UMinho

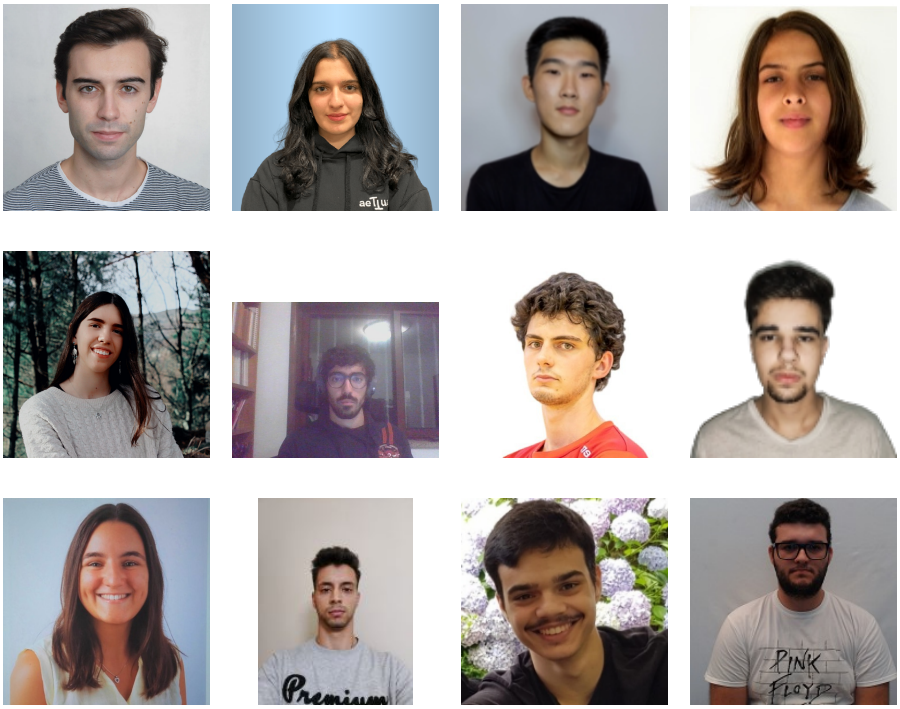
Mestrado Engenharia Informática

Requisitos e Arquiteturas de Software (2023/24)

PROBUM v.1

Grupo 1-A, PL1 / Entrega 3

Afonso Ferreira - pg52669
Catarina Costa - pg52676
Daniel Du - pg53751
Francisca Barros - pg53816
Joana Pereira - pg53895
João Alvim - pg53902
José Martins - pg53968
José Pereira - a89596
Marta Aguiar - pg52469
Telmo Oliveira - pg54247
Ricardo Araújo - pg54190
Xavier Mota - pg54276



Braga, January 5, 2024

Prefácio

Durante a fase de implementação do projeto, enfrentamos desafios notáveis, especialmente relacionados à integração das diferentes partes e à comunicação entre os microserviços. Um dos motivos destas dificuldades foi a escolha da linguagem de programação, para cada microserviço, ter sido tomada separadamente por equipes diferentes. Para superar essas dificuldades, implementamos reuniões regulares entre as equipes responsáveis por cada microserviço. O objetivo dessas reuniões era garantir a consistência nas decisões tomadas e promover a coesão entre os diferentes componentes do sistema. Assim conseguimos garantir uma implementação coesa e bem-sucedida do projeto, mesmo diante das complexidades introduzidas pela escolha separada das linguagens para cada microserviço. Por fim, analisamos o contributo de cada elemento do grupo para o desenvolvimento do trabalho. Assim, cada elemento é avaliado com 0 caso o contributo seja próximo da média, com + caso o contributo seja acima da média e - caso contrário.

Frontend	Afonso Ferreira	0
	Catarina Costa	0
	Marta Aguiar	0
Gestão de Utilizadores	Daniel Du	0
	João Alvim	0
Gestão de Provas	Francisca Barros	0
	Joana Pereira	0
	José Martins	0
Gestão de Salas	José Pereira	0
	Telmo Oliveira	0
Gestão de Notificações	Ricardo Araújo	0
	Xavier Mota	0

Contents

1 Introdução e Objetivos

Nesta 3ª do projeto, realizamos a implementação, levando em consideração o trabalho desenvolvido nas fases anteriores.

