UNIVERSIDADE
LUSÓFONA

Gestão da qualidade de implementação de user stories

Trabalho Final de curso

Intercalar 2º Semestre

Afonso Lopes, a22204080, Engenharia Informatica

André Guimarães,a22204198, Engenharia Informatica

Orientador: Prof. Luís Gomes

Co-orientador: Prof. José Brás

Entidade Externa: CGI TI Portugal

Departamento de Engenharia Informática da Universidade Lusófona

Centro Universitário de Lisboa

07/04/2026

www.ulusofona.pt

Formatted: Font: +Headings (Cambria), 28 pt

Formatted: Centered

Formatted: Line spacing: Multiple 1.15 li

Formatted: Justified

Formatted: Not Highlight

Formatted: Justified

Direitos de cópia

Agradecimentos

Resumo

O trabalho "Gestão da qualidade de implementação de user stories" propõe o desenvolvimento de uma solução inovadora para automatizar a validação do código face às user stories durante o ciclo de desenvolvimento de software.

O sistema tem como principal objetivo garantir que o código implementado cumpre exatamente os critérios de aceitação definidos, promovendo a qualidade, a rastreabilidade e a eficiência das equipas de desenvolvimento.

A solução desenvolvida é composta por uma API REST em Python/FastAPI que recebe o código submetido, associa-o à respetiva user story e executa uma validação automática recorrendo a um modelo de linguagem (LLaMA, via Groq API). O sistema devolve uma pontuação de 0 a 100, um resumo da avaliação, os critérios cumpridos e falhados, e sugestões concretas de melhoria. Para além da avaliação de código, o sistema avalia também a qualidade das próprias user stories face às boas práticas Scrum (critérios INVEST e formato Given/When/Then). A integração com Azure DevOps e Azure Repos permite importar work items e código diretamente dos repositórios da equipa.

Este ciclo — validação, feedback e revalidação — repete-se até atingir a conformidade total, garantindo que as entregas de software respeitam os requisitos propostos. O modelo de dados foi estruturado em 3ª forma normal (3FN), assegurando integridade e rastreabilidade entre utilizadores, user stories, submissões, validações e feedbacks.

O projeto adota uma metodologia Agile, com desenvolvimento iterativo e validação contínua, resultando numa solução escalável e aplicável a contextos académicos e empresariais.

O trabalho pode ser encontrado neste repositório : https://github.com/afonsolopes22/TFC-UserStories

Palavras chave:

Commented [JB1]: As palavras-chave devem estar imediatamente a seguir ao resumo.
Colocar na pagina anterior

Abstract

Formatted: Normal

Resumo em inglês.

The project **"Quality Management of User Story Implementation"** proposes an innovative solution to automate code validation against user stories during the software development lifecycle.

Its main goal is to ensure that the implemented code strictly meets the defined acceptance criteria, improving quality, traceability, and team efficiency.

The solution consists of a **Python/FastAPI REST API** that receives submitted code, links it to the corresponding user story, and performs automatic validation using a language model (LLaMA via Groq API). The system returns a score from 0 to 100, an evaluation summary, fulfilled and unfulfilled criteria, and concrete improvement suggestions.

In addition to code evaluation, the system also assesses the quality of the user stories themselves, based on Scrum best practices (INVEST criteria and Given/When/Then format).

Integration with Azure DevOps and Azure Repos enables direct import of work items and source code from team repositories.

This cycle—validation, feedback, and revalidation—repeats until full compliance is achieved, ensuring that software deliveries meet the proposed requirements.

The data model is structured in **Third Normal Form (3NF)**, ensuring integrity and traceability across users, user stories, submissions, validations, and feedback.

The project follows an **Agile methodology**, with iterative development and continuous validation, resulting in a scalable solution applicable in both academic and professional environments.

Repository: https://github.com/afonsolopes22/TFC-UserStories

Keywords:

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas

API Interface de Programação de Aplicações

HTML Linguagem de Marcação de Hipertexto

QA Testes de Controlo de Qualidade (Quality Assurance)

ODS **Objetivos de Desenvolvimento Sustentável**

RF **Requisitos Funcionais**

RNF **Requisitos Não Funcionais**

CA **Critérios de Aceitação**

1 Introdução

1.1 Enquadramento

O trabalho, embora inserido no âmbito do trabalho final de curso, situa-se mais concretamente na área de engenharia de software, controlo de qualidade e validação de código.

No desenvolvimento ágil, as equipas lidam, com frequência, com a dificuldade em garantir com que o codigo cumpra o que é descrito nas User Stories.

Este projeto procura dar uma resposta viável a essa necessidade, através da criação de um sistema de back-end automatizado capaz de verificar se o código implementado cumpre os critérios de aceitação definidos, recorrendo a modelos de linguagem de grande escala (LLM) para produzir avaliações estruturadas e feedback acionável. O sistema integra-se com o Azure DevOps e Azure Repos, permitindo que as equipas consumam a solução diretamente a partir dos seus ambientes de trabalho habituais.

1.2 Motivação e Identificação do Problema

A motivação para o desenvolvimento deste projeto prende-se com a importância de assegurar consistência entre os requisitos definidos e a implementação final. Muitas equipas enfrentam dificuldades em garantir que o código reflete o que foi solicitado nas User Stories, o que se traduz em inconsistências e falhas de qualidade.

O problema identificado é a falta de um processo automatizado que valide o código face as user stories e forneça feedback imediato aos programadores.A solução proposta visa resolver este grande problema ao criar um sistema capaz de validar automaticamente o código submetido, gerar relatórios de conformidade e permitir a repetição até que todos os critérios estejam cumpridos.

Commented [JB2]: Atenção aos erros.... Coloquem o corretor automático..

1.3 Objetivos

O problema reflete um problema real vivido por equipas de software e o nosso objetivo é apresentar e desenvolver um sistema automático que valide se o código implementa corretamente o que é solicitado nas user stories.

Objetivos detalhados:

- Módulo de Avaliação de Código: compara o código submetido com os critérios de aceitação definidos, recorrendo a um LLM (LLaMA via Groq API), devolvendo uma pontuação (0–100), resumo, critérios cumpridos/falhados e sugestões de melhoria.
- Módulo de Avaliação de User Stories: avalia a qualidade das próprias user stories face aos critérios INVEST e ao formato Given/When/Then, atribuindo uma classificação (Muito bom / Bom / Satisfatório / Mau).Ciclo de Revalidação: Ciclo continuo até que o código esteja em conformidade total.
- Integração com Azure DevOps e Azure Repos: permite importar work items e código diretamente dos repositórios da equipa, suportando múltiplas organizações.
- Sistema de Streaming: suporte a respostas em tempo real via Server-Sent Events (SSE), para feedback token a token.
- Rastreabilidade: garantir consistência entre user stories, código submetido e resultados de validação, com suporte multi-tenant por organização.

