proteger contra injeções.

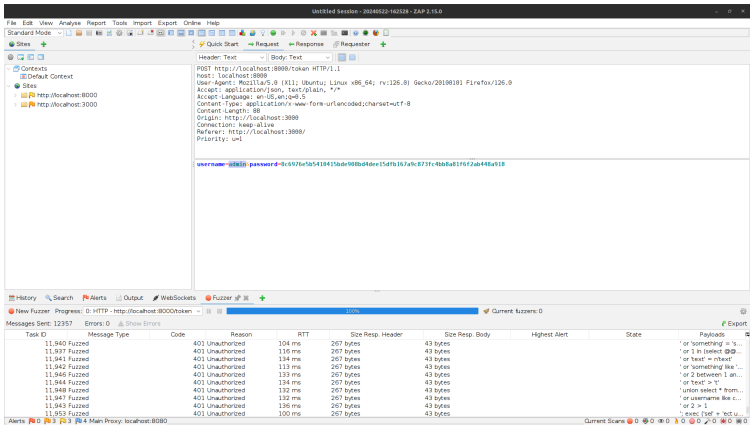


Figure 17: Análise ZAP para o campo 'username'

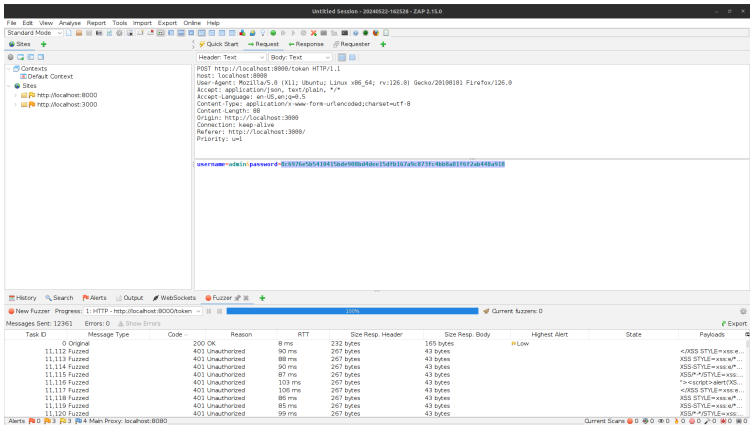


Figure 18: Análise ZAP para o campo 'password'

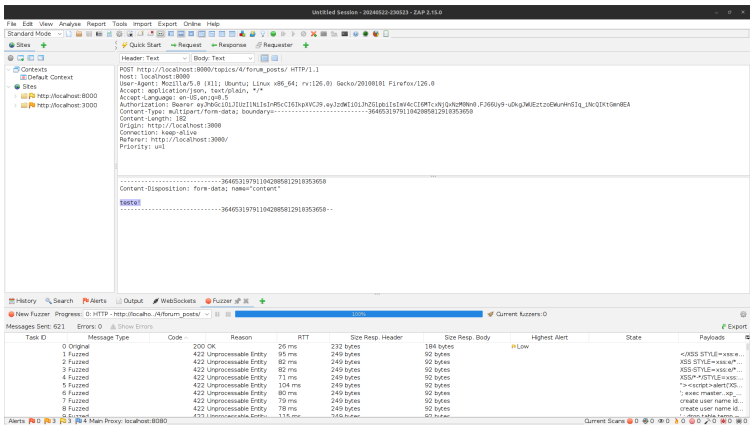


Figure 19: Análise ZAP para o campo 'tópico'

6 Conclusão

No âmbito deste projeto, tivemos a oportunidade de repensar o concetio de segunraça em software não só como uma medida reativa, mas também como uma medida proativa, integrando-a em todas as fases de desenvolvimento de código. Como tal, fizemos o design, implementamos e analisamos uma aplicação de aprendizagem feita de raíz por nós para este projeto. Foi possível construir uma aplicação segura, em que existe proteção contra uma variedade de atques que poderiam comprometer a confidencialidade e integridade da aplicação. Queriamos também ressalvar que, enquanto que os resultados das ferramentas de segurança testadas tenham sido poisitivos, apresentam limitações, nomeadamente na resistência a injeção por XSS. A espressão regular que foi implementad apara impedir este tipo de ataques filtra os caractéres '<' e '>', porque consideramos que os *payloads* de XSS apresentam geralmente os mesmos. Contudo, nem sempre é verdade, e podem haver *payloads* de XSS sem esses caractéres que iram poder ser injetados. Por fim, concluímos com a importância de integrar e pensar na segurança desde a primeira fase do desenvolvimento de software, para acautelar e facilitar outras fases que ganham com esta metodologia.

References

[1] J. Jürjens. Secure systems development with uml. In *Security and Cryptology*, page 318. Springer, 2005.

[2] M.U.A. Khan and M. Zulkernine. Quantifying security in secure software development phases. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference*, pages 955–960, 2008.

[3] M.U.A. Khan and M. Zulkernine. On selecting appropriate development processes and requirements engineering methods for secure software. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference*, pages 353–358, 2009.

[4] Infosys Limited. Speed and security: How to achieve both in agile development. `https://www.infosys.com/agile-devops/documents/speed-security.pdf`. Accessed: 2024-05-20.

[5] G. McGraw. *Software Security: Building Security In*. Addison Wesley, 2006.

[6] Nabil M. Mohammed, Mahmood Niazi, Mohammad Alshayeb, and Sajjad Mahmood. Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards Interfaces*, 50:107–115, 2017.

[7] Hugo Pacheco. Project documentation. `https://github.com/hpacheco/ses/blob/main/project/Project.md`, 2024. Accessed: 2024-05-20.

[8] P. Salini and S. Kanmani. Security requirements engineering process for web applications. *Procedia Engineering*, 38:2799–2807, 2012.

[9] P. Salini and S. Kanmani. Survey and analysis on security requirements engineering. *Computers & Electrical Engineering*, 38(6):1785–1797, 2012.

[10] Kiran Kumar Voruganti. Implementing security by design practice with devsecops shift left approach. *Journal of Technological Innovations*, 2(1), 2021.