O Probum tem como principal objetivo otimizar processos e aprimorar a experiência de alunos e docentes. Em resumo, a proposta é permitir que alunos de uma unidade curricular em um curso universitário ou politécnico realizem suas avaliações acadêmicas utilizando as infraestruturas informáticas de sua própria instituição de ensino superior (IES), mesmo que estas apresentem limitações em termos de dimensão, disponibilidade e capacidade.

Nesta fase, é esperado, no mínimo, o Produto Mínimo Viável (MVP) do sistema especificado e arquitetado nas fases anteriores. O MVP deve proporcionar as seguintes funcionalidades:

- **Criação de Provas por Docentes:** Os docentes devem ser capazes de criar provas compostas por questões de escolha múltipla.
- **Realização de Provas por Alunos:** Os alunos devem poder responder a uma prova, ou seja, realizar uma avaliação.
- **Correção Automática por Docentes:** Os docentes têm a capacidade de acionar a correção automática das respostas fornecidas pelos alunos.
- **Consulta de Respostas e Avaliação por Alunos:** Os alunos podem consultar as respostas dadas a uma prova e sua respetiva avaliação.
- **Notificação aos Alunos:** Deve existir um sistema de notificação para informar os alunos quando a prova estiver disponível.

  Essas funcionalidades constituem a base do sistema e visam atender às necessidades essenciais dos docentes e alunos, proporcionando uma plataforma funcional e eficiente para a realização de avaliações acadêmicas.

1.1 Visão Geral dos Requisitos

Existem 68 requisitos funcionais e 24 requisitos não funcionais. Estes requisitos foram identificados e explicitados no relatório da Fase 1 nos capítulos denominados Requisitos Funcionais e Requisitos Não Funcionais.

2 Implementação

Nesta etapa, o grupo foi organizado em 5 subgrupos especializados. Quatro dos subgrupos concentraram-se no desenvolvimento de um microserviço específico e outro para o frontend, contribuindo assim para a criação de uma arquitetura robusta e altamente modular para a aplicação. Essa estratégia permitiu uma clara divisão de responsabilidades, promovendo o progresso simultâneo em várias partes do sistema. Em seguida serão exploradas em mais detalhe as implementações feitas para cada microserviço.

2.1 Front-end

Para a implementação do frontend, o nosso grupo decidiu utilizar a tecnologia React, por ser intuitiva e de fácil integração e implementação.

O front-end contém as páginas e funcionalidades essencias, para prefazer os use cases definidos no relatório da fase anterior.

Apesar de estar preparado para ter todas as suas funcionalidades ligadas com o *backend*, apenas parte delas se encontram conectadas.

2.2 Microserviço - Utilizadores

O microserviço dos utilizadores é o microserviço responsável pela gestão dos utilizadores, ou seja, tudo que engloba a adição, remoção e edição de um utilizador da base de dados.

Para a implementação deste microserviço foi usado *Python* e *MySqlite* como base de dados. A escolha de Python para o backend deste microserviço deve-se à sua simplicidade de aprendizagem, implementação e manutenção. Quanto à escolha de MySqlite deve-se à sua natureza relacional e transacional, propocionando uma escalabilidade robusta.

Para assegurar um microserviço robusto e coeso, foi realizado um modelo de base de dados estruturados. Com este modelo normalizado os dados ficam organizados, minimizando incoerências e redundâncias.

Nas figuras abaixo encontram-se os diagramas conceptual e lógico, respetivamente. Através do modelo conceptual, conseguimos ver os atributos que cada entidade tem e as relações entre elas. Já no modelo lógico, conseguimos observar que temos quatro tabelas, a dos utilizadores, alunos, docentes e uc. A tabela dos utilizadores corresponde a todos utilizadores do sistema, sendo que tem como atributos: id (chave primária), nome, password, email e por último o seu tipo (1-aluno, 2-docente e 3 técnico). As tabelas *aluno* e *docente* têm o id do utilizador como chave estrangeira e uma chave estrangeira para as ucs que frequenta ou leciona, respetivamente. Por último, a tabela das *ucs* tem como chave primária o id da Uc e um atributo que é a sua descrição.

