deti
universidade de aveiro
departamento de eletrónica, telecomunicações e informática

TAÇA UA

Relatório Técnico

Autores:
Bernardo Borges 103592
Alexandre Regalado 124574
Mateus Rocha 122949
Diego Aguilar 117122
Gonçalo Simões 119412

Orientadores:
Osvaldo Pacheco
Carlos Costa

Coordenador:
João Luís

Abstract

Taça UA represents one of the most significant academic sporting events at the University of Aveiro, involving extensive organizational coordination and community engagement. Ensuring efficient management is essential to sustaining its operational quality and institutional relevance.

Despite its scale, the competition is currently managed through fragmented tools and informal communication channels, resulting in inconsistent data handling, scheduling challenges, and limited transparency. No integrated digital platform exists that addresses the specific administrative and public needs of the event.

This project aims to conceptualize and specify a unified, web-based management system capable of centralizing operational workflows, improving information reliability, and supporting the diverse requirements of administrators, athletes and the general public.

The study employs requirement elicitation, comparative benchmarking of existing sports-management platforms, persona-based modelling and the development of functional specifications, non-functional requirements, user stories, and use-case diagrams to establish a rigid initial structure.

Analysis indicates a strong need for automation in specific areas, standardized data governance and real-time dissemination of the schedules, results and classifications. The system architecture derived from the study organizes responsibilities into three role-based profiles and modules covering tournaments, teams, scoring, scheduling and regulation management.

The proposed solution significantly reduces organizational overhead, enhances the accuracy and timeliness of competition data, and strengthens transparency for all users. It provides a scalable digital foundation for the modernization and long-term evolution of academic sports events of the University of Aveiro.

Abstract 1
1. Introdução 4
1.1. Contexto 5
1.2. Objetivos 5
1.3. Estrutura do documento 6
2. Estado de Arte 7
2.1. Análise Comparativa 8
2.2. Análise das Soluções Existentes 9
2.3. Síntese Crítica 10
3. Metodologia 10
3.1. Ferramentas de Comunicação 11
3.2. Gestão de tarefas em Jira 11
3.3. Equipa 12
4. Requisitos do Sistema 13
4.1. Levantamento de requisitos 14
4.2. Requisitos funcionais 14
4.2.1. Gestão de Utilizadores e Autenticação 14
4.2.2. Gestão da Competição 14
4.2.3. Gestão de Modalidades, Torneios e Jogos 15
4.2.4. Gestão de Equipas e Atletas por Curso 15
4.2.5. Gestão de Jogos ao Nível do Curso 15
4.2.6. Transparência, Consulta Pública e Comunicação 16
4.2.7. Sistema de Pontuação e Classificação Geral 16
4.3. Requisitos não funcionais 16
4.3.1. Usabilidade 17
4.3.2. Desempenho 17
4.3.3. Compatibilidade 17
4.3.4. Manutenibilidade e Evolução 18
4.4. Atores e Personas 18
4.4.1. Atores 18
4.4.2. Personas 18
4.4.3. Casos de Uso 20

5. Arquitetura 24
5.1. Camada de Apresentação 25
5.2. Camada de Autenticação e Autorização 26
5.3. Camada de APIs e Orquestração 26
5.4. Serviços de Domínio 27
5.5. Comunicação Assíncrona e Processamento de Eventos 27
5.6. Persistência e Gestão de Dados 27
5.7. Observabilidade e Monitorização 28
6. Implementação 28
6.1. Gestão dos repositórios em GitHub 29
6.2. Frontend 29
6.2.1. Tecnologias Usadas 30
6.2.2. Estrutura no Repositório 30
6.2.3. Interfaces 31
6.2.3.1. Interface Publica 31
6.2.3.2. Painel do Administrador 31
6.3. Backend 32
6.3.1. Tecnologias Usadas 33
6.3.2. Serviços implementados 33
6.3.3. Estrutura no Repositório 34
6.3.3.1. APIs Centrais 35
6.3.3.2. Micro-serviços 35
6.3.3.3. Componentes Partilhados 36
6.3.3.4. Ferramentas Auxiliares 37
6.4. DevOps & CI/CD 37
6.4.1. Docker 37
6.4.2. Continuous Integration 38
6.4.3. Continuous Deployment 39
6.5. Modelo de Dados 39
6.6. Segurança 40
6.6.1. Autenticação e Controlo de Acesso 41
6.6.2. Segurança na Camada de Transporte 42

6.7. Problemas Encontrados 44
6.7.1. Integração com Microserviços 44
6.7.2. Falta de Dados Reais 44
6.7.3. Problemas de Desempenho 45
6.7.4. Configuração do NGINX em Ambiente de Staging 45
6.7.5 Atraso na Disponibilização da VM de Staging 46
7. Documentação da API 46
7.1. Ferramentas e Abordagem 47
7.2. Conteúdo da Documentação 47
8. Testes e Resultados 48
8.1. Testes de Usabilidade 49
8.1.1. Amostra 49
8.1.2. Método 49
8.1.3. Resultados 50
8.2. Testes de Integração 52
8.3. Testes de Desempenho 53
8.3.1. Resultados Obtidos 54
9. Conclusão 56
9.1. Resultados Obtidos 57
9.2. Trabalho Futuro 57
10. Referencias 58
Apêndice A 59
Apêndice B 63
Apêndice C 63
Apêndice D 63

Lista de Abreviaturas

Sigla	Significado
AAUAv	Associação Académica da Universidade de Aveiro
API	Application Programming Interface
CD	Continuous Deployment
CI	Continuous Integration
CSS	Cascading Style Sheets
FADU	Federação Académica do Desporto Universitário
HTTP	Hypertext Transfer Protocol
JWT	JSON Web Token
MVC	Model–View–Controller
NGINX	Engine X (servidor web e reverse proxy)
OIDC	OpenID Connect
PDF	Portable Document Format
RBAC	Role-Based Access Control
REST	Representational State Transfer
SLO	Service Layer Objective
SOLID	Conjunto de princípios de design orientado a objetos
SUS	System Usability Scale
TLS	Transport Layer Security
UA	Universidade de Aveiro
UML	Unified Modeling Language
VU	Virtual User

1. Introdução

O presente relatório técnico documenta o trabalho realizado pelo grupo 15 no âmbito da unidade curricular de Projeto em Engenharia Informática (43675) da Licenciatura em Engenharia Informática, durante o ano letivo de 2025/2026. O projeto foi realizado sobre a orientação dos Prof. Dr. Carlos Costa, Prof. Dr. Osvaldo Pacheco e João Luís.

1.1. Contexto

A gestão da taça UA constitui atualmente um processo complexo, fragmentado e fortemente dependente de comunicação informal. A ausência de um sistema unificado leva à utilização simultânea de múltiplas ferramentas, originando dificuldades significativas na coordenação entre organizadores, núcleos e participantes.

No que diz a respeito à organização das modalidades e dos torneios, identificam-se várias limitações práticas. A calendarização dos jogos, frequentemente construída manualmente, é propensa a erros e exige constantes atualizações distribuídas por diferentes canais. A gestão de equipas e jogadores revela-se igualmente morosa, com o registo e validação de atletas, a associação de equipas por núcleo e a verificação do estatuto de sócio dependentes de processos repetitivos e não integrados. A publicação de regulamentos carece de um repositório central, resultando a inconsistências e atualizações demoradas.

A experiência dos administradores de núcleo enfrenta desafios paralelos. A submissão de jogadores elegíveis para cada jogo e o preenchimento da ficha de jogo envolvem tarefas redundantes e frequentemente duplicadas. A inexistência de mecanismos automáticos para validação de informação, como o estatuto de sócio, contribui ainda para ambiguidades e atrasos operacionais.

De ponto de vista do publico, a consulta de resultados, calendários e classificações encontra-se distribuída por várias páginas e publicações informais, dificultando uma perceção clara do estado da competição. A falta de transparência e centralização afeta a experiência global da comunidade académica, reduzindo o impacto e visibilidade do evento.

É neste cenário de ineficiências acumuladas que se insere o presente projeto. A proposta visa no desenvolvimento de uma plataforma web integrada que centralize a gestão de modalidades, torneios, equipas e resultados, garantindo atualizações em tempo real, uniformização da informação e uma experiência mais fluida para administradores, estudantes e público.

1.2. Objetivos

O desenvolvimento da plataforma digital para a Taça UA tem como propósito principal modernizar, unificar e otimizar as operações essenciais à gestão da competição, garantindo melhor eficiência, transparência e coerência organizacional. Assim, os objetivos do sistema são:

- Centralizar a gestão da Taça UA numa única plataforma web, eliminando a necessidade de ferramentas dispersas e comunicação manual.
- Apoiar a organização desportiva através de módulos dedicados, incluindo gestão de modalidades, torneios, equipas, jogos e classificações.
- Automatizar o cálculo de pontuações e classificações, reduzindo erros manuais e assegurando atualizações consistentes em tempo real.
- Disponibilizar informação publica atualizada, como calendários, resultados, classificações gerais e regulamentos, promovendo uma maior transparência para participantes e jogadores
- Facilitar a operação dos Administradores de Núcleo através de funcionalidades como a criação de equipas, gestão de jogadores e geração de fichas de jogo.

Estes objetivos refletem a necessidade de transformar a gestão da Taça UA num processo mais estruturado, eficiente e sustentável, alinhando com os requisitos modernos de informação, coordenação e fiabilidade num evento desportivo académico.

1.3. Estrutura do documento

Este relatório encontra-se organizado de forma a apresentar, de maneira clara e progressiva, o desenvolvimento do sistema de gestão da Taça UA, desde o enquadramento teórico e metodológico até à implementação e análise dos resultados obtidos. A estrutura do documento é a seguinte:

1. Estado da Arte: Apresenta o enquadramento do problema e uma análise comparativa de soluções existentes no domínio da gestão de competições e eventos desportivos académicos, identificando limitações e oportunidades que fundamentam a proposta do sistema desenvolvido.

2. Metodologia: Descreve a abordagem metodológica adotada ao longo do projeto, incluindo as ferramentas de comunicação, organização da equipa e gestão de tarefas, bem como as práticas utilizadas para garantir coordenação e acompanhamento do desenvolvimento.

3. Requisitos do Sistema: Define os requisitos do sistema com base no levantamento efetuado junto dos intervenientes, incluindo a identificação de atores, personas e casos de uso, bem como a especificação dos requisitos funcionais e não funcionais que orientaram o desenvolvimento da solução.

4. Arquitetura: Apresenta a arquitetura geral da solução, descrevendo a separação entre frontend, backend e serviços de suporte, bem como as principais decisões arquiteturais adotadas para garantir escalabilidade, segurança e manutenibilidade.

5. Implementação: Detalha o processo de implementação do sistema, abordando a gestão de repositórios, tecnologias utilizadas no frontend e backend, práticas de DevOps e CI/CD, bem como o modelo de dados adotado.

6. Documentação da API: Descreve a interface de programação disponibilizada pelo sistema, incluindo os principais endpoints, formatos de dados e mecanismos de autenticação, servindo de suporte à integração e manutenção futura.

7. Conclusão: Sintetiza os principais contributos do projeto, avaliando o cumprimento dos objetivos propostos e apontando direções para trabalho futuro.

8. Referências: Lista das fontes bibliográficas, documentação técnica e outros recursos utilizados ao longo do projeto.

2. Estado de Arte

A organização e gestão de competições desportivas académicas têm vindo a beneficiar de soluções digitais que procuram centralizar informação, automatizar processos e melhorar a comunicação entre entidades organizadoras, participantes e público. Em contexto nacional, a Federação Académica do Desporto Universitário (FADU) disponibiliza plataformas que suportam a gestão de competições universitárias a nível interinstitucional, servindo como referência funcional para eventos desportivos académicos de grande escala.

Contudo, a Taça UA apresenta características organizacionais específicas, nomeadamente a gestão por cursos, a existência de múltiplas modalidades em simultâneo e a forte dependência de administradores locais (Núcleos), que não são totalmente contempladas pelas soluções existentes. Atualmente, a gestão da Taça UA assenta num conjunto disperso de ferramentas, como folhas de cálculo, formulários online e comunicação informal, resultando em processos manuais, redundância de informação e maior probabilidade de erro.

Neste contexto, torna-se pertinente analisar o estado da arte das soluções existentes, identificando as suas funcionalidades, limitações e o posicionamento do projeto proposto face às práticas atuais.

2.1. Análise Comparativa

A tabela abaixo resume as funcionalidades disponíveis em cada aplicação:

Critério	FADU	Taça UA (Atual)	Taça UA (Projeto)
Gestão de torneios / modalidades	SIM	PARCIAL (Excel + Google Forms)	SIM
Gestão por Núcleos	NÃO	MANUAL (Via Responsável)	SIM
Hierarquia Administrativa	NÃO	GESTAO INFORMAL	SIM
Agendamento de Jogos	SIM	MANUAL	SIM
Registo de resultados	SIM	MANUAL (Redes Sociais)	SIM
Pontuação automática por modalidade	NÃO	EXCEL	SIM
Classificação geral intermodalidades	NÃO	EXCEL	SIM
Inscrição de atletas	SIM	FORMS	SIM
Validação de sócios	NÃO	MANUAL	MANUAL
Fichas de Jogo (PDF)	NÃO	CRIADAS MANUALMENTE	FORA DA APLICAÇÃO
Histórico multi-época	SIM	EXCEL	SIM
Calendário público com filtros	SIM	NÃO	SIM

Tabela 2.1: Comparação de funcionalidades entre FADU, a solução atual da Taça UA e do projeto da Taça UA

2.2. Análise das Soluções Existentes

A plataforma da FADU constitui uma referência no contexto do desporto universitário nacional, oferecendo suporte à gestão de modalidades, torneios, inscrições de atletas e resultados. Destaca-se pela disponibilização de calendários públicos e históricos multi-época. No entanto, trata-se de uma solução orientada para competições interuniversitárias, não contemplando a gestão interna por núcleos, nem uma hierarquia administrativa adaptada à realidade organizacional da Taça UA.

A versão atual da Taça UA recorre a um conjunto de ferramentas heterogéneas, incluindo folhas de cálculo, formulários online e redes sociais. Embora estas permitam assegurar o funcionamento básico da competição, grande parte dos processos, como o agendamento de jogos, o registo de resultados, o cálculo de pontuações e a gestão de equipas, é realizada manualmente. Esta abordagem dificulta a consistência dos dados, aumenta a carga administrativa e compromete a transparência e atualização em tempo real da informação disponibilizada ao público.

O projeto da Taça UA propõe uma solução integrada e centralizada, concebida especificamente para responder às necessidades organizacionais da competição. A plataforma suporta uma hierarquia administrativa clara, com distinção entre Administrador Geral e Administradores de Núcleo, automatiza o cálculo de pontuações e classificações, centraliza a gestão de equipas e jogos, e disponibiliza informação pública filtrável e atualizada em tempo real. Embora algumas funcionalidades, como a validação de sócios e a geração final das fichas de jogo em PDF, ainda dependam de processos externos ou complementares, a solução representa um avanço significativo face às práticas atuais.

2.3. Síntese Crítica

A análise comparativa evidencia que, apesar da existência de plataformas consolidadas como a da FADU, estas não respondem de forma plena às especificidades da Taça UA. Por outro lado, a abordagem atualmente utilizada pela organização carece de integração, automatização e escalabilidade. O projeto proposto posiciona-se, assim, como uma solução intermédia e especializada, combinando boas práticas existentes com funcionalidades adaptadas ao contexto académico local, contribuindo para uma gestão mais eficiente, transparente e digitalmente sustentável da Taça UA.

