UNIVERSIDADE
LUSÓFONA

Geração automática de testes de avaliação usando Large Language Models

Relatório Intercalar 2º Semestre

Rodrigo Pereira - a22301147

Orientador : Bruno Cipriano
Co-orientador : Rodrigo Correia

Trabalho Final de Curso | LEI | abril 2026

Direitos de cópia

Agradecimentos

Agradeço ao meu orientador, Professor Bruno Cipriano e ao meu Co-Orientador, Professor Rodrigo Correia, pela disponibilidade, orientação e sugestões ao longo do desenvolvimento deste trabalho.

Resumo

Este trabalho final de curso investiga o uso de Modelos de Linguagem de Grande Escala (LLM) para apoiar docentes na geração de mini testes de programação a partir de exemplos de treino previamente utilizados em unidades curriculares como Fundamentos de Programação, Algoritmia e Estruturas de Dados e Linguagens de Programação II. A criação manual e recorrente destes mini testes é uma tarefa demorada, que exige consistência pedagógica, diversidade de enunciados e alinhamento com mecanismos de avaliação automática.

O objetivo principal consiste na conceção e implementação de uma aplicação desktop em Java que permite ao utilizador selecionar uma pasta com exemplos de treino, onde cada exemplo inclui um ficheiro de enunciado, implementações de referência e testes unitários em Java. A aplicação lê e agrega estes artefactos, constrói um pedido em formato JSON e envia-o, via HTTP, para um LLM escolhido pelo utilizador, solicitando a geração de novas versões de exercícios.

As respostas do modelo são devolvidas também em formato JSON, contendo novos enunciados, implementações de referência e testes unitários, sendo posteriormente apresentadas ao utilizador. Prevê-se, em fases futuras, a persistência automática destes resultados em disco e a integração com plataformas externas, como repositórios GitHub e recursos multimodais.

Neste relatório intercalar descrevem-se o enquadramento, os objetivos, os requisitos, a modelação e a arquitetura da solução, bem como o protótipo atualmente implementado. Na próxima fase, pretende-se aprofundar os testes funcionais e conduzir uma avaliação pedagógica com docentes, analisando a qualidade, variedade e coerência das versões geradas para apoiar o desenvolvimento de materiais de avaliação em programação.

Palavras-chave: LLM, Geração automática de testes, ensino de programação, Java, avaliação automática, Geração Procedimental de Conteúdos

Abstract

This final degree project investigates the use of Large Language Models (LLMs) to support instructors in generating programming mini-tests from training examples previously used in courses such as Programming Fundamentals, Algorithms and Data Structures and Programming Languages II. Manually creating and updating these mini-tests is time-consuming and requires pedagogical consistency, diversity of problem statements and alignment with automatic assessment mechanisms.

The main goal is to design and implement a Java desktop application that allows the user to select a folder containing training examples, where each example includes a problem statement file (instructions.md), a reference implementation and unit tests in Java. The application reads and aggregates these artefacts, builds a JSON request and sends it, via HTTP, to an LLM chosen by the user, asking it to generate new versions of the statements and corresponding tests. The model's responses are returned in JSON and presented to the user; future iterations of the system will add persistent storage of the generated versions, integration with GitHub repositories and support for multimodal (text + image) examples.

This interim report describes the project context, objectives, requirements, modelling and system architecture, as well as the current prototype. In the next phase, the plan is to extend functional testing and carry out a pedagogical evaluation with instructors, analysing the quality, variety and coherence of the generated versions, in order to support the development of assessment materials for programming courses.

Keywords: LLM, automatic test generation, programming education, Java, automated assessment

Índice

Lista de Tabelas

Lista de Figuras

Lista de Siglas

LLM - Large Language Model (Modelo de Linguagem de Grande Escala)

AED - Algoritmia e Estruturas de Dados

FP - Fundamentos de Programação

LP II - Linguagens de Programação II

CM - Computação Móvel

HTTP - HyperText Transfer Protocol (Protocolo de comunicação usado na Web)

API - Application Programming Interface (Interface de Programação de Aplicações)

UI - User Interface (Interface de Utilizador)

IDE - Integrated Development Environment (Ambiente de Desenvolvimento Integrado)

IA - Inteligência Artificial

ODS - Objetivos de Desenvolvimento Sustentável

1 Introdução

1.1 Enquadramento

Os Modelos de Linguagem de Grande Escala (LLM) têm demonstrado capacidade para gerar e transformar texto e código, abrindo oportunidades relevantes no ensino da programação. Em contextos avaliativos, a criação recorrente de mini-testes exige tempo, consistência pedagógica e alinhamento com os objetivos de aprendizagem. Em paralelo, existe interesse em variar enunciados sem perder o nível de dificuldade e em automatizar a validação através de testes unitários.

Este projeto insere-se nesse contexto, explorando a utilização de LLMs para gerar novas versões de mini-testes a partir de exemplos fornecidos pelo docente. Em particular, pretende-se estudar até que ponto um LLM consegue produzir, de forma automática, um conjunto coerente de três elementos: enunciado (instructions), implementação de referência (reference) e testes unitários (tests), mantendo consistência entre si e respeitando restrições típicas de avaliações de programação.

1.2 Motivação e Identificação do Problema

A criação de testes práticos de programação é uma tarefa exigente: requer criatividade, consistência, controlo do nível de dificuldade e, quando existe correção automática, exige também a produção de testes unitários adequados. Na prática, é comum existir um custo elevado na produção de várias versões equivalentes do mesmo tipo de mini-teste.

O problema central que motiva este trabalho é a ausência de um método sistemático que permita gerar novas versões de mini-testes que sejam simultaneamente:

- Pedagogicamente alinhadas com o exemplo fornecido (tipo de requisitos e dificuldade);
- Variadas no tema/domínio para reduzir repetição;
- Automaticamente verificáveis, garantindo que enunciado, solução e testes coincidem.

Assim, este projeto investiga se um LLM, ao receber exemplos de treino, consegue aprender padrões e produzir variantes utilizáveis, com especial foco na coerência entre os três ficheiros gerados.

1.3 Objetivo

Este TFC tem como objetivo conceber e implementar um sistema em Java que leia um Exemplo de Treino composto por:

- instructions.md (enunciado)
- ficheiros java de referência
- ficheiros java de testes unitários

A partir de uma pasta seleccionada pelo utilizador, agregue o respetivo conteúdo e o envie para o LLM, pedindo ao mesmo que gere novas versões dos enunciados e dos testes.
O sistema permitirá escolher o modelo a utilizar e o número de versões a produzir, apresentando as respostas do LLM em formato JSON.
Em termos de investigação, o trabalho visa explorar as capacidades e limitações dos LLM na geração automática de testes de programação, nomeadamente a coerência entre enunciado e testes e a variedade entre versões, com vista ao apoio ao desenvolvimento de materiais de avaliação para o ensino da programação.

1.4 Estrutura do Documento

Este relatório está organizado da seguinte forma:

1. No Capítulo 1 apresenta-se o enquadramento, motivação, objetivos e principais conceitos do trabalho.
2. O Capítulo 2 discute a pertinência e viabilidade do projeto.
3. No Capítulo 3 são descritos os requisitos e a modelação da solução.
4. O Capítulo 4 apresenta a solução proposta, incluindo arquitetura e componentes.
5. O Capítulo 5 descreve a estratégia de testes e validação.
6. O Capítulo 6 detalha o método e o planeamento do trabalho.
7. O Capítulo 7 resume o estado atual dos resultados
8. O Capítulo 8 apresenta o Questionário de IA.
9. O Capítulo 9 apresenta as conclusões.

1.5 Definições/Conceitos

Para bem entender este relatório existe um conceito fundamental que aproveitamos para definir. É o conceito de Exemplo de Treino.

Um exemplo de Treino é Um conjunto de ficheiros composto por 3 coisas:

- um ficheiro de instruções
- ficheiros Java com a implementação de referência
- ficheiros Java com os testes unitários

A partir deste exemplo, o sistema pede ao LLM a geração de novas versões mantendo o tipo de requisitos e o nível de dificuldade, garantindo que os conteúdos gerados se interligam.

2 Pertinência e Viabilidade

2.1 Pertinência

O projeto é pertinente por combinar IA generativa com o ensino de programação, respondendo a necessidades reais de eficiência, variedade e coerência na produção de enunciados para exercícios de avaliação.
Em contexto educacional, a solução permite reduzir o esforço docente na criação de novos enunciados, aumentar a diversidade de exercícios disponibilizados aos estudantes e automatizar a avaliação através de testes unitários,favorecendo práticas de aprendizagem ativa e mitigando fenômenos de repetição ou plágio entre alunos.
Do ponto de vista científico e tecnológico, o projeto constitui uma prova de conceito para avaliar a utilidade prática de modelos de linguagem de grande dimensão (Large Language Models - LLMs) na geração de artefatos pedagógicos coerentes, nomeadamente enunciados de exercícios, implementações de referência e testes unitários, contribuindo para o debate sobre qualidade, limitações e condições de utilização destes modelos em contexto académico.
Pela sua simplicidade tecnológica (Java, Swing e comunicação via HTTP/JSON) e pelo seu foco pedagógico, o projeto apresenta-se como oportuno, exequível e com potencial impacto em unidades curriculares como Algoritmia e Estruturas de Dados (AED), Fundamentos de Programação (FP) e Linguagens de Programação II (LP II).

2.2 Viabilidade

Para enquadrar o restante documento, salientam-se dois eixos centrais que sustentam este projeto: por um lado, a pertinência pedagógica da geração automática de exercícios a partir de exemplos reais fornecidos pelos docentes; por outro, a viabilidade técnica e operacional da implementação de um protótipo em Java capaz de ler esses exemplos, construir pedidos estruturados em formato JSON e registar as respostas produzidas por um modelo de linguagem.
Com base nestes dois eixos, apresenta-se de seguida a avaliação de viabilidade do projeto.

- Alinhamento com Objectos de Desenvolvimento Sustentável (ODS) da ONU [1]

O projeto enquadra-se no Objectivo 4 - Educação de Qualidade, ao explorar o uso de tecnologias de inteligência artificial para apoiar a criação de recursos educativos e melhorar processos de avaliação no ensino da programação.

- Viabilidade técnica

Do ponto de vista técnico, é possível comunicar com modelos de linguagem através de APIs que utilizam protocolos HTTP e respostas em formato estruturado (como JSON). Isto permite integrar LLMs em aplicações desenvolvidas em linguagens como Java.
Adicionalmente, técnicas de Prompt Engineering, como One-Shot, Few-Shot ou Role Playing, permitem orientar o comportamento do modelo e produzir respostas mais adequadas ao contexto pretendido.

- Viabilidade económica

O projeto será desenvolvido em regime pro bono, não implicando custos diretos de desenvolvimento.
Não exige infraestrutura pois irá ser executado localmente na máquina dos docentes interessados, não sendo necessária infraestrutura adicional.
Relativamente aos custos associados ao uso de modelos de IA generativa, a Universidade Lusófona disponibiliza uma plataforma que permite aos professores aceder a estes serviços sem custos adicionais [2].

- Viabilidade social

O projeto é desenvolvido em contexto académico e encontra-se a ser orientado por dois docentes da área da programação, que poderão avaliar e promover a sua utilidade no contexto do ensino universitário.

2.3 Análise Comparativa com Soluções Existentes

2.3.1 Soluções existentes

Não foi identificada uma solução diretamente equivalente à proposta neste trabalho, uma vez que o formato de exemplos de treino utilizado é específico do contexto dos docentes orientadores.
Ainda assim, a literatura apresenta abordagens relacionadas que exploram o uso de modelos de linguagem de grande dimensão (LLMs) em contextos de ensino da programação e geração automática de testes.
De forma geral, os trabalhos encontrados enquadram-se em três linhas principais:

- Utilização de LLMs em contexto educativo para avaliar o desempenho dos modelos em provas ou exercícios da área da informática, permitindo analisar o seu potencial e limitações em cenários académicos [3];

- Geração automática de testes com apoio de LLMs a partir de código produzido por estudantes, com o objetivo de apoiar a avaliação automática e melhorar a deteção de erros em contexto pedagógico [4];

- Integração de LLMs em processos de desenvolvimento e validação de software, nomeadamente na geração, correção e melhoria de testes unitários em ambientes de desenvolvimento.