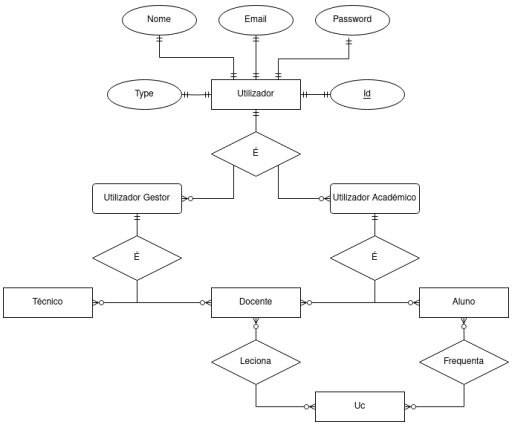


Figure 1: Modelo conceptual

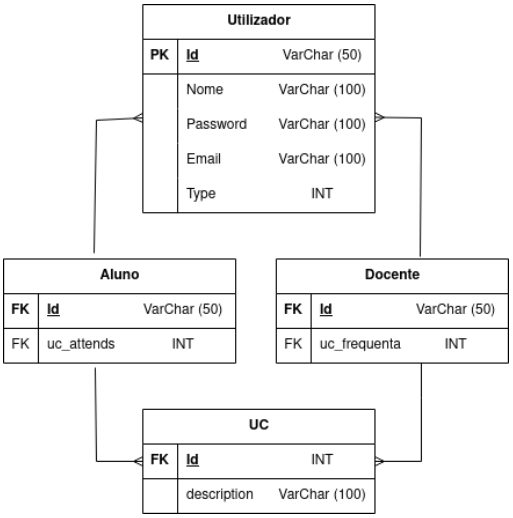


Figure 2: Modelo lógico

Método	Rota	Descrição
GET	/autenticaEmail	Autenticação um utilizador com o seu email
GET	/autenticaId	Autenticação um utilizador com o seu ID.
GET	/getResgistoGestorForm	Rota que retorna JSON com campos para preencher
GET	/getResgistoAlunoForm	Rota que retorna JSON com campos para preencher
GET	/getResgistoDocenteForm	Rota que retorna JSON com campos para preencher
GET	/registarAluno	Regista um aluno
GET	/registarDocente	Regista um docente
GET	/registarGestor	Regista um gestor
POST	/registarLista	Envia um ficheiro JSON com uma lista de dicionários para registrar docentes, alunos ou tecnicos.
GET	/getPerfil	Recebe o Perfil de um determinado utilizador
GET	/editarPassword	Permite alterar uma password de um utilizador
GET	/editarEmail	Permite alterar um email de um utilizador
GET	/getAll	Retorna de todos os utilizadores, usado em Debug

2.3 Microserviço - Notificações

O microserviço de notificações desempenha um papel importantissímo na aplicação Probum, sendo responsável por a criação e distribuição das notificações pelos diversos utilizadores.

Para a implementação deste microserviço, optamos pelo uso do Node.js e MongoDB como base de dados. A escolha do Node.js para o backend foi motivada pela familiaridade com JavaScript, eficiência no desenvolvimento e criação facilitada de uma API. Quanto ao MongoDB, escolhemos este banco de dados devido à sua natureza não relacional, o que permite a fácil implementação de uma grande diversidade de notificações.

Sobre a estrutura da *schema* das notificações, tivemos uma pequena alteração em relação ao modelo de classes da fase anterior. Nesta fase, optamos pela implementação de uma lista de várias notificações (data, título e conteúdo) relacionadas com um *número de utilizador*.

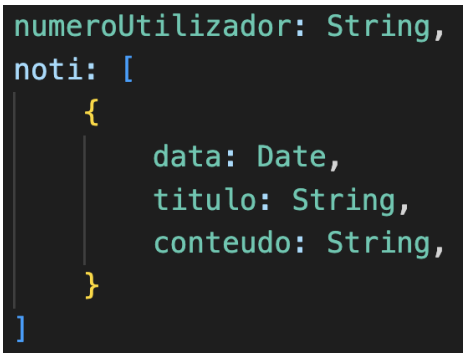


Figure 3: Nova estrutura das Notificações.

A nossa solução tem um mecanismo de criação de notificações costumizadas, ou seja, dependendo da rota de onde originou o pedido irá ser devolvida uma notificação customizada para esse evento (pedido).

Para realçar as principais rotas implementadas, elaboramos uma tabela que apresenta o método e a descrição de cada rota.