3. Metodologia

Ao longo do desenvolvimento da plataforma de gestão da Taça UA, optou-se por uma abordagem incremental e colaborativa. O trabalho foi estruturado em diferentes fases, desde a definição e análise dos requisitos até à sua correta implementação, garantindo a coerência entre os objetivos definidos e as soluções propostas.

Em relação à coordenação e comunicação envolvidas no projeto, foram utilizadas principalmente duas ferramentas digitais: o Discord, como meio de comunicação, e o Jira para a gestão de tarefas. Os aspetos referidos serão aprofundados ao longo das secções seguintes.

3.1. Ferramentas de Comunicação

A comunicação entre os membros da equipa, o cliente e os orientadores revelou-se fundamental ao longo das diversas etapas de desenvolvimento do projeto. Esta ocorreu tanto de forma presencial como virtual, assegurando um acompanhamento contínuo e eficaz.

Conforme referido anteriormente, a ferramenta mais utilizada foi o Discord, que possibilitou a troca de informação e de recursos relevantes, a discussão de decisões e o acompanhamento constante do progresso do projeto. Foram criados dois canais distintos: um que incluía o cliente, destinado à recolha de requisitos, partilha de progressos e esclarecimento de dúvidas; e outro restrito aos membros da equipa, mais focado na organização e planeamento das tarefas. Adicionalmente, através desta plataforma, foram realizadas reuniões remotas entre os membros da equipa, recorrendo a chamadas de voz e à partilha de ecrã, o que facilitou a análise conjunta de documentos e código.

No que respeita à comunicação com os orientadores, recorreu-se à plataforma Zoom para a realização de reuniões virtuais, complementadas por diversos encontros presenciais. Cada uma destas reuniões foi fundamental para o progresso do projeto, permitindo esclarecer dúvidas relacionadas com a estrutura do trabalho, a definição de prioridades e os aspetos a melhorar.

Tudo o que foi descrito foi transcendental no processo de validação das decisões e no alinhamento do projeto com os objetivos e requisitos previamente definidos.

3.2. Gestão de tarefas em Jira

As diversas tarefas em cada uma das etapas do projeto foram geridas com a ajuda do Jira, permitindo organizar o projeto de forma ordenada e acompanhar de forma clara a sua evolução.

Cada um dos diferentes aspetos a desenvolver foi registado na plataforma, originando um quadro do tipo *Kanban*, organizado em colunas que representavam os diferentes estados das tarefas (to do, in progress, in review, in testing, done). Esta estrutura facilitou a visualização do estado global do projeto, bem como a identificação atempada de obstáculos ou atrasos que pudessem surgir ao longo do seu desenvolvimento. Cada uma das tarefas criadas foi atribuída a um membro da equipa, de acordo com a sua função no projeto, o que permitiu uma distribuição clara das responsabilidades e facilitou a identificação do progresso individual e coletivo.

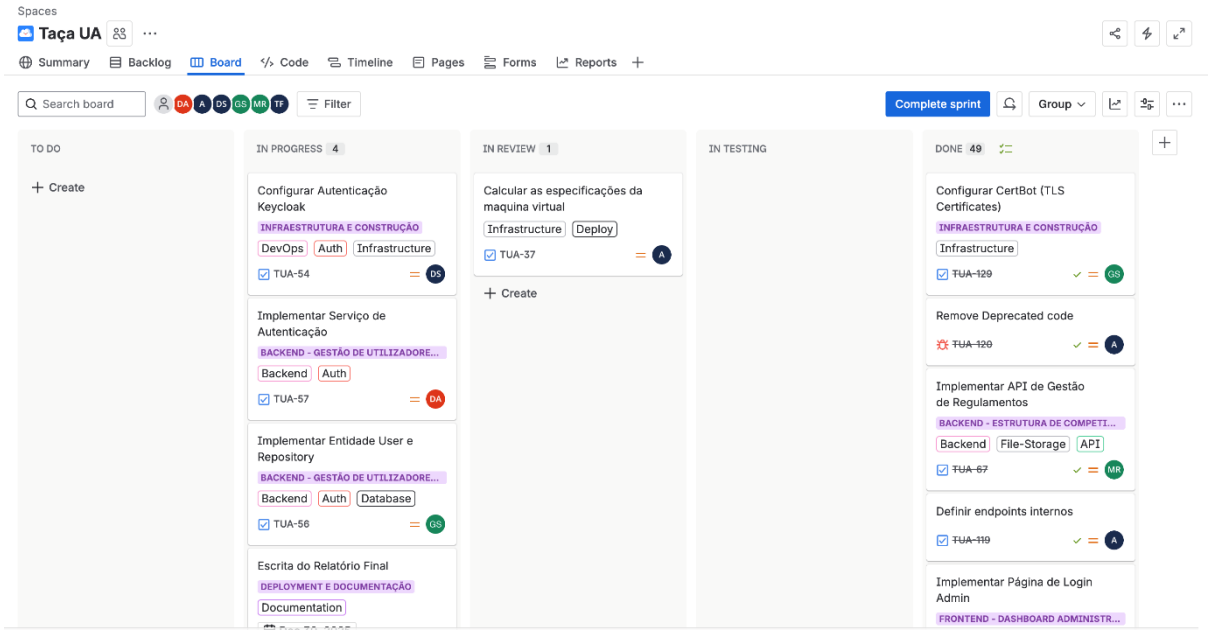


Figura 3.1: Board do Jira utilizada na gestão das tarefas do projeto.

3.3. Equipa

A equipa de desenvolvimento foi constituída por seis elementos, todos estudantes da Licenciatura em Engenharia Informática, que colaboraram ativamente em todas as etapas do desenvolvimento do projeto.

Cada um dos membros assumiu funções e responsabilidades específicas, de acordo com as suas competências e interesses individuais.

Membro	**Função**
Bernardo Borges	Project Owner, Frontend
Alexandre Regalado	DevOps, Backend
Mateus Rocha	Arquiteto, Backend
Gonçalo Simões	Frontend
Diego Aguilar	Frontend
Dinis Sousa	Backend

Tabela 3.1: Tabela das funções de cada membro da equipa

4. Requisitos do Sistema

4.1. Levantamento de requisitos

O levantamento de requisitos do sistema foi realizado com base na análise do contexto atual da gestão da Taça UA, em reuniões informais com os responsáveis pela organização, bem como na observação dos processos atualmente utilizados, maioritariamente suportados por folhas de cálculo, formulários externos e comunicação manual. Este processo permitiu identificar necessidades funcionais e não funcionais associadas à gestão administrativa da competição, à organização das equipas e jogos, e ao acesso público à informação. De forma a garantir que o sistema responde de forma adequada às necessidades reais dos utilizadores, foi adotada uma abordagem centrada no utilizador, recorrendo à definição de atores e personas representativas dos principais perfis de utilização da plataforma.

4.2. Requisitos funcionais

Os requisitos funcionais definem as funcionalidades que o sistema deve disponibilizar de forma a suportar a gestão integral da Taça UA, abrangendo tanto as necessidades administrativas como o acesso público à informação. Estes requisitos encontram-se organizados por áreas funcionais, facilitando a compreensão das responsabilidades associadas a cada tipo de utilizador e garantindo uma cobertura completa dos processos identificados durante o levantamento de requisitos.

4.2.1. Gestão de Utilizadores e Autenticação

- O sistema deve permitir a autenticação de Administradores Gerais e Administradores de Núcleo através de credenciais válidas.
- O sistema deve atribuir permissões de acesso de acordo com o perfil do utilizador, assegurando uma separação clara de responsabilidades.
- O Administrador Geral deve ser capaz de criar, gerir e remover contas de Administradores de Núcleo.
- O público em geral deve conseguir aceder à informação pública da plataforma sem necessidade de autenticação.

4.2.2. Gestão da Competição

- O Administrador Geral deve ser capaz de criar, editar e remover cursos participantes na Taça UA.
- O Administrador Geral deve poder associar Administradores de Núcleo a cada curso.
- O sistema deve permitir a gestão e publicação de regulamentos gerais da competição, bem como regulamentos específicos por modalidade.
- O Administrador Geral deve ser capaz de iniciar e terminar épocas desportivas, definindo o ciclo competitivo de cada edição da Taça UA.

4.2.3. Gestão de Modalidades, Torneios e Jogos

- O Administrador Geral deve ser capaz de criar e configurar torneios para as diferentes modalidades.
- O sistema deve permitir o agendamento e edição de jogos, incluindo a definição de equipas, data, hora e local.
- O Administrador Geral deve poder gerir o calendário completo de jogos de cada modalidade.
- O sistema deve permitir o registo e atualização de resultados dos jogos.
- O Administrador Geral deve poder definir o sistema de pontuação de cada modalidade e tipo de torneio, em conformidade com os regulamentos aplicáveis.

4.2.4. Gestão de Equipas e Atletas por Curso

- O Administrador de Núcleo deve ser capaz de criar, editar e remover jogadores associados ao seu curso.
- O Administrador de Núcleo deve poder criar e gerir equipas por modalidade.
- O sistema deve permitir a inscrição de atletas nas equipas, incluindo o respetivo número mecanográfico.
- O sistema deve verificar automaticamente se o atleta é sócio, com base numa lista fornecida pelo Administrador Geral, sinalizando visualmente situações de incumprimento.

4.2.5. Gestão de Jogos ao Nível do Curso

- O Administrador de Núcleo deve ser capaz de atribuir aos jogos apenas os jogadores elegíveis pertencentes às equipas relevantes.

- O sistema deve permitir a atribuição de números de equipamento aos jogadores convocados.
- O Administrador de Núcleo deve poder adicionar comentários associados a cada jogo.
- O sistema deve permitir a geração de fichas de jogo em formato PDF, contendo toda a informação relevante do encontro.

4.2.6. Transparência, Consulta Pública e Comunicação

- O sistema deve disponibilizar um calendário completo da competição, organizado por dias.
- O público deve poder filtrar o calendário por modalidade e/ou curso.
- Os resultados dos jogos devem ser públicos e atualizados automaticamente.
- O público deve poder consultar classificações por modalidade e a classificação geral da Taça UA.
- O sistema deve permitir o acesso a regulamentos gerais e específicos.
- Deve ser possível consultar resultados e classificações de edições anteriores da competição.

4.2.7. Sistema de Pontuação e Classificação Geral

- O sistema deve calcular automaticamente a pontuação dos cursos por modalidade.
- O sistema deve atualizar de forma automática e consistente a Classificação Geral da Taça UA.

4.3. Requisitos não funcionais

Os requisitos não funcionais definem os critérios de qualidade do sistema, assegurando níveis adequados de desempenho, usabilidade, segurança, fiabilidade e capacidade de evolução. Estes requisitos são essenciais para garantir a aceitação da plataforma pelos utilizadores e a sua sustentabilidade técnica a longo prazo.

4.3.1. Usabilidade

- A plataforma deve apresentar um design responsivo, adaptado a dispositivos desktop, tablet e smartphone.
- A interface deve ser intuitiva e utilizar terminologia reconhecida pela comunidade académica.
- As atualizações de dados devem ocorrer de forma automática, sem necessidade de atualização manual da página.
- A navegação deve permitir o acesso a qualquer funcionalidade principal a partir da página principal em, no máximo, quatro cliques.
- Devem existir mecanismos de pesquisa e filtros eficientes para calendários, equipas e resultados.

4.3.2. Desempenho

- O tempo de resposta do sistema deve ser inferior a dois segundos em 95% das requisições.
- Devem ser implementadas técnicas de cache para reduzir a carga associada a consultas frequentes, como calendários e resultados.
- A arquitetura do sistema deve suportar escalabilidade horizontal, permitindo acomodar um aumento futuro de utilizadores e dados.

4.3.3. Compatibilidade

- A plataforma deve ser compatível com os navegadores mais utilizados, nomeadamente Chrome, Firefox, Edge e Safari.

4.3.4. Manutenibilidade e Evolução

- O código deve seguir boas práticas de desenvolvimento, como os princípios SOLID e arquiteturas MVC e RESTful.
- O sistema deve permitir atualizações contínuas com tempo de indisponibilidade mínimo.
- A documentação técnica deve ser atualizada a cada nova versão do sistema.
- A arquitetura deve permitir a inclusão futura de novas modalidades e categorias sem necessidade de reestruturação profunda.
- Devem existir manuais de utilizador dirigidos a administradores sem conhecimentos técnicos.

4.4. Atores e Personas

De forma a compreender melhor a interação entre os utilizadores e o sistema, foram identificados os principais atores e definidas personas representativas, permitindo alinhar os requisitos técnicos com necessidades reais de utilização.

4.4.1. Atores

Os atores do sistema correspondem aos diferentes tipos de utilizadores que interagem com a plataforma, cada um com permissões e responsabilidades distintas:

- Administrador Geral: responsável pela gestão global da competição, incluindo modalidades, torneios, jogos, regulamentos e classificação geral.
- Administrador de Núcleo: responsável pela gestão das equipas e atletas de um curso específico.
- Utilizador Público: qualquer utilizador sem autenticação que acede à plataforma para consultar informação pública.
- Sistema: ator lógico responsável por processos automáticos, como o cálculo de pontuações e atualização de classificações.

4.4.2. Personas

De forma a compreender melhor os objetivos, motivações e dificuldades dos diferentes tipos de utilizadores, foram definidas várias personas representativas, descritas de seguida.

Michael Hondt – Estudante / Participante

- Idade: 22 anos
- Curso: Química
- Papel: Participante na Taça UA e utilizador público da plataforma

Michael participa regularmente na Taça UA e acompanha ativamente o desempenho do seu curso. Atualmente, sente frustração devido à dispersão da informação e aos atrasos na divulgação de resultados e horários. Procura uma plataforma centralizada onde possa consultar rapidamente calendários, resultados e classificações atualizadas em tempo real, de forma simples e intuitiva.

Lahra Cough – Administradora Geral

- Idade: 26 anos
- Cargo: Membro da Direção da AAUAv
- Papel: Administradora Geral da plataforma

Lahra integra a comissão organizadora da Taça UA e é responsável pela coordenação global da competição. No modelo atual, depende fortemente de folhas de Excel, formulários e comunicação manual, o que compromete a eficiência e o controlo do processo. O seu objetivo principal é gerir modalidades, cursos e responsáveis, torneios, equipas e regulamentos num único sistema digital, reduzindo erros e tempo administrativo.

Zach Daniel – Administrador Geral

- Idade: 24 anos
- Cargo: Membro da Administração da AAUAv
- Papel: Administrador Geral, responsável pelas modalidades

Zach desempenha funções de gestão transversal das modalidades da Taça UA. É responsável pela criação de torneios, agendamento de jogos, atualização de resultados e validação das classificações. Enfrenta dificuldades associadas a atrasos na atualização de dados e correções manuais frequentes. Procura uma solução que lhe permita operar com rapidez, precisão e confiança, garantindo classificações automáticas e comunicação imediata com os participantes.

Teelor Shift – Administradora de Núcleo

- Idade: 24 anos
- Curso: Engenharia Informática
- Papel: Representante do Núcleo de Engenharia Informática

Teelor é responsável pela gestão das equipas do seu núcleo, incluindo a inscrição de atletas e a organização das equipas por modalidade. A comunicação fragmentada e alterações tardias nos horários afetam a logística das equipas. O seu objetivo é dispor de uma plataforma que lhe permita gerir jogadores, equipas e fichas de jogo de forma clara, centralizada e fiável.

Clara Almeida – Utilizadora Pública

- Idade: 19 anos
- Curso: Engenharia Mecânica
- Papel: Utilizadora pública da plataforma

Clara não participa diretamente na competição, mas acompanha os jogos para apoiar o seu curso. Sente dificuldade em encontrar horários e locais atualizados através dos meios atuais. Procura uma plataforma simples, acessível e visualmente clara, onde possa consultar rapidamente os jogos do dia, resultados e classificações.