1.4 Estrutura do Documento

O relatório está organizado da seguinte forma:

- Na Secção 1 (Introdução), é apresentado o enquadramento, as motivações, os objetivos e a estrutura geral do trabalho.
- Na Secção 2 (Pertinência e Viabilidade), é analisada a relevância do problema e a viabilidade técnica, económica, social e ambiental da solução.
- Na Secção 3 (Especificação e Modelação), são descritos os requisitos funcionais, não funcionais e de sistema, bem como os diagramas UML e o modelo de dados.
- Na Secção 4 (Solução Desenvolvida), são apresentadas a arquitetura real implementada, as tecnologias utilizadas, os componentes e os ambientes de teste e produção.
- Na Secção 5 (Testes e Validação), são descritos os testes realizados, os resultados obtidos e a estratégia de validação adotada.
- Na Secção 6 (Método e Planeamento), é descrita a metodologia Scrum adotada, o planeamento por sprints e a análise crítica ao progresso.

- Na Secção 7 (Resultados), são apresentados os resultados dos testes e o grau de cumprimento dos requisitos definidos.
- Na Secção 8 (Conclusão), são apresentadas as conclusões do trabalho desenvolvido e as perspetivas de trabalho futuro.

2 Pertinência e Viabilidade

2.1 Pertinência

No desenvolvimento ágil, as equipas trabalham com ciclos curtos de entrega, grande volume de user stories e sucessivos increments de código. Apesar disso, a validação entre o que está descrito na user story e o que efetivamente foi implementado continua, na maioria dos casos, a ser feita de forma manual, dispersa entre testes, revisões de código e validações informais com o Product Owner.

Esta realidade conduz frequentemente a:

- **inconsistências** entre requisitos e implementação;
- **retrabalho** devido a critérios de aceitação não cumpridos;
- **dificuldade em medir a conformidade** das entregas face às user stories;
- **falta de rastreabilidade** entre *user story*, código submetido e resultados de validação.

Commented [JB3]: As palavras em Inglês, devem estar em Itálico

Formatted: Font: Italic

O sistema proposto responde diretamente a estes problemas, ao:

- centralizar a **inserção e gestão de *user stories***, com critérios de aceitação explícitos;
- permitir a **submissão de código** associada a cada user story;
- executar uma **validação automática** entre código e critérios de aceitação;
- gerar **feedback estruturado e rastreavel**;
- suportar um **ciclo iterativo de revalidação**, até atingir conformidade total.

Commented [JB4]: idem

Formatted: Font: Italic

Os resultados preliminares de recolha de necessidades junto de estudantes e profissionais de desenvolvimento (a detalhar em anexo, sob a forma de inquérito) indicam que:

- há **perceção clara** de que a validação de user stories é um ponto crítico de qualidade;
- existe **carência de ferramentas** que façam a ponte direta entre user story, código e critérios de aceitação;
- a maioria dos respondentes considera **relevante** a existência de um sistema que automatize esta verificação e forneça métricas de conformidade.

Deste modo, o projeto é pertinente tanto em contexto académico (apoio à aprendizagem de boas práticas de engenharia de software) como em contexto empresarial (melhoria da qualidade e previsibilidade das entregas).

2.2 Viabilidade

A viabilidade do projeto é analisada em quatro dimensões principais: técnica, económica, social e ambiental, bem como no enquadramento com os Objetivos de Desenvolvimento Sustentável (ODS).

- **2.2.1 Viabilidade técnica**

Do ponto de vista técnico, a solução é viável porque:

- se baseia numa **arquitetura em camadas** (*front-end*, API/*back-end,* módulo de validação e base de dados), já descrita na secção 4.2;
- utiliza uma **base de dados relacional** estruturada em 3ª forma normal (3FN), garantindo integridade e rastreabilidade entre utilizadores, *user stories*, submissões, validações e feedbacks (secção 3.2);
- o **módulo de avaliação** recorre ao modelo de linguagem LLaMA, disponibilizado via Groq API, que processa o código e os critérios de aceitação e devolve uma avaliação estruturada em JSON — pontuação, resumo, critérios cumpridos/falhados e sugestões de melhoria — num formato diretamente consumível pelo frontend;
- as tecnologias utilizadas — **Python, FastAPI, SQLAlchemy, SQLite, pytest e Groq API** — são maduras, largamente documentadas e compatíveis com infraestruturas correntes; o sistema está atualmente em produção no Render (https://tfc-userstories.onrender.com), validando a viabilidade do deployment..

Os requisitos funcionais e não funcionais identificados (secção 3.1) são tecnicamente exequíveis com os recursos atualmente disponíveis, não exigindo hardware especializado nem algoritmos de elevada complexidade.

- **2.2.2 Viabilidade económica**

A viabilidade económica assenta nos seguintes pontos:

- **Custos de desenvolvimento controlados**, uma vez que o sistema pode ser construído de forma incremental, começando por um protótipo funcional que cubra o ciclo básico: inserção de user story → submissão de código → validação → feedback;
- **Redução de custos de retrabalho**, graças à deteção precoce de não conformidades, o que diminui o número de iterações informais entre equipas de desenvolvimento, QA e Product Owner;
- **Potencial de reutilização** em múltiplos projetos, tanto académicos como empresariais, diluindo o investimento inicial;
- possibilidade de evolução para um **modelo de negócio SaaS**, com subscrição por equipa ou por projeto, gerando receita recorrente com custos operacionais previsíveis.

Uma análise custo-benefício preliminar sugere que a diminuição de bugs em produção, o aumento da qualidade das entregas e a melhoria da eficiência das equipas podem compensar rapidamente o investimento de desenvolvimento e manutenção da plataforma.

- **2.2.3 Viabilidade social**

A viabilidade social está relacionada com a adoção e aceitação da solução pelas partes interessadas:

- **Programadores**: beneficiam de feedback imediato e estruturado, com indicações claras sobre que critérios de aceitação falharam e que melhorias são recomendadas;
- ***Product Owners* / analistas de negócio**: passam a dispor de **evidência objetiva** de conformidade entre user story e código, facilitando decisões de aceitação ou rejeição;

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Italic

- **Equipa de QA**: pode focar-se em testes exploratórios e de alto nível, deixando a validação sistemática e repetitiva para o sistema automatizado;
- **Contexto académico**: docentes podem usar a ferramenta para avaliar TPCs e projetos, medindo o grau de cumprimento de requisitos e fornecendo feedback mais consistente aos alunos.