Método	Rota	Descrição
GET	/notificacao/:idUtilizador	Lista das notificações de um utilizador
DELETE	/notificacao/:idUtilizador	Remove as notificações de um utilizador
POST	/notification/registoAluno	Criação de uma notificação quando um Aluno é criado
POST	/notification/registoDocente	Criação de uma notificação quando um Docente é criado
POST	/notification/criaProva	Cria notificação/ões sobre a calendarização da prova
POST	/notification/salaPartilha	Cria notificação/ões sobre a partilha da prova pos docentes
POST	/notification/provaClassificada	Cria notificação/ões sobre a prova estar classificada
POST	/notification/provaDisponivel	Cria notificação/ões sobre a prova esta disponivel para consulta
POST	/notification/salasRemovidas	Cria notificação/ões sobre as salas serem removidas

2.4 Microserviço - Salas

O microserviço de salas desempenha um papel essencial na aplicação Probum, pois neste são implementadas funcionalidades como a de criar sala, a de reservar sala a de mostrar as salas disponiveis para um determinado dia e hora, tem tambem a funcionalidade de mostrar a combinaçao de salas para um dado dia e uma dada hora dado um numero de alunos e outras como a de eliminar sala, a de eliminar reserva, a de obter todas as salas e a de obter todas as reservas.

Método	Rota	Descrição
GET	/salas/disponiveis/<data>/<horaInicio>/<duracao>	Fornece todas as salas disponíveis para uma determinada data, horário e as respetivas capacidades
GET	/salas/disponiveisPAlunos/<alunosTotais>/<data>/<horaInicio>/<duracao>	Combinação de salas a reservar para um determinado número de alunos em uma data e horário específicos
DELETE	/salas/<idsala>	Remove uma sala
DELETE	/salas/<idreserva>	Remove a reserva de uma sala
POST	/salas/inserirReservas/<idSala>/<data>/<hora_inicio>/<hora_fim>	Adiciona a reserva de uma sala
POST	/salas/inserirSalas/<idEdificio>/<piso>/<capacidade>	Adiciona uma sala

2.5 Microserviço - Provas

O microserviço de provas desempenha um papel central na aplicação Probum, sendo responsável pela criação, realização, correção e consulta de provas acadêmicas.

Para a implementação deste microserviço, optamos pelo uso do Node.js e do MySQL. A escolha do Node.js foi motivada pela familiaridade com JavaScript, eficiência no desenvolvimento e integração facilitada com o frontend. Quanto ao MySQL, escolhemos este banco de dados devido à sua natureza relacional e transacional, proporcionando uma escalabilidade robusta.

Para assegurar uma implementação coesa e eficiente do microserviço, foi realizado um modelo de base de dados estruturado. Com a realização deste modelo é possível ter os dados melhor organizados, minimizando incoerências e simplificando o processo de implementação. A realização do mesmo é também

importante para criar uma estrutura robusta, reduzindo a probabilidade de alterações significativas no futuro, proporcionando uma base estável e confiável para o desenvolvimento contínuo do microserviço. Conforme o grupo foi avançando e comunicando com outros microserviços foram sendo feitas alterações, ficando no final com este modelo:

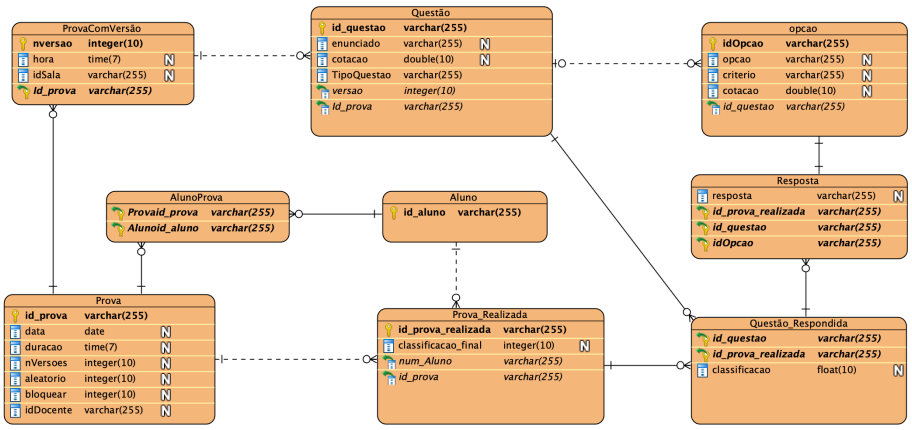


Figure 4: Diagrama relacional sobre o microserviço das provas.

Assim conseguimos ver que houve mudanças em relação ao diagrama de classes feito na fase anterior, necessárias de modo a facilitar a posterior implementação e mais coeso.