4.4.3. Casos de Uso

Na Figura 4.1 apresenta-se o diagrama UML de casos de uso do sistema proposto para a gestão da Taça UA. Este diagrama ilustra as principais interações entre os diferentes atores e a plataforma, evidenciando as funcionalidades disponibilizadas a cada perfil de utilizador.

Os casos de uso foram definidos com base no levantamento de requisitos e nas responsabilidades associadas a cada ator, permitindo descrever de forma clara e estruturada as operações suportadas pelo sistema. De forma complementar ao diagrama, a Tabela 4.1 apresenta a descrição detalhada de cada caso de uso identificado.

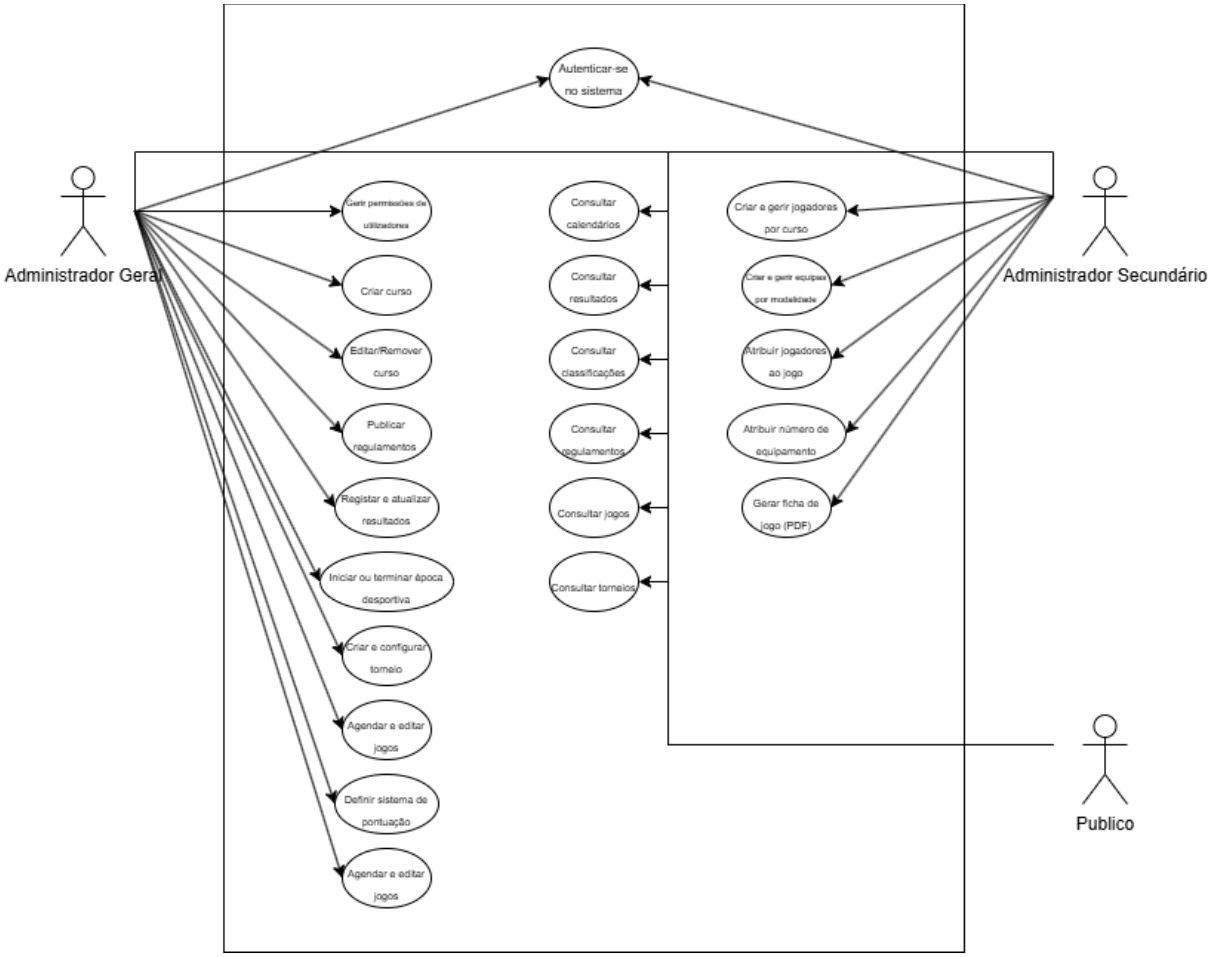


Figura 4.1: Diagrama de Casos de Uso do sistema de gestão da Taça UA.

ID	Ator	Caso de Uso	Descrição
UC1	Todos os Administradores	Autenticar-se no sistema	Permite ao Administrador Geral e ao Administrador de Núcleo autenticar-se na plataforma através de credenciais válidas, garantindo acesso seguro às funcionalidades restritas.
UC2	Administrador Geral	Gerir permissões de utilizadores	Permite atribuir e gerir permissões de acesso de acordo com o tipo de administrador, assegurando controlo hierárquico
UC3	Administrador Geral	Criar curso	Permite criar cursos participantes na Taça UA.
UC4	Administrador Geral	Editar ou remover curso	Permite alterar ou remover cursos previamente criados.
UC5	Administrador Geral	Publicar regulamentos	Permite gerir e publicar regulamentos gerais e específicos por modalidade.
UC6	Administrador Geral	Iniciar ou terminar época desportiva	Permite definir o ciclo competitivo de cada edição da Taça UA.
UC7	Administrador Geral	Criar e configurar torneios	Permite criar torneios por modalidade e definir a sua estrutura competitiva.
UC8	Administrador Geral	Agendar e editar jogos	Permite definir equipas, data, hora e local dos jogos, bem como efetuar alterações posteriores.
UC9	Administrador Geral	Definir sistema de pontuação	Permite configurar o sistema de pontuação por modalidade e tipo de torneio, conforme regulamentos.
UC10	Administrador Geral	Registar e atualizar resultados	Permite introduzir e corrigir resultados dos jogos, desencadeando atualizações automáticas das classificações.
UC11	Administrador Geral	Consultar torneios	Permite visualizar torneios existentes e respetiva informação associada.

ID	Ator	Caso de Uso	Descrição
UC12	Administrador Geral	Consultar jogos	Permite consultar jogos agendados ou já realizados.
UC13	Utilizador Público	Consultar calendários	Permite visualizar o calendário geral de jogos da competição.
UC14	Utilizador Público	Consultar calendário com filtros	Permite filtrar o calendário por modalidade e/ou curso, facilitando o acesso à informação relevante.
UC15	Utilizador Público	Consultar resultados	Permite consultar resultados atualizados dos jogos.
UC16	Utilizador Público	Consultar classificações	Permite visualizar classificações por modalidade e a classificação geral da Taça UA.
UC17	Utilizador Público	Consultar regulamentos	Permite aceder aos regulamentos gerais e específicos das competições.
UC18	Administrador de Núcleo	Criar e gerir jogadores por curso	Permite criar, editar e remover jogadores associados ao respetivo curso.
UC19	Administrador de Núcleo	Criar e gerir equipas por modalidade	Permite criar e organizar equipas do curso por modalidade.
UC20	Administrador de Núcleo	Atribuir jogadores ao jogo	Permite selecionar os jogadores elegíveis para participar num jogo específico
UC21	Administrador de Núcleo	Atribuir número de equipamento	Permite associar um número de equipamento a cada jogador convocado.
UC22	Administrador de Núcleo	Gerar ficha de jogo (PDF)	Permite gerar uma ficha de jogo em formato PDF contendo jogadores, números de equipamento e comentários associados.

Tabela 4.1: Tabela de Casos de Uso do sistema de gestão da Taça UA

5. Arquitetura

A arquitetura do sistema proposto para a gestão da Taça UA foi concebida com base numa abordagem modular, distribuída e orientada a serviços, tendo como principais objetivos a escalabilidade, a manutenibilidade e a separação clara de responsabilidades. A solução adota um modelo de sistema web multicamada, no qual as diferentes componentes comunicam através de interfaces bem definidas, recorrendo a APIs REST e a mecanismos de comunicação assíncrona baseados em eventos.

A Figura 5.1 apresenta a arquitetura global do sistema, ilustrando os principais módulos funcionais, os fluxos de comunicação estabelecidos entre eles e a interação com os diferentes tipos de utilizadores. A arquitetura encontra-se organizada em camadas distintas, nomeadamente: camada de apresentação, camada de autenticação e autorização, camada de orquestração e APIs, serviços de domínio, persistência de dados e observabilidade.

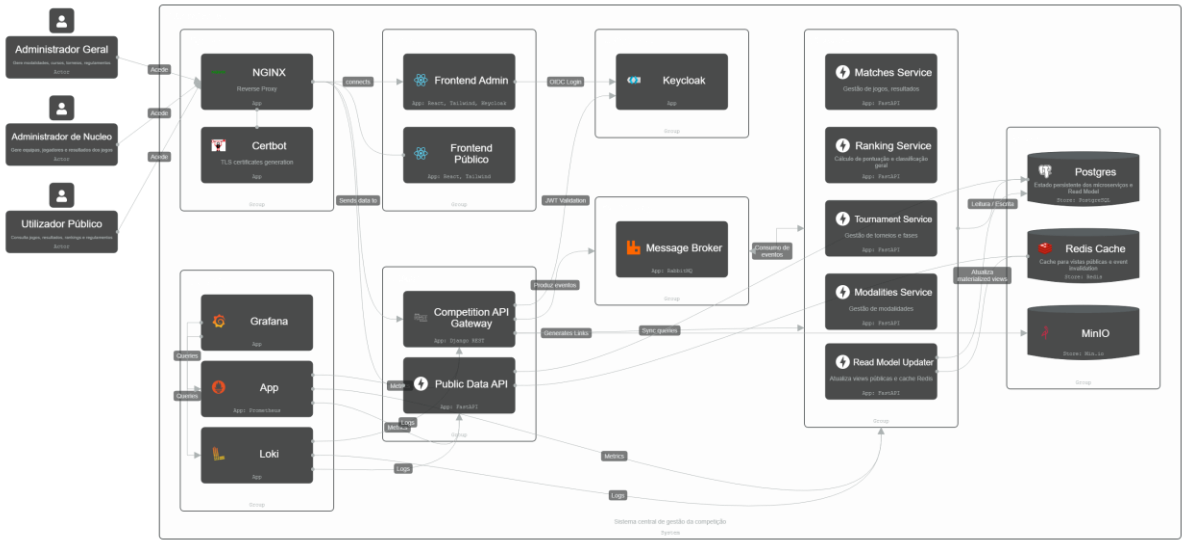


Figura 5.1: Visão geral da arquitetura do sistema

5.1. Camada de Apresentação

A camada de apresentação é responsável pela interação direta entre os utilizadores e o sistema, sendo composta por duas aplicações web distintas:

- Frontend Administrativo, destinado aos Administradores Gerais e Administradores de Núcleo;
- Frontend Público, acessível a utilizadores não autenticados.

Ambas as interfaces foram desenvolvidas com recurso à biblioteca React, em conjunto com Tailwind CSS, garantindo consistência visual, reutilização de componentes e adaptação a diferentes dispositivos. O acesso às aplicações é efetuado através de um NGINX Reverse Proxy, responsável pelo encaminhamento de pedidos HTTP, bem como pela integração com certificados TLS geridos automaticamente pelo Certbot, assegurando comunicações seguras.

5.2. Camada de Autenticação e Autorização

A autenticação e a autorização são elementos fundamentais do sistema, uma vez que a plataforma suporta diferentes perfis de utilizador com níveis distintos de acesso às suas funcionalidades. Para garantir a segurança e o controlo de acessos, o sistema recorre ao Keycloak como serviço centralizado de gestão de identidade.

A autenticação é realizada através do protocolo OpenID Connect (OIDC). Os utilizadores administrativos são redirecionados para o serviço de autenticação, onde validam as suas credenciais. Após autenticação bem-sucedida, é emitido um JSON Web Token (JWT), que identifica o utilizador e é utilizado nos pedidos subsequentes às APIs do sistema, permitindo uma arquitetura “stateless” e escalável.

A autorização baseia-se num modelo de controlo de acesso por papéis (RBAC). Cada utilizador possui papéis específicos, como Administrador Geral ou Administrador de Núcleo, que determinam as operações que pode executar. Os serviços validam o token recebido, verificando os papéis associados antes de permitir o acesso aos recursos solicitados.

O servidor Keycloak é configurado através de um ficheiro de exportação de realm (src/configs/keycloak/realm-export.json), que define de forma reproduzível e versionável toda a configuração do realm taca-ua. Este ficheiro inclui a definição dos clientes OIDC, os mapeadores de claims responsáveis por incluir os papéis do utilizador no token JWT (claim realm_access.roles), e os dois papéis de realm existentes: general_admin, atribuído aos administradores gerais da plataforma, e nucleo_admin, atribuído aos administradores de núcleo. A gestão da configuração por ficheiro de

exportação garante que o estado do servidor de identidade pode ser replicado de forma determinista em qualquer ambiente, sem configuração manual.

A interface de autenticação do Keycloak foi personalizada através de um tema próprio (src/configs/keycloak/themes/taca-ua/login/), que adapta a página de login ao contexto visual da plataforma Taça UA. Este tema substitui o tema predefinido do Keycloak para o realm taca-ua, sendo carregado automaticamente aquando do arranque do contentor.

5.3. Camada de APIs e Orquestração

A comunicação entre os frontends e a lógica de negócio do sistema é realizada através de um conjunto de APIs REST, organizadas de acordo com o tipo de acesso. O Competition API Gateway centraliza o acesso às funcionalidades administrativas, funcionando como ponto único de entrada para operações que requerem autenticação e autorização.

Em paralelo, a Public Data API é responsável por disponibilizar dados públicos, como calendários e resultados, otimizados para consultas frequentes e de elevada concorrência. Esta separação permite melhorar o desempenho global do sistema, ao mesmo tempo que reduz a exposição direta dos serviços internos.

O uso de um gateway de APIs facilita ainda a aplicação uniforme de políticas de segurança, validação de tokens, controlo de acessos e futura integração com outros sistemas ou serviços externos.

5.4. Serviços de Domínio

A lógica de negócio do sistema encontra-se distribuída por vários serviços independentes, cada um responsável por um conjunto específico de funcionalidades, como a gestão de torneios, jogos, equipas, atletas e regulamentos. Esta abordagem promove uma separação clara de responsabilidades e reduz o acoplamento entre componentes.

Um dos serviços centrais é o responsável pelo cálculo automático das classificações, que aplica as regras de pontuação definidas para cada modalidade. Sempre que são registados ou atualizados resultados, este serviço assegura a atualização imediata das classificações, garantindo consistência e fiabilidade da informação disponibilizada.

5.5. Comunicação Assíncrona e Processamento de Eventos

Para suportar processos que não requerem resposta imediata e reduzir dependências diretas entre serviços, o sistema recorre a um mecanismo de comunicação assíncrona baseado num “message broker”. Este componente permite a publicação e consumo de eventos relevantes, como atualizações de resultados ou alterações na configuração de torneios.

A utilização de eventos possibilita que múltiplos serviços reajam automaticamente a alterações no sistema, sem necessidade de comunicação síncrona direta. Por exemplo, após o registo de um resultado, podem ser desencadeados processos de atualização de classificações e invalidação de cache.

5.6. Persistência e Gestão de Dados

A persistência da informação é assegurada por uma base de dados relacional (PostgreSQL), utilizada para armazenar dados estruturados, como utilizadores, cursos, equipas, jogos, resultados e regulamentos. A escolha de uma base de dados relacional garante integridade referencial e suporte a consultas complexas.