Os inquéritos e entrevistas previstos (e a incluir como anexo) permitirão validar estas perceções, mas, conceptualmente, a solução está alinhada com a necessidade crescente de **transparência, colaboração e comunicação clara** em equipas ágeis.

- **2.2.4 Viabilidade ambiental e enquadramento nos ODS**

Embora se trate de um projeto essencialmente digital, é possível identificar impacto ambiental indireto e alinhamento com os ODS:

- **ODS 8 – Trabalho digno e crescimento económico**: ao melhorar a eficiência e qualidade do desenvolvimento de software, contribui para práticas de trabalho mais sustentáveis e produtivas.
- **ODS 9 – Indústria, inovação e infraestruturas**: promove inovação na forma como a qualidade é gerida no ciclo de desenvolvimento, suportando infraestruturas digitais mais robustas.
- **ODS 12 – Produção e consumo responsáveis**: ao reduzir retrabalho, deploys falhados e iterações de correção, contribui para uma utilização mais eficiente de recursos computacionais e humanos.

O consumo de recursos adicionais (servidores, armazenamento, rede) é relativamente reduzido e pode ser otimizado com boas práticas de desenvolvimento e operação (uso de ambientes partilhados, escalabilidade elástica, monitorização).

2.3 Análise Comparativa com Soluções Existentes

Realizamos uma análise comparativa de ferramentas já existentes que realizam validação automática com controlo de qualidade do código. As soluções analisadas foram*: GitHub Actions*, *GitLab CI*, *Jira Xray*, *SonarQube* e *CodeGrade*.

	Valida código	Associa a US	Feedback automático	Ciclo de revalidação	Rastreabilidade
GitHub Actions	☑	✗	✗	✗	✗
GitLab CI	☑	✗	✗	☑	✗
Jira + Xray	✗	☑	✗	✗	☑
SonarQube	☑	✗	☑	✗	✗
CodeGrade	☑	✗	☑	☑	☑
Solução Proposta	☑	☑	☑	☑	☑

Com esta comparação conseguimos deduzir que o que vamos fazer é algo que ,até ao momento, não existe no mercado.

Commented [JB5]: Criar sigla para isto

Commented [JB6]: Criar sigla

2.4 Proposta de inovação e mais-valias

A inovação da solução proposta reside, sobretudo, em três dimensões:

1. **Foco na user story como unidade de qualidade**
    - A maioria das ferramentas foca-se em código, testes ou issues de forma isolada.
    - Neste projeto, a unidade central é a **user story com critérios de aceitação**, garantindo que toda a validação converge para a pergunta: *"o código cumpre exatamente o que esta user story define?"*.
2. **Ciclo automático de validação–feedback–revalidação**
    - O sistema não se limita a executar testes:
        - recebe o código;
        - associa-o à user story correspondente;
        - valida os critérios;
        - gera **feedback estruturado**;
        - permite **novo envio** até conformidade total.
    - Este ciclo reforça a cultura de **melhoria contínua** e reduz o tempo entre erro e correção.
3. **Rastreabilidade e métricas de conformidade**
    - Cada submissão, resultado de validação e feedback é registado, permitindo:
        - rastrear a evolução de uma user story ao longo de várias submissões;
        - medir a **taxa de conformidade** por sprint;
        - identificar padrões de erro recorrentes, úteis para ações de formação ou revisão de processos.

As principais mais-valias da solução são:

- **Para equipas de desenvolvimento**: maior confiança nas entregas, redução de retrabalho e feedback mais claro.
- **Para gestão de projeto / Product Owners**: visibilidade objetiva sobre o cumprimento de requisitos e suporte à decisão na aceitação de funcionalidades.
- **Para contexto académico**: ferramenta de apoio à avaliação de projetos, reforçando a ligação entre requisitos, implementação e testes.

2.5 Identificação de oportunidade de negócio

A solução tem potencial para evoluir de projeto académico para **produto comercial** com várias linhas de exploração:

- **Modelo SaaS (Software as a Service)**:

  - disponibilização da plataforma na nuvem, com planos de subscrição por utilizador, por equipa ou por número de projetos;
  - integração com ferramentas amplamente utilizadas (GitHub, GitLab, Jira, Trello, Azure DevOps).

- **Segmentos-alvo**:
  - **Pequenas e médias empresas de software**, que não dispõem de equipas QA muito dimensionadas, mas precisam de garantir qualidade e rastreabilidade;
  - **Empresas de consultoria** que trabalham com múltiplos clientes e projetos em paralelo, necessitando de métricas de qualidade comparáveis;
  - **Instituições de ensino superior**, que podem usar a plataforma para avaliar trabalhos de alunos, reforçando boas práticas de engenharia de software.
- **Serviços complementares**:
  - consultoria na **definição de critérios de aceitação** e desenho de pipelines de validação;
  - formação em **boas práticas de user stories e testes automatizados**;
  - customização da solução para contextos específicos (por exemplo, setores regulados).

Assim, para além de responder a uma necessidade concreta identificada no desenvolvimento ágil, o projeto abre espaço para a criação de um **produto diferenciador**, com capacidade de gerar valor económico e académico, alinhado com as tendências atuais de automação e melhoria contínua na engenharia de software.

3 Especificação e Modelação

Esta secção apresenta as caracteristicas da solução a desenvolver, dividindo entre requisitos funcionais, não funcionais, epics, features e user stories.

3.1 Análise de Requisitos

O sistema foi concebido para responder à necessidade de validação automática entre o código e as user stories, com feedback e análise continua. Os requisitos foram definidos considerando a insereção das user stories, o armazenamento em base de dados, a validação do código, a emissão de feedback e a correção até correspondência total.

O projeto foi desenvolvido com base nas práticas de Engenharia de Requisitos e Testes e Engenharia de Software, utilizando metodologias e para estruturar o sistema e os requisitos de maneira clara e objetiva.

3.1.1 Enumeração de Requisitos

Requisitos Funcionais (RF)

Tabela1-Req. Funcionais

ID	Tipo	Descrição	Critérios de Aceitação (CA)
RF_01	Funcional	Submeter código associado a um ID de User Story para iniciar o processo de validação.	**CA-RF_01-1:** Aceitar submissão com código e ID de user story válidos e iniciar validação automaticamente. **CA-RF_01-2:** Rejeitar submissões com campos obrigatórios em falta, devolvendo HTTP 422. **CA-RF_01-3:** Devolver resposta estruturada com score, resumo, critérios e sugestões.
RF_02	Funcional	Guardar submissões no histórico.	**CA-RF_02-1:** Cada submissão gera um registo persistente com timestamp e estado. **CA-RF_02-2:** O histórico pode ser consultado por user story. **CA-RF_02-3:** Submissões anteriores não são alteradas por novas submissões.