Primeiro o grupo começou por criar a tabela **Prova** que serve como a entidade principal para registrar os detalhes gerais de cada prova. Esta armazena informações fundamentais como o seu ID, a data, a duração, o número de versões, além de indicadores como se a prova é aleatória e se é bloqueada (não poder voltar atrás nas perguntas). Esta tamém contém o id do Docente que a criou.

A tabela **ProvaComVersão** foi criada para lidar com diferentes versões das provas, uma vez que cada versão terá as suas próprias questões. Esta tabela está relacionada à tabela *Prova* por meio do identificador da prova.

A tabela **Questão** é responsável por armazenar detalhes das questões associadas às provas. Cada questão é identificada exclusivamente pelo *id_questao* e está vinculada à tabela **ProvaComVersao** para indicar a versão específica à qual pertence. Tem também informação quanto à descrição e o valor da questão.

A tabela **Opção** trata das opções disponíveis para as questões. Ela contém informações sobre cada opção, incluindo identificador único (*idopcao*), o enunciado da mesma (opção), o critério e quanto vale. O critério é uma String de modo a funcionar para qualquer tipo de questão. Quando é o caso da escolha múltipla uma opção selecionada conta como *True* e uma não seleciona conta como *False*.

A tabela **Prova_realizada** regista uma prova realizada por um dado aluno, contendo o número do mesmo, a classificação final e referência à prova original. Uma prova prova_realizada só é inserida na tabela após o aluno responder a uma dada prova. Esta é inserida com a classificação final a NULL, e só posteriormente, após a correção da prova é que é alterado esse valor. A classificação final é calculado através da soma das cotações de cada questão. A chave estrangeira *id_prova* está vinculada à tabela *Prova*

A tabela **Questão_Respondida** armazena as questões específicas respondidas em uma prova realizada, juntamente com a classificação obtida, inicialmente a NULL, calculado só posteriormente após a correção da prova. O cálculo é feito através da soma das cotações obtidas em cada opção. As chaves estrangeiras **id_prova_realizada** e **id_questao** garantem a ligação com as tabelas **Prova_realizada** e **Questao**.

A tabela **Resposta** regista as respostas dadas pelos alunos para cada opção de uma dada questão.É através da comparação da resposta guardada nesta tabela com o critério da opção que é calculado a classificação de uma dada questão. Esta tabela conecta as tabelas **prova_realizada**, **Questao** e **Opcao** através de chaves estrangeiras.

Por fim foi acrescentada ainda a **AlunoProva** que contém todas as provas em que o aluno está registado.

De modo a destacar as principais rotas implementadas, criámos uma tabela com o método e descrição de cada rota. Essas rotas abrangem desde a consulta de informações específicas até ações como responder a provas, correção automática e gestão de dados.

Método	Rota	Descrição
GET	/provas/docente/<idD>	Lista das provas de um docente
GET	/provas/<idP>/versao/<nV>	Informações sobre versão de prova
GET	/provas/<idP>/versao/<nV>/questoes	Lista de questões de uma versão
GET	/provas/<idP>/versao/<nV>/questoes/<idQ>	Informação de uma questão específica
GET	/provas/prova_realizada/<idP>	Prova realizada para consulta
GET	/provas/aluno/<idA>	Lista de provas e o seu estado
GET	/provas/<idP>	Informações de prova
POST	/provas/<idP>/responder	Respostas do aluno
POST	/provas/	Adicionar provas à base de dados
POST	/provas/<idP>/versao/<nV>/questoes	Adicionar questões a uma versão da prova
PUT	/provas/<idP>/corrigir	Corrige uma prova automaticamente
PUT	/provas/docente/<idD>/corrigir	Corrige todas as provas de um docente automaticamente
DELETE	/provas/<idP>	Apaga um prova
DELETE	/provas/<idP>/versao/<nV>	Apaga uma versão da prova
DELETE	/provas/<idP>/versao/<nV>/questoes/<idQ>	Apaga uma questão da prova
DELETE	/provas/<idP>/versao/<nV>/opcoes/<idO>	Apaga uma opcão de uma questão

3 Grau de Cobertura

Este capítulo é dedicado à avaliação da solução proposta em termos de sua abrangência em relação aos requisitos estabelecidos. A mensuração do grau de cobertura é fundamental para compreender quão completamente a solução atende tanto aos requisitos funcionais quanto aos não funcionais. Para isso foi realizada uma tabela com cada requisito, indicando a prioridade e se foi ou não satisfeito pela implementação. Essa abordagem oferece informações claras sobre a eficácia da solução, permitindo a identificação de potencial áreas de aprimoramento futuro.