Para melhorar o desempenho e reduzir a carga sobre a base de dados principal, o sistema utiliza um mecanismo de cache em memória (Redis). Este componente é particularmente relevante para dados de leitura frequente, como calendários, resultados e classificações públicas.

O armazenamento de ficheiros, nomeadamente documentos em formato PDF e imagens, é realizado através de um serviço de armazenamento de objetos (MinIO), garantindo persistência fiável e acesso controlado a conteúdos não estruturados.

5.7. Observabilidade e Monitorização

A monitorização do sistema é assegurada através da integração de ferramentas de observabilidade, como Prometheus, Grafana e Loki. Estas permitem a recolha de métricas, visualização do estado dos serviços e análise centralizada de logs.

A disponibilização de métricas e registos detalhados facilita a deteção precoce de falhas, a análise de desempenho e a identificação de gargalos no sistema. Esta abordagem é essencial para garantir a fiabilidade e a continuidade do serviço durante o período competitivo da Taça UA.

6. Implementação

O processo de implementação da plataforma de gestão da Taça UA foi orientado por princípios de engenharia de software modernos, privilegiando a modularidade, a escalabilidade, a manutenibilidade e a automação dos processos de desenvolvimento e integração. As decisões técnicas tomadas refletem a necessidade de suportar um sistema distribuído, com múltiplos perfis de utilizador, elevado volume de acessos públicos e requisitos de fiabilidade e segurança.

6.1. Gestão dos repositórios em GitHub

O desenvolvimento do sistema foi suportado por um repositório principal alojado no GitHub. A estrutura do repositório reflete uma separação clara entre frontend, backend, micro-serviços, configurações de infraestrutura e documentação técnica, facilitando o desenvolvimento paralelo e a manutenção do código.

Foram adotadas boas práticas de versionamento e colaboração, nomeadamente:

- Utilização de branches dedicadas ao desenvolvimento de funcionalidades específicas (feature branches);
- Submissão de pull requests com revisão de código antes da integração na branch de desenvolvimento principal;
- Integração contínua automatizada através do GitHub Actions, garantindo validação sistemática do código.

A documentação técnica encontra-se centralizada na diretoria “docs/”, permitindo um acesso estruturado a informação relevante sobre autenticação, definição de endpoints, modelos de dados, migrações e configuração do ambiente de desenvolvimento. Esta abordagem contribui para a rastreabilidade das decisões técnicas e para a sustentabilidade do projeto a longo prazo.

6.2. Frontend

O frontend da plataforma foi desenvolvido com o objetivo de proporcionar uma experiência de utilização intuitiva, responsiva e adequada aos diferentes perfis de utilizador identificados. A separação entre interface pública e painel administrativo permite adaptar a complexidade e o nível de interação às necessidades específicas de cada tipo de utilizador.

6.2.1. Tecnologias Usadas

O desenvolvimento do frontend baseou-se num conjunto de tecnologias modernas amplamente utilizadas no ecossistema web:

- React: desenvolvimento de interfaces dinâmicas baseadas em componentes reutilizáveis;
- TypeScript: reforço da robustez do código através de tipagem estática;
- Vite: ambiente de desenvolvimento e *build* otimizado;
- Tailwind CSS: definição de estilos consistentes e responsivos;
- Docker: containerização das aplicações frontend.

No painel administrativo foi integrado o sistema de autenticação baseado em Keycloak, assegurando o controlo de acessos e a correta gestão de perfis e permissões.

6.2.2. Estrutura no Repositório

O frontend encontra-se organizado em duas aplicações distintas, localizadas na diretoria “src/frontend/”:

- *admin-panel*: painel administrativo destinado ao Administrador Geral e aos Administradores de Núcleo;
- *public-website*: interface pública para consulta de calendários, resultados, classificações e regulamentos.

Ambas as aplicações seguem uma estrutura interna consistente, organizada por responsabilidades:

- pages/: páginas principais da aplicação;
- components/: componentes reutilizáveis;
- api/: módulos de comunicação com as APIs backend;
- contexts/ e hooks/: gestão de estado global e autenticação;

- routes/: definição de rotas da aplicação.

Esta organização promove a separação de preocupações (separation of concerns) e facilita a evolução futura das interfaces.

6.2.3. Interfaces

As interfaces do sistema foram concebidas de acordo com uma separação clara de perfis de utilização, refletindo os diferentes níveis de responsabilidade e necessidades de acesso à informação identificados durante o levantamento de requisitos. Esta abordagem permite otimizar a experiência do utilizador, reduzir a complexidade das interações e reforçar os mecanismos de segurança, ao disponibilizar apenas as funcionalidades estritamente necessárias a cada perfil.

O frontend encontra-se, assim, organizado em três interfaces principais: interface pública e painel do Administrador.

6.2.3.1. Interface Publica

A interface pública destina-se a utilizadores não autenticados e tem como principal objetivo assegurar a transparência e a divulgação eficiente da informação relativa à Taça UA junto da comunidade académica. Esta interface privilegia a simplicidade de navegação, a clareza visual e o acesso rápido à informação mais relevante, uma vez que representa o principal ponto de contacto entre o sistema e o público em geral.

Através desta interface, é possível:

- Consultar o calendário de jogos da competição, com suporte a filtros por curso e modalidade, facilitando a localização de eventos de interesse específico;
- Aceder aos resultados dos jogos, atualizados de forma automática após o seu registo no sistema;
- Visualizar classificações por modalidade, por torneio e a classificação geral da Taça UA, refletindo o estado competitivo atualizado;
- Consultar regulamentos gerais da competição e regulamentos específicos por modalidade;
- Aceder ao histórico de edições anteriores da Taça UA, permitindo uma análise evolutiva da competição ao longo do tempo.

Esta interface foi otimizada para consultas frequentes e elevada concorrência, recorrendo a modelos de leitura e mecanismos de cache descritos nas secções anteriores.

6.2.3.2. Painel do Administrador

O painel do Administrador constitui o núcleo central de gestão da plataforma, disponibilizando funcionalidades de carácter estratégico e transversal a toda a competição. Este perfil é responsável pela configuração global do sistema e pela definição das estruturas organizativas que suportam cada edição da Taça UA.

Através deste painel, um Administrador Geral pode:

- Gerir cursos participantes e respetivos núcleos, estabelecendo a hierarquia administrativa da competição;
- Criar, configurar e administrar modalidades e torneios, definindo a sua estrutura competitiva;
- Gerir e publicar regulamentos gerais e específicos, assegurando a consistência normativa da competição;
- Controlar o ciclo de vida das épocas desportivas, incluindo a abertura e o encerramento de cada edição da Taça UA.

Um Administrador de Núcleo pode:

- Gerir atletas e membros associados ao curso, incluindo criação, edição e remoção de registos.
- Criação e organização de equipas por modalidade, de acordo com as regras definidas;
- Consulta e acompanhamento dos jogos em que as equipas do curso participam;
- Atribuição de jogadores a jogos específicos, assegurando que apenas atletas elegíveis são convocados;

O acesso a estas funcionalidades encontra-se protegido por mecanismos de autenticação e autorização baseados em RBAC, garantindo que apenas utilizadores devidamente autorizados podem executar operações críticas para o funcionamento do sistema.

6.3. Backend

O backend da plataforma da Taça UA foi concebido segundo uma arquitetura distribuída baseada em micro-serviços, com o objetivo de garantir escalabilidade, modularidade e facilidade de manutenção. Esta abordagem permite que diferentes componentes do sistema evoluam de forma independente, reduzindo o acoplamento entre domínios funcionais e facilitando a introdução de novas funcionalidades ou serviços no futuro.

6.3.1. Tecnologias Usadas

A seleção das tecnologias do backend teve como critério principal a robustez, o desempenho e o suporte a arquiteturas orientadas a serviços. As principais tecnologias utilizadas são:

- FastAPI, utilizada no desenvolvimento de micro-serviços de alto desempenho, beneficiando de validação automática de dados, tipagem estática e geração de documentação OpenAPI;
- Django REST Framework, adotado na implementação da API administrativa, funcionando também como ponto de entrada centralizado (API Gateway) para operações autenticadas;
- PostgreSQL, como sistema de gestão de base de dados relacional, garantindo integridade referencial e suporte a consultas complexas;
- Redis, utilizado como mecanismo de cache para otimização de consultas públicas de elevada frequência;
- RabbitMQ, responsável pela comunicação assíncrona baseada em eventos entre serviços;
- Keycloak, como serviço centralizado de autenticação, autorização e gestão de controlo de acessos baseado em papéis (RBAC);
- Docker, utilizado para a containerização de todos os serviços, assegurando portabilidade e consistência entre ambientes.

Esta combinação tecnológica permite responder de forma eficaz aos requisitos funcionais e não funcionais identificados, nomeadamente em termos de desempenho, segurança e escalabilidade.

6.3.2. Serviços implementados

O backend encontra-se organizado em vários serviços especializados, cada um responsável por um domínio funcional específico do sistema. Esta decomposição

facilita o desenvolvimento independente e a manutenção de cada componente. Os principais serviços implementados são:

- Competition API (Django): atua como API Gateway, concentrando os endpoints administrativos, assegurando a validação de tokens JWT e o controlo de permissões de acordo com os papéis do utilizador;
- Public API (FastAPI): disponibiliza endpoints otimizados para consumo público, baseados em modelos de leitura e preparados para elevada concorrência;
- Matches Service: responsável pela gestão de jogos, covocatorias e respetivos resultados;
- Modalities Service: gere modalidades, tipos de modalidade, cursos, membros, equipas, regulamentos e épocas desportivas (seasons);
- Tournaments Service: responsável pela definição da estrutura dos torneios e respetivas fases;
- Ranking Service: encarregado do cálculo automático das pontuações e classificações;
- Read Model Updater: responsável pela atualização dos modelos de leitura e invalidação de cache com base em eventos.

Apesar de partilharem a mesma base de dados física, cada serviço opera sobre domínios bem definidos (schemas), com migrações independentes, reduzindo o acoplamento lógico e facilitando a evolução incremental do sistema.

6.3.3. Estrutura no Repositório

A organização do código-fonte do backend reflete diretamente a arquitetura baseada em micro-serviços adotada pelo sistema. A estrutura do repositório foi concebida com o objetivo de promover uma separação clara de responsabilidades, facilitar a manutenção do código e permitir o desenvolvimento, teste e evolução independentes de cada componente.

O código do backend encontra-se maioritariamente concentrado no diretório src/, o qual está organizado em quatro grandes áreas funcionais: APIs centrais, micro-serviços, componentes partilhados e ferramentas auxiliares. Esta organização contribui para uma visão clara da arquitetura do sistema e reduz o acoplamento entre diferentes domínios funcionais.

6.3.3.1. APIs Centrais

O diretório "src/apis/" agrega as aplicações responsáveis pela interação direta com os frontends e pela orquestração dos restantes serviços do sistema. Estas APIs assumem um papel central na validação de acessos, exposição de endpoints e encaminhamento de pedidos.

As principais APIs centrais são:

- competition-api: aplicação desenvolvida em Django que atua como API Gateway do sistema. É responsável pela exposição dos endpoints administrativos, pela validação de tokens JWT, pela aplicação das regras de autenticação e autorização, bem como pelo controlo de permissões baseado em RBAC.
- public-api: aplicação desenvolvida em FastAPI, dedicada à disponibilização de informação pública da competição. Esta API baseia-se em modelos de leitura otimizados, permitindo consultas eficientes a calendários, resultados, classificações e regulamentos, sem expor diretamente a lógica interna dos serviços de escrita.

Esta separação entre APIs administrativas e públicas contribui para uma maior segurança, desempenho e clareza na definição dos contratos de acesso ao sistema.

6.3.3.2. Micro-serviços

Os micro-serviços do sistema encontram-se organizados no diretório "src/microservices/", sendo cada serviço responsável por um domínio funcional específico da plataforma. Esta abordagem permite isolar a lógica de negócio, reduzir dependências diretas entre componentes e facilitar a evolução independente de cada serviço.

Os micro-serviços atualmente implementados incluem:

- matches-service: responsável pela gestão de jogos e resultados;
- modalities-service: responsável pela gestão de modalidades, cursos, membros e equipas;
- tournaments-service: responsável pela definição e gestão da estrutura dos torneios e respetivas fases;
- ranking-service: responsável pelo cálculo automático das pontuações e pela atualização das classificações;
- read-model-updater: responsável pela atualização dos modelos de leitura utilizados pela API pública, bem como pela invalidação de cache sempre que ocorrem alterações relevantes.

Cada micro-serviço segue uma estrutura interna consistente, promovendo uniformidade e facilidade de manutenção:

- app/: contém a lógica principal da aplicação, incluindo modelos, rotas, esquemas de validação e gestão de eventos;
- alembic/: responsável pela gestão de migrações da base de dados associadas ao serviço;
- Dockerfile e Dockerfile.env: definem os ambientes de execução do serviço;
- requirements.txt: lista as dependências específicas de cada micro-serviço.

6.3.3.3. Componentes Partilhados

O diretório src/shared/ concentra bibliotecas e módulos comuns utilizados por múltiplos serviços, promovendo a reutilização de código e evitando duplicação de lógica transversal ao sistema. Os componentes partilhados encontram-se divididos em dois subdiretórios: src/shared/platform/, para bibliotecas de infraestrutura transversal, e src/shared/contracts/, para os contratos de dados partilhados entre serviços.

Entre os principais componentes partilhados destacam-se:

taca-logging: A biblioteca taca-logging fornece uma configuração centralizada de logging estruturado para todos os serviços da plataforma, com base na biblioteca structlog. A função configure_logging() inicializa a pipeline de processadores, garantindo

que todos os serviços emitem registos num formato uniforme, compatível com a ingestão pelo Loki. A função get_logger() é o ponto de acesso padrão para obter um logger estruturado em qualquer serviço. Para os serviços FastAPI, a biblioteca expõe adicionalmente o StructlogMiddleware, que associa automaticamente o identificador do pedido (request_id) e o método HTTP ao contexto de cada log emitido durante o processamento de um pedido.

taca-messaging: Fornece uma interface uniforme para a comunicação assíncrona baseada em RabbitMQ, encapsulando a lógica de publicação e consumo de eventos e uniformizando a interação entre serviços. Todos os microserviços utilizam esta biblioteca para publicar e subscrever eventos de domínio sem dependência direta do cliente RabbitMQ.

t**aca-outbox:** Implementa o padrão Transictional Outbox para garantir a entrega fiável de eventos ao RabbitMQ mesmo em caso de falha do broker. A biblioteca fornece a função de fábrica create_outbox_model(), que cada microserviço invoca com o nome do seu schema PostgreSQL para gerar a tabela outbox_event correspondente. Esta tabela regista cada evento a publicar com campos event_type, aggregate_type, aggregate_id, payload, published e retry_count. O processo OutboxPublisher percorre periodicamente os eventos não publicados e encarrega-se de os entregar ao RabbitMQ, incrementando o contador de tentativas em caso de falha e garantindo que nenhum evento é perdido silenciosamente.

taca-rebuild: Fornece a infraestrutura de reconstrução do modelo de leitura via Server-Sent Events (SSE). A classe abstrata BaseSSERebuildService define o contrato que o read-model-updater implementa: os microserviços de escrita expõem um endpoint SSE interno (/internal/snapshot/stream) que transmite o estado atual do seu domínio de forma incremental; o read-model-updater consome esses streams e reconstrói as tabelas do modelo de leitura sem carregar todos os dados em memória de uma só vez. A biblioteca inclui também endpoints de controlo de reconstrução (pause/resume), acessíveis via /internal/rebuild/pause e /internal/rebuild/resume, que permitem interromper e retomar o processo sem reiniciar o serviço.

taca-storage: Abstrai o acesso ao MinIO, o sistema de armazenamento de objetos utilizado para fichas de jogo em PDF e imagens de logótipos de núcleos. A classe MinioService encapsula a criação de buckets com política de leitura pública, o upload e download de ficheiros, e a gestão de URLs públicas. O FileService, construído sobre o MinioService, fornece operações de mais alto nível (upload com validação de tipo MIME, geração de URL pública) utilizadas diretamente pela competition-api-v2 nos endpoints de gestão de ficheiros.

taca-events: Atua como registo de esquemas de eventos (schema registry) da plataforma. Define as estruturas Pydantic de cada evento de domínio, com

versionamento explícito (por exemplo, match.created.v1), permitindo que qualquer serviço importe e deserialize um evento recebido como um objeto tipado em vez de um dicionário arbitrário. Esta abordagem garante contratos estáveis entre serviços produtores e consumidores e facilita a deteção de incompatibilidades em tempo de desenvolvimento.

taca-snapshots: Define os DTOs (Data Transfer Objects) Pydantic utilizados nos payloads de snapshot transmitidos pelo mecanismo SSE de reconstrução. Cada domínio tem o seu módulo de DTOs (matches.py, modalities.py, ranking.py, tournaments.py), que formaliza a estrutura dos dados transmitidos entre os microserviços de escrita e o read-model-updater. A separação destes contratos numa biblioteca independente garante que produtor e consumidor evoluem de forma coordenada.

read-model-shared: Define os modelos partilhados utilizados nos modelos de leitura, sendo consumido tanto pela public-api como pelo read-model-updater. Garante que a estrutura das tabelas de leitura é consistente entre o serviço que as atualiza e o serviço que as expõe.