ID	Tipo	Descrição	Critérios de Aceitação (CA)
RF_03	Funcional	Rejeitar submissões inválidas.	**CA-RF_03-1:** Código vazio ou ausente é rejeitado com erro HTTP 422. **CA-RF_03-2:** Critérios de aceitação ausentes ou malformados são rejeitados. **CA-RF_03-3:** A mensagem de erro identifica claramente o campo inválido.
RF_04	Funcional	Feedback Detalhado.	**CA-RF_04-1:** Identificar e listar os critérios de aceitação cumpridos. **CA-RF_04-2:** Identificar e listar os critérios de aceitação falhados. **CA-RF_04-3:** Apresentar exatamente 3 sugestões concretas de melhoria.
RF_05	Funcional	Calcular percentagem global.	**CA-RF_05-1:** Devolver um score de 0 a 100 em cada avaliação. **CA-RF_05-2:** O score reflete proporcionalmente o número de critérios cumpridos. **CA-RF_05-3:** Código irrelevante ou vazio deve obter score 0.
RF_06	Funcional	Guardar resultado da validação.	**CA-RF_06-1:** Cada avaliação gera um registo com score, resumo e critérios. **CA-RF_06-2:** Os resultados ficam associados à submissão correspondente. **CA-RF_06-3:** Resultados são acessíveis via endpoint de consulta.
RF_07	Funcional	Calcular a percentagem de aceitação de cada user story.	**CA-RF_07-1:** Calcular corretamente a percentagem de critérios cumpridos. **CA-RF_07-2:** Garantir consistência entre o cálculo e os dados em base de dados. **CA-RF_07-3:** Mostrar a percentagem ao utilizador no resultado da avaliação.

ID	Tipo	Descrição	Critérios de Aceitação (CA)
RF_08	Funcional	Devolver resultado em JSON com score e feedback.	**CA-RF_08-1:** A resposta inclui os campos: score, summary, criteria_met, criteria_failed, improvements. **CA-RF_08-2:** O JSON é válido e diretamente consumível pelo frontend sem transformações. **CA-RF_08-3:** Em caso de erro, devolver errorCode e errorDescription.
RF_09	Funcional	Guardar no histórico todas as validações anteriores.	**CA-RF_09-1:** Todas as avaliações ficam registadas com timestamp. **CA-RF_09-2:** O histórico está ordenado cronologicamente. **CA-RF_09-3:** Não é possível eliminar avaliações do histórico.
RF_10	Funcional	Gerar mensagens descritivas para cada critério falhado.	**CA-RF_10-1:** Cada critério falhado tem uma mensagem explicativa associada. **CA-RF_10-2:** As mensagens são em linguagem simples, sem jargão técnico excessivo. **CA-RF_10-3:** As mensagens incluem uma indicação de como corrigir.
RF_11	Funcional	Visualizar feedback detalhado.	**CA-RF_11-1:** O feedback é devolvido na resposta do endpoint de avaliação. **CA-RF_11-2:** O frontend consegue apresentar o feedback sem processamento adicional. **CA-RF_11-3:** O feedback distingue claramente critérios cumpridos e falhados.
RF_12	Funcional	Associar feedback à tentativa correspondente.	**CA-RF_12-1:** Nenhum feedback pode existir sem uma avaliação correspondente. **CA-RF_12-2:** A ligação entre feedback e submissão é garantida por chave estrangeira na BD. **CA-RF_12-3:** Consultar uma submissão devolve sempre o feedback associado.

ID	Tipo	Descrição	Critérios de Aceitação (CA)
RF_13	Funcional	Aprovar automaticamente a user story quando o nível de aceitação >= 95%.	**CA-RF_13-1:** Score >= 95 desencadeia aprovação automática da user story. **CA-RF_13-2:** O estado da user story é atualizado para "Aprovado" na base de dados. **CA-RF_13-3:** O resultado da avaliação indica claramente que a user story foi aprovada.
RF_14	Funcional	Alterar estado para "Aceite" e armazenar o score final.	**CA-RF_14-1:** O estado "Aceite" é persistido na base de dados. **CA-RF_14-2:** O score final que originou a aprovação é armazenado. **CA-RF_14-3:** O estado não pode ser revertido automaticamente após aprovação.
RF_15	Funcional	Visualizar o histórico completo de tentativas e evolução da percentagem.	**CA-RF_15-1:** Listar todas as submissões de uma user story com o score de cada uma. **CA-RF_15-2:** A listagem está ordenada cronologicamente. **CA-RF_15-3:** É visível a progressão do score entre tentativas.

Requisitos Não Funcionais (RNF)

Tabela2-Req. Não Funcionais

ID	Tipo	Descrição	Critérios de Aceitação (CA)
RNF_01	Não Funcional	O sistema deve ser capaz de processar múltiplas submissões de código simultaneamente sem degradação no desempenho.	**CA-RNF_01-1:** O sistema deve processar até 100 submissões por minuto sem falhas.

ID	Tipo	Descrição	Critérios de Aceitação (CA)
RNF_02	Não Funcional	O sistema deve garantir a segurança do código enviado, evitando a sua execução direta no servidor.	**CA-RNF_02-1:** O código submetido é enviado ao LLM para análise, não executado diretamente no servidor. **CA-RNF_02-2:** Nenhum input do utilizador é interpretado como código executável no backend.
RNF_03	Não Funcional	A interface do sistema deve ser amigável e fácil de usar.	**CA-RNF_03-1:** O formulário de submissão deve ser intuitivo. **CA-RNF_03-2:** O dashboard de feedback deve ser claro e acessível.
RNF_04	Não Funcional	O sistema deve ser capaz de escalar horizontalmente para suportar aumento no número de submissões.	**CA-RNF_04-1:** O sistema deve permitir a adição de instâncias de forma transparente. **CA-RNF_04-2:** O deployment no Render suporta escalabilidade automática.
RNF_05	Não Funcional	O sistema deve implementar procedimentos de backup e recuperação de dados.	**CA-RNF_05-1:** O sistema realiza backups periódicos automáticos. **CA-RNF_05-2:** O sistema permite a recuperação de dados em tempo útil.

Requisitos de Sistema (RS)

Tabela3-Req de Sistema

ID	Tipo	Descrição	Critérios de Aceitação (CA)
RS_01	Sistema	Reconhecer IDs de user stories provenientes do Github.	**CA-RS_01-1:** O sistema aceita um ID de work item do Azure DevOps como referência de user story. **CA-RS_01-2:** O sistema valida a existência do work item antes de iniciar a avaliação. **CA-RS_01-3:** Work items inexistentes ou inacessíveis devolvem erro HTTP 404.

ID	Tipo	Descrição	Critérios de Aceitação (CA)
RS_02	Sistema	Verificar se o Github é válido antes da validação.	**CA-RS_02-1:** Credenciais inválidas devolvem erro HTTP 401. **CA-RS_02-2:** Organização não registada devolve mensagem de erro descritiva. **CA-RS_02-3:** A validação ocorre antes de qualquer chamada ao repositório.
RS_03	Sistema	Guardar Github associado a cada resultado de validação.	**CA-RS_03-1:** Cada avaliação regista a organização/repositório de origem. **CA-RS_03-2:** A informação de origem é acessível na consulta do histórico. **CA-RS_03-3:** A integridade referencial é garantida por chave estrangeira na BD.
RS_04	Sistema	Enviar código ao front-end.	**CA-RS_04-1:** O backend responde com códigos HTTP adequados (200, 201, 422). **CA-RS_04-2:** Em caso de dados inválidos, devolve HTTP 400/422 com mensagem clara. **CA-RS_04-3:** Apenas endpoints documentados podem ser utilizados pelo frontend.
RS_05	Sistema	Como back-end, quero receber o resultado da validação do front end.	**CA-RS_05-1:** O endpoint POST /evaluate/code aceita JSON body com código e critérios. **CA-RS_05-2:** O endpoint POST /evaluate/azure aceita JSON body com referência ao repositório. **CA-RS_05-3:** Todos os endpoints devolvem respostas em JSON válido com códigos HTTP adequados.
RS_06	Sistema	Revalidar automaticamente quando o código é corrigido.	**CA-RS_06-1:** Cada submissão gera uma nova avaliação independente. **CA-RS_06-2:** O histórico de avaliações anteriores é preservado. **CA-RS_06-3:** O estado da user story é atualizado conforme o resultado da nova avaliação.

3.1.2 Descrição detalhada dos requisitos principais

Requisito	Critérios de Aceitação (CA)	Regras de Validação (RV)
RF_01	**CA-RF_01-1:** Aceitar submissão com código e ID de user story válidos e iniciar validação automaticamente. **CA-RF_01-2:** Rejeitar submissões com campos obrigatórios em falta, devolvendo HTTP 422. **CA-RF_01-3:** Devolver resposta estruturada com score, resumo, critérios e sugestões.	**RV-RF_01-1:** O FE valida os campos obrigatórios antes de enviar o pedido. **RV-RF_01-2:** O BE devolve erro HTTP 422 se o código ou os critérios estiverem ausentes. **RV-RF_01-3:** O LLM é invocado apenas após validação de input bem-sucedida.
RF_04	**CA-RF_04-1:** Identificar e listar os critérios de aceitação cumpridos. **CA-RF_04-2:** Identificar e listar os critérios de aceitação falhados. **CA-RF_04-3:** Apresentar exatamente 3 sugestões concretas de melhoria.	**RV-RF_04-1:** O prompt enviado ao LLM obriga à devolução de criteria_met e criteria_failed. **RV-RF_04-2:** O BE valida que o campo improvements contém exatamente 3 entradas. **RV-RF_04-3:** Se o LLM não cumprir o formato, o BE rejeita a resposta e devolve erro.
RF_08	**CA-RF_08-1:** A resposta inclui os campos: score, summary, criteria_met, criteria_failed, improvements. **CA-RF_08-2:** O JSON é válido e diretamente consumível pelo frontend sem transformações. **CA-RF_08-3:** Em caso de erro, devolver errorCode e errorDescription.	**RV-RF_08-1:** O BE usa modelos Pydantic para garantir a estrutura do JSON de resposta. **RV-RF_08-2:** Qualquer campo em falta na resposta do LLM resulta em erro controlado. **RV-RF_08-3:** O FE não necessita de transformar a resposta para a apresentar ao utilizador.

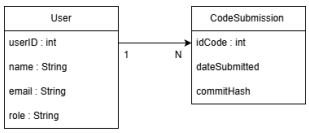
Requisito	Critérios de Aceitação (CA)	Regras de Validação (RV)
RS_05	**CA-RS_05-1:** O endpoint POST /evaluate/code aceita JSON body com código e critérios. **CA-RS_05-2:** O endpoint POST /evaluate/azure aceita JSON body com referência ao repositório. **CA-RS_05-3:** Todos os endpoints devolvem respostas em JSON válido com códigos HTTP adequados.	**RV-RS_05-1:** O FE envia sempre JSON no body, nunca query params. **RV-RS_05-2:** O BE usa modelos Pydantic para validação automática do body. **RV-RS_05-3:** Em caso de erro, a resposta inclui sempre errorCode e errorDescription.

Tabela4- Criterios de Aceitação dos Principais Requesitos

3.2 Modelação

Diagrama de Diagrama de Classes UML

O diagrama de caso de uso apresenta, as interações entre o utilizador (Developer) e as funcionalidades disponibilizadas pela solução. O objetivo é representar o que o sistema deve permitir fazer.



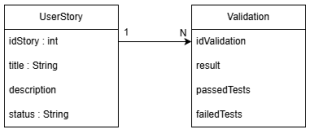


Diagrama de Atividade

Commented [JB7]: Faltam:
Diagramas de caso de uso;
Diagramas de atividade
Diagrama de ER
Diagramas de classes

Commented [JB8]: Que modelo é este?????
Falta legenda
Falta texto introdutório e a referencia á figura.....

Formatted: Font: 14 pt

Formatted: Font: 14 pt

Formatted: Font: (Default) +Body (Calibri), 11 pt

Formatted: Font: (Default) +Body (Calibri), 11 pt

Formatted: Font: (Default) +Body (Calibri), 16 pt

O diagrama de atividade ilustra o fluxo sequencial do processo de validação automática do código submetido.

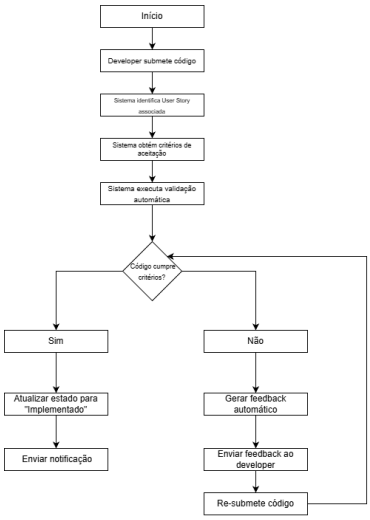
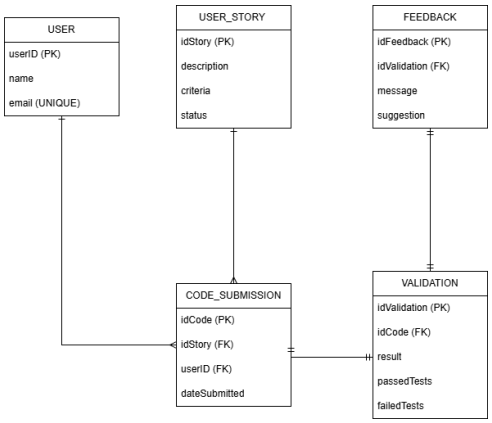


Diagrama Entidade-Relação (ER)

O diagrama entidade-relação modela a estrutura da base de dados, representando as principais entidades do sistema, os seus atributos e os relacionamentos entre elas. Este modelo foi normalizado até à 3ª forma normal .



3.3 Protótipos de Interface

Os protótipos de interface foram definidos em conjunto com a equipa de frontend, que consome a API REST desenvolvida. As principais vistas identificadas são:

- **Página de Submissão de Código:** formulário para inserir o código e selecionar a user story associada, chamando o endpoint POST /evaluate/code.
- **Página de Avaliação de User Stories:** formulário para inserir uma user story e os seus critérios de aceitação, chamando o endpoint POST /evaluate/userstory.
- **Dashboard de Resultados:** visualização da pontuação (0–100), resumo da avaliação, critérios cumpridos e falhados, e sugestões de melhoria.
- **Integração Azure DevOps:** interface para configurar a organização e importar work items e código diretamente do Azure Repos, via POST /evaluate/azure

4 Solução Proposta

4.1 Apresentação

A solução desenvolvida é uma API REST em Python/FastAPI que avalia automaticamente código e user stories recorrendo a inteligência artificial. O sistema está em produção no Render (https://tfc-userstories.onrender.com) e é consumido por uma equipa de frontend dedicada.

Principais funcionalidades implementadas:

- Avaliação automática de código face a critérios de aceitação (`POST /evaluate/code`), com pontuação 0–100, resumo, critérios cumpridos/falhados e 3 sugestões de melhoria;
- Avaliação da qualidade de user stories face às boas práticas Scrum (`POST /evaluate/userstory`), com classificação Muito bom / Bom / Satisfatório / Mau;
- Integração com Azure DevOps para importação de work items (`GET /azure/workitems`) e com Azure Repos para importação de código (`POST /evaluate/azure`);
- Suporte a múltiplas organizações com gestão de credenciais por organização (`/organizations`);
- Streaming de respostas em tempo real via Server-Sent Events (`POST /evaluate/stream`).

4.2 Arquitetura

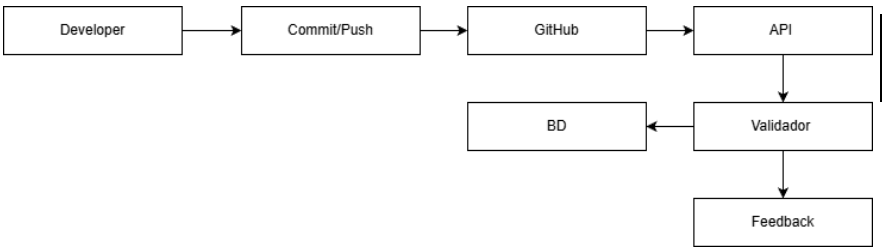
Commented [JB9]: Fazer um esquema para ilustrar a arquitetura em conjunto com o stack tecnologico

A arquitetura do sistema segue uma estrutura em camadas. O frontend comunica exclusivamente com a API REST (FastAPI), que coordena os restantes módulos. O fluxo principal de avaliação de código é o seguinte:

1. O frontend envia o código e os critérios de aceitação via POST /evaluate/code (ou POST /evaluate/azure para código proveniente do Azure Repos).
2. A API constrói um prompt estruturado e envia-o ao modelo LLaMA via Groq API.
3. O modelo devolve uma avaliação em JSON com pontuação, resumo, critérios cumpridos/falhados e sugestões.
4. A API valida, normaliza e devolve a resposta ao frontend.

Para o fluxo de avaliação de user stories (POST /evaluate/userstory), o processo é semelhante, mas o modelo avalia a qualidade da user story face aos critérios INVEST e ao formato Given/When/Then. A base de dados SQLite (gerida via SQLAlchemy) armazena as organizações e respetivas credenciais para suporte multi-tenant

Esquema da Arquitetura:



4.3 Tecnologias e Ferramentas Utilizadas

As tecnologias e ferramentas utilizadas no desenvolvimento da solução são:

- **Python 3.11**: linguagem principal de desenvolvimento do backend.
- **FastAPI**: framework web para construção da API REST, com suporte nativo a documentação automática (Swagger/OpenAPI).
- **SQLAlchemy + SQLite**: ORM e base de dados relacional para persistência das organizações e credenciais.
- **Groq API (modelo LLaMA)**: serviço de inferência de LLM utilizado para avaliação de código e user stories; os prompts são estruturados para devolver JSON normalizado.
- **Azure DevOps REST API**: integração para importação de work items do projeto ustProduction da organização ustQuality.
- **Azure Repos**: integração para importação de ficheiros de código via endpoint /items?recursionLevel=Full.
- **pytest + HTTPX**: framework de testes e cliente HTTP para testes de integração dos endpoints (suite de 114 testes, todos a passar).
- **Render**: plataforma de deployment do backend em produção (https://tfc-userstories.onrender.com).
- **Git / GitHub**: controlo de versão e repositório do projeto (https://github.com/afonsolopes22/TFC-UserStories)

4.4 Ambientes de Teste e de Produção

A solução opera em dois ambientes distintos:

Ambiente de Desenvolvimento/Teste (Local)

- Execução local com uvicorn, base de dados SQLite em memória para isolamento total entre testes (StaticPool, recriação de esquema por teste).
- Suite de 114 testes automatizados com pytest + HTTPX, cobrindo todos os endpoints e cenários de avaliação.

Ambiente de Produção (Render)

- API disponível publicamente em https://tfc-userstories.onrender.com.
- Deployment automático a partir do repositório GitHub.
- Base de dados SQLite persistente no servidor Render.

4.5 Abrangência

Este projeto aplica conhecimentos das seguintes unidades curriculares:

- **Engenharia de Software:** modelação, requisitos e gestão de user stories;
- **Desenvolvimento de Interfaces Web/ Programação Web:** criação de front-end e comunicação com API;
- **Sistemas de Suporte à Decisão:** análise de dados e relatórios de conformidade;

4.6 Componentes

Componente 1 — Motor de Avaliação por LLM (`app/services/`)) Constrói os prompts estruturados para avaliação de código e de user stories, envia-os ao modelo LLaMA via Groq API e processa a resposta JSON devolvida. Garante normalização do output (pontuação 0–100, campos obrigatórios, exatamente 3 sugestões de melhoria).

Componente 2 — API REST (`app/routers/`)) Expõe os endpoints públicos consumidos pelo frontend: `POST /evaluate/code`, `POST /evaluate/userstory`, `POST /evaluate/azure`, `POST /evaluate/stream`, `GET /azure/workitems` e CRUD de `/organizations`. Valida os inputs com Pydantic e devolve respostas estruturadas em JSON.

Componente 3 — Integração Azure (`app/integrations/`)) Módulos dedicados para comunicação com a Azure DevOps REST API (importação de work items) e Azure Repos (importação de ficheiros via `/items?recursionLevel=Full`), com suporte a múltiplas organizações e credenciais.

Componente 4 — Base de Dados (`app/models/`, `app/database.py`)) Modelo relacional em SQLite gerido por SQLAlchemy, com a tabela `organizations` para suporte multi-tenant. Garante persistência das credenciais e integridade referencial.

4.7 Interfaces

5 Testes e Validação

5.1 Estratégia de Testes

A estratégia de testes adotada assenta em testes automatizados com pytest e HTTPX, cobrindo todos os endpoints da API e os principais cenários de avaliação. O isolamento entre testes é garantido através de uma base de dados SQLite em memória (StaticPool), com recriação completa do esquema antes de cada teste, eliminando dependências entre casos de teste.

5.2 Resultados dos Testes

A suite atual é composta por **114 testes, todos a passar (114/114)**. Os testes cobrem:

- **Endpoints de avaliação de código** (POST /evaluate/code): validação de inputs, estrutura da resposta (score, summary, criteria_met, criteria_failed, improvements), e comportamento para inputs de alta, média, baixa e nula qualidade.
- **Endpoints de avaliação de user stories** (POST /evaluate/userstory): verificação da classificação devolvida (Muito bom / Bom / Satisfatório / Mau), resumo e 3 sugestões de melhoria.
- **Endpoints de integração Azure** (GET /azure/workitems, POST /evaluate/azure): validação da comunicação com Azure DevOps e Azure Repos, tratamento de credenciais inválidas e respostas de erro.
- **CRUD de organizações** (/organizations): criação, listagem, atualização e remoção de organizações, com validação de campos obrigatórios.
- **Streaming** (POST /evaluate/stream): verificação de que a resposta é devolvida como SSE (text/event-stream).

5.3 Calibração do Modelo de Avaliação

Para validar que o modelo de avaliação produz scores coerentes com a qualidade do input, foram definidos casos de teste deliberadamente calibrados em quatro níveis: alto (score esperado ≥ 80), médio (40–79), baixo (1–39) e nulo (score = 0 para código vazio ou irrelevante). Esta calibração garante que o modelo não produz avaliações uniformes independentemente da qualidade do código submetido.

6 Método e Planeamento

6.1 Planeamento inicial

O desenvolvimento do projeto segue uma abordagem Agile/Scrum, com entregas iterativas e incrementais organizadas em sprints. O backlog é gerido no Azure DevOps (organização `ustQuality`, projeto `ustProduction`), onde as user stories e tarefas são criadas e acompanhadas ao longo do desenvolvimento.

A equipa é composta pelo grupo de backend (responsável pela API e integrações) e por uma equipa de frontend dedicada, que consome a API REST desenvolvida. O Scrum Master/orientador acompanha o progresso nas revisões de sprint e valida as entregas.

O planeamento inicial previa os seguintes grandes marcos:

- **Sprint 1**: definição de requisitos, modelação da base de dados e estrutura base da API FastAPI.
- **Sprint 2**: implementação do motor de avaliação por LLM (código e user stories) e suite de testes inicial.
- **Sprint 3**: integração com Azure DevOps e Azure Repos, suporte multi-tenant e documentação da API para o frontend.
- **Sprint 4**: streaming SSE, estabilização da suite de testes (114/114) e deployment em produção no Render."

6.2 Análise Crítica ao Planeamento

O desenvolvimento decorreu de forma globalmente alinhada com o planeamento inicial, com algumas adaptações relevantes:

O que correu de acordo com o planeamento: a estrutura em camadas (API, serviços, integrações, base de dados) foi implementada conforme previsto; a suite de testes cresceu de forma contínua, atingindo 114 testes a passar; o deployment em produção no Render foi concretizado ainda antes da 2ª entrega.

O que foi ajustado face ao plano inicial: a tecnologia de validação sofreu uma mudança significativa — a ideia original de treinar um chatbot próprio foi substituída pela integração com a Groq API (modelo LLaMA), uma decisão tomada após confirmação com a entidade externa (CGI) e que se revelou mais rápida e robusta. A integração de repositórios, inicialmente prevista para GitHub/GitLab, foi direcionada para Azure DevOps e Azure Repos, em linha com o ambiente de trabalho da entidade externa. O contrato da API sofreu também ajustes a pedido da equipa de frontend (por exemplo, migração de query params para JSON body no endpoint `POST /evaluate/azure`), o que implicou coordenação ativa entre as duas equipas.

Principais riscos identificados para a fase seguinte: a dependência da Groq API para todas as avaliações cria um ponto único de falha; a estabilidade do deployment no Render (cold starts) pode afetar a experiência do utilizador final.

7 Resultados

7.1 Resultados dos Testes

7.2 Cumprimento de requisitos

8 Conclusão

8.1 Conclusão

8.2 Trabalhos Futuros

Bibliografia

[DEISI24] DEISI, Regulamento de Trabalho Final de Curso, Out. 2024.

[DEISI24b] DEISI, www.deisi.ulusofona.pt, Out. 2024.

[TaWe20] Tanenbaum,A. e Wetherall,D., *Computer Networks*, 6ª Edição, Prentice Hall, 2020.

[ULHT21] Universidade Lusófona de Humanidades e Tecnologia, www.ulusofona.pt, acedido em Out. 2024.

Anexo 1- Formulário de declaração de uso de ferramentas de Inteligência Artificial a anexar a relatório

Formulário de declaração de uso de ferramentas de Inteligência Artificial a anexar a relatório

Todos os relatórios deverão incluir anexo com cópia, devidamente preenchida, do formulário abaixo.

Assinalar as opções aplicáveis e completar os campos solicitados.

1. Utilização de IA

[ ] Não foram utilizadas ferramentas de IA na realização deste trabalho.
[X] Foram utilizadas ferramentas de IA na realização deste trabalho.

---

2. Ferramentas utilizadas

Assinalar todas as que se aplicam.