Em comum, estas abordagens partem normalmente de artefatos já existentes - como código, documentação ou respostas de estudantes - e utilizam os LLMs para gerar ou adaptar testes, frequentemente seguindo um ciclo de geração, execução e correção com base nos erros observados.
No entanto, nenhuma destas propostas aborda exatamente o cenário específico deste projeto, em que se pretende gerar, a partir de exemplos reais fornecidos pelo docente, um conjunto coerente de artefatos pedagógicos composto por enunciado, implementação de referência e testes unitários, no formato particular utilizado na unidade curricular.
Assim, o presente TFC procura colmatar essa lacuna, adaptando o uso de LLMs ao contexto concreto do ensino da programação e aos artefatos pedagógicos efetivamente usados pelos docentes orientadores.
Esta análise permite concluir que, embora existam trabalhos relacionados com o uso de LLMs na geração de testes e apoio ao ensino da programação, permanece em aberto a adaptação destas abordagens ao formato pedagógico específico adotado neste projeto.

2.3.2 Análise de benchmarking

Dada a inexistência de soluções diretamente comparáveis ao formato específico de exemplos de treino utilizado neste trabalho, não se apresenta uma tabela formal de benchmarking. Em alternativa, a Secção 2.3.1 sintetiza trabalhos relacionados que exploram ideias semelhantes, permitindo posicionar a presente proposta no contexto do estado da arte.

2.4 Proposta de inovação e mais-valias

Não tendo sido identificadas soluções diretamente equivalentes ao formato pedagógico utilizado pelos docentes, este projeto propõe um conjunto de inovações técnicas e mais-valias pedagógicas centradas na geração assistida de material de avaliação com apoio de modelos de linguagem.

Inovações técnicas

- Geração few-shot a partir de exemplos reais: o pedido ao LLM é construído diretamente com exemplos reais do curso, preservando o estilo, o nível de dificuldade e o vocabulário esperado.
- Geração encadeada dos artefactos pedagógicos: em vez de gerar apenas um único elemento, o sistema produz de forma sequencial e dependente o enunciado (*instructions*), a implementação de referência (*reference*) e os testes unitários (*tests*), melhorando a coerência entre os diferentes ficheiros gerados.
- Esquema JSON canónico (pedido e resposta): facilita persistência, comparação automática entre versões e futura integração com outras ferramentas (CI, dashboards, etc.).
- Parametrização do processo: escolha do modelo LLM e do nº de versões a gerar, permitindo experiências controladas (benchmark interno).
- Orquestração com validações e tratamento de respostas: fluxo robusto com tratamento de erros de I/O e HTTP, bem como mecanismos de extração do conteúdo devolvido pelo modelo, reduzindo problemas de formatação e possíveis crashes.
- Controlo adicional por parte do utilizador: a interface permite introduzir contexto extra editável, possibilitando ajustar a prompt e influenciar o comportamento do LLM de acordo com restrições pedagógicas específicas.
- Preparado para extensões: pontos de extensão para leitura de dados via GitHub e exemplos multimodais (texto+imagem), sem reescrever o núcleo.

Mais-valias pedagógicas e operacionais

- Aumento de produtividade docente: gera rapidamente variantes de mini testes (segundos/minutos), libertando tempo para feedback e tutoria.
- Diversidade e mitigação de cópia: múltiplas versões coerentes do mesmo tema diminuem a repetição entre turmas/anos.
- Manutenção de coerência: enunciado, implementação de referência e testes são gerados de forma encadeada, reduzindo o risco de desalinhamento.
- Reutilização de conhecimento: exemplos de anos anteriores passam a alimentar o gerador, capitalizando o investimento pedagógico já feito.

Diferenciadores face a trabalhos existentes

- Foco no conjunto pedagógico completo (*instructions*, *reference* e *tests*), e não apenas na geração isolada de testes unitários genéricos.

- Aderência ao contexto da UC: o sistema opera com a estrutura de pastas e convenções usadas pelos docentes, sem exigir mudanças do processo atual.

- Base para avaliação sistemática: ao fixar o formato e o pipeline, torna viável medir qualidade, cobertura e variedade das versões geradas ao longo do tempo.

2.5 Identificação de oportunidade de negócio

Sendo um projecto de índole académica, não se pretende que se torne uma oportunidade de negócio.

3. Especificação e Modelação

3.1 Análise de Requisitos

Nesta secção apresentam-se os requisitos do projeto, organizados em duas tabelas: Tabela 1 - Requisitos Funcionais (RF) e Tabela 2 - Requisitos Não Funcionais (RNF).
Cada requisito tem um ID (p.ex., RF1, RNF1), uma categoria (p.ex., User Interface, Validações, Tratamento das respostas), e uma tipificação (Obrigatório ou Opcional).

3.1.1 Enumeração de Requisitos

Tabela 1 - Requisitos funcionais

ID	Categoria	Requisito	Tipo
RF1	User Interface	O sistema deve permitir ao utilizador selecionar uma pasta contendo exemplos de treino, onde cada exemplo é composto por: - um ficheiro de instruções (.md); - um ou mais ficheiros Java de implementação de referência; - um ou mais ficheiros Java contendo testes unitários.	Obrigatório
RF2	User Interface	O sistema deve ler todos os Exemplos de Treino da pasta selecionada, reunindo o conteúdo de cada exemplo (instructions, reference e tests) de forma a construir as prompts que serão enviados ao LLM.	Obrigatório
RF3	User Interface	O sistema opcionalmente irá suportar a leitura de exemplos de treino diretamente a partir de um repositório GitHub remoto	Opcional
RF4	User Interface	O sistema irá suportar a funcionalidade de leitura de ficheiros de imagem como parte dos exemplos de treino, permitindo testar o processamento multimodal (texto e imagem)	Opcional

RF5	User Interface: validações	O sistema deve validar os inputs (pasta e número de versões) e apresentar mensagens de erro claras e acionáveis (p. ex., “pasta inválida”, “ausência de exemplos”, “erro HTTP”), evitando terminações inesperadas da execução	Obrigatório
RF6	User Interface: validações	O sistema deve validar se a pasta selecionada contém exemplos de treino válidos, verificando que cada exemplo inclui: - um ficheiro de instruções (instructions.md); - pelo menos um ficheiro Java de implementação de referência; - pelo menos um ficheiro Java de testes unitários (identificados por prefixo "Test"). O sistema deve apresentar mensagens de erro caso a estrutura esperada não seja cumprida.	Obrigatório
RF7	User Interface	O sistema deve permitir ao utilizador configurar o pedido ao modelo de linguagem, indicando o modelo (LLM) a utilizar (ex: GPT, Gemini, etc) e o número de versões a gerar	Obrigatório
RF8	Tratamento das respostas	O sistema deve guardar as respostas recebidas do LLM numa pasta do disco local, incluindo as versões geradas de testes e instruções, para análise e comparação pelo utilizador	Obrigatório

Tabela 2 - Requisitos não funcionais

ID	Categoria	Requisito	Tipo
RNF1	Envio de pedidos	O sistema deve enviar pedidos ao LLM contendo todos os exemplos de treino num formato estruturado (JSON), incluindo os campos instructions, reference e tests, de forma a permitir ao modelo compreender o papel de cada componente.	Obrigatório
RNF2	Envio de pedidos	A API Key não deve ser incluída no repositório GitHub, deve ser fornecida por variável de ambiente ou outro mecanismo seguro durante a execução	Obrigatório
RNF3	Consumo	O sistema deve monitorizar e registar o consumo por pedido (por exemplo, número de tokens das perguntas enviadas ao LLM, das respostas e total, bem como o custo estimado sempre que a API o disponibilizar).	Opcional

3.1.2 Descrição detalhada dos requisitos principais

RF1 - Selecionar pasta com Exemplos de Treino
Objetivo. Permitir ao utilizador indicar a pasta raiz que contém os exemplos (cada exemplo com instructions.md, implementação de referência e tests.java).
Dependências. Nenhuma. Pré-requisito para RF2, RF5 e RF6.
Critérios de aceitação.

- Dado que o utilizador abre o seletor de pastas, quando confirma uma pasta existente, então o caminho fica visível na UI.

- Dado que a pasta é inválida/inexistente, quando tenta confirmar, então é apresentado um erro claro e a operação é cancelada.

RF2 - Ler Exemplos de Treino da pasta selecionada
Objetivo. Percorrer diretórios e carregar os artefactos relevantes de cada exemplo.
Dependências. RF1.
Critérios de aceitação.

- Dado que a pasta contém um ficheiro instructions.md e ficheiros .java válidos, quando o sistema lê os ficheiros, então identifica automaticamente ficheiros de testes (prefixo "Test") e ficheiros de implementação de referência, listando o número de exemplos lidos na área de mensagens.

- Dado que faltam ficheiros obrigatórios num exemplo, quando a leitura ocorre, então o sistema sinaliza o exemplo como inválido e não prossegue para envio.

Notas. O formato dos diretórios segue a estrutura ilustrada na secção“Estrutura dos Diretórios”.

RF5/RF6 - Validação de inputs e de exemplos
Objetivo. Garantir robustez do fluxo antes do envio, validando que cada exemplo contém os ficheiros obrigatórios.
Dependências. RF1, RF2, RF7.
Critérios de aceitação.

- Dado que a pasta é vazia/sem exemplos válidos, quando o utilizador tenta submeter, então a UI mostra mensagem (“pasta inválida”, “ausência de exemplos”, “erro HTTP”) e não envia o pedido.

RF7 - Configurar pedido (modelo LLM e nº de versões)
Objetivo. Parametrizar o pedido ao LLM (modelo e quantidade de versões).
Dependências. RF1 (pasta), RF2 (conteúdos lidos).
Critérios de aceitação.

- Dado que o utilizador seleciona um modelo, quando confirma a submissão, então o modelo escolhido é incluído no JSON do pedido.

- Dado que o nº de versões tem de ser maior que 1 ou não numérico, quando tenta submeter, então é apresentado erro e a submissão é bloqueada.

RF8 - Guardar respostas do LLM em disco
Objetivo. Persistir as versões devolvidas (JSON) para análise posterior.
Dependências. Envio e receção válidos (RNF1).
Critérios de aceitação.

- Dado que a resposta do LLM é recebida, quando termina o processamento, então são criados ficheiros no diretório local "generated", contendo os conteúdos gerados (instructions, reference e tests).

RNF1 - Estrutura canónica do pedido/resposta (JSON)
Objetivo. Uniformizar payloads para facilitar validação e evolução.
Dependências. RF2, RF7 (pedido) e RF8 (resposta).
Critérios de aceitação.

- Pedido contém um array de exemplos com id, instructions, reference e tests; resposta contém um array de versões com id, instructions, reference e tests, conforme os exemplos ilustrativos no relatório.

Notas. No protótipo atual, cada execução gera apenas uma versão. O suporte a múltiplas versões é realizado através de execuções repetidas do processo.

RNF2 - Gestão segura da API Key
Objetivo. Evitar guardar credenciais no código/repositório.
Dependências. Ambiente de execução.
Critérios de aceitação.

- O repositório não inclui chaves; a aplicação usa variável de ambiente ou mecanismo equivalente. No protótipo, a chave pode estar vazia apenas para efeitos de compilação/demonstração.

RNF3 - Consumo
Objetivo. O sistema deve monitorizar e registar o consumo por pedido (por exemplo, número de tokens das perguntas enviadas ao LLM, das respostas e total, bem como o custo estimado sempre que a API o disponibilizar).

Notas. Esta funcionalidade não se encontra implementada no protótipo atual, sendo considerada uma evolução futura.