A centralização destes componentes facilita a manutenção, melhora a consistência do sistema e reduz o esforço de desenvolvimento.

6.3.3.4. Ferramentas Auxiliares

Para além do código aplicacional, o repositório inclui um conjunto de diretórios dedicados à configuração da infraestrutura e a ferramentas de suporte ao desenvolvimento e operação do sistema.

Estes encontram-se organizados da seguinte forma:

- src/configs/: contém as configurações associadas a componentes de infraestrutura, como NGINX, Prometheus, Grafana, Loki e Certbot;
- tools/: reúne ferramentas auxiliares destinadas a testes, povoamento de dados e apoio ao desenvolvimento.

Esta organização contribui para uma separação clara entre código aplicacional e configurações operacionais, facilitando o deployment e a gestão do sistema em diferentes ambientes.

6.4. DevOps & CI/CD

A estratégia de DevOps adotada visa garantir consistência entre ambientes, automatização de processos e redução de erros humanos durante o ciclo de desenvolvimento e entrega do software.

6.4.1. Docker

Todos os componentes do sistema encontram-se containerizados, incluindo frontend, backend, micro-serviços e ferramentas de observabilidade. Foram definidos ficheiros Dockerfile e docker-compose distintos para ambientes de desenvolvimento e produção, permitindo reproduzir localmente condições próximas do ambiente final de execução e facilitando o processo de deploy.

6.4.2. Continuous Integration

A integração contínua é suportada por GitHub Actions, automatizando tarefas como a validação de builds, execução de testes e verificação da integridade das imagens Docker. Complementarmente, são utilizados pre-commit hooks para garantir conformidade com normas de estilo e qualidade de código antes da submissão de alterações para o repositório.

Esta abordagem contribui para a deteção precoce de erros e para a manutenção de um elevado padrão de qualidade ao longo do desenvolvimento.

6.4.3. Continuous Deployment

O processo de deployment foi concebido para suportar atualizações contínuas do sistema. O acesso externo é mediado por um reverse proxy NGINX, responsável pelo encaminhamento de pedidos e pela integração com certificados TLS geridos automaticamente através do Certbot, assegurando comunicações seguras.

A pipeline de deploy e acionada sempre que é feito um PR para a branch principal do projeto.

Este processo envolve vários passos:

No ambiente geral:

1. Construção de imagens docker de produção
2. Publicação de imagens construídas para o “GitHub Container Registry” (GHCR)

No ambiente de produção:

3. Criação do ficheiro “.env”. Injetado todos os segredos referentes a palavras-passe default de vários serviços guardados com o “GitHub Secrets”.
4. Sincronizar “docker-compose” de produção e as várias configurações em “src/configs”
5. Download das imagens mais recentes presentes no GHCR.
6. Levantar novas imagens.
7. Esperar 10 segundos e verificar se os serviços estão a correr.
8. Apagar as imagens antigas.

6.5. Modelo de Dados

O modelo de dados do sistema foi concebido de acordo com os princípios da arquitetura de microserviços, atribuindo a cada serviço a responsabilidade pelas suas próprias entidades e regras de integridade. Cada serviço opera sobre um schema PostgreSQL

dedicado, com migrações independentes geridas pelo Alembic. Esta abordagem reduz o acoplamento entre serviços e facilita a manutenção e evolução independente de cada domínio.

De forma a otimizar o acesso público à informação, foi adotada uma separação entre modelos de escrita e modelos de leitura. Os modelos de escrita residem nos microserviços de domínio; os modelos de leitura residem no schema public_read, mantido pelo read-model-updater e sincronizado por eventos assíncronos. Esta estratégia melhora o desempenho das consultas públicas e garante consistência eventual dos dados.

6.5.1. modalities-service

Season - representa uma época desportiva. Agrega toda a atividade competitiva de um período letivo. Atributos: id (inteiro autoincrementado), name, created_by, finished_by, created_at, finished_at. Relaciona-se com ModalityType, Course, Team e Regulation através de season_id.

Nucleo - representa um núcleo académico participante na competição. Atributos: id (UUID), name, abbreviation (única), logo_url, admins_ids (array de UUIDs dos administradores associados), created_by, created_at, updated_at.

Course - representa um curso universitário associado a um núcleo. Atributos: id (UUID), name, abbreviation (única), nucleo_id (FK → Nucleo), created_by, created_at, updated_at. Um curso pode participar em múltiplas épocas.

ModalityType - define um tipo de modalidade desportiva e as suas regras de pontuação. Atributos: id (UUID), name, description, mode (enum: "modality" ou "points"), escaloes (JSON com escalões, limites de participantes e tabela de pontos), tournament_competitor_type (individual ou equipa), season_id (FK - Season), created_by, created_at, updated_at.

Modality - representa uma modalidade desportiva concreta (ex.: Futebol 7). Atributos: id (UUID), name (único), created_by, created_at, updated_at. Relaciona-se com ModalityType por época através da entidade SeasonModality.

SeasonModality - entidade de associação que liga uma Modality a um ModalityType numa Season específica. Permite que a mesma modalidade tenha tipos e regulamentos

diferentes em épocas distintas. Atributos: season_id, modality_id, modality_type_id, regulation_id.

Member - classe base (herança de tabela única) para os participantes registados. Atributos comuns: id (UUID), full_name, member_type (discriminador), created_by, created_at, updated_at.

Student (extends Member) - atleta/sócio inscrito. Atributos adicionais: student_number, course_id (FK - Course), date_of_birth, photo_url.

Staff (extends Member) - membro de staff técnico. Atributos adicionais: role (função no staff), nucleo_id (FK - Nucleo).

Team - equipa registada numa modalidade e época. Atributos: id (UUID), name, modality_id (FK - Modality), course_id (FK - Course), season_id (FK - Season), logo_url, created_by, created_at, updated_at.

Regulation - documento regulamentar associado a uma modalidade numa época. Atributos: id (UUID), title, description, file_url (URL do PDF armazenado no MinIO), season_id (FK - Season), created_at, updated_at.

6.5.2. matches-service

Match - representa uma ocorrência competitiva (jogo). Não assume desporto nem estrutura específica. Atributos: id (UUID), tournament_id (referência ao domínio de torneios), location, start_time, status (enum: SCHEDULED, IN_PROGRESS, FINISHED), journey (ronda do torneio), created_by, created_at, updated_at.

MatchParticipant - regista um participante num jogo e o seu resultado. Chave composta por match_id e participant (UUID do competidor no domínio de torneios). Atributos: score (pontuação obtida) e position (posição final).

Lineup - convocatória de uma equipa para um jogo. Chave composta por match_id, participant e player_id. Atributos: jersey_number (número de camisola) e is_starter (indica se o jogador está no onze inicial). Permite gerir a lista de atletas convocados por equipa em cada jogo.

MatchLineupStaff - membros do staff técnico presentes num jogo. Chave composta por match_id, participant_id e staff_id. Regista os elementos do staff por equipa participante em cada jogo.

Comment - comentários ou notas associadas a um jogo. Atributos: id (UUID), match_id (FK - Match), message (texto livre), created_by, created_at.

OutboxEvent (schema: matches) - tabela de outbox para publicação fiável de eventos ao RabbitMQ. Gerada pela função create_outbox_model() da biblioteca taca-outbox. Atributos: id (UUID), event_type, aggregate_type, aggregate_id, payload (JSON), published (booleano), published_at, created_at, retry_count. Instâncias análogas existem nos schemas tournaments, modalities e ranking.

6.5.3. tournaments-service

Tournament - representa um torneio. Suporta formatos múltiplos através de herança polimórfica (campo format: "free" ou "league"). Atributos: id (UUID), modality_id, name, status (draft/active/finished), start_date,scoring_format_id, competitor_type (enum: TEAM ou ATHLETE), season_id,format, created_by, created_at, updated_at, finished_at, finished_by.

TournamentCompetitor - competidor inscrito num torneio. Atributos: id (UUID), tournament_id (FK - Tournament), competitor_type (enum: TEAM ou ATHLETE), team_id, athlete_id, competitor_course_id, created_at. Suporta tanto torneios de equipas como torneios individuais.

TournamentRankingPosition - classificação final de um competidor num torneio. Atributos: id (UUID), tournament_id (FK - Tournament), competitor_id, position, created_at.

6.5.4. ranking-service

O ranking-service mantém projeções de leitura dos dados necessários ao cálculo de classificações, alimentadas por eventos dos restantes serviços. As entidades ModalityTypeEscalao, ModalityType, Modality, Tournament, TournamentCompetitor, TournamentResult e Course são projeções locais sincronizadas por eventos.

ModalityTypeEscalao - define os escalões e a tabela de pontuação de um tipo de modalidade. Atributos: _id (inteiro), modality_type_id, name, min_participants, max_participants, points (array de inteiros com os pontos atribuídos por posição).

TournamentResult - resultado final de um competidor num torneio, utilizado para calcular pontuações. Atributos: tournament_id, competitor_id, position.

GeneralRanking - classificação geral por curso e época. Atributos: season_id, course_id, points. Tabela derivada, recalculada pelo ranking-service após cada atualização de resultados.

ModalityRanking - classificação por modalidade, curso e época. Atributos: season_id, modality_id, course_id, points.

CourseRanking - classificação consolidada por curso e época, com discriminação de pontos por modalidade. Atributos: season_id, course_id, points, modality_breakdown (array de inteiros com os pontos por modalidade).

6.6. Segurança

A plataforma Taça UA implementa segurança em duas camadas distintas: autenticação e controlo de acesso ao painel de administração, e segurança na camada de transporte para todas as comunicações externas. A API pública é intencionalmente acessível sem autenticação, uma vez que expõe exclusivamente dados de leitura sem caráter confidencial.

6.6.1. Autenticação e Controlo de Acesso

A autenticação dos utilizadores do painel de administração é delegada ao Keycloak 26.3.2, que atua como servidor de identidade segundo o protocolo OpenID Connect (OIDC). O servidor Keycloak é configurado a partir de um ficheiro de exportação de *realm* (*src/configs/keycloak/realm-export.json*), que define de forma reproduzível todas as configurações do *realm* taca-ua — incluindo clientes OIDC, mapeadores de *claims* e os dois papéis de aplicação existentes: *general_admin*, atribuído aos administradores gerais, e *nucleo_admin*, atribuído aos administradores de núcleo.

Após autenticação bem-sucedida, o Keycloak emite um JSON Web Token (JWT) assinado com o algoritmo RS256. Os papéis do utilizador são incluídos no JWT através do *claim realm_access.roles*, conforme configurado no mapeador de *claims* do *realm*.

A validação dos tokens no servidor é realizada pelo *middleware KeycloakJWTMiddleware*, implementado em *src/apis/competition-api-v2/admin_api/middleware.py*. O *middleware* é aplicado a todas as rotas da API de administração e funciona da seguinte forma:

- O cabeçalho Authorization: Bearer <*token*> é extraído de cada pedido recebido.
- O middleware obtém as chaves públicas de assinatura do Keycloak através do endpoint JWKS (JSON Web Key Set), cujo URI é: *{KEYCLOAK_INTERNAL_URL}/realms/taca-ua/protocol/openid-connect/certs*.
  As chaves são obtidas através da biblioteca PyJWKClient e mantidas em cache local por um período de uma hora (lifespan=3600), minimizando a latência das verificações subsequentes.
- O *token* é verificado com a chave pública correspondente. São validados: a assinatura (RS256), a data de expiração (exp), o emissor (iss) e, opcionalmente, o público-alvo (aud).
- Em caso de sucesso, os atributos r*equest.user_id* (claim sub) e *request.roles* (claim realm_access.roles) são populados no objeto de pedido e ficam disponíveis para as camadas de autorização subsequentes.

O controlo de acesso ao nível das vistas é implementado por um conjunto de decoradores de autorização (admin_api/utils/decorators.py), que constituem uma segunda camada de verificação sobre os atributos populados pelo middleware. O decorador require_auth garante que o utilizador está autenticado (HTTP 401 caso contrário); o decorador require_roles (*roles) verifica que o utilizador possui todos os papéis exigidos (HTTP 403 caso contrário). Esta separação de responsabilidades - validação do token no middleware, aplicação de políticas nas vistas - permite reutilizar a lógica de validação em toda a API sem duplicação.

Para facilitar o desenvolvimento local sem dependência do servidor Keycloak, o middleware suporta um modo de contorno de autenticação (dev bypass) ativado exclusivamente por variáveis de ambiente (DEV_AUTH_BYPASS_ENABLED=true e DEV_AUTH_BYPASS_TOKEN). Quando ativo, um pedido com o cabeçalho X-Dev-Auth-Token: <token-configurado> contorna a validação JWT e assume os papéis definidos em DEV_AUTH_BYPASS_ROLES. Este mecanismo nunca deve ser ativado em ambiente de produção.

6.6.2. Segurança na Camada de Transporte

Todo o tráfego externo da plataforma é encaminhado através do NGINX, que atua como reverse proxy com terminação TLS na porta 443. Os serviços internos (competition-api-v2, public-api, admin-panel, public-website, Keycloak) não são expostos diretamente ao exterior, comunicando exclusivamente através da rede interna Docker.

Os certificados TLS são emitidos pela autoridade de certificação Let's Encrypt e geridos automaticamente pelo Certbot. A renovação dos certificados é configurada através do Docker Compose de produção, não exigindo intervenção manual. O processo de obtenção do certificado inicial segue o desafio ACME HTTP-01: o NGINX inicia temporariamente em modo HTTP na porta 80, o Certbot solicita o certificado ao Let's Encrypt e, após validação, o NGINX recarrega a configuração com HTTPS ativo.

A configuração NGINX define ainda os seguintes cabeçalhos de segurança HTTP, aplicados a todas as respostas:

- X-Frame-Options: SAMEORIGIN - impede o enquadramento da página em iframes de origens externas, mitigando ataques de clickjacking.
- X-Content-Type-Options: nosniff - previne que o navegador interprete respostas com um tipo MIME diferente do declarado.
- X-XSS-Protection: 1; mode=block - ativa o filtro XSS dos navegadores e instrui o bloqueio da página em caso de deteção de ataque.