Assistência geral à escrita, análise ou ideação

[ ] ChatGPT
[ ] Microsoft Copilot
[ ] Gemini
[X] Claude
[ ] Perplexity
[ ] Outras. Quais? _____

Assistência à programação / desenvolvimento

[X] GitHub Copilot
[X] Claude
[ ] OpenAI Codex
[ ] Cursor
[ ] Tabnine
[ ] Amazon CodeWhisperer / Amazon Q
[ ] Outras. Quais? _____

Geração de imagem / design / multimédia

[ ] DALL·E
[ ] Midjourney
[ ] Stable Diffusion
[ ] Canva AI / Magic Design
[ ] Outras. Quais? _____

Outros usos

[ ] Contexto: Ferramentas? _____

3. Fases do trabalho em que foi utilizada IA

[X] Planeamento do trabalho
[X] Pesquisa exploratória / levantamento inicial de informação
[] Documentação técnica
[X] Redação do relatório
[] Desenho / modelação / arquitetura
[ ] Design / prototipagem / interface
[X] Geração de código
[X] Revisão / refatoração / debugging de código
[X] Criação de testes / casos de teste
[ ] Análise de resultados
[ ] Preparação de apresentação ou materiais auxiliares
[ ] Outros. Quais? _____

4. Tipo de utilização

Descrever sucintamente como a IA foi utilizada.

Exemplos: brainstorming, estruturação de secções, revisão linguística, sugestão de arquitetura, geração de exemplos, explicação de conceitos, geração parcial de código, correção de erros, criação de casos de teste, apoio ao design.

O Claude foi utilizado como assistente principal ao longo de todo o projeto. No desenvolvimento do backend, foi usado para gerar e rever código Python/FastAPI, sugerir arquitetura de endpoints, implementar integrações com a API do Groq e com o Azure DevOps/Repos, e criar a suite de testes automatizados com pytest. Na documentação, auxiliou na estruturação e redação dos relatórios intercalares. No planeamento, apoiou a definição da arquitetura do sistema e a elaboração de prompts de avaliação de User Stories e código com o modelo LLaMA via Groq.

5. Partes do trabalho afetadas

Indicar as secções, componentes, módulos, ficheiros, entregáveis ou atividades que foram influenciados pelo uso de IA.

Backend Python/FastAPI: todos os módulos e endpoints (app/main.py, app/routers/, app/integrations/azure_repos.py, app/integrations/groq_client.py, app/models/, app/database.py). Prompts de avaliação de código e de User Stories enviados ao modelo

LLaMA. Suite de testes pytest (tests/). Relatórios intercalares (Word). Documento de instruções para a equipa de frontend.

---

6. Exemplos de *prompt*

Inserir exemplos de *prompt*, diferenciando por âmbito (enquadrado na questão 2) e fase (enquadrado na questão 4)

Assistência à programação / Geração de código: "Implementa um endpoint FastAPI POST /evaluate/azure que receba body JSON com organization_id, repo_name e us_id, faça fetch do código no Azure Repos e devolva a avaliação do LLaMA."

Assistência à programação / Criação de testes: "Cria testes pytest para o endpoint /evaluate/userstory usando StaticPool, cobrindo: US válida, US sem critérios de aceitação, e body em falta."

Assistência geral / Documentação: "Redige a secção de Arquitetura do Sistema para o relatório intercalar, descrevendo os componentes FastAPI, Groq API, Azure DevOps e SQLite e como interagem entre si."

Assistência geral / Planeamento: "Sugere uma estrutura de prompt para que o LLaMA avalie uma User Story com critérios INVEST e Given/When/Then, devolvendo rating (Muito bom/Bom/Satisfatório/Mau), sumário e 3 sugestões de melhoria."

Assistência à programação / Debugging: "O endpoint /evaluate/azure devolve 404 ao usar /trees na API do Azure Repos. Analisa o erro e sugere uma alternativa para listar ficheiros recursivamente."

---

7. Validação, revisão e intervenção dos autores

Descrever que verificação, revisão, correção, adaptação ou reescrita foi realizada pelos autores.

Nota: se a IA tiver sido usada em código, testes, scripts, modelos, consultas, configurações ou outros artefactos técnicos, deve ser indicado de que forma os autores validaram o funcionamento e confirmaram a sua compreensão.

Todo o código gerado foi revisto, testado e integrado manualmente pelos autores. Os endpoints foram validados com a suite de 114 testes automatizados (pytest), cobrindo casos positivos e negativos. Os prompts de avaliação foram calibrados com quatro implementações

Java de qualidade distinta para verificar consistência do scoring. O conteúdo dos relatórios foi revisto pelos autores, garantindo conformidade com os requisitos académicos e com o estado real da implementação.

---

8. Grau de utilização

[ ] Residual
[ ] Moderado
[X] Extensivo

[ ] Utilização homogénea
[X] Grau de uso diferenciado por fase ou componente de trabalho

Descrever sucintamente os diferentes usos.

O grau variou por fase: na geração e revisão de código o uso foi extensivo, com o Claude a produzir rascunhos completos de módulos posteriormente adaptados e validados. Na criação de testes o uso foi igualmente extensivo. Na redação do relatório o uso foi moderado, servindo para estruturar secções e rever linguagem técnica. No design da arquitetura e nos prompts de avaliação o uso foi moderado, com forte intervenção dos autores na definição dos critérios finais.

---

9. Trabalhos em parceria

Protecção de dados confidenciais e recursos proprietários de parceiros

[X] O trabalho foi realizado em parceria com entidade externa ao DEISI

No caso da resposta anterior ser verdadeira, responder às seguintes questões:

[X] O parceiro tem regras para restringir submissão de dados
[ ] As submissões validam aplicação de regras de tratamento de dados
[X] Foram implementados mecanismos para restringir a partilha de recursos proprietários

---

10. Declaração de responsabilidade

Ao assinarem a presente declaração, os autores declaram que:

- a informação acima é verdadeira e reflete o uso efetivo de ferramentas de IA na realização do trabalho;
- compreendem que a IA não substitui autoria nem responsabilidade académica;
- verificaram a validaram e veracidade das referências bibliográficas incluídas no relatório

- assumem integralmente a responsabilidade técnica, científica, ética e académica por todo o conteúdo submetido, incluindo texto, código, modelos, testes, imagens, diagramas e restantes artefactos entregues.

---

11. Identificação dos autores

Nome(s): André Guimarães, Afonso Lopes

Número(s): a22204193, a22204030

Data: 12 / 04 / 2026

Assinatura(s): André Guimarães, Afonso Lopes

Glossário

LEI	Licenciatura em Engenharia Informática
TFC	Trabalho Final de Curso