3.1.3 Casos de Uso/User Stories

Actores principais

- **Utilizador** - Utilizador da aplicação que seleciona exemplos de treino, configura o pedido ao LLM e analisa as versões geradas.

- **LLM** - Serviço externo acessível via API que recebe pedidos em JSON e devolve novas versões de enunciados e testes (actor externo/sistema).

User Stories

- **US1 - Selecionar exemplos de treino**
  Como utilizador, quero selecionar uma pasta com exemplos de treino para que o sistema carregue automaticamente os enunciados, implementações de referência e testes associados.

- **US2 - Configurar pedido ao LLM**
  Como utilizador, quero escolher o LLM a utilizar e o número de versões a gerar para que possa controlar o custo e a variedade das soluções geradas.

- **US3 - Enviar pedido e receber resposta**
  Como utilizador, quero enviar um pedido em JSON com todos os exemplos de treino reunidos para que o LLM devolva novas versões de enunciados e testes para cada exemplo.

- **US4 - Visualizar resultados**
  Como utilizador, quero ver no ecrã as respostas do LLM organizadas por exemplo de treino para que possa avaliar rapidamente se são utilizáveis em testes de avaliação reais.

- **US5 - Monitorizar consumo**
  Como utilizador, quero consultar o consumo por pedido (tokens e custo estimado) para que no futuro consiga perceber o impacto do uso do LLM.

3.2 Modelação

A modelação proposta organiza a informação em torno do conceito central de TrainingExample (Exemplo de Treino), que representa a unidade base de entrada para o sistema.

Cada TrainingExample agrega os diferentes artefactos necessários à geração automática de novas versões de exercícios através de um LLM, incluindo o enunciado (instructions), a implementação de referência (reference) e os testes unitários (tests).

Esta estrutura permite ao sistema preparar os dados de forma consistente antes de construir as prompts enviadas ao LLM.

Na Figura 1 apresenta-se o modelo de dados adotado.

A classe TrainingExample contém um identificador único (id), correspondente ao nome da pasta do exemplo, e o conteúdo das instruções associado ao exercício.

Cada exemplo inclui:
- o enunciado (instructions), correspondente ao conteúdo do ficheiro instructions.md;
- o código de referência, resultante da agregação do conteúdo dos ficheiros Java de implementação;
- o código de testes unitários, também agregado a partir dos ficheiros de testes.

Esta separação permite distinguir claramente o papel de cada tipo de artefacto, facilitando a sua posterior utilização na construção de prompts para o LLM.

Internamente, esta estrutura é convertida para um formato JSON antes de ser enviada ao LLM, utilizando os campos instructions, reference e tests.

Importa salientar que o LLM não tem acesso direto ao sistema de ficheiros, sendo apenas enviado o conteúdo textual dos ficheiros.

Espera-se que o LLM consiga inferir automaticamente o papel de cada componente com base nesta estrutura. Caso essa inferência não seja suficiente, poderão ser incluídas instruções adicionais na prompt para clarificar o significado de cada campo.

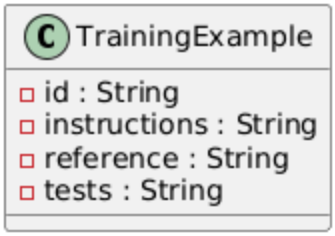

Figura 1 - Modelação

3.3 Protótipos de Interface

O protótipo de interface foi desenvolvido com o objetivo de validar o fluxo principal de utilização da aplicação e a interação com o LLM, não correspondendo ainda à versão final da interface.

A interface principal foi concebida para suportar as operações essenciais do sistema:

- seleção da pasta com exemplos de treino;
- escolha do modelo LLM;
- definição do número de versões a gerar;
- introdução de contexto adicional para instructions, reference e tests;
- submissão do pedido;
- visualização de mensagens de progresso, erro e resultados.

Dado que o sistema possui atualmente uma interface simples, centrada num único ecrã principal, a navegação aplicacional é reduzida.
Existe ainda um ecrã complementar de configuração, destinado à definição do servidor e da API Key.

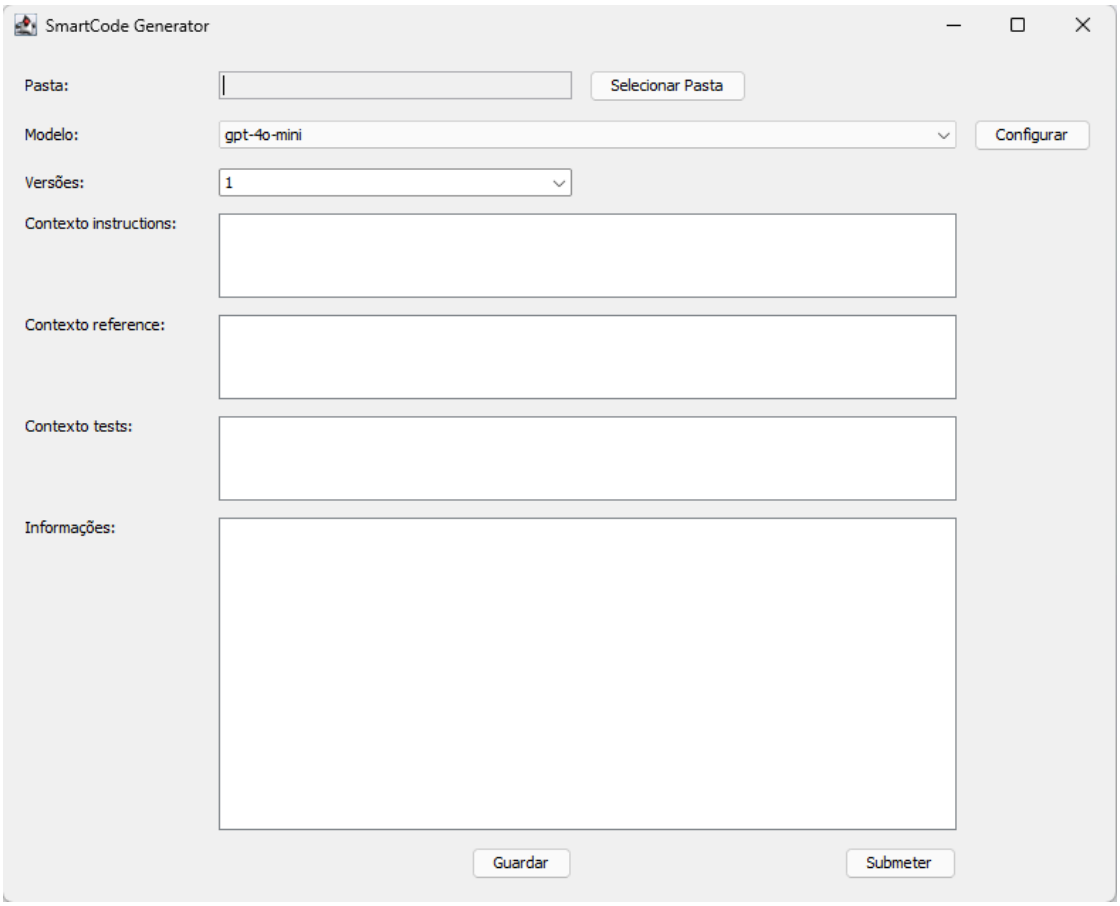

Figura 2 - Interface do protótipo desenvolvido

4. Solução Proposta

4.1 Apresentação

A solução proposta consiste numa aplicação desktop desenvolvida em Java que permite a geração automática de novas versões de exercícios de programação com recurso a modelos de linguagem (LLM).

A aplicação permite ao utilizador:

- selecionar uma pasta contendo exemplos de treino, onde cada exemplo inclui um ficheiro instructions.md, ficheiros de implementação de referência e ficheiros de testes;
- configurar o pedido ao LLM, escolhendo o modelo a utilizar e o número de versões a gerar;
- introduzir contexto adicional para as componentes instructions, reference e tests, influenciando a geração de conteúdo;
- enviar pedidos ao LLM através de uma API externa;
- visualizar mensagens de progresso, erros e resultados diretamente na interface;
- guardar os resultados gerados em ficheiros locais para análise posterior.

O sistema implementa um fluxo sequencial de geração em três etapas:

1. geração do enunciado (instructions);
2. geração da implementação de referência (reference), com base no enunciado gerado;
3. geração dos testes unitários (tests), com base no enunciado e na implementação.

Este fluxo permite garantir coerência entre os diferentes ficheiros gerados.

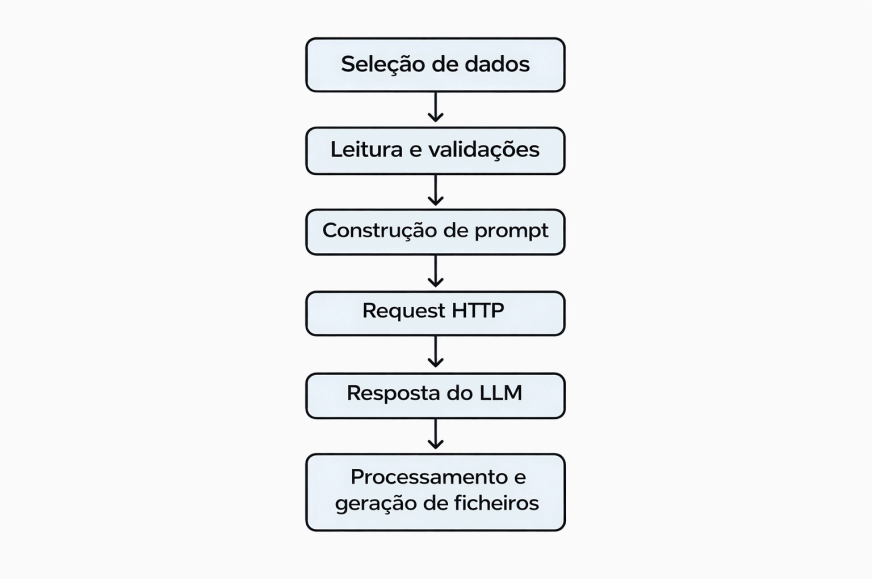

Figura 3 - Fluxo

A comunicação com o LLM é realizada através de pedidos HTTP, incluindo mecanismos de retry e tratamento de erros (ex.: timeouts).

O protótipo desenvolvido encontra-se disponível no repositório GitHub do projeto, integrado na organização da unidade curricular:

- https://github.com/DEISI-ULHT-TFC-2025-26/TFC-DEISI1234-Geracao-automatica-de-testes-de-avaliacao-usando-Large-Language-Models-ULHT2026

4.2 Arquitetura

A arquitetura do sistema segue uma abordagem baseada em cliente, em que a aplicação desktop desenvolvida em Java comunica com um serviço externo de LLM através de uma API HTTP.

No lado do cliente, a aplicação é responsável por:

- ler e validar os exemplos de treino selecionados pelo utilizador;
- estruturas internas que representam cada exemplo de treino;
- recolher as opções definidas na interface, como o modelo LLM, o número de versões a gerar e o contexto adicional para instructions, reference e tests;
- construir as prompts a enviar ao LLM;
- processar as respostas recebidas e extrair o conteúdo gerado (enunciado, implementação e testes);
- apresentar os resultados ao utilizador e permitir a sua gravação em disco.

A geração de conteúdo é realizada de forma sequencial em três etapas:

1. geração do enunciado (instructions);
2. geração da implementação de referência (reference), com base no enunciado gerado;
3. geração dos testes unitários (tests), com base no enunciado e na implementação gerados.

A aplicação comunica com um serviço externo de LLM, acedido via API, que recebe os pedidos em formato JSON e devolve as respostas geradas.

A Figura 4 apresenta, de forma resumida, a arquitetura atual do sistema.

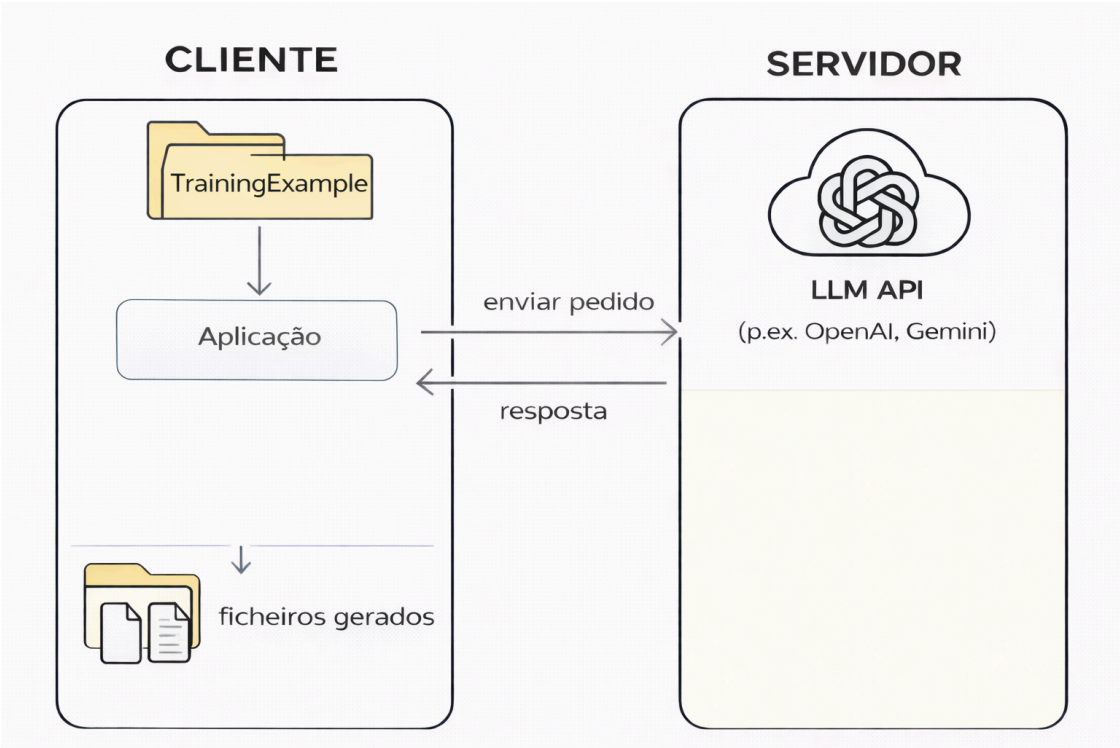

Figura 4 - Arquitetura

4.3 Estrutura dos Diretórios

Os exemplos de treino utilizados pela aplicação são organizados em diretórios independentes, onde cada pasta representa um exemplo completo.

Todos os exemplos seguem a mesma estrutura base, garantindo consistência no processamento automático pelo sistema.

Estrutura de um exemplo de treino:

```
TrainingExample/
      instructions.md
      Main.java
      TestMain.java
```

Cada exemplo de treino é considerado válido quando contém:

- Um ficheiro de instruções (instructions.md)
- Um ficheiro Java de implementação de referência (Main.java)
- Um menos um ficheiro Java de testes unitários (TestMain.java)

A distinção entre ficheiros de referência e testes é feita automaticamente com base no nome do ficheiro:

- Ficheiros cujo nome começa por "Test" são considerados testes unitários
- Os restantes ficheiros ".java" são considerados implementação de referência

Esta abordagem permite simplificar a estrutura dos diretórios e reduzir a complexidade da organização dos ficheiros, mantendo ao mesmo tempo a flexibilidade necessária para diferentes exercícios.

Após a leitura, os conteúdos são agregados em formato textual e utilizados na construção das prompts enviadas ao LLM.

4.3.1 Pedido ao LLM

Segue-se um exemplo do pedido enviado ao LLM, estruturado em formato JSON.

O sistema processa um exemplo de treino de cada vez, sendo cada exemplo correspondente a uma pasta no sistema de ficheiros.

O conteúdo dessa pasta é lido e transformado numa representação textual antes de ser enviado ao LLM.

Cada exemplo é composto por:

- id - identificador único do exemplo (ex.: "exemplo1");
- instructions - conteúdo textual do ficheiro instructions.md;
- reference - conteúdo agregado dos ficheiros Java de implementação de referência;
- tests - conteúdo agregado dos ficheiros Java de testes unitários.

Importa salientar que o LLM não tem acesso direto ao sistema de ficheiros, sendo apenas enviado o conteúdo textual dos ficheiros.

Desta forma, o modelo interpreta os diferentes blocos com base no contexto fornecido na prompt.

Assume-se que o LLM consegue inferir automaticamente o papel de cada componente com base na estrutura fornecida. No entanto, esta inferência depende da qualidade da prompt e da consistência dos exemplos de treino, podendo ser necessário reforçar instruções explícitas para garantir resultados mais previsíveis.

Exemplo de pedido:

```
{
  "id": "exemplo1",
  "instructions": "conteúdo do instructions.md",
  "reference": "conteúdo dos ficheiros .java de referência",
  "tests": "conteúdo dos ficheiros de testes"
}
```

4.3.2 Resposta do LLM

A resposta do LLM é devolvida em formato JSON e contém o conteúdo gerado para um novo exercício com base nos exemplos de treino fornecidos.

Cada resposta inclui os seguintes campos:

- id - identificador do exemplo gerado (no protótipo atual, baseado no nome do exemplo de entrada)
- instructions - novo enunciado gerado para o exercício;
- reference - implementação de referência correspondente ao novo enunciado;
- tests - testes unitários coerentes com o enunciado e com a implementação gerada.

O conteúdo de cada campo é tratado como texto simples, correspondendo ao conteúdo completo dos ficheiros gerados. O sistema não recebe estruturas de ficheiros explícitas, sendo a distinção entre enunciado, implementação e testes feita com base no contexto definido na prompt.

No protótipo atual, cada exemplo de treino é identificado pelo nome da respetiva pasta (por exemplo, "exemplo1"), sendo este valor utilizado internamente pelo sistema.
Embora o LLM possa devolver um identificador de versão (por exemplo, "v1"), este não é atualmente utilizado de forma estruturada.

O suporte a múltiplas versões encontra-se ainda em evolução.
De momento, quando o utilizador solicita várias versões, o sistema executa o processo de geração repetidamente, produzindo uma versão de cada vez, sem agregação numa única estrutura de resposta.

Exemplo de resposta:

```
{
  "id": "nome da pasta",
  "instructions": "Novo enunciado gerado",
  "reference": "Código Java gerado",
  "tests": "Testes unitários gerados"
}
```

4.4 Tecnologias e Ferramentas Utilizadas

- Linguagem de programação: Java
- Interface gráfica: Java Swing (com GridBagLayout)
- Ambiente de desenvolvimento: IntelliJ IDEA
- Integração com LLM: API HTTP de serviços de IA Generativa

4.5 Ambientes de Teste e de Produção

Esta secção descreve os requisitos mínimos para execução e teste do sistema.

- Computador com acesso à Internet, necessário para comunicação com o serviço externo de LLM;
- Chave de acesso (API Key) válida para um fornecedor de IA Generativa (p.e OpenAI ou Gemini);
- Ambiente de execução com suporte para Java.

O protótipo foi testado em ambiente local, não existindo ainda distinção formal entre ambientes de teste e produção.

4.6 Abrangência

A solução aplica conteúdos de: Fundamentos de Programação (algoritmia básica), Algoritmia e Estruturas de Dados (Java, leitura de ficheiros, etc), Linguagens de Programação II (uso de classes em Java).
Adicionalmente, relaciona-se com o desenvolvimento de interfaces gráficas (Java Swing) e com a utilização de APIs Web, incluindo formatos de comunicação como JSON.
Didaticamente, o sistema pode ser aplicado no contexto de unidades curriculares como FP/AED/LP2, permitindo a geração automática de múltiplas variações de exercícios, mantendo coerência entre enunciado, implementação de referência e testes unitários.

4.7 Componentes

A solução é composta por um componente principal (cliente) e por um serviço externo de suporte.
O sistema desenvolvido corresponde a uma aplicação cliente, responsável por toda a lógica de leitura, processamento e interação com o utilizador. Adicionalmente, comunica com um serviço externo de LLM, acessível via API, que é responsável pela geração dos conteúdos.

Componentes:
- Cliente (aplicação desktop)
- Serviço externo de LLM (via API)

4.7.1 Cliente

Seguem-se as principais características do Cliente a desenvolver:

4.7.1.1 Janela Principal

O que faz: corresponde ao ecrã onde o utilizador seleciona a pasta com os exemplos, o modelo de IA e o número de versões, e onde carrega em “Submeter”.
Como é composta:
- Pasta: campo de texto só de leitura com botão “Selecionar Pasta” (abre a janela do sistema para escolher a pasta).
- Modelo: lista de opções com os modelos disponíveis.
- Nº de versões: um campo editável onde o utilizador indica quantas versões quer.
- Área de mensagens: caixa de texto que mostra o progresso e eventuais erros.
- Campos de contexto adicional: áreas de texto opcionais que permitem ao utilizador fornecer instruções extra para a geração de instructions, reference e tests.

Regras de validação:
- A pasta não pode estar vazia.
- Tem de existir um modelo selecionado.
- O número de versões tem de ser pelo menos 1.

4.7.1.2 Leitura de Exemplos

Como funciona:

- Abre a pasta e percorre os ficheiros existentes;
- Identifica o ficheiro de instruções (instructions.md) e os ficheiros Java;
- Distingue automaticamente ficheiros de testes dos de referência com base no nome (ficheiros cujo nome começa por "Test" são considerados testes);
- Lê e agrega o conteúdo textual dos ficheiros para posterior utilização na construção dos prompts.

4.7.1.3 Processar Pedido

O que faz: administra o sistema - coordena todas as etapas e atualiza a interface.
Entradas:

- Pasta selecionada,
- Modelo escolhido,
- Número de versões a gerar,
- Área de mensagens (onde aparece o progresso).

Como funciona:

1. Valida a pasta.
2. Lê os exemplos através da função de leitura.
3. Mostra quantos ficheiros foram lidos na área de mensagens.
4. Repete o envio para o LLM o número de vezes indicado (uma vez por versão pedida).
5. Atualiza o progresso e os erros na área de mensagens, para o utilizador acompanhar.
6. Imprime a resposta completa (JSON) na consola para consulta técnica.

4.7.1.4 Envio do Pedido para o Servidor

O que faz: prepara o conteúdo a enviar e comunica com o serviço externo do LLM através de API HTTP.
Entradas:

- Os textos lidos dos ficheiros (instruções e testes).
- O modelo de IA escolhido pelo utilizador.

Como funciona:

- Constrói um pedido em JSON com o modelo e as mensagens.
- Envia esse pedido por HTTP para a API do LLM.
- Usa cabeçalhos apropriados (tipo de conteúdo e chave de acesso, quando existir).

Saídas: o texto de resposta da API (também em JSON).
O sistema inclui mecanismos de repetição (retries) em caso de falha na comunicação com a API.

4.7.2 Servidor

A componente responsável pela geração de conteúdo corresponde a um serviço externo de LLM, acessível através de API (por exemplo, OpenAI, Gemini, entre outros).
Este serviço recebe os pedidos em formato JSON e devolve respostas com os conteúdos gerados.
No âmbito deste projeto, esta componente não é desenvolvida nem controlada diretamente, sendo utilizada como serviço externo.

4.8 Interfaces

Esta secção apresenta os ecrãs mais representativos da aplicação implementada, descrevendo a sua função, a forma como foram construídos e as principais decisões tomadas ao nível da interface.

4.8.1 Janela Principal

A janela principal constitui o ecrã central da aplicação, concentrando as operações essenciais do sistema.

Através desta interface, o utilizador pode selecionar a pasta com os exemplos de treino, escolher o modelo de linguagem a utilizar, definir o número de versões a gerar e introduzir contexto adicional para as fases de geração de instructions, reference e tests. A mesma janela inclui ainda uma área de texto destinada à apresentação de mensagens de progresso, erros e resultados, bem como botões para submeter o pedido e guardar os ficheiros gerados.

A interface foi implementada em Java Swing, recorrendo a GridBagLayout para organizar os componentes. Esta opção permitiu estruturar os elementos de forma flexível e manter uma disposição simples e legível.

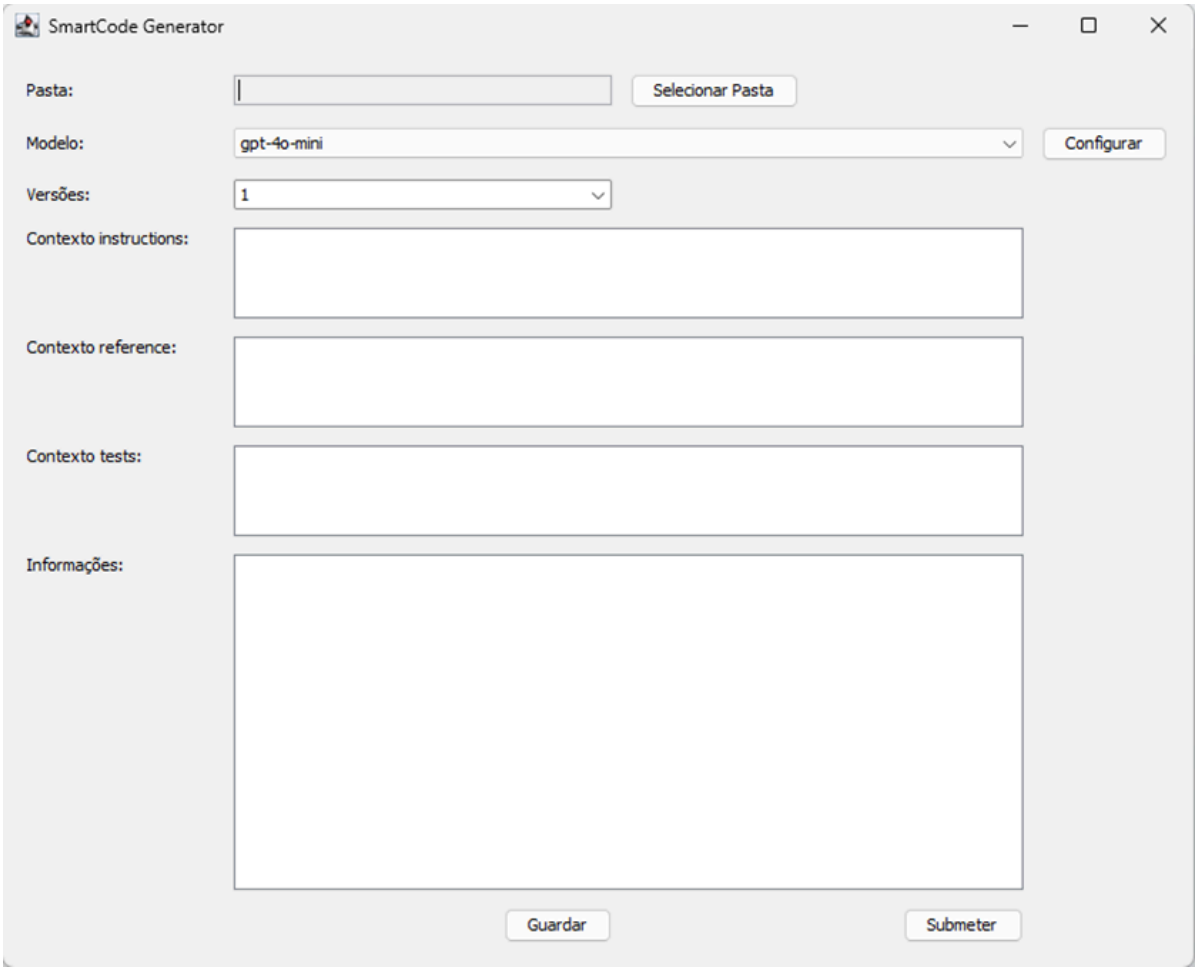

Figura 5 - Janela Principal

Os principais elementos da interface são:

- Campo “Pasta” - apresenta o caminho da pasta selecionada, sendo preenchido através do botão “Selecionar Pasta”;
- Botão “Selecionar Pasta” - abre o explorador de ficheiros para escolha da pasta com os exemplos de treino;
- Seleção de modelo - lista de modelos LLM disponíveis para utilização (ex.: gpt-4o-mini);
- Botão “Configurar” - abre a janela de configuração onde o utilizador pode definir o servidor e a API Key;
- Campo “Versões” - permite definir o número de versões a gerar;
- Campos de contexto adicional - áreas de texto destinadas à introdução de instruções específicas para instructions, reference e tests;
- Área “Informações” - apresenta mensagens de progresso, erros e resultados gerados pelo sistema;
- Botão “Submeter” - inicia o processo de geração, enviando o pedido ao LLM;
- Botão “Guardar” - permite guardar localmente os conteúdos gerados.

.

4.8.2 Janela de configuração

A janela de configuração permite ao utilizador definir os parâmetros necessários para a comunicação com o serviço externo de LLM, nomeadamente o servidor e a API Key.

Esta funcionalidade foi implementada numa janela separada da interface principal, com o objetivo de reduzir a complexidade visual do ecrã principal e isolar os parâmetros técnicos de configuração. Os valores introduzidos são guardados localmente no ficheiro config.properties, permitindo a sua reutilização em execuções futuras.

Figura 6 - Janela de Configuração

Os principais elementos da janela de configuração são:

- Campo “Servidor” – permite indicar o endpoint do serviço LLM;
- Campo “API Key” – utilizado para autenticação nos pedidos à API;
- Botão “Guardar” – guarda os parâmetros no ficheiro config.properties;
- Botão “Cancelar” – fecha a janela sem aplicar alterações.

Os valores introduzidos são persistidos localmente no ficheiro config.properties, garantindo a sua reutilização automática em execuções futuras da aplicação.

5 Testes e Validação

5.1 Abordagem de Testes

O projecto vai ser testado de duas formas:

- Testes funcionais realizados por mim;
- Testes quantitativos, realizados pelos professores, com o sentido de avaliar a utilidade da solução.

A validação da solução foi realizada através de testes funcionais e operacionais, com o objetivo de verificar o cumprimento dos requisitos definidos e avaliar a aplicabilidade da solução em contexto real.

A abordagem adotada baseia-se na execução de cenários representativos do fluxo de utilização da aplicação, incluindo situações de sucesso e de erro, de forma a garantir a robustez do sistema.

Os testes incidiram sobre:

- validação da leitura de exemplos de treino;
- construção e envio de pedidos ao LLM;
- processamento das respostas geradas;
- tratamento de erros e validação de entradas;
- persistência dos resultados gerados.

Foram utilizados exemplos reais de exercícios de programação, simulando um cenário de utilização por docentes.

5.2 Testes Realizados

Teste 1 - Geração de exercício com sucesso

Objetivo: Validar o fluxo completo de geração de conteúdos.

Cenário:
- Seleção de uma pasta válida com exemplos de treino;
- Escolha de modelo LLM;
- Definição de número de versões = 1;
- Submissão do pedido.

Resultado esperado:
- Leitura correta dos ficheiros;
- Envio do pedido ao LLM;
- Receção de resposta válida;
- Apresentação dos conteúdos gerados na interface.

Resultado obtido:
O sistema executou corretamente todas as etapas, gerando um novo enunciado, código de referência e testes unitários.

Outros cenários adicionais e capturas detalhadas encontram-se no Anexo A.

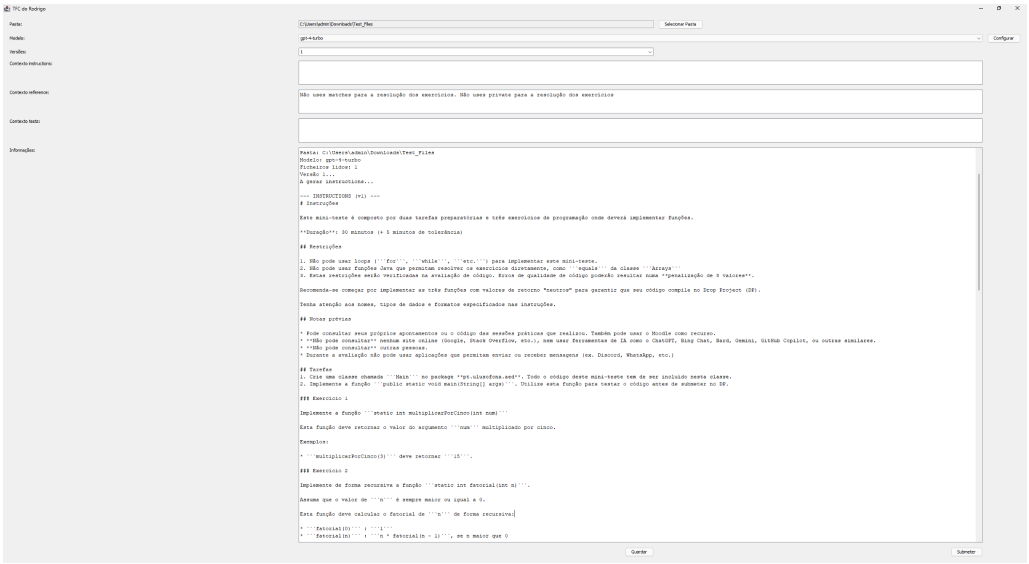

Figura 7 - Geração com Sucesso

Teste 2 - Submissão sem API Key

Objetivo: Validar a verificação de configuração obrigatória.

Cenário:
- API Key não definida;
- URL server não definido;
- Tentativa de submissão do pedido.

Resultado esperado:
- Bloqueio da operação;
- Apresentação de mensagem de erro.

Resultado obtido:
O sistema impediu a submissão e apresentou uma mensagem clara ao utilizador.

Outros cenários adicionais e capturas detalhadas encontram-se no Anexo B.

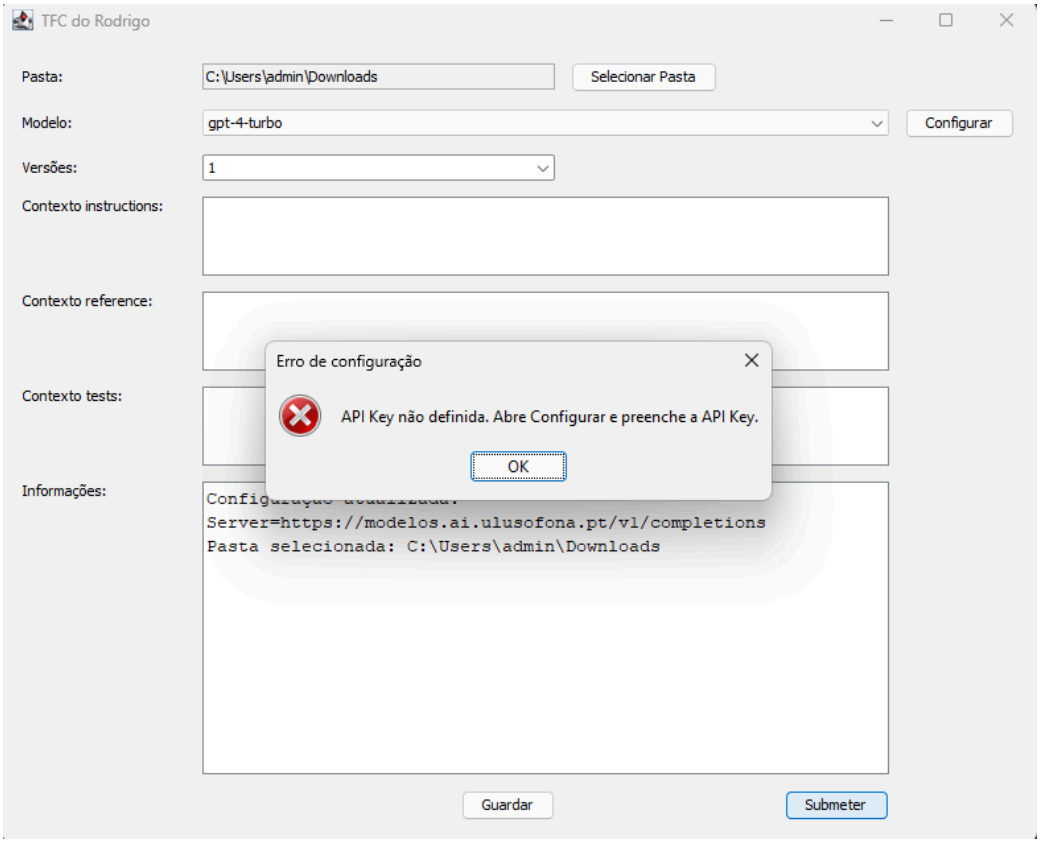

Figura 8 - Submissão sem API Key

Teste 3 - Falha na comunicação com o servidor

Objetivo: Avaliar o comportamento em caso de erro de comunicação.

Cenário:
- Pedido enviado ao LLM;
- Falha na resposta do servidor (erro HTTP).

Resultado esperado:
- Tratamento do erro;
- Apresentação de mensagem ao utilizador.

Resultado obtido:
O sistema apresentou mensagem de erro, mantendo a aplicação funcional

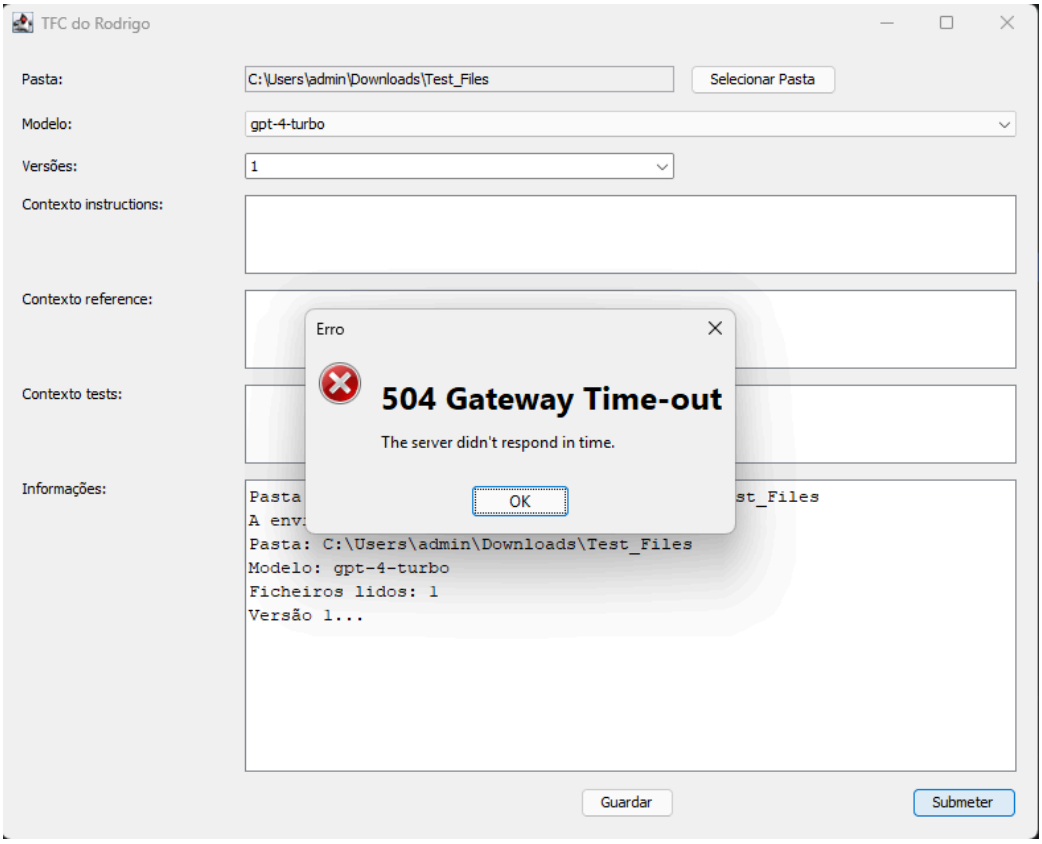

Figura 9 - Erro Gateway Time-out

Teste 4 - Submissão sem pasta selecionada ou pasta inválida

Objetivo: Avaliar o comportamento em caso de erro de comunicação.

Cenário:
- Pasta não selecionada;
- Pasta inválida (pasta sem instructions, reference e tests);
- Tentativa de submissão do pedido.

Resultado esperado:
- Bloqueio da operação;
- Apresentação de mensagem de aviso ao utilizador.

Resultado obtido:
O sistema impediu a submissão e apresentou uma mensagem indicando que é necessário selecionar uma pasta.

Outros cenários adicionais e capturas detalhadas encontram-se no Anexo C.

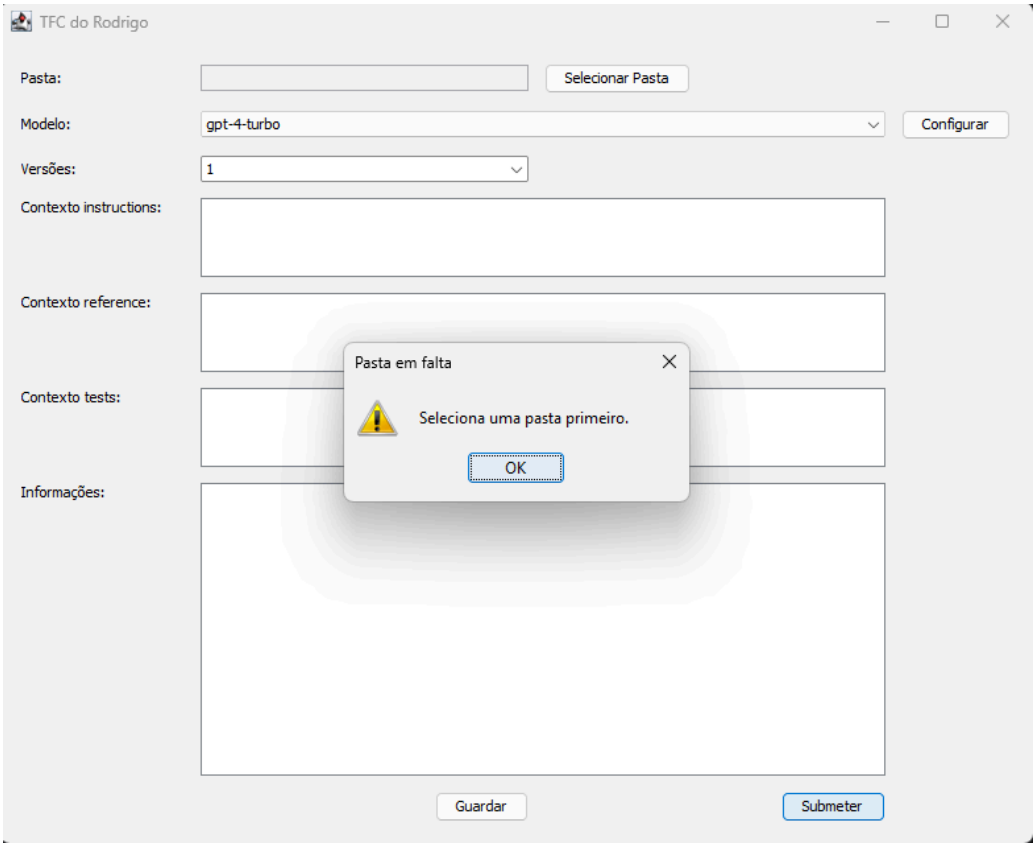

Figura 10 - Pasta vazia

5.3 Limitações e Trabalho Futuro

Apesar dos resultados positivos, foram identificadas algumas limitações:

- ausência de monitorização de consumo (tokens e custos);
- dependência de serviços externos (LLM);
- suporte limitado a múltiplas versões numa única execução.

Como trabalho futuro, prevê-se:

- implementação de métricas de consumo;
- melhoria da robustez da comunicação com o LLM;
- integração com a plataforma do GitHub;

6 Método e Planeamento

6.1 Planeamento inicial

O desenvolvimento do projeto foi organizado por fases, combinando atividades de análise, modelação, prototipagem, implementação e validação.

Numa fase inicial foi desenvolvido um protótipo com o objetivo de compreender melhor o problema, testar a viabilidade técnica da solução e apoiar a definição mais rigorosa dos requisitos. Posteriormente, o trabalho foi estruturado em fases de definição de requisitos, modelação da arquitetura, elaboração do relatório intercalar e implementação progressiva das funcionalidades obrigatórias.

O planeamento adotado segue uma abordagem faseada, adequada à natureza académica do projeto, permitindo acompanhar a evolução do trabalho desde a prova de conceito até à futura validação com docentes.

A Tabela 3 apresenta o cronograma de trabalho atualmente definido.

Tabela 3 - Método e Planeamento

Fase	Descrição	Data Início	Data Fim
1	Desenvolvimento de um protótipo para melhor compreensão dos objetivos do projeto	25/09/2025	09/10/2025
2	Definição de requisitos	23/10/2025	25/10/2025
3	Definição da arquitectura do sistema	23/10/2025	28/10/2025
4	Relatório 1	16/10/2025	27/11/2025
5	Implementação de requisitos obrigatórios	01/12/2025	Em aberto
6	Fase Experimental 1	23/01/2025	02/03/2026
7	Ajustes e melhorias	04/03/2026	Em aberto
8	Relatório 2	12/04/2026	Em aberto
9	Implementação de requisitos opcionais	Em aberto	Em aberto
10	Fase Experimental 2	Em aberto	Em aberto
11	Ajustes e melhorias	Em aberto	Em aberto
12	Relatório final	Em aberto	Em aberto

6.2 Análise Crítica ao Planeamento

Até ao momento, o planeamento inicial foi cumprido de forma global nas fases de prototipagem, definição de requisitos, modelação da solução e elaboração do relatório intercalar.
O desenvolvimento do protótipo revelou-se particularmente importante, uma vez que permitiu clarificar melhor a estrutura dos exemplos de treino, a sequência de geração de conteúdos (instructions, reference e tests) e os requisitos de validação da aplicação.

Ao longo do trabalho foram introduzidos alguns ajustamentos relativamente à ideia inicial. Em particular, verificou-se a necessidade de:

- reforçar a componente de validação de inputs e configuração;
- melhorar a definição dos requisitos funcionais e não funcionais;
- simplificar a arquitetura apresentada no relatório, de forma a torná-la mais coerente com o estado atual do protótipo;
- aprofundar a secção de testes e validação com cenários reais de sucesso e erro.

Uma das principais dificuldades encontradas esteve relacionada com a necessidade de alinhar o comportamento do LLM com a estrutura pretendida para os artefactos gerados, bem como com a necessidade de tornar o relatório coerente com a implementação efetivamente realizada.
Relativamente às fases futuras, o planeamento mantém-se em aberto em algumas tarefas, nomeadamente na implementação dos requisitos opcionais, avaliação com docentes e ajustes finais, por depender da evolução do protótipo e do feedback recolhido nas próximas entregas.

De forma geral, considera-se que o progresso do trabalho tem sido positivo, tendo já sido alcançada uma prova de conceito funcional, embora ainda existam aspetos a consolidar na geração de múltiplas versões, persistência dos resultados e avaliação pedagógica da solução.

7 Resultados

7.1 Resultados dos Testes

Nesta fase do projeto foram realizados diversos testes com o objetivo de avaliar o comportamento do sistema em cenários representativos de utilização, incluindo situações de sucesso e de erro.
Os resultados obtidos demonstram que o sistema é capaz de executar corretamente o fluxo principal de geração automática de conteúdos, nomeadamente:

- leitura de exemplos de treino;
- construção e envio de pedidos ao LLM;
- receção e processamento das respostas;
- apresentação dos conteúdos gerados ao utilizador.

No cenário de sucesso (Teste 1), o sistema gerou automaticamente novos enunciados, código de referência e testes unitários, mantendo uma estrutura consistente com os exemplos fornecidos.
Nos cenários de erro (Testes 2, 3 e 4), o sistema demonstrou robustez ao:

- bloquear operações inválidas (ex: ausência de API Key ou pasta);
- apresentar mensagens de erro claras ao utilizador;
- manter a aplicação estável mesmo perante falhas de comunicação com o servidor (erro HTTP 504).

Os resultados detalhados dos testes, bem como capturas adicionais, encontram-se disponíveis nos anexos (Anexo A, B e C).
Apesar dos resultados positivos, a avaliação da qualidade pedagógica dos conteúdos gerados ainda não foi realizada com utilizadores finais (docentes), estando prevista para uma fase futura do projeto.

7.2 Cumprimento de Requisitos

Tabela 4 - Cumprimento de Requisitos

ID	Requisito	Tipo	Estado	Justificação
RF1	Selecionar pasta com exemplos de treino	Obrigatório	Realizado	A aplicação permite selecionar uma pasta válida através da interface
RF2	Ler exemplos de treino	Obrigatório	Realizado	Os exemplos são corretamente lidos e agregados (instructions, reference, tests)
RF3	Leitura de exemplos via GitHub	Opcional	Não realizado	Funcionalidade não implementada nesta fase
RF4	Suporte a imagens	Opcional	Não realizado	Não existe ainda suporte para inputs multimodais
RF5	Validação de inputs	Obrigatório	Realizado	O sistema valida inputs e apresenta erros (ex: API Key, pasta inválida)
RF6	Validação de exemplos	Obrigatório	Realizado	O sistema verifica se os exemplos têm estrutura válida (instructions, Java, tests)
RF7	Configuração do pedido (modelo e versões)	Obrigatório	Parcialmente realizado	O modelo é aplicado corretamente, mas o suporte a múltiplas

				versões apresenta limitações
RF8	Guardar respostas do LLM	Obrigatório	Parcialmente realizado	Os resultados são apresentados e podem ser guardados, mas a persistência estruturada e organização automática ainda não estão totalmente implementadas
RNF 1	Estrutura JSON	Obrigatório	Realizado	Pedido e resposta seguem formato estruturado consistente
RNF 2	Gestão da API Key	Obrigatório	Realizado	A API Key não está no código e é validada antes do envio
RNF 3	Monitorização de consumo	Opcional	Não realizado	Funcionalidade não implementada

8 Questionário IA

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

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

Assistência à programação / desenvolvimento

[x] GitHub Copilot

[ ] 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
[x] Outras. Quais? Chat Gpt

Outros usos

[ ] Contexto: Ferramentas? _____

---

3. Fases do trabalho em que foi utilizada IA

[ ] Planeamento do trabalho
[ ] Pesquisa exploratória / levantamento inicial de informação
[ ] Documentação técnica
[ ] Redação do relatório
[x] Desenho / modelação / arquitetura
[ ] Design / prototipagem / interface
[x] Geração de código
[x] Revisão / refatoração / debugging de código
[ ] 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.

Geração parcial de código e revisão do mesmo, Geração das imagens da arquitetura do sistema,

Esclarecimento de conceitos, apoio na estruturação do relatório e revisão e melhoria de textos.

---

5. Partes do trabalho afetadas

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

A utilização de IA influenciou principalmente na estruturação de algumas secções do relatório,

Na definição da arquitetura do sistema e no apoio na implementação de algumas partes do código

---

6. Exemplos de *prompt*

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

Gera-me em uma imagem que represente a arquitetura do sistema

Tenho um trecho de código Java responsável pela leitura de ficheiros. Pretendo distinguir automaticamente entre ficheiros de testes unitários e ficheiros de implementação. Ajuda-me a refatorar o código para que ficheiros cujo nome começa por 'Test' sejam considerados testes, e os restantes sejam considerados implementação de referência.

Ajuda-me a melhorar a redação desta secção do relatório técnico, tornando-a mais clara, estruturada e adequada a um contexto académico, mantendo o conteúdo original e evitando linguagem informal.

---

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.

Todos os conteúdos gerados com apoio de IA foram revistos, validados e adaptados manualmente pelo autor.

No caso do código, foram realizados testes de execução para garantir o correto funcionamento. No caso do relatório, os textos foram ajustados para garantir coerência, clareza e adequação ao contexto do projeto.

A IA foi utilizada apenas como ferramenta de apoio, não substituindo o trabalho de análise, decisão e implementação do autor.

---

8. Grau de utilização

[ ] Residual
[x] Moderado
[ ] Extensivo

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

Descrever sucintamente os diferentes usos.

A utilização de IA foi moderada e distribuída de forma homogénea ao longo do projeto, sendo utilizada como ferramenta de apoio em várias fases, mas sem dependência total da mesma.

---

9. Trabalhos em parceria

Protecção de dados confidenciais e recursos proprietários de parceiros

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

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

[ ] O parceiro tem regras para restringir submissão de dados

[ ] As submissões validam aplicação de regras de tratamento de dados

[ ] 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): Rodrigo Pereira

Número(s): 22301147

Data: 08 / 04 / 2026

Assinatura(s): 

9 Conclusão

9.1 Conclusão

Nesta fase do projeto foi desenvolvido um protótipo funcional da aplicação, com o objetivo de validar a viabilidade técnica da solução proposta para a geração automática de exercícios de programação com recurso a Modelos de Linguagem de Grande Escala (LLM).

A aplicação implementada permite a leitura de exemplos de treino a partir de uma pasta local, a validação de inputs (pasta, modelo e número de versões), a construção de pedidos em formato JSON e o envio desses pedidos via HTTP para um LLM, bem como a receção e apresentação dos conteúdos gerados ao utilizador.

Apesar de não cobrir todos os requisitos definidos, a solução desenvolvida permite validar o fluxo principal do sistema, nomeadamente a transformação de exemplos de treino em novos enunciados, código de referência e testes unitários, demonstrando assim a exequibilidade da abordagem proposta.

Foram igualmente identificadas algumas limitações nesta fase, nomeadamente:

- ausência de persistência estruturada dos resultados gerados;
- suporte limitado à geração de múltiplas versões numa única execução;
- inexistência de integração com repositórios remotos (ex.: GitHub);
- ausência de suporte a inputs multimodais;

Como trabalho futuro, prevê-se:

- estabilizar e evoluir o esquema de pedido/resposta em JSON;
- implementar mecanismos de persistência e organização dos resultados;
- integrar fontes externas de dados e suporte multimodal;
- melhorar a parametrização do modelo utilizado;
- realizar uma avaliação com docentes, de forma a validar a qualidade, coerência e utilidade dos exercícios gerados.

De forma geral, considera-se que o trabalho desenvolvido constitui uma base sólida para a evolução do sistema, evidenciando o potencial dos LLM no apoio à criação de materiais de avaliação em contexto de ensino de programação.

Bibliografia

[1] Wikipédia. Objetivos de Desenvolvimento Sustentável. Disponível em: https://pt.wikipedia.org/wiki/Objetivos_de_Desenvolvimento_Sustent%C3%A1vel. Acesso em: 20 nov. 2025.

[2] LiteLLM API - Swagger UI. Modelos.ai - Ulusofona. Disponível em: https://modelos.ai.ulusofona.pt/. Acesso em: 20 nov. 2025.

[3] IEEE Xplore - LLM Performance Assessment in Computer Science Graduate Entrance Exams. Disponível em: https://ieeexplore.ieee.org/document/10843484. Acesso em: 25 nov. 2025

[4] arXiv.org e-Print archive - Using Large Language Models for Student-Code Guided Test Case Generation in Computer Science Education. Disponível em: https://arxiv.org/pdf/2402.07081. Acesso em: 25 nov. 2025

[5] ACM Digital Library - ChatUniTest: A Framework for LLM-Based Test Generation. Disponível em: https://dl.acm.org/doi/abs/10.1145/3663529.3663801. Acesso em: 25 nov. 2025

Anexo A - Resultados complementares do Teste 1

Este anexo apresenta capturas complementares do teste de geração com sucesso, evidenciando o conteúdo produzido nas fases de implementação de referência e de testes unitários

.

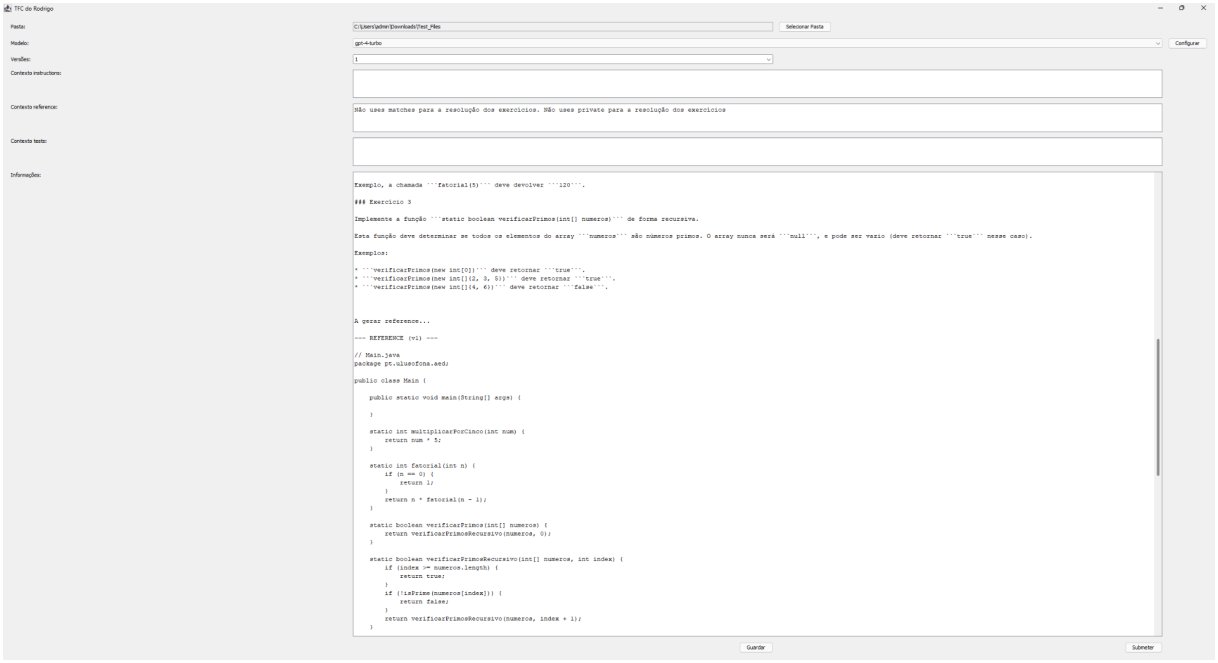

Figura A.1 - Conteúdo gerado para a implementação de referência

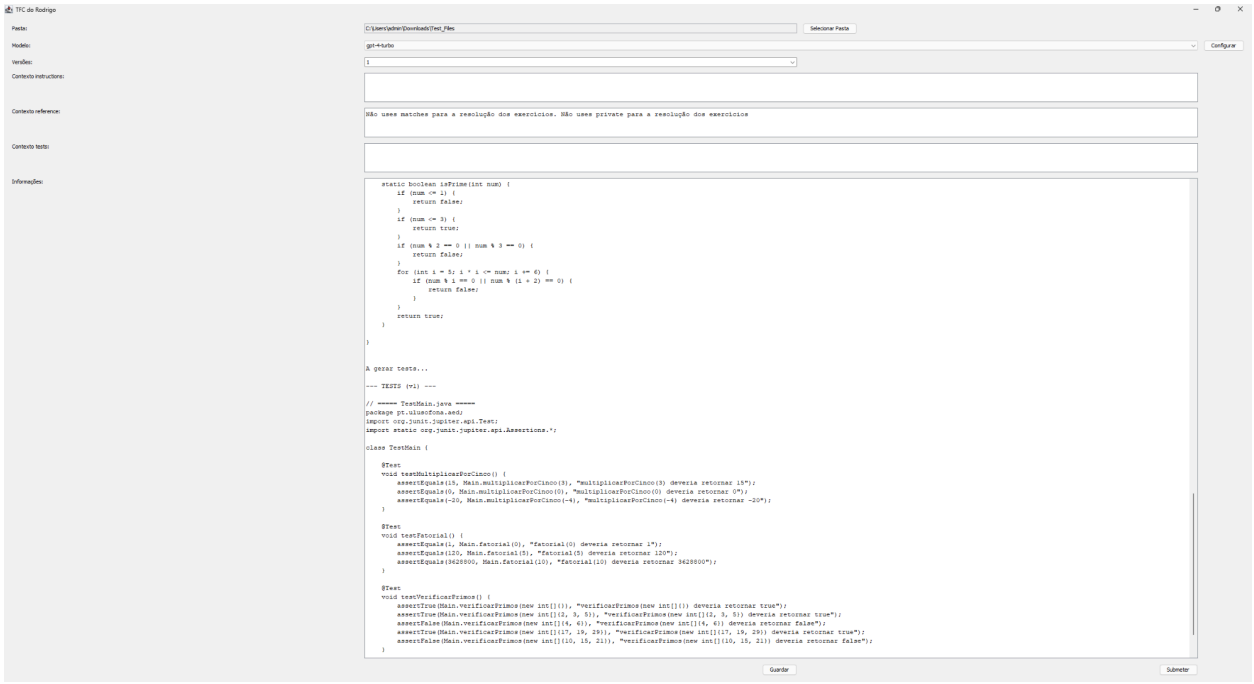

Figura A.2 - Conteúdo gerado para os testes unitários

Anexo B - Validações de configuração

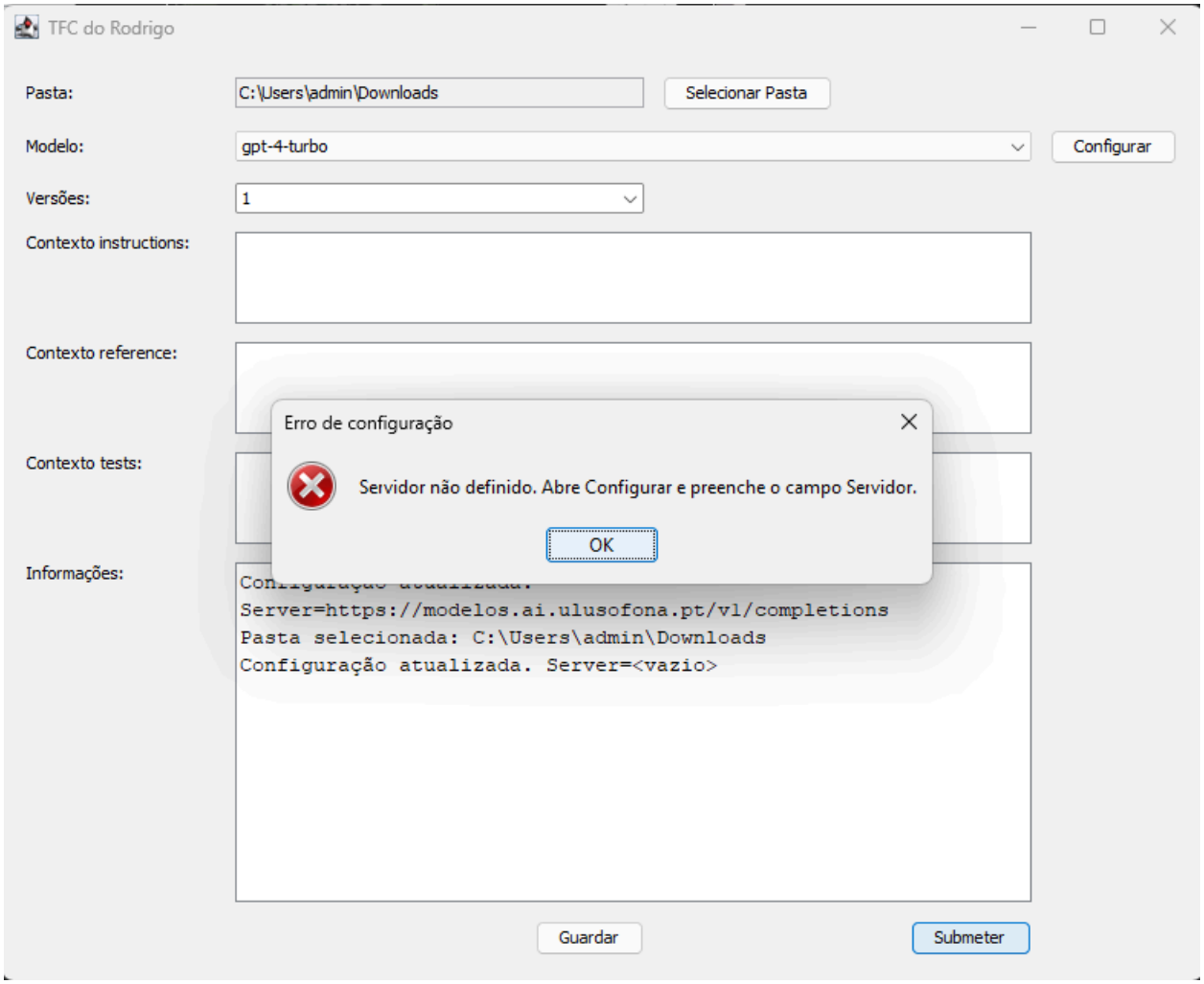

Figura B.1 - Submissão sem servidor definido

Anexo C - Validação de pasta de entrada

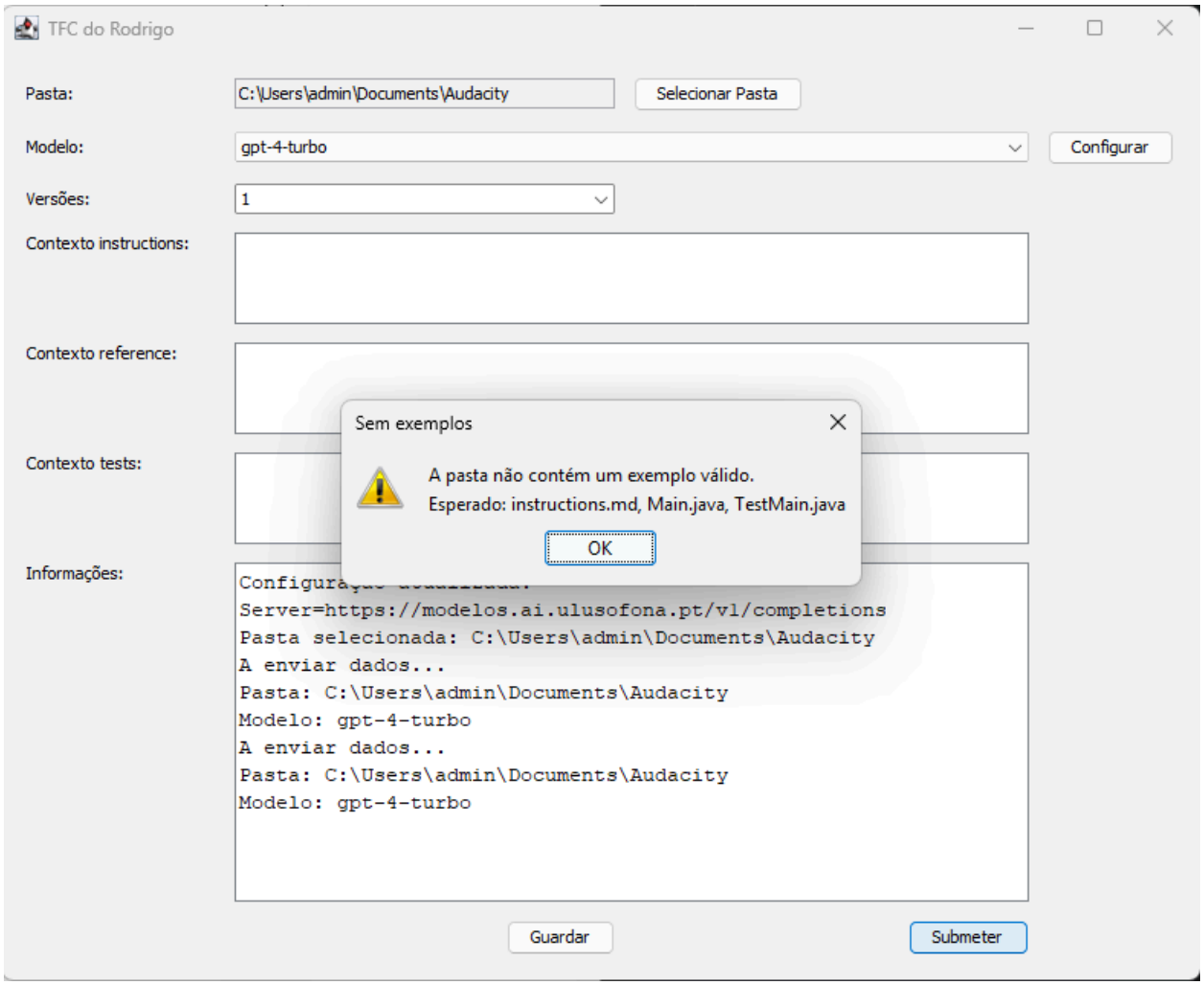

Figura C.1 – Pasta sem exemplo válido

Glossário

LEI - Licenciatura em Engenharia Informática

TFC - Trabalho Final de Curso

Exemplo de Treino - Conjunto de ficheiros composto por instructions.md, implementação de referência em Java e testes unitários, utilizado como entrada para o LLM com o objetivo de gerar novos exercícios

LLM (Large Language Model) - Modelo de Inteligência Artificial treinado com grandes volumes de dados, capaz de compreender e gerar texto e código com base em instruções fornecidas

Implementação de referência - Solução considerada correta para um determinado exercício, utilizada como base para validação e geração de novos conteúdos

Teste unitário - Teste automático que verifica o comportamento de uma função ou método de forma isolada, garantindo a sua correção

Avaliação automática - Processo de correção de soluções de programação através da execução de testes automatizados, sem intervenção manual

JSON (JavaScript Object Notation) - Formato estruturado de representação de dados em texto, amplamente utilizado na comunicação entre sistemas e APIs

GitHub - Plataforma online para alojamento e gestão de código, baseada no sistema de controlo de versões Git

OpenAI - Organização responsável pelo desenvolvimento de modelos de Inteligência Artificial utilizados neste trabalho através de API

Multimodalidade - Capacidade de um sistema processar e integrar diferentes tipos de dados, como texto e imagem

GridBagLayout - Gestor de layout da biblioteca Swing (Java) que permite organizar componentes numa grelha flexível na interface gráfica

API Key - Chave secreta utilizada para autenticar pedidos a uma API, identificando o acesso ao serviço de IA generativa, não devendo ser armazenada em repositórios públicos

UC - Unidade Curricular

Prompt - Texto ou conjunto de instruções enviado ao LLM que orienta a geração de conteúdo

Prompt Engineering - Técnica de conceção e ajuste de prompts com o objetivo de melhorar a qualidade e previsibilidade das respostas geradas pelo LLM

Test Case - Cenário de teste específico com condições, ações e resultados esperados, utilizado para validar o comportamento do sistema.

Validação de inputs - Processo de verificação dos dados introduzidos pelo utilizador, garantindo que cumprem os requisitos esperados antes de serem processados.