6.7. Problemas Encontrados

O desenvolvimento da plataforma Taça UA envolveu a resolução de um conjunto de problemas técnicos e organizacionais que condicionaram o ritmo de trabalho e influenciaram algumas decisões de arquitetura. Esta secção descreve os principais obstáculos encontrados ao longo do projeto e as soluções adotadas em cada caso.

6.7.1. Integração com Microserviços

A adoção de uma arquitetura de microserviços introduziu desafios significativos ao nível da comunicação assíncrona entre serviços e da propagação de erros. Em sistemas distribuidos que comunicam por mensagens, a ausência de uma transação global torna difícil de rastrear a causa da raiz de uma falha quando este se propaga através de vários serviços. Um erro que ocorre no matches-service, por exemplo, pode manifestar-se de forma silenciosa ou com atraso no ranking-service, tornando o diagnóstico demoroso sem ferramentas adequadas.

Para mitigar este problema, foi implementado o Loki como sistema centralizado de recolha e indexação de registos. Todos os microserviços emitem registos estruturados em formato JSON através da biblioteca taca-logging - construída sobre o structlog - que são reencaminhados para o Loki e consultáveis em tempo real através do Grafana. Esta abordagem permite correlacionar eventos entre serviços distintos e identificar rapidamente o ponto de falha numa cadeia de processamento assíncrono.

A fiabilidade da entrega de eventos RabbitMQ constituiu um problema adicional neste contexto: em caso de falha do broker no momento da publicação, um evento poderia ser perdido de forma irreversível. A solução adotada foi a implementação do padrão Transactional Outbox em cada microserviço, garantindo que os eventos são persistidos atomicamente na base de dados antes de serem publicados, e que qualquer evento não entregue é reenviado pelo processo de relay da outbox.

6.7.2. Falta de Dados Reais

O desenvolvimento e o teste de funcionalidades como rankings, torneios e listagens de equipas requerem um volume de dados coerente e representativo da realidade operacional da plataforma. No entanto, a Taça UA não disponibilizou dados históricos estruturados em formato adequado para importação direta.

Para contornar esta limitação, foi desenvolvido um conjunto de ferramentas de apoio (tools/scraper.py, tools/populate_db2.py) que permitiram extrair e processar informação de fontes públicas existentes, como listas de cursos e núcleos da Universidade de Aveiro, e criar dados simulados coerentes com o modelo de domínio da plataforma. Os dados processados foram organizados em três estágios (tools/data/step1-raw-data/, step2-processed-data/, step3-final-data/), permitindo reproduzir o processo de importação em qualquer ambiente de desenvolvimento ou demonstração.

6.7.3. Problemas de Desempenho

A execução dos testes de desempenho com a ferramenta k6 permitiu identificar um bottleneck na API pública sob carga elevada. A base de dados PostgreSQL, configurada com um pool de ligações limitado, tornava-se o componente condicionante à medida que o número de utilizadores virtuais aumentava, resultando em tempos de resposta crescentes e, a partir de determinado limiar, em erros de esgotamento do pool.

Para resolver este problema, foi introduzida uma camada de cache Redis na API pública, com TTLs configuráveis por tipo de recurso. A maioria dos pedidos de leitura passa a ser servida diretamente a partir da cache, sem acesso à base de dados, o que reduziu os valores de p95 de latência para valores na ordem dos 10 a 14 ms, conforme demonstrado pelos resultados do teste de latency apresentados na secção 8.3.

6.7.4. Configuração do NGINX em Ambiente de Staging

A configuração do NGINX como reverse proxy revelou-se problemática durante a fase de implantação na máquina virtual de staging. Especificamente, o encaminhamento correto de pedidos para os vários serviços internos (painel de administração, API pública, API de administração, Keycloak) exigiu ajustes iterativos às regras de localização, em particular no que diz respeito ao tratamento dos cabeçalhos X-Forwarded-Proto e X-Forwarded-Host necessários para o correto funcionamento do Keycloak por detrás do proxy.

A resolução deste problema implicou a revisão da configuração do nginx.conf e a adição de um script de entrada (docker-entrypoint.sh) que adapta dinamicamente a configuração consoante o modo de arranque (HTTP para validação ACME ou HTTPS para operação normal).

6.7.5 Atraso na Disponibilização da VM de Staging

A máquina virtual de staging, fornecida pelos serviços de infraestrutura da Universidade de Aveiro, sofreu um atraso de disponibilização devido a mudanças internas nos procedimentos de provisionamento e na própria estrutura das máquinas. Este atraso condicionou o calendário de testes de integração e de validação em ambiente próximo da produção, obrigando a adiar parte das atividades de deployment para uma fase mais tardia do projeto.

7. Documentação da API

A documentação da API da plataforma de gestão da Taça UA foi concebida de forma a garantir clareza, consistência e alinhamento contínuo com a implementação do sistema. Para esse efeito, adotou-se o padrão OpenAPI, amplamente utilizado na especificação formal de interfaces de serviços web REST, permitindo a geração automática de documentação técnica estruturada e interativa.

A documentação é disponibilizada através da ferramenta ReDoc, que possibilita uma navegação clara e organizada pelos diferentes recursos expostos pela API. Esta abordagem assegura que a documentação reflete fielmente o estado atual do sistema, reduzindo o risco de divergências entre a especificação e a implementação efetiva dos serviços.

Importa salientar que a documentação é gerada automaticamente a partir das definições dos endpoints, modelos de dados e esquemas de validação presentes no código-fonte dos serviços backend, não sendo necessária manutenção manual adicional.

7.1. Ferramentas e Abordagem

Para a especificação e geração da documentação da API foram utilizadas as seguintes tecnologias e frameworks:

- OpenAPI, para a definição formal dos contratos das APIs, incluindo endpoints, métodos, parâmetros e esquemas de dados;
- ReDoc, para a visualização estruturada e interativa da documentação;
- FastAPI e Django Rest Framework, utilizados no backend, que permitem a geração automática dos esquemas OpenAPI a partir do código da aplicação.

A adoção desta abordagem integrada garante que a documentação acompanha automaticamente a evolução do backend, constituindo um recurso fiável tanto para a compreensão da arquitetura dos serviços como para o desenvolvimento, manutenção e futura extensão da plataforma.

7.2. Conteúdo da Documentação

A documentação da API encontra-se organizada por recursos e serviços, refletindo a estrutura lógica do sistema. Para cada endpoint são disponibilizadas informações essenciais, nomeadamente:

- Método HTTP e respetivo caminho do endpoint;
- Descrição funcional da operação;
- Parâmetros de entrada, incluindo parâmetros de caminho, query e corpo do pedido;
- Estrutura dos pedidos e das respostas, com esquemas de dados associados;
- Códigos de resposta HTTP e respetivos significados;
- Requisitos de autenticação e autorização, quando aplicável.

Estão devidamente documentadas tanto a API administrativa, responsável pela gestão interna da competição (modalidades, torneios, jogos, equipas e classificações), como a API pública, destinada à consulta de informação aberta, incluindo calendários, resultados, classificações e regulamentos.

Esta documentação constitui, assim, um elemento fundamental de suporte à utilização correta da API, à integração com outros sistemas e à evolução futura da plataforma, promovendo boas práticas de engenharia de software e facilitando a continuidade do projeto.

8. Testes e Resultados

Com o objetivo de validar a qualidade e o desempenho da plataforma desenvolvida, foram conduzidas três categorias de testes: testes de usabilidade com utilizadores reais, testes de integração ao nível do frontend e testes de desempenho sobre a API pública. Esta abordagem permite cobrir, de forma complementar, a experiência do utilizador final, a correta integração dos componentes de interface com o backend e a capacidade de resposta do sistema sob diferentes condições de carga.

8.1. Testes de Usabilidade

Com o objetivo de avaliar a adequação da interface do painel de administração às necessidades dos seus utilizadores reais, foram conduzidos testes de usabilidade formais com o recurso ao método System Usability Scale (SUS). Os testes foram realizados diretamente sobre a aplicação funcional, permitindo uma avaliação mais próxima da experiência de utilização real do que aquela que seria obtida através de um protótipo de baixa fidelidade.

8.1.1. Amostra

Os testes contaram com a participação de sete utilizadores, com perfis e experiência variada. Cada participante interagiu de forma independente com a plataforma, sendo observado por um membro da equipa de desenvolvimento ao longo de toda a sessão. Antes da realização das tarefas, foi recolhida informação sobre o sexo, a idade e a experiência anterior com este tipo de aplicação de gestão, através de um questionário pós-tarefas (Apêndice D).

8.1.2. Método

Cada sessão foi conduzida por um observador pertencente à equipa de desenvolvimento, utilizando um Guião do Observador (Apêndice C) que permitiu registar, para cada uma das sete tarefas propostas, se a tarefa foi concluída, se o resultado obtido estava correto, o tempo estimado de conclusão, o número e severidade dos erros cometidos, se o utilizador ficou perdido em algum momento, se necessitou de ajuda e a dificuldade global observada. As sessões não foram conduzidas com assistência direta - o observador intervinha apenas em situações de bloqueio total que impedissem a progressão do participante.

As sete tarefas definidas para o teste cobrem o fluxo de trabalho administrativo central da plataforma, desde a gestão de utilizadores até à condução de um jogo, e foram

concebidas de forma a envolver os dois perfis de administrador existentes (administrador geral e administrador de núcleo), conforme a Tabela 8.1.

Tarefa	Perfil	Descrição
T1	Geral	Fazer login como administrador geral e indicar quantos administradores estão presentes, discriminando quantos são de núcleo (X/Y)
T2	Geral	Adicionar um novo administrador de núcleo, associado a 3 núcleos, e indicar ao observador os cursos associados.
T3	Geral	Adicionar uma nova modalidade de um tipo à escolha.
T4	Núcleo	Fazer login como o administrador de núcleo criado e criar um membro participante
T5	Núcleo	Criar uma equipa usando a modalidade criada na T3 e adicionar o participante criado na T4 à equipa
T6	Geral	Voltar ao administrador geral. Criar um torneio com a modalidade criada, selecionar duas equipas participantes, criar um jogo entre elas e ativar o torneio
T7	Núcleo	Voltar ao administrador de núcleo. Localizar o jogo criado, editar a convocatória para incluir o participante criado, consultar a ficha de equipa e verificar a data do jogo no calendário

Tabela 8.1.2: Tabela com as tarefas dos testes de usabilidade

No final de cada sessão, os participantes preencheram o questionário pós-tarefas (Apêndice B), que inclui as dez afirmações do SUS avaliadas numa escala de Likert de um a cinco - de "Concordo plenamente" a "Discordo completamente" - a partir das quais é calculada uma pontuação normalizada de usabilidade no intervalo entre zero e cem. O questionário incluiu ainda um campo de comentários livres sobre a experiência de utilização.

8.1.3. Resultados

Todos os participantes completaram as sete tarefas propostas com uma taxa de conclusão de 100%. A Tarefa 1 constituiu a única exceção parcial: tratando-se de uma tarefa de leitura e interpretação de dados (indicar quantos administradores existem na plataforma), cinco dos sete participantes (71,4%) forneceram a resposta correta ("2/1"), um participante forneceu uma resposta equivalente e um forneceu um valor incorreto, o que indica que a navegação foi bem-sucedida, mas a leitura dos dados requer atenção.

A Tabela 8.2 sintetiza a distribuição das avaliações de dificuldade, numa escala de 1 (Muito Difícil) a 5 (Muito Fácil), para cada tarefa.

Tarefa	Muito Difícil (1)	Difícil (2)	Média (3)	Fácil (4)	Muito Fácil (5)
T1	-	-	14,3%	57,1%	28,6%
T2	-	-	42,9%	42,9%	14,3%
T3	-	-	14,3%	28,6%	57,1%
T4	-	-	14,3%	28,6%	57,1%
T5	-	28,6%	14,3%	28,6%	28,6%
T6	-	-	28,6%	42,9%	28,6%
T7	-	42,9%	14,3%	14,3%	28,6%

Tabela 8.1.3: Distribuição de dificuldade percebida por tarefa (n=7)

Nenhuma tarefa registou avaliações de "Muito Difícil", o que constitui um indicador positivo de usabilidade geral. As tarefas T3 e T4, relativas à criação de modalidades e de membros participantes, foram as mais bem avaliadas, com 57,1% dos participantes a classificá-las como "Muito Fácil". A tarefa T7 - que envolvia localizar um jogo, editar a convocatória, consultar a ficha de equipa e verificar a data no calendário - foi a mais desafiante, com 42,9% dos participantes a avaliá-la como "Difícil", o que sugere que o fluxo desta funcionalidade pode beneficiar de melhorias de navegação. A tarefa T5, relativa à criação de equipas com associação de participantes, registou igualmente 28,6% de avaliações "Difícil".

A pontuação SUS média obtida foi de 80,7, o que, segundo a escala de adjetivos de Bangor et al., corresponde a uma classificação de "Bom" (Good). Dado que valores superiores a 68 são considerados acima da média, este resultado confirma que a plataforma satisfaz os requisitos de usabilidade enunciados na secção 4.3.1.

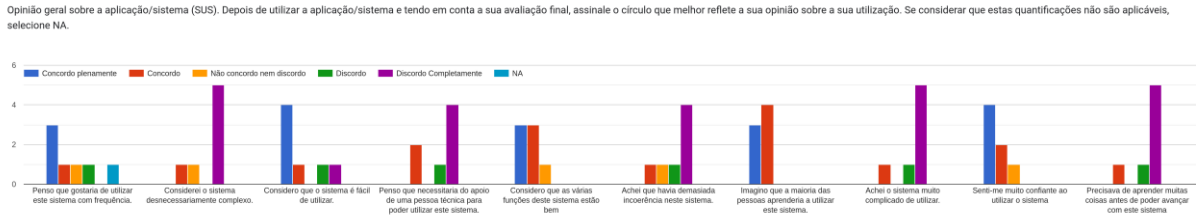


Fig. 8.1: Resultados do questionário pós-tarefas

8.2. Testes de Integração

A validação funcional do painel de administração foi realizada através de testes de integração, desenvolvidos com recurso à framework Vitest em conjunto com a biblioteca React Testing Library. Esta combinação permite simular interações reais de utilizador sobre os componentes React, verificando o comportamento da interface em resposta a ações como a filtragem de dados, a navegação entre vistas e a submissão de formulários.

Os testes executam num ambiente jsdom, que emula o modelo de objetos do documento (DOM) de um navegador web, e recorrem a respostas simuladas (mocks) da API para isolar o frontend do backend durante a fase de validação. A configuração do ambiente de testes encontra-se definida em vitest.config.ts, e os ficheiros de teste estão organizados em src/frontend/admin-panel/tests/integration/.

Foram desenvolvidos testes de integração para os doze módulos funcionais do painel de administração, conforme apresentado na Tabela 8.2.

#	Módulo	Ficheiro de Teste
1	Cursos	Courses.integration.test.tsx
2	Modalidade	Modalities.integration.test.tsx
3	Detalhes de Modalidade	ModalityDetail.integration.test.tsx
4	Equipas	Teams.integration.test.tsx
5	Jogos	Jogos.integration.test.tsx
6	Detalhes de Jogo	JogosDetails.integration.test.tsx
7	Torneios	Tournament.integration.test.tsx
8	Detalhes de Torneio	Tournament.detail.integration.test.tsx
9	Núcleo	Nucleos.integration.test.tsx
10	Sócios	Socios.integration.test.tsx
11	Membros de Staff	Staff.integration.test.tsx
12	Regulamentos	Regulamentos.integration.test.tsx

Tabela 8.2: Módulos do painel de administração cobertos por testes de integração

A título exemplificativo, o teste relativo ao módulo de Jogos valida que a lista de jogos é carregada e apresentada corretamente, que o mecanismo de filtragem por estado de jogo funciona como esperado - devolvendo uma mensagem informativa quando não existem jogos no estado selecionado - e que a transição para a vista de calendário apresenta o estado correto para o dia em curso. Os testes podem ser executados localmente através do comando 'npm run test', sendo a sua integração no pipeline de CI uma melhoria identificada para trabalho futuro.