Requisito	**Prioridade**	**Feito**
F1	Must	X
F2	Could	X
F3	Must	X
F4	Must	X
F5	Must	X
F6	Must	
F7	Must	X
F8	Must	X
F9	Must	
F10	Must	X
F11	Must	
F12	Should	X
F13	Could	X
F14	Must	X
F15	Must	X
F16	Must	

Requisito	**Prioridade**	**Feito**
F17	Must	X
F18	Must	
F19	Must	X
F20	Should	
F21	Must	X
F22	Must	X
F23	Must	X
F24	Must	
F25	Must	
F26	Must	X
F27	Could	
F28	Should	
F29	Must	X
F30	Must	X
F31	Must	
F32	Could	

Requisito	Prioridade	Feito
F33	Must	
F34	Must	
F35	Must	
F36	Must	
F37	Must	
F38	Must	
F39	Must	
F40	Must	
F41	Must	
F42	Should	
F43	Should	
F44	Could	
F45	Could	
F46	Could	
F47	could	
F48	Must	X
F49	Must	X
F50	Must	X

Requisito	Prioridade	Feito
F51	Must	X
F52	Must	X
F53	Must	
F54	Must	
F55	Must	
F56	Must	
F57	Must	X
F58	Must	X
F59	Must	
F60	Should	X
F61	Should	X
F62	Must	X
F63	Must	X
F64	Could	X
F65	Must	
F66	Should	
F67	Should	X
F68	Should	X

Requisito	Prioridade	Feito
NF1	Must	X
NF2	Could	X
NF3	Must	X
NF4	Must	X
NF5	Must	X
NF6	Must	
NF7	Must	X
NF8	Must	
NF9	Must	
NF10	Must	
NF11	Must	
NF12	Should	

Requisito	Prioridade	Feito
NF13	Could	X
NF14	Must	
NF15	Must	X
NF16	Must	
NF17	Must	
NF18	Must	
NF19	Must	
NF20	Should	
NF21	Must	
NF22	Must	
NF23	Must	
NF24	Must	X

Através da análise das tabelas acima é possível identificar que dos 68 requisitos funcionais identificados, foi possível realizar com sucesso 34. Dentro desses 34, foram cumpridos 26 com prioridade *Must*, 3 de prioridade *Could* e 5 de prioridade *Should*. Acreditamos que alguns requisitos não estão bem classificados em relação à sua prioridade devido ao facto que, para cumprir determinados requisitos de prioridade *Must* tiveram que ser realizados alguns requisitos de prioridade inferior (*Should* e *Could*).

Acrescentamos também que o *frontend* está preparado para a realização de alguns dos requisitos definidos, assim como algumas funcionalidades do *backend*, porém, a ligação entre os mesmos não foi cumprida, portanto, a elaboração de certos requisitos não foi possível de ser completada.

Ao observarmos os requisitos não funcionais, dos 24 identificados, 9 foram efetivamente atendidos. Destes, 7 foram de prioridade *Must* e apenas 2 foram de prioridade *Could*.

Assim conlui-se que, apesar da dificuldade de elaborar todos os requisitos funcionais, aplicamos um foco nos requisitos com maior grau de prioridade (prioridade *Must*), tendo sido a maioria destes de prioridade máxima.

4 Conclusão

Ao longo deste trabalho, empenhamo-nos na concepção, desenvolvimento e avaliação da nossa solução, focando especialmente na integração e comunicação eficiente entre microserviços.

Durante a avaliação do grau de cobertura dos requisitos, constatámos um desempenho global positivo. Os dados refletem a realização eficaz de uma considerável quantidade de requisitos funcionais e não funcionais, destacando a solidez do desenvolvimento em várias áreas cruciais do projeto.

Reconhecemos que, embora tenhamos obtido êxito em muitos aspectos, houve desafios específicos no desenvolvimento de requisitos classificados como *must*. Algumas dessas necessidades essenciais não foram desenvolvidas, evidenciando áreas de ajuste futuro.

No entanto, é importante enfatizar que, no geral, estamos contentes com os resultados alcançados. A solução apresenta um equilíbrio notável entre as funcionalidades desejadas e a eficácia na comunicação entre microserviços.

Este trabalho proporcionou uma base sólida para futuras iterações e aprimoramentos, destacando a nossa dedicação em alcançar uma solução eficiente e adaptável às necessidades em constante evolução.