8.3. Testes de Desempenho

Para avaliar o comportamento da API pública (/api/public) sob condições de carga variável, foi desenvolvida uma suite de testes de desempenho utilizando a ferramenta k6. A suite é composta por sete cenários distintos, cada um com objetivos e perfis de carga específicos, e incide sobre os endpoints de maior utilização esperada: classificação geral, classificação por modalidade, torneios, jogos, equipas e núcleos.

Os níveis de serviço (SLOs - Service Level Objectives) definidos como critério de aceitação para os testes são os seguintes: o percentil 95 da latência de resposta (p95) deve ser inferior a 500 milissegundos, e a taxa de erros deve ser inferior a 1%.

Os sete cenários implementados encontram-se descritos na Tabela 8.2.

Cenário	Duração	VUs	Objetivo
Smoke	30 s	1	Verificação de sanidade – todos os endpoints devem devolver HTTP 200
Latency	4 min	20	Medir p50/p95/p99 de latência por endpoint a carga estável
Throughput	7 min	5 a 80	Rampa gradual para identificar o débito maximo sustentável (RPS)
Peak	6 min	20 a 200	Simular pico realista de carga – validar SLOs
Spike	5 min	5 a 200	Surge instância sem aquecimento – testar resiliência e recuperação
Stress	7 min	30 a 400	Pressionar o sistema até o ponto de rotura – identificar o bottleneck
Soak	30 min	15	Carga sustentada para detetar fugas de memória e degradação progressiva de latencia

Tabela 8.3: Cenários da suite de testes de desempenho

O cenário de smoke constitui o ponto de partida obrigatório de qualquer execução, verificando que todos os endpoints respondem corretamente antes de sujeitar o sistema a carga. O cenário de latency recorre a vinte utilizadores virtuais em regime estável, fornecendo uma caracterização detalhada dos tempos de resposta em condições normais de operação. O cenário de peak modela o momento de maior relevância operacional da plataforma - a publicação de resultados no final de um dia de competição - simulando uma subida de vinte para duzentos utilizadores virtuais durante dois minutos, seguida de retorno gradual ao estado de carga normal. Este cenário visa confirmar que os SLOs definidos são cumpridos durante o pico de tráfego mais exigente esperado em produção. O cenário de stress incrementa progressivamente a carga entre trinta e trezentos utilizadores virtuais, com o propósito de identificar o ponto de rotura do sistema e verificar que o mesmo retoma o funcionamento normal após a re4om um limite de dez ligações ativas e vinte de overflow. O cenário de soak mantém quinze utilizadores virtuais durante trinta minutos, com o objetivo de detetar problemas de natureza temporal que apenas se manifestam sob carga prolongada, nomeadamente fugas de memória nos contentores e degradação progressiva da latência (latency rot). ‘0

Durante a execução dos testes, o estado do sistema pode ser monitorizado em tempo real através do Grafana, com dashboards dedicados para a latência HTTP, a taxa de pedidos por segundo, as ligações ativas à base de dados e o consumo de recursos por contentor. Os registos de erro são acessíveis via Loki, filtráveis por nível de severidade.

8.3.1. Resultados Obtidos

Foram realçados três testes da suite: latency, throughput (débito) e stress. A Tabela 8.3 apresenta o resumo geral dos três cenários executados.

Teste	VUs	Duração	Pedidos	Débito	Taxa de Erro	Resultados
Latency	50	3min45s	10k	42.2	0%	PASS
Débito	Até 250	11min	145k	220.59	0%	PASS
Stress	Até 400	10min30s	55k*	157.91	4.19%	FAIL

Tabela 8.3.1.1: Resumo geral dos testes de desempenho executados

** Pedidos contabilizados até rutura*

A Tabela 8.4 detalha os valores de p95 de latência por endpoint, medidos durante o teste de latency (50 VUs, carga estável).

Endpoint	p95 (ms)	SLO p95
GET /ranking/geral	10,6	< 500 ms
GET /ranking/modality	10.8	< 500 ms
GET /teams	11.2	< 500 ms
GET /nucleos	11.8	< 300 ms
GET /matches	13.2	< 500 ms
GET /tournaments	13.4	< 500 ms

Tabela 8.3.1.2: Latência p95 por endpoint (teste de latency, 50 VUs)

A análise dos resultados permite retirar as seguintes conclusões. No teste de latency, todos os endpoints cumpriram os SLOs definidos com ampla margem: o valor mais elevado de p95 foi de 13,4 ms no endpoint GET /tournaments, muito abaixo do limite de 500 ms, e o endpoint GET /nucleos (limite de 300 ms) registou 11,8 ms. A taxa de erros foi de 0%, confirmando a estabilidade do sistema a carga normal.

No teste de throughput, o sistema sustentou até 250 utilizadores virtuais em rampa progressiva, processando 145 mill pedidos a um débito máximo de 220,59 req/s, sem qualquer erro. Este valor representa uma capacidade muito superior à carga realista esperada em produção, confirmando que a infraestrutura é adequada ao volume de tráfego previsto para a plataforma.

No teste de stress, o ponto de rotura foi atingido com 291 utilizadores virtuais, registando-se uma taxa de erros de 4,18% e um débito de 157,91 req/s. O resultado é classificado como FAIL, conforme esperado para este tipo de teste, cujo objetivo é precisamente identificar o limite de capacidade do sistema e não validar SLOs. A identificação deste ponto de rotura permite dimensionar adequadamente a infraestrutura e definir estratégias de escalabilidade horizontal caso o volume de utilizadores aumente em edições futuras da Taça UA.

Os valores de latência e bottleneck registados são consistentes com respostas servidas a partir da cache Redis integrada na API pública, que armazena as respostas dos endpoints de leitura por um período configurável, eliminando assim a necessidade de consultar a base de dados PostgreSQL em cada pedido.

Durante o desenvolvimento, a base de dados PostgreSQL foi identificada como o principal bottleneck sob carga elevada, dado o pool de ligações configurado com um limite de dez ligações ativas e vinte de overflow. Para mitigar este constrangimento, foi introduzida uma cache Redis com TTLs configuráveis por tipo de recurso - de 30 segundos para listas de jogos até duas horas para núcleos e regulamentos - evitando assim que a maioria dos pedidos de leitura chegue à base de dados.

9. Conclusão

O desenvolvimento do sistema de gestão da Taça UA permitiu concretizar uma solução técnica coerente com os requisitos inicialmente levantados, validando as decisões de engenharia adotadas ao longo do projeto. A plataforma resultante apresenta uma arquitetura modular, orientada a serviços e suportada por princípios consolidados aprendidos no curso de Engenharia Informática, assegurando separação de responsabilidades, escalabilidade e facilidade de manutenção.

9.1. Resultados Obtidos

O desenvolvimento do sistema de gestão da Taça UA permitiu concretizar uma solução técnica coerente com os requisitos inicialmente levantados, validando as decisões de engenharia adotadas ao longo do projeto. A plataforma resultante apresenta uma arquitetura modular, orientada a serviços e suportada por princípios consolidados da Engenharia Informática, assegurando separação de responsabilidades, escalabilidade e facilidade de manutenção.

A nível arquitetural, a adoção de uma arquitetura multicamada, baseada em APIs REST e comunicação assíncrona por eventos, permitiu validar uma abordagem moderna e alinhada com boas práticas de sistemas distribuídos. A integração de serviços como Keycloak para autenticação e autorização, Redis para cache, PostgreSQL para persistência relacional e ferramentas de observabilidade demonstra a aplicação consistente de conceitos fundamentais de Engenharia Informática, nomeadamente segurança, desempenho e monitorização.

Adicionalmente, a definição clara de atores, personas e casos de uso revelou-se essencial para alinhar o desenvolvimento técnico com as necessidades reais dos utilizadores, contribuindo para uma solução funcionalmente adequada e conceptualmente bem estruturada.

9.2. Trabalho Futuro

Apesar dos resultados alcançados com o desenvolvimento da plataforma de gestão da Taça UA, foram identificadas linhas de trabalho que permitirão consolidar a solução e prepará-la para utilização operacional sustentada.

No plano imediato, a transição para ambiente de produção permanece como o passo mais crítico. Embora o pipeline de entrega contínua (CD) esteja implementado e operacional acionando automaticamente o deployment após cada build bem-sucedida na branch main, a ativação do ambiente de produção está dependente de confirmação por parte da Associação Académica da Universidade de Aveiro (AAUAv), entidade responsável pela operação da Taça UA. À data de conclusão deste projeto, a plataforma encontra-se em funcionamento num ambiente de staging, aguardando formalização do processo de transição para produção.

Em paralelo, identificam-se questões de natureza legal e organizacional que carecem de resolução antes da entrega definitiva do projeto. Concretamente, a transferência de titularidade do projeto para a AAUAv não foi ainda formalizada, sendo necessário definir os termos de utilização, manutenção e evolução futura da plataforma pela organização destinatária.

Ao nível funcional, identificou-se uma oportunidade de melhoria na experiência dos administradores de núcleo: a implementação de uma dashboard de tarefas pendentes (to-do dashboard) que sintetize, num único ecrã, as ações administrativas pendentes - tais como convocatórias por submeter, resultados de jogos por registar ou regulamentos em falta. Esta funcionalidade reduziria o esforço cognitivo dos administradores na gestão do dia-a-dia da competição e aumentaria a probabilidade de cumprimento atempado dos processos administrativos.

10. Referencias

[1] Federação Académica do Desporto Universitário. *Plataforma de Gestão Desportiva Universitária*

[2] System Usability Scale (SUS)

[3] Escala de Likert. *Escala usada para em questionários*

[4] Grafana K6. *Documentação do Grafana k6*

[5] NGINX. Documentação do NGINX

[6] Keycloak. Documentação do Keycloak

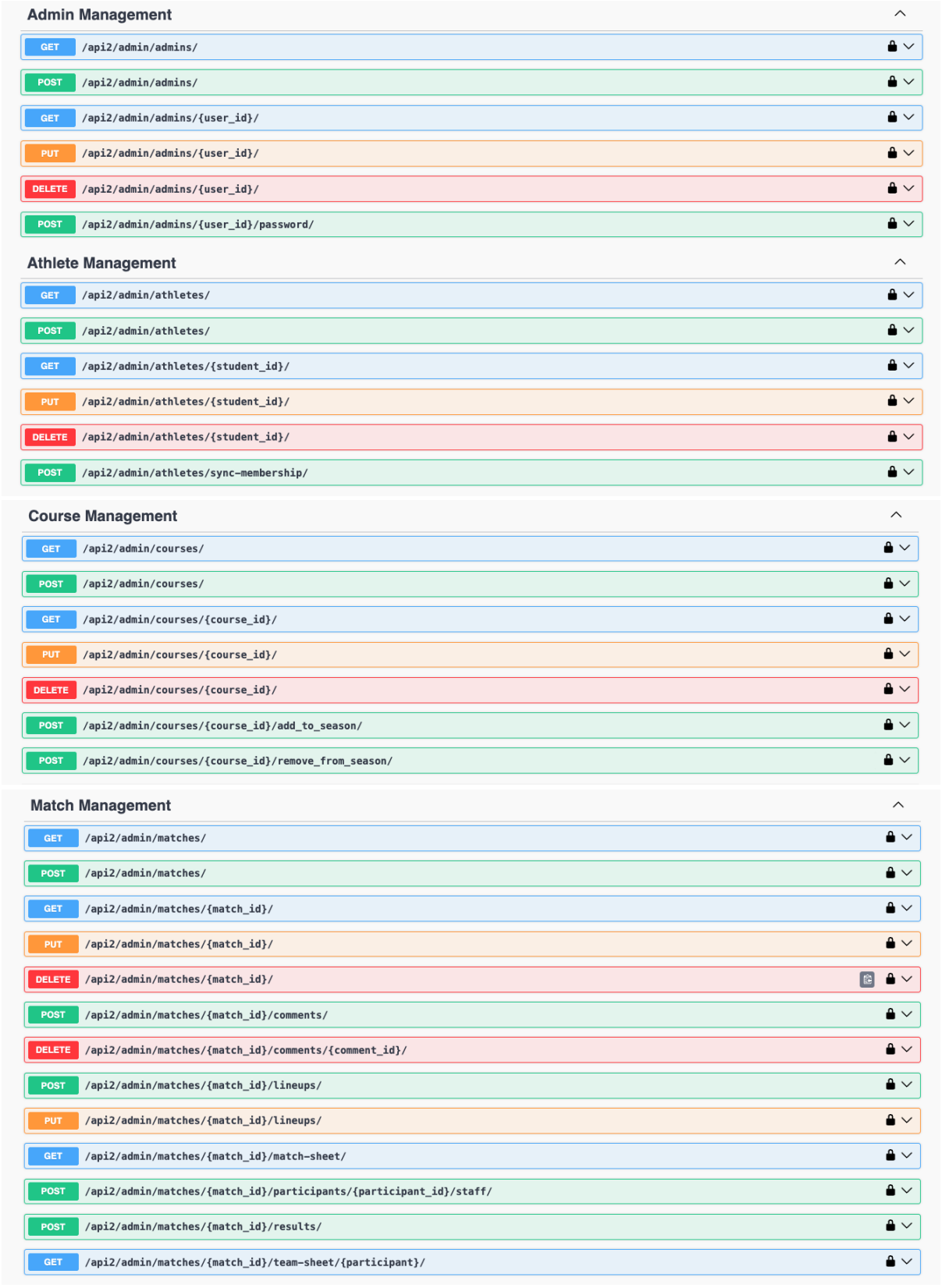
Apêndice A

Documentação da API

Public API

- GET `/teams` List all teams
- GET `/teams/{team_id}` Get team by ID
- GET `/students` List all students
- GET `/students/{student_id}` Get student by ID
- GET `/students/by-number/{student_number}` Get student by number
- GET `/tournaments` List all tournaments
- GET `/tournaments/{tournament_id}` Get tournament by ID
- GET `/tournaments/{tournament_id}/standings` Get tournament standings
- GET `/matches` List all matches
- GET `/matches/{match_id}` Get match by ID
- GET `/competitors/{competitor_id}/standings` Get competitor standings
- GET `/ranking/general` Get general ranking
- GET `/ranking/general/course/{course_id}` Get course ranking
- GET `/nucleos` List all nucleos
- GET `/nucleos/{nucleo_id}` Get nucleo by ID
- GET `/regulations` List all regulations
- GET `/ranking/modality` Get modality rankings
- GET `/ranking/modality/course/{course_id}` Get course modality rankings
- GET `/seasons` List all seasons
- GET `/metrics` Metrics
- GET `/` Read Root
- GET `/health` Health Check
- GET `/cache/stats` Get Cache Statistics
- POST `/cache/clear` Clear Cache

Admin API


Admin Management
GET /api2/admin/admins/
POST /api2/admin/admins/
GET /api2/admin/admins/{user_id}/
PUT /api2/admin/admins/{user_id}/
DELETE /api2/admin/admins/{user_id}/
POST /api2/admin/admins/{user_id}/password/
Athlete Management
GET /api2/admin/athletes/
POST /api2/admin/athletes/
GET /api2/admin/athletes/{student_id}/
PUT /api2/admin/athletes/{student_id}/
DELETE /api2/admin/athletes/{student_id}/
POST /api2/admin/athletes/sync-membership/
Course Management
GET /api2/admin/courses/
POST /api2/admin/courses/
GET /api2/admin/courses/{course_id}/
PUT /api2/admin/courses/{course_id}/
DELETE /api2/admin/courses/{course_id}/
POST /api2/admin/courses/{course_id}/add_to_season/
POST /api2/admin/courses/{course_id}/remove_from_season/
Match Management
GET /api2/admin/matches/
POST /api2/admin/matches/
GET /api2/admin/matches/{match_id}/
PUT /api2/admin/matches/{match_id}/
DELETE /api2/admin/matches/{match_id}/
POST /api2/admin/matches/{match_id}/comments/
DELETE /api2/admin/matches/{match_id}/comments/{comment_id}/
POST /api2/admin/matches/{match_id}/lineups/
PUT /api2/admin/matches/{match_id}/lineups/
GET /api2/admin/matches/{match_id}/match-sheet/
POST /api2/admin/matches/{match_id}/participants/{participant_id}/staff/
POST /api2/admin/matches/{match_id}/results/
GET /api2/admin/matches/{match_id}/team-sheet/{participant}/

Modality Management

- GET /api2/admin/modalities/
- POST /api2/admin/modalities/
- GET /api2/admin/modalities/{modality_id}/
- PUT /api2/admin/modalities/{modality_id}/
- DELETE /api2/admin/modalities/{modality_id}/
- PUT /api2/admin/modalities/{modality_id}/remove-from-season/
- PUT /api2/admin/modalities/{modality_id}/update-regulation/

Modality Ttype Management

- GET /api2/admin/modality-types/
- POST /api2/admin/modality-types/
- GET /api2/admin/modality-types/{modality_type_id}/
- PUT /api2/admin/modality-types/{modality_type_id}/
- DELETE /api2/admin/modality-types/{modality_type_id}/
- GET /api2/admin/modality-types/minimal/

Nucleo Management

- GET /api2/admin/nucleos/
- POST /api2/admin/nucleos/
- GET /api2/admin/nucleos/{nucleo_id}/
- PUT /api2/admin/nucleos/{nucleo_id}/
- DELETE /api2/admin/nucleos/{nucleo_id}/

Regulation Management

- GET /api2/admin/regulations/
- POST /api2/admin/regulations/
- GET /api2/admin/regulations/{regulation_id}/
- PUT /api2/admin/regulations/{regulation_id}/
- DELETE /api2/admin/regulations/{regulation_id}/

Season Management

- GET /api2/admin/seasons/
- POST /api2/admin/seasons/
- GET /api2/admin/seasons/current/
- GET /api2/admin/seasons/summary/

Staff Management

- GET /api2/admin/staff/
- POST /api2/admin/staff/
- GET /api2/admin/staff/{staff_id}/
- PUT /api2/admin/staff/{staff_id}/
- DELETE /api2/admin/staff/{staff_id}/

Team Management

- GET /api2/admin/teams/
- POST /api2/admin/teams/
- GET /api2/admin/teams/{team_id}/
- PUT /api2/admin/teams/{team_id}/
- DELETE /api2/admin/teams/{team_id}/

Tournament Management

- GET /api2/admin/tournaments/
- POST /api2/admin/tournaments/
- GET /api2/admin/tournaments/{tournament_id}/
- PUT /api2/admin/tournaments/{tournament_id}/
- DELETE /api2/admin/tournaments/{tournament_id}/
- PUT /api2/admin/tournaments/{tournament_id}/competitors/add/
- PUT /api2/admin/tournaments/{tournament_id}/competitors/remove/
- POST /api2/admin/tournaments/{tournament_id}/finish/
- PUT /api2/admin/tournaments/{tournament_id}/format-meta/
- GET /api2/admin/tournaments/{tournament_id}/rounds/
- GET /api2/admin/tournaments/{tournament_id}/standings/

Apêndice B

Guião do Teste de Usabilidade

Teste de Usabilidade

* Indica uma pergunta obrigatória

1. **Tarefa 1** - Fazer login como admin geral e indicar quantos administradores estão presentes no website, indicando quantos deles são administradores de núcleo (X/Y). *

2. Dificuldade da **Tarefa 1** *

   Marcar apenas uma oval.

   - ◯ 1 (Muito difícil)
   - ◯ 2 (Difícil)
   - ◯ 3 (Média)
   - ◯ 4 (Fácil)
   - ◯ 5 (Muito Fácil)

3. **Tarefa 2** - Adicionar um novo administrador de núcleo. Escolher 3 núcleos. Indicar ao observador quais cursos estão associados a esse administrador. *

   Marcar tudo o que for aplicável.

   - ☐ Completo

4. Dificuldade da **Tarefa 2** *

Marcar apenas uma oval.

- ◯ 1 (Muito difícil)
- ◯ 2 (Difícil)
- ◯ 3 (Média)
- ◯ 4 (Fácil)
- ◯ 5 (Muito Fácil)

5. **Tarefa 3** - Adicionar uma nova modalidade de um tipo à escolha.

Marcar tudo o que for aplicável.

- ☐ Completo

6. Dificuldade da **Tarefa 3** *

Marcar apenas uma oval.

- ◯ 1 (Muito difícil)
- ◯ 2 (Difícil)
- ◯ 3 (Média)
- ◯ 4 (Fácil)
- ◯ 5 (Muito Fácil)

7. **Tarefa 4** - Fazer login como admin de núcleo. Criar um membro participante novo.

Marcar tudo o que for aplicável.

- ☐ Completo

8. Dificuldade da **Tarefa 4** *

Marcar apenas uma oval.

- 1 (Muito difícil)
- 2 (Difícil)
- 3 (Média)
- 4 (Fácil)
- 5 (Muito Fácil)

9. **Tarefa 5 -** Criar uma equipa nova usando a modalidade criada previamente. Adicionar o participante novo à equipa.

Marcar tudo o que for aplicável.

- [ ] Completo

10. Dificuldade da **Tarefa 5** *

Marcar apenas uma oval.

- 1 (Muito difícil)
- 2 (Difícil)
- 3 (Média)
- 4 (Fácil)
- 5 (Muito Fácil)

11. **Tarefa 6** - Voltar ao admin geral. Criar um torneio novo com a modalidade criada. Escolher team 1 e a equipa criada como participantes. Após esta ser criada, ver detalhes e criar um jogo entre essas duas equipas. Ativa o torneio.

Marcar tudo o que for aplicável.

- [ ] Completo

12. Dificuldade da **Tarefa 6** *

Marcar apenas uma oval.

- 1 (Muito difícil)
- 2 (Difícil)
- 3 (Média)
- 4 (Fácil)
- 5 (Muito Fácil)

13. **Tarefa 7 -** Voltar ao admin de núcleo. Procurar o jogo e editar a convocatória para adicionar o participante criado. Ver a ficha de equipa. Verificar a data do jogo no calendário.

Marcar tudo o que for aplicável.

- [ ] Completo

14. Dificuldade da **Tarefa 7** *

Marcar apenas uma oval.

- 1 (Muito difícil)
- 2 (Difícil)
- 3 (Média)
- 4 (Fácil)
- 5 (Muito Fácil)

15. Comentários adicionais

Apêndice C

Guião do Observador

Guião do Observador

* Indica uma pergunta obrigatória

1. Escolher o nome do observador *

Marcar apenas uma oval.

- Bernardo
- Alexandre
- Diego
- Gonçalo
- Dinis
- Mateus

2. O utilizador conclui a tarefa: *

Marcar apenas uma oval por linha.

	Sim	Não
Tarefa 1	◯	◯
Tarefa 2	◯	◯
Tarefa 3	◯	◯
Tarefa 4	◯	◯
Tarefa 5	◯	◯
Tarefa 6	◯	◯
Tarefa 7	◯	◯

3. O utilizador completou a task com os resultados corretos: *

Marcar apenas uma oval por linha.

	Sim	Não
Tarefa 1	◯	◯

4. Tempo estimado para a completação de cada uma das tarefas *

Marcar apenas uma oval por linha.

	30s-40s	20s-30s	10s-30s	40s-50s	50s-1min	1min-2min	2min-4min
Tarefa 1							
Tarefa 2							
Tarefa 3							
Tarefa 4							
Tarefa 5							
Tarefa 6							
Tarefa 7							

5. Número de erros cometidos *

Marcar apenas uma oval por linha.

	1	2	3	4	5+	0
Tarefa 1	◯	◯	◯	◯	◯	◯
Tarefa 2	◯	◯	◯	◯	◯	◯
Tarefa 3	◯	◯	◯	◯	◯	◯
Tarefa 4	◯	◯	◯	◯	◯	◯
Tarefa 5	◯	◯	◯	◯	◯	◯
Tarefa 6	◯	◯	◯	◯	◯	◯
Tarefa 7	◯	◯	◯	◯	◯	◯

6. Severidade dos erros cometidos *

Marcar apenas uma oval por linha.

	Nenhuma	Pouca	Alguma	Muita	Extrema
Tarefa 1	◯	◯	◯	◯	◯
Tarefa 2	◯	◯	◯	◯	◯
Tarefa 3	◯	◯	◯	◯	◯
Tarefa 4	◯	◯	◯	◯	◯
Tarefa 5	◯	◯	◯	◯	◯
Tarefa 6	◯	◯	◯	◯	◯
Tarefa 7	◯	◯	◯	◯	◯

7. O utilizador ficou perdido em algum momento: *

Marcar apenas uma oval por linha.

	Sim	Não
Tarefa 1	◯	◯
Tarefa 2	◯	◯
Tarefa 3	◯	◯
Tarefa 4	◯	◯
Tarefa 5	◯	◯
Tarefa 6	◯	◯
Tarefa 7	◯	◯

8. O utilizador precisou de ajuda: *

Marcar apenas uma oval por linha.

	Sim	Não
Tarefa 1	◯	◯
Tarefa 2	◯	◯
Tarefa 3	◯	◯
Tarefa 4	◯	◯
Tarefa 5	◯	◯
Tarefa 6	◯	◯
Tarefa 7	◯	◯

9. Facilidade/Dificuldade observada na completação da task: *

Marcar apenas uma oval por linha.

	Muito Complexo	Complexo	Razoável	Simples	Muito fácil
Tarefa 1					
Tarefa 2					
Tarefa 3					
Tarefa 4					
Tarefa 5					
Tarefa 6					
Tarefa 7					

10. Observações adicionais

Apêndice D

Questionário Pós-Tarefas

Questionário pós-tarefas

Agradecemos a sua colaboração neste estudo, que tem como objetivo avaliar a interface do utilizador da aplicação/sistema administrativo da Taça UA e tentar melhorá-la de acordo com os critérios de Usabilidade.
A sua colaboração é importante para o sucesso desta avaliação, pelo que lhe pedimos que preencha este
questionário, cujos dados serão utilizados em total anonimato, apenas para fins académicos.

* Indica uma pergunta obrigatória

1. Sexo *

   Marcar apenas uma oval.

   - ◯ Feminino
   - ◯ Masculino
   - ◯ Outro

2. Idade *

   _____

3. Experiência anterior com este tipo de aplicação/sistema: *

   Marcar apenas uma oval.

   - ◯ Nenhuma
   - ◯ Alguma
   - ◯ Muita

4. Opinião geral sobre a aplicação/sistema (SUS). *
Depois de utilizar a aplicação/sistema e tendo em conta a sua avaliação final, assinale o círculo que melhor
reflete a sua opinião sobre a sua utilização. Se considerar que estas quantificações não são aplicáveis, selecione NA.

Marcar apenas uma oval por linha.

	Concordo plenamente	Concordo	Não concordo nem discordo	Discordo	Discordo Completamente
Penso que gostaria de utilizar este sistema com frequência.					
Considerei o sistema desnecessariamente complexo.					
Considero que o sistema é fácil de utilizar.					
Penso que necessitaria do apoio de uma pessoa técnica para poder utilizar este sistema.					
Considero que as várias funções deste sistema estão bem					
Achei que havia demasiada incoerência neste sistema.					
Imagino que a maioria das pessoas aprenderia a utilizar este sistema.					
Achei o sistema muito complicado de utilizar.					

Guião do Observador

* Indica uma pergunta obrigatória

1. Escolher o nome do observador *

Marcar apenas uma oval.

- Bernardo
- Alexandre
- Diego
- Gonçalo
- Dinis
- Mateus

2. O utilizador conclui a tarefa: *

Marcar apenas uma oval por linha.

	Sim	Não
Tarefa 1	◯	◯
Tarefa 2	◯	◯
Tarefa 3	◯	◯
Tarefa 4	◯	◯
Tarefa 5	◯	◯
Tarefa 6	◯	◯
Tarefa 7	◯	◯

3. O utilizador completou a task com os resultados corretos: *

Marcar apenas uma oval por linha.

	Sim	Não
Tarefa 1	◯	◯

4. Tempo estimado para a completação de cada uma das tarefas *

Marcar apenas uma oval por linha.

	30s-40s	20s-30s	10s-30s	40s-50s	50s-1min	1min-2min	2min-4min
Tarefa 1							
Tarefa 2							
Tarefa 3							
Tarefa 4							
Tarefa 5							
Tarefa 6							
Tarefa 7							

5. Número de erros cometidos *

Marcar apenas uma oval por linha.

	1	2	3	4	5+	0
Tarefa 1	◯	◯	◯	◯	◯	◯
Tarefa 2	◯	◯	◯	◯	◯	◯
Tarefa 3	◯	◯	◯	◯	◯	◯
Tarefa 4	◯	◯	◯	◯	◯	◯
Tarefa 5	◯	◯	◯	◯	◯	◯
Tarefa 6	◯	◯	◯	◯	◯	◯
Tarefa 7	◯	◯	◯	◯	◯	◯

6. Severidade dos erros cometidos *

Marcar apenas uma oval por linha.

	Nenhuma	Pouca	Alguma	Muita	Extrema
Tarefa 1	◯	◯	◯	◯	◯
Tarefa 2	◯	◯	◯	◯	◯
Tarefa 3	◯	◯	◯	◯	◯
Tarefa 4	◯	◯	◯	◯	◯
Tarefa 5	◯	◯	◯	◯	◯
Tarefa 6	◯	◯	◯	◯	◯
Tarefa 7	◯	◯	◯	◯	◯

7. O utilizador ficou perdido em algum momento: *

Marcar apenas uma oval por linha.

	Sim	Não
Tarefa 1	◯	◯
Tarefa 2	◯	◯
Tarefa 3	◯	◯
Tarefa 4	◯	◯
Tarefa 5	◯	◯
Tarefa 6	◯	◯
Tarefa 7	◯	◯

8. O utilizador precisou de ajuda: *

Marcar apenas uma oval por linha.

	Sim	Não
Tarefa 1	◯	◯
Tarefa 2	◯	◯
Tarefa 3	◯	◯
Tarefa 4	◯	◯
Tarefa 5	◯	◯
Tarefa 6	◯	◯
Tarefa 7	◯	◯

9. Facilidade/Dificuldade observada na completação da task: *

Marcar apenas uma oval por linha.

	Muito Complexo	Complexo	Razoável	Simples	Muito fácil
Tarefa 1					
Tarefa 2					
Tarefa 3					
Tarefa 4					
Tarefa 5					
Tarefa 6					
Tarefa 7					

10. Observações adicionais

Senti-me muito confiante ao utilizar o sistema						
Precisava de aprender muitas coisas antes de poder avançar com este sistema						

5. Deixe os seus comentários sobre a experiência do utilizador proporcionada pela aplicação/sistema: