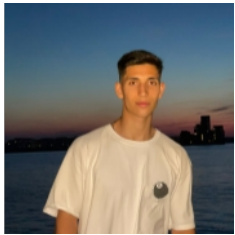


**Universidade do Minho**  
Mestrado em Engenharia Informática

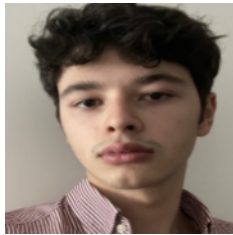
# **Unidade Curricular de Requisitos e Arquiteturas de Software**

Ano Letivo de 2024/2025

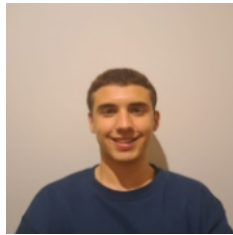
## **Grupo E**



**Filipe Gonçalves**  
PG55940



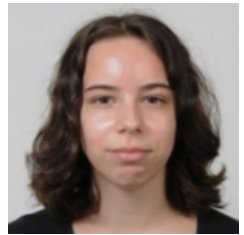
**José Correia**  
PG55967



**Leonardo Barroso**  
PG55974



**Lucas Oliveira**  
PG57886



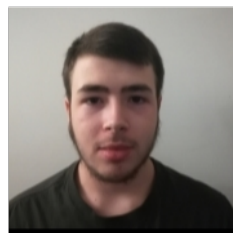
**Marta Rodrigues**  
PG55982



**Pedro Viana**  
PG57895



**Rafael Gomes**  
PG56000



**Ricardo Silva**  
PG55626



**Rui Lopes**  
PG56009

## Resumo

Este relatório tem como objetivo apresentar o processo de desenvolvimento e implementação do PictuRAS, que está a ser desenvolvido no âmbito da unidade curricular de Requisitos e Arquiteturas de Software, do Mestrado em Engenharia Informática, da Universidade do Minho.

Nesta terceira etapa, o foco incide sobre a implementação do sistema planeado na fase anterior. Abordando aspetos essenciais como o desenho e a integração dos microsserviços que compõem a arquitetura do sistema, a comunicação entre os componentes através de APIs e outros mecanismos, a gestão de dados e de estados, bem como os mecanismos de segurança e escalabilidade. Serão também detalhados os testes realizados para validar o funcionamento do sistema e garantir a sua conformidade com os requisitos definidos anteriormente.

**Área de Aplicação:** Processamento de imagens.

**Palavras-Chave:** Engenharia de *Software*, Desenho e Arquitetura de *Software*, Planeamento de *Software*, Modelação de Sistemas, *Unified Modelling Language*, Arquiteturas de Microsserviços, Comunicação Assíncrona, Aplicações para a Web, Bases de Dados, RabbitMQ, Vue.js, Python, Elixir, Structured Query Language.

# Índice

<b>1. Preâmbulo</b>	<b>1</b>
1.1. Requisitos do Sistema	1
1.2. Requisitos Funcionais	1
1.3. Requisitos Não Funcionais	3
1.4. Grau de Completude	5
<b>2. Estratégia da Solução</b>	<b>6</b>
2.1. Identificação de Microserviços	6
2.2. Sistema de Dados	8
<b>3. Implementação</b>	<b>13</b>
3.1. Frontend	13
3.2. API Gateway	13
3.3. Microserviço de Utilizadores	14
3.4. Microserviço de uma ferramenta	15
3.5. Microserviço de Projetos	16
3.6. Message Broker	18
3.7. Bases de Dados	18
3.8. Diagrama Arquitetural Final	19
<b>4. Páginas</b>	<b>20</b>
4.1. Login	20
4.2. Registo	21
4.3. Página Principal	22
4.4. Edição de Projeto	22
4.5. Modals	23
4.6. Mobile	25
<b>5. Organização da Equipa</b>	<b>26</b>
<b>6. Deployment</b>	<b>27</b>
<b>7. Reflexão</b>	<b>28</b>
7.1. Pontos Positivos	28
7.2. Pontos Negativos	28
7.3. Trabalho Futuro	28
7.4. Conclusão	29

# 1. Preâmbulo

## 1.1. Requisitos do Sistema

Durante a fase inicial deste projeto, foi elaborado um documento de requisitos cujo foco incidu sobre o domínio do problema. Posteriormente, a equipa de trabalho definiu os requisitos funcionais e não funcionais indispensáveis para garantir, no mínimo, a implementação de uma versão estável e funcional do PictuRAS.

Na última etapa deste projeto, a equipa apostou os seus esforços na realização e concretização do MVP do PictuRAS, de forma a garantir que a aplicação fosse estável, funcional e capaz de atender aos requisitos essenciais definidos. O objetivo foi criar uma versão que proporcionasse uma experiência de utilização fluída e eficiente, permitindo validar as funcionalidades principais. Esta abordagem visou possibilitar a identificação de melhorias e ajustes necessários para as próximas iterações, assegurando, assim, uma base sólida para o desenvolvimento contínuo da plataforma.

## 1.2. Requisitos Funcionais

De seguida encontra-se a lista dos requisitos funcionais que compõem o MVP expectável do PictuRAS.

Requisito	Descrição	Prioridade	Concluído
Req1	O utilizador autentica-se utilizando as suas credenciais.	Must	Sim
Req2	O utilizador anónimo regista-se.	Must	Não
Req3 e Req4	O utilizador escolhe o plano de subscrição (gratuito ou <i>premium</i> ).	Must	Sim
Req6	O utilizador cria um projeto.	Must	Sim
Req7	O utilizador lista os seus projetos.	Must	Sim
Req8	O utilizador acede à área de edição de um projeto.	Must	Sim
Req9	O utilizador carrega imagens para um projeto.	Must	Sim
Req11	O utilizador adiciona uma ferramenta de edição ao projeto.	Must	Sim
Req12	O utilizador desencadeia o processamento de um projeto.	Must	Sim

Requisito	Descrição	Prioridade	Concluído
Req13	O utilizador transfere o resultado de um projeto para o dispositivo local	Must	Sim
Req18	O utilizador registado acede às suas informações de perfil.	Should	Não
Req21	O utilizador <i>premium</i> cancela a sua subscrição <i>premium</i> .	Should	Sim
Req22	O utilizador registado termina a sua sessão.	Must	Sim
Req25	O utilizador recorta manualmente imagens.	Must	Sim
Req26	O utilizador escala imagens para dimensões específicas.	Must	Sim
Req27	O utilizador adiciona bordas a imagens.	Must	Sim
Req29	O utilizador ajusta o brilho a imagens.	Must	Sim
Req30	O utilizador ajusta o contraste a imagens.	Must	Sim
Req32	O utilizador roda imagens.	Must	Sim
Req35	O utilizador aplica um algoritmo de recorte automático de imagens com base no seu conteúdo.	Must	Sim

Tabela 1: Lista tabular dos requisitos funcionais prioritários

Para além dos requisitos considerados essenciais, foram, anteriormente, identificados outros requisitos que poderiam ser ótimas adições a iterações futuras da aplicação.

Requisito	Descrição	Prioridade	Concluído
Req17	O utilizador cancela o processamento de um projeto durante a sua execução.	Should	Não
Req19	O utilizador edita o seu perfil.	Should	Não

Tabela 2: Lista tabular dos requisitos funcionais ideias para iterações futuras

Adicionalmente, aos requisitos funcionais implementados, foram incluídos certos requisitos que não constavam na anterior fase como prioritário, mas que acabamos por incluir instintivamente.

Requisito	Descrição	Prioridade
Req10	O utilizador remove uma imagem do projeto.	Must
Req15	O utilizador altera a ordem das ferramentas de um projeto.	Must
Req16	O utilizador altera os parâmetros das ferramentas.	Must
Req37	O utilizador aplica um algoritmo de reconhecimento de objetos em imagens.	Must

Tabela 3: Lista tabular dos requisitos funcionais não prioritários, mas implementados

### 1.3. Requisitos Não Funcionais

Agora, apresentamos a lista dos requisitos não funcionais que compõem o MVP do PictuRAS, tendo a equipa selecionado aqueles que considera terem maior impacto no desempenho e na experiência do utilizador. Esta seleção foi realizada a partir de várias perspetivas, com o objetivo de garantir que as características não funcionais escolhidas sejam devidamente priorizadas na arquitetura do sistema.

Requisito	Descrição	Tipo	Prioridade	Concluído
RNF5	A página de um projeto distingue visualmente entre as ferramentas básicas e avançadas.	Usabilidade	Must	Sim
RNF7	O utilizador cria um projeto implicitamente ao arrastar ficheiros para o <i>dashboard</i> da aplicação.	Usabilidade	Must	Sim
RNF8	O utilizador carrega imagens arquivadas num único ficheiro .zip.	Usabilidade	Must	Sim
RNF11	O utilizador é mantido informado acerca do estado de processamento de um projeto em tempo real.	Usabilidade	Must	Não
RNF14	A visualização de imagens grandes ou pequenas é auxiliada pelo utilitário zoom.	Usabilidade	Must	Sim
RNF15	A visualização de imagens permite que se navegue em todas as duas direções arrastando o rato.	Usabilidade	Must	Sim
RNF18	A aplicação é compatível com diferentes plataformas e browsers, incluindo de dispositivos móveis e <i>desktop</i> .	Usabilidade	Must	Sim

Requisito	Descrição	Tipo	Prioridade	Concluído
RNF19	A aplicação fornece <i>feedback</i> visual claro e imediato ao utilizador em caso de erro ou falha durante um procedimento demorado.	Usabilidade	Should	Sim
RNF22	O sistema deve processar até 100 imagens ao mesmo tempo, sem quebras perceptíveis no desempenho.	Escalabilidade e Elasticidade	Must	Sim
RNF23	A aplicação deve ser capaz de escalar horizontalmente de forma elástica para suportar o aumento de utilizadores e volume de processamentos, mantendo o desempenho mas também os custos controlados.	Escalabilidade e Elasticidade	Must	Não
RNF25	A aplicação deve ser integrável com outras plataformas e serviços de terceiros.	Extensibilidade	Must	Sim
RNF28	A aplicação deve ser facilmente estendida com novas ferramentas de edição.	Extensibilidade	Must	Sim
RNF32	O sistema deve ser projetado para facilitar a execução de testes.	Manutenibilidade	Must	Sim
RNF33	A aplicação deve realizar <i>backups</i> automáticos dos dados e imagens dos utilizadores.	Manutenibilidade	Should	Não

Tabela 4: Lista tabular dos requisitos não funcionais prioritários

Para além destes requisitos não funcionais prioritários, acabamos por implementar também os seguintes requisitos.

Requisito	Descrição	Prioridade
RNF1	A aplicação deve conter uma interface simples e clara.	Must
RNF10	A listagem dos projetos é pesquisável por nome.	Could

Tabela 5: Lista tabular dos requisitos não funcionais não prioritários, mas implementados

## 1.4. Grau de Completude

Para avaliar o grau de completude da solução, nesta secção apresentaremos o rácio entre o número de requisitos funcionais e não funcionais implementados e o número total de requisitos definidos no documento de requisitos, abrangendo tanto os requisitos funcionais quanto os não funcionais.

	Requisitos Funcionais	Requisitos Não Funcionais
Requisitos implementados	23	13
Requisitos totais	27	16
Rácio	85.18%	81.25%

Tabela 6: Grau de cobertura da solução

Do total de 27 requisitos funcionais definidos para o PictuRAS, foram implementados com sucesso 23, o que resulta em um rácio de completude de aproximadamente 85%.

No que diz respeito aos requisitos não funcionais, a equipa conseguiu implementar 13 dos 16 requisitos planeados, alcançando um rácio de 81%.

Embora a equipa de desenvolvimento tenha alcançado um progresso significativo em ambos os tipos de requisitos, não foi possível implementar todos os requisitos previstos inicialmente. Essa limitação deve-se principalmente a restrições temporais que afetaram o ritmo de desenvolvimento, impossibilitando a execução de algumas funcionalidades e características planeadas.

No entanto, apesar dessas lacunas, a equipa está satisfeita com os resultados alcançados até o momento. A aplicação alcançou um bom nível de maturidade e estabilidade, atendendo à maioria das necessidades essenciais, tanto funcionais quanto não funcionais.



## 2. Estratégia da Solução

A aplicação PictuRAS segue uma arquitetura de microsserviços, o que permite um alto grau de escalabilidade, flexibilidade e manutenção simplificada.

Esta abordagem facilita o desenvolvimento modular, onde cada serviço é responsável por uma funcionalidade específica, podendo ser implementado, testado e implementado de forma independente. Além disso, a arquitetura de microsserviços contribui para uma melhor gestão de falhas, já que cada serviço opera de maneira isolada, minimizando o impacto de eventuais problemas.

Como forma de atender a alguns requisitos específicos de desempenho e comunicação em tempo real, a equipa decidiu incorporar nuances de uma arquitetura event-driven. Para isso, será utilizada uma fila de mensagens (*message broker*) que facilitará a comunicação assíncrona entre alguns serviços, garantindo maior eficiência na troca de informação e um nível de *coupling* muito mais baixo.

### 2.1. Identificação de Microsserviços

#### 2.1.1. Fase Arquitetural

Durante a fase arquitetural (anterior) foram definidos quatro tipos de microsserviços, nomeadamente e, passamos a citar:

##### **Microsserviço de Subscrições**

Este microsserviço será responsável pela gestão das subscrições, e respetivos pagamentos, dos utilizadores. Este, terá que se certificar que tudo ocorre em conformidade e os utilizadores ficam satisfeitos.

##### **Microsserviço de Utilizadores**

Na vasta maioria dos casos, as arquiteturas de microsserviços são acompanhadas por um componente denominado Proxy ou Gateway, que é responsável por encaminhar os pedidos realizados pelos clientes para os diferentes microsserviços.

Deste modo, a equipa de trabalho considera que a gestão de utilizadores deve estar dividida entre duas componentes da solução final: um microsserviço totalmente dedicada à criação, edição e remoção de utilizadores e uma camada de autenticação e autorização presente no dito Gateway. Esta decisão visa minimizar ao máximo a carga que poderia ser imposta no microsserviço de utilizadores caso a autenticação de um utilizador sempre dependesse dele.

##### **Microsserviço para cada uma das ferramentas**

Neste caso, tratam-se de vários microsserviços, com um dedicado a cada ferramenta existente no sistema.

Primeiramente, desta forma é muito mais simples e eficaz escalar cada uma das ferramentas individualmente, e de acordo com a necessidade dos utilizadores num dado período do tempo.

Depois, esta separação permite o desenvolvimento de cada uma das ferramentas em paralelo, por equipas especializadas distintas e em tecnologias diferentes.

Finalmente, a extensão do sistema para novas e futuras ferramentas também se constitui significativamente mais fácil, exigindo apenas a criação e configuração de um novo microsserviço.

### **Microserviço de Projetos**

Este microserviço será responsável pela gestão dos projetos dos utilizadores. Desde a criação, edição e remoção até ao manuseamento, armazenamento e processamento das imagens.

### **2.1.2. Fase de Desenvolvimento**

A equipa de desenvolvimento decidiu não implementar o microserviço de subscrições nesta fase inicial da aplicação. A principal razão para essa escolha foi a avaliação de que, para a versão inicial, esse componente não era essencial.

Essa decisão permitiu à equipa concentrar-se em outros aspetos da aplicação que eram mais críticos para garantir a funcionalidade e estabilidade do sistema, assegurando uma base sólida antes de expandir para funcionalidades adicionais.

Além do microserviço de subscrições, o microserviço de projetos também passou por algumas alterações. A equipa de desenvolvimento decidiu adicionar uma funcionalidade extra, que permite ao criador de um projeto convidar utilizadores registados na aplicação para visualizar e fazer *download* do projeto.

Os demais microserviços foram implementados conforme planeado.

## 2.2. Sistema de Dados

Tal como na vasta maioria das aplicações *web* atuais, a conceção de um sistema de dados, e consequente existência de uma base de dados é crucial. O PictuRAS não é exceção.

Além disso, uma base de dados potencia a capacidade de gerar relatórios e estatísticas que possuem informações úteis para uma análise mais profunda da utilização, e seus padrões, da plataforma. Por exemplo, poderão ser geradas estatísticas que informem qual a ferramenta mais utilizada no último mês.

Finalmente, as bases de dados oferecem implementações de armazenamento muito eficientes, o que permite uma consulta rápida dos mesmos, fornecendo, assim, uma experiência final mais agradável ao utilizador.

Durante a fase anterior, a equipa de trabalho optou por desenvolver um diagrama ER, usando a notação Crow's Foot. Este diagrama permite obter uma visão geral de como se relacionam as diferentes tabelas de dados do sistema.

O mesmo apresentava, inicialmente, a seguinte estrutura.

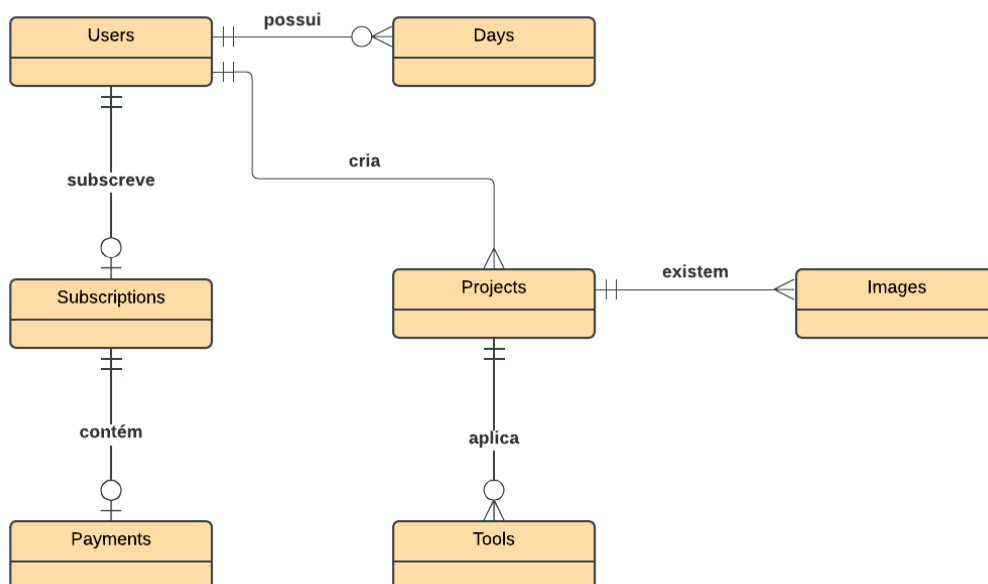


Figura 10: Diagrama ER - Fase Arquitetural

Como já mencionado anteriormente existem funcionalidades e microserviços que não foram implementados e como tal o diagrama ER inicialmente definido sofreu algumas alterações.

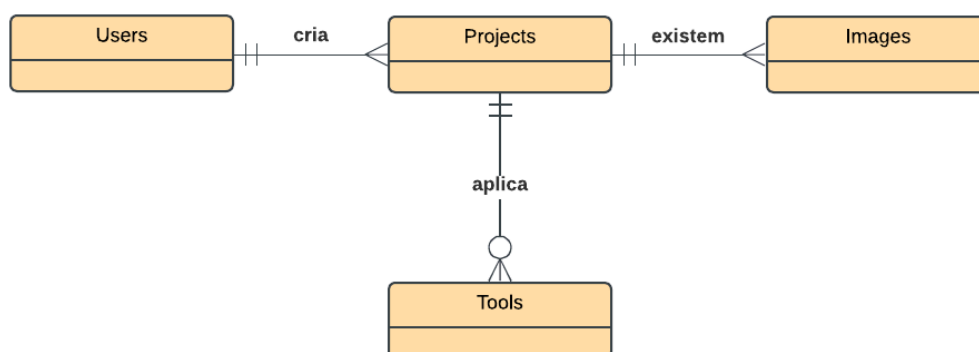


Figura 11: Diagrama ER - Fase de Desenvolvimento

Segue-se uma lista dos modelos de dados necessários em cada um dos microsserviços apresentados, acompanhada de um exemplo prático para cada um deles. Os campos destacados a laranja correspondem às chaves primárias, enquanto os campos destacados a cinzento correspondem às chaves estrangeiras.

### 2.2.1. Utilizadores

Neste caso, existirá apenas uma tabela dedicada aos utilizadores do sistema. De realçar que esta tabela deverá suportar a existência de utilizadores anónimos. Futuramente, deverá existir uma forma de contabilizar os processamentos realizadas pelo utilizador num dado dia.

Users		
Campo	Tipo	Descrição
id	uuid	Identificador único de um utilizador
name	varchar(200)	Nome do utilizador
email	unique citext	Email do utilizador
hashed_password	varchar(72)	Hash da palavra-passe do utilizador
type	enum	Tipo de utilizador: normal ( <i>default</i> ) ou premium

Tabela 7: Modelo de dados de um utilizador

Campo	Valor
id	0e6d0ce7-08c1-4de9-b7ff-82554bad32d8
name	Rui Lopes
email	mail@ruilopesm.com
hashed_password	\$2a\$12\$M.P4DK7okN/GuDhloUh6B.Qorkv0p6tJ3rrcdVx1loaDs8MUB9NPY
type	premium

Tabela 8: Exemplo de uma entrada da tabela dos utilizadores

### 2.2.2. Projetos

No que toca aos projetos existirão quatro tabelas. Primeiramente, uma tabela principal para dar conta da informação direta do projeto. A segunda, destina-se a guardar os URIs das imagens, já que um projeto pode ter várias imagens associadas. A terceira tabela é dedicada às ferramentas, para guardar, de forma ordenada, as últimas ferramentas aplicadas no projeto, juntamente com os respetivos parâmetros. Além disso, existe uma tabela dedicada a armazenar os utilizadores convidados num projeto, associando-os aos projetos aos quais têm acesso.

Projects		
Campo	Tipo	Descrição
id	uuid	Identificador único de um projeto
name	varchar(200)	Nome do projeto
state	enum	Estado do projeto: idle ( <i>default</i> ), processing, failed ou canceled
completion	int	Progresso do projeto
owner_id	uuid	Utilizador que criou o projeto

Tabela 9: Modelo de dados de um projeto

Campo	Valor
id	516d3862-2a87-4e04-baf8-5bf640b94838
name	Fotografias de Astronomia
state	processing
completion	50
owner_id	0e6d0ce7-08c1-4de9-b7ff-82554bad32d8

Tabela 10: Exemplo de uma entrada na tabela dos projetos

Images		
Campo	Tipo	Descrição
id	uuid	Identificador único da imagem
uri	text	Link para a imagem
type	enum	Valor que indica se uma é initial ou final
project_id	uuid	Projeto ao qual a imagem pertence

Tabela 11: Modelo de dados de uma imagem

Campo	Valor
id	516d3862-2a87-4e04-baf8-5bf640b94838
uri	<a href="https://s3.picturas.com/d3011aad-7c20-4f4a-8191-7749325a49ae.jpeg">https://s3.picturas.com/d3011aad-7c20-4f4a-8191-7749325a49ae.jpeg</a>

Campo	Valor
type	initial
project_id	516d3862-2a87-4e04-baf8-5bf640b94838

Tabela 12: Exemplo de entrada na tabela das imagens

Tools		
Campo	Tipo	Descrição
id	uuid	Identificador único da ferramenta
procedure	string	Nome do procedimento
position	int	Posição na lista virtual de ferramentas
parameters	map	Mapeamento do nome de um parâmetro para o valor do mesmo
project_id	uuid	Projeto ao qual a ferramenta pertence

Tabela 13: Modelo de dados de uma ferramenta

Campo	Valor
id	d3011aad-7c20-4f4a-8191-7749325a49ae
procedure	blur
position	3
parameters	{"opacity": 0.5, "method": "gauss"}
project_id	516d3862-2a87-4e04-baf8-5bf640b94838

Tabela 14: Exemplo de entrada na tabela das ferramentas

Guests		
Campo	Tipo	Descrição
id	uuid	Identificador único da ferramenta
user_id	uuid	Utilizador convidado
project_id	uuid	Projeto ao qual o utilizar convidado pertence

Tabela 15: Modelo de dados de um convidado

Campo	Valor
id	d3011aad-7c20-4f4a-8191-7749325a49ae
user_id	0e6d0ce7-08c1-4de9-b7ff-82554bad32d8
project_id	516d3862-2a87-4e04-baf8-5bf640b94838

Tabela 16: Exemplo de entrada na tabela dos convidados

## 3. Implementação

Esta secção tem por objetivo descrever o processo de implementação de cada um dos componentes do PictuRAS, bem como as funcionalidades e sacrifícios associados.

### 3.1. Frontend

Para o desenvolvimento do frontend, a equipa de trabalho decidiu utilizar a linguagem Typescript. A escolha desta linguagem em relação a Javascript prende-se no facto da mesma possuir um sistema de tipos fortes, que foi utilizado, principalmente, para “tipar” informação vinda do API Gateway, facilitando e acelerando, assim, o desenvolvimento de novas funcionalidades.

Decidiu-se, ainda, utilizar a *framework* Vue.js com o *bootstraper* e *bundler* Vite + Bun. Esta *framework* preza-se por ser moderna, simples e intuitiva. Além disso, existem imensos recursos de apoio ao desenvolvimento espalhados pela internet, fruto de ser imensamente utilizada em vários sistemas atualmente em produção.

Para a estilização das páginas existentes no sistema, utilizamos a *framework* Tailwind, para além de alguns estilos globais por via de CSS puro. No entanto, a sua utilização não foi muito intensiva, uma vez que pretendemos utilizar uma biblioteca de componentes bem estabelecida no mercado, Shadcn.

#### Interação com o API Gateway

A interação com o API Gateway, via HTTP, foi feita com recurso à biblioteca Axios.

#### Componentes

A equipa teve o cuidado de realizar uma separação clara em componentes mais pequenos e reutilizáveis em diferentes vistas (páginas da aplicação web). Estes componentes podem ser encontrados na diretoria `src/components`.

### 3.2. API Gateway

O API Gateway é um componente central na arquitetura de microsserviços, servindo como o ponto de entrada único para todos os pedidos realizados pelos clientes, desde que estes sejam via HTTP. Este, atua como um intermediário que se limita a encaminhar pedidos para os microsserviços apropriados. Além disso, o mesmo trata de garantir uma correta autenticação dos utilizadores e autorização aquando do acesso a recursos.

A implementação do API Gateway não seguiu a solução inicialmente definida. A equipe havia considerado o uso da linguagem Elixir para desenvolver este componente, mas, devido a restrições no *deployment* particularmente relacionadas à máquina virtual fornecida, essa abordagem não pôde ser utilizada.

Como alternativa, o API Gateway foi implementado utilizando NGINX para contornar essa limitação.

#### Autenticação

O processo de autenticação no API Gateway utiliza JSON Web Tokens, JWT, armazenados do lado do cliente.

O grupo optou por utilizar um sistema de chaves pública e privada como forma de garantir a segurança e a autenticidade dos tokens gerados. A chave privada é utilizada pelo microserviço de utilizadores para assinar



os tokens, garantindo que somente ele possa gerar tokens válidos. Já a chave pública, disponibilizada no API Gateway, é utilizada para validar a assinatura do token enviado pelo cliente.

Essa abordagem oferece várias vantagens, como a separação de responsabilidades e a segurança reforçada. A chave privada nunca é exposta, o que minimiza o risco de comprometimento, enquanto a chave pública pode ser amplamente distribuída sem comprometer a integridade do sistema.

Além disso, a utilização desse mecanismo permite uma escalabilidade eficiente, pois o API Gateway pode validar os tokens sem necessidade de aceder diretamente ao microserviço de utilizadores, melhorando a performance e a flexibilidade da arquitetura.

A ideia é que uma *token* guarde informação não volátil sobre a identificação do utilizador.

#### **Autorização**

Num primeiro nível, é realizada a autorização do utilizador a aceder a um determinado recurso comparando o identificador presente na rota com o identificador presente na token JWT. No entanto, nem sempre esta autorização é suficiente - por exemplo, no caso em que um utilizador convidado tenta aceder ao projeto de outro utilizador. Dessa forma, existirão níveis de autorização subsequentes em cada um dos microserviços, se necessário.

### **3.3. Microserviço de Utilizadores**

A descrição deste microserviço já foi feita em detalhe na fase anterior. Ficou decidido utilizar a linguagem Elixir (com Phoenix).

Esta escolha implica a utilização de uma arquitetura MVC (Model-View-Controller), que estrutura a interação entre os módulos de acordo com uma clara separação de responsabilidades.

Além disso, este microserviço possui um único ponto de entrada (denominada *router*), capaz de delegar os pedidos HTTP efetuados para os diferentes controladores disponíveis. Falamos, portanto, de uma espécie de *facade*.

#### **Autenticação**

Tal como referido anteriormente, este microserviço é responsável pela geração de tokens JWT. Para tal, foi utilizada a biblioteca Guardian.

#### **Passwords**

Como forma de garantir que os requisitos legais do projeto são cumpridos, as *passwords* dos utilizadores são armazenadas de forma encriptada. Para tal, recorremos ao algoritmo de *hashing* bem conhecido no mercado, o Argon2.

#### **Interação com o Sistema de Dados**

O microserviço utiliza o Ecto, uma biblioteca ORM em Elixir, para interagir com a base de dados. Este tipo de bibliotecas facilitam a integração entre estruturas de dados Elixir e tabelas SQL, agindo como um mapeador de informação.

#### **Mailer**

Apesar de não utilizado, foi implementado um sistema capaz de gerar emails HTML e enviá-los através de um servidor SMTP. Esta funcionalidade tinha como objetivo enviar emails aquando do registo de um utilizador; ou após o término do processamento de um projeto, por exemplo.

#### **Rotas**

Este microserviço é constituído por um total de quatro rotas, sendo as mesmas:

`POST /api/v1/register`

Esta rota trata-se de um pedido POST no qual é fornecido no body o nome do utilizador, o email e respetiva password. O sistema responde com o utilizador registado.

`POST /api/v1/login`

Esta rota trata-se de um pedido POST no qual é fornecido no body o email e password do utilizador a autenticar. O sistema responde fazendo *login* do utilizador, ou seja, gera e devolve um JWT que é armazenado em local storage do utilizador evitando que o mesmo sempre que aceda à aplicação tenha de fazer *login*.

Idealmente, esta implementação passaria pela utilização de cookies HTTP - retirando, de todo, a necessidade do frontend ter de lidar com autenticação. Além disso, a utilização destas cookies permitiria uma segurança redobrada, evitando ataques do tipo CSRF.

POST `/api/v1/me`

Esta rota trata-se de um pedido POST onde apenas é necessário, que quando feito o pedido, exista um token JWT válida. Caso o API Gateway valide o token, o sistema responde com as informações do utilizador respeitante.

GET `/internal/users`

Esta rota trata-se de um pedido GET no qual é fornecido no body um email. Esta trata-se de uma rota interna utilizada entre o microserviço de utilizadores e o microserviços de projetos. A existência da mesma surge devido à funcionalidade, já mencionada anteriormente, em que é possível convidar utilizadores para um projeto, exigindo validar se um email de utilizador realmente existe.

## 3.4. Microserviço de uma ferramenta

Neste caso, falamos de um microserviço responsável unicamente pelo processamento de imagens. Desta vez optamos por Python, visto que esta linguagem possui um ecossistema muito mais maduro no que toca a este quesito - manipulação de imagens.

### Bibliotecas

Optamos por escolher as bibliotecas Tesseract OCR, para o processamento de texto em imagens; Open CV e Pillow, para a manipulação mais genérica de imagens; Rembg com o único propósito de remover o *background* de imagens e YOLO para a integração de modelos complexos de *machine learning* para a deteção de objetos em imagens. Desta forma, foi possível cumprir com todos os requisitos impostos relacionados ao processamento de imagens.

### Interação com o Message Broker

As interações realizadas com o message broker fazem uso da biblioteca Pika. De facto, a equipa aproveitou a maioria do código fornecido pelos docentes, para a ferramenta que aplica marcas de água.

### 3.4.1. Lista de ferramentas desenvolvidas

As ferramentas desenvolvidas até ao momento foram as seguintes:

- Ajustar o contraste de uma imagem
- Adicionar uma borda colorida a uma imagem
- Remover o fundo de uma imagem
- Ajustar a saturação de uma imagem
- Inverter as cores de uma imagem
- Aplicar um filtro do tipo blur a uma imagem
- Ajustar o HUE de uma imagem
- Ajudar a saturação de uma imagem
- Rodar uma imagem
- Ajustar a nitidez de uma imagem
- Redimensionar uma imagem
- Recortar uma imagem
- Reconhecimento de objetos numa imagem

A ferramenta capaz de realizar OCR (detecção de texto) a uma imagem foi desenvolvida, mas não implementada ao nível do frontend. Esta decisão tem a ver com o facto do grupo não ter encontrado uma solução suficiente genérica capaz de acomodar diferentes tipos de outputs (texto, neste caso) produzidos pelas ferramentas.

## 3.5. Microserviço de Projetos

Mais uma vez, optamos por utilizar Elixir (com Phoenix).

### Interação com o Sistema de Dados

O microserviço utiliza o Ecto, uma biblioteca ORM em Elixir, para interagir com a base de dados. Este tipo de bibliotecas facilitam a integração entre estruturas de dados Elixir e tabelas SQL, agindo como um mapeador de informação.

### Interação com o Armazenamento

De forma a armazenar as imagens, a equipa de desenvolvimento optou por utilizar o armazenamento local em vez de um armazenamento externo, como o S3, que havia sido definido anteriormente.

Numa fase inicial, a equipa considerou que o armazenamento local seria mais simples de configurar e suficiente para atender às necessidades imediatas do projeto, reduzindo a complexidade e associada à integração com serviços externos.

Essa decisão também permitiu agilizar o desenvolvimento, concentrando esforços na funcionalidade principal do sistema. No entanto, a solução permanece escalável, com a possibilidade de integrar um serviço externo de armazenamento, como o S3, no futuro, caso o volume de dados ou os requisitos do projeto evoluam.

Neste momento, este armazenamento local está configurado como um volume Docker partilhado entre os diferentes microserviços que necessitam dele.

### Rotas

Este microserviço é constituído por um total de onze rotas, nomeadamente:

**POST** `/api/v1/users/:user_id/projects`

Esta rota trata-se de um pedido POST no qual poderá ser fornecido no body o nome do projeto (opcional), o sistema responde criando um projeto com o respetivo nome e associando-o ao utilizador que fez o pedido.

Caso um nome não seja fornecido, é gerado um aleatoriamente.

**GET** `/api/v1/users/:user_id/projects`

Esta rota trata-se de um pedido GET que tem como objetivo listar todos os projetos de um utilizador.

É importante destacar que, caso um utilizador seja convidado para um projeto, então esse projeto também será listado.

**GET** `/api/v1/users/:user_id/projects/:project_id`

Esta rota trata-se de um pedido GET que tem como objetivo mostrar um projeto específico de um determinado utilizador.

É importante destacar que, caso um utilizador seja convidado para um projeto, ele também terá a possibilidade de aceder às informações relacionadas a esse projeto, mas não realizar todas as operações sobre o mesmo.

**GET** `/api/v1/users/:user_id/projects/:project_id/:image_id`

Esta rota trata-se de um pedido GET que tem como objetivo fazer download de uma dada imagem, num dado projeto.

É importante destacar que, caso um utilizador seja convidado para um projeto, ele também consegue fazer download das imagens desse projeto.

`PUT /api/v1/users/:user_id/projects/:project_id`

Esta rota trata-se de um pedido PUT que contém no seu body o novo nome do projeto. O sistema altera o nome do projeto, já existente para o nome contido no body do pedido.

`DELETE /api/v1/users/:user_id/projects/:project_id`

Esta rota trata-se de um pedido DELETE que tem como objetivo apagar um projeto existente.

`POST /api/v1/users/:user_id/projects/:project_id/invite`

Esta rota trata-se de um pedido POST que contém no seu body o email do utilizador a ser convidado. O sistema, se o email for de um utilizador registado, associa esse utilizador aos convidados do projeto.

`PUT /api/v1/users/:user_id/projects/:project_id/revoke`

Esta rota trata-se de um pedido DELETE que contém no seu body o email do utilizador a ser removido. O sistema, se esse utilizador é um convidado do projeto, remove dos convidados do projeto o utilizador que está associado ao email enviado no body. A mesma não é do tipo DELETE, apesar de remover algo, visto que as normas REST não permitem que pedidos do tipo DELETE possuam um body associado.

`POST /api/v1/users/:user_id/projects/:project_id/upload`

Esta rota trata-se de um pedido POST que contém no seu body uma lista de imagens a serem adicionadas ao projeto. O sistema, caso o limite de imagens não ultrapasse os 100 MiB (limite definido pelo API Gateway), adiciona as imagens ao projeto.

De facto, caso o sistema de subscrições tivesse sido implementado, deveriam existir mecanismos de verificação associados ao limite de tamanho de uma (ou várias) imagens consoante o tipo de utilizador.

`DELETE /api/v1/users/:user_id/projects/:project_id/image_id`

Esta rota trata-se de um pedido DELETE que tem como objetivo apagar uma imagem de um projeto.

`POST /api/v1/users/:user_id/projects/:project_id/process`

Esta rota corresponde a um pedido POST que inclui no seu body uma lista ordenada das ferramentas a serem aplicadas em todas as imagens do projeto. Cada ferramenta contém, ainda, os parâmetros selecionados pelo utilizador, permitindo que o sistema aplique as configurações específicas conforme solicitado.

`GET /api/v1/users/:user_id/tools`

Esta rota corresponde a um pedido GET cujo objetivo passa por ser listar as ferramentas disponíveis para utilização no sistema. Esta lista constituiu um schema genérico (com o nome da ferramenta, descrição, parâmetros, tipos e valores *default*) que é, depois, utilizada pelo frontend para renderizar a configuração correta das ferramentas.

### Orquestrador

O orquestrador é a peça responsável pela correta gestão e realização do encadeamento de ferramentas num conjunto de imagens. O mesmo trata-se um processo de Elixir isolado (GenServer), capaz de receber chamadas assíncronas vindouras do message broker e processar de forma concorrente diversos pedidos em diversos projetos.

O estado do mesmo é definido num processo de Elixir, também, isolado (Agent) que permite a escalabilidade para vários servidores orquestradores (até em diferentes máquinas) a acederem a um mesmo estado. Isto é possível dados os mecanismos de comunicação providenciados pela máquina virtual em que o Elixir está assente, a de Erlang, BEAM.

Ainda, como forma de tornar este orquestrador resiliente a uma morte do processo responsável pela gestão de estado, seria necessário persistir a informação do estado numa base de dados, por exemplo Redis. Apesar do grupo não ter implementado tal funcionalidade, a mesma seria de fácil implementação, uma vez que as chamadas efetuadas ao estado atual estão bem definidas e encapsuladas.

### **3.6. Message Broker**

O Message Broker é uma peça central na arquitetura de microsserviços do nosso sistema, garantindo a comunicação assíncrona entre os diversos componentes. Ele permite que os microsserviços realizem uma comunicação de forma desacoplada, facilitando a escalabilidade, resiliência e organização do sistema.

Para atender às necessidades do sistema, foi escolhido o RabbitMQ, uma solução robusta e amplamente adotada, baseada no protocolo AMQP.

### **3.7. Bases de Dados**

As bases de dados desempenham um papel essencial no armazenamento e gerenciamento de informações relacionadas aos utilizadores e projetos. A escolha da tecnologia PostgreSQL foi fundamentada na sua robustez, flexibilidade e suporte a recursos avançados, como armazenamento de dados JSON.

### 3.8. Diagrama Arquitetural Final

Dadas as mudanças e sacrifícios feitos durante esta fase de implementação, a arquitetura final ficou da seguinte forma:

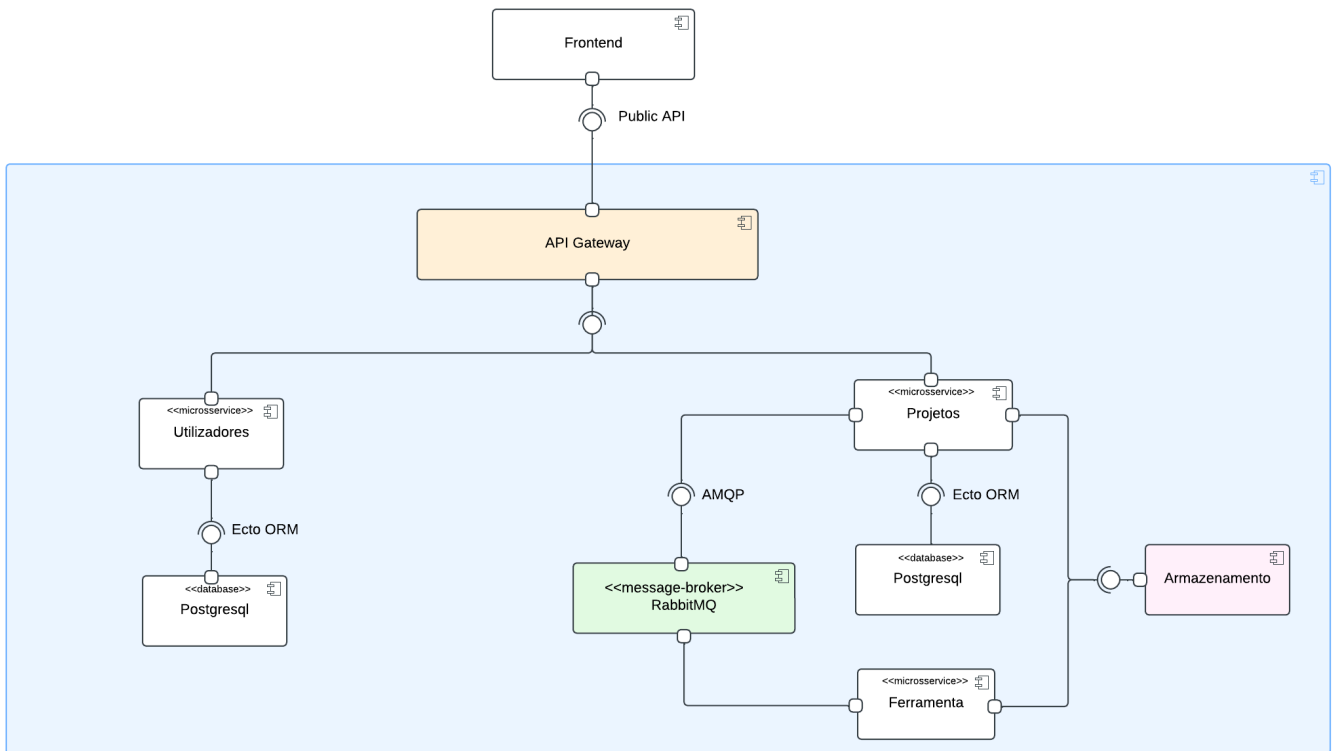


Figura 12: Diagrama de componentes Layer-0 do PictuRAS

## 4. Páginas

### 4.1. Login

### Welcome to Picturas

Email \*

Password \*

Login

---

Don't have an account? [Register](#)

## 4.2. Registo

### Create an account

Name \*

Email \*

Password \*

Confirm Password \*

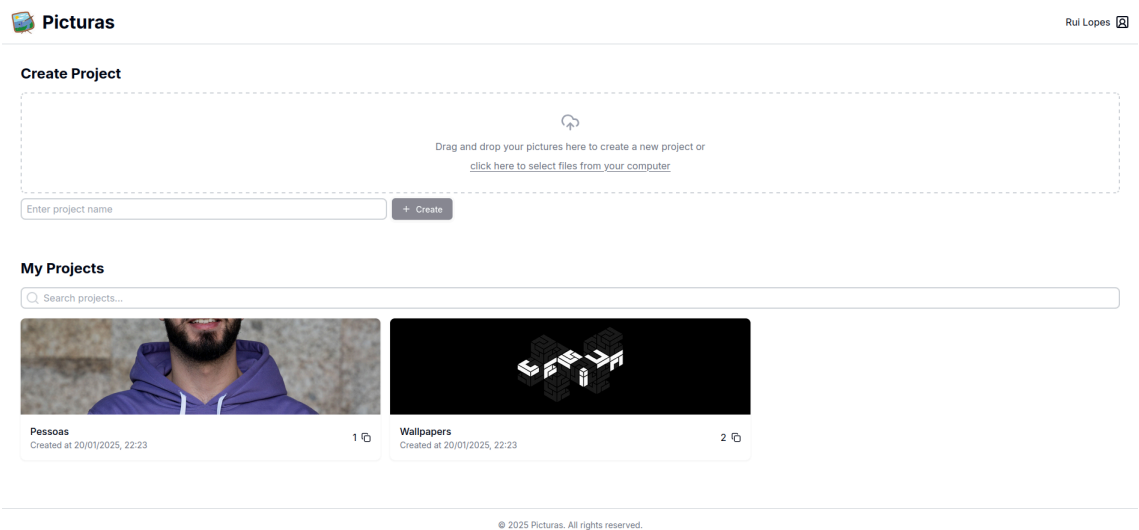
[Register](#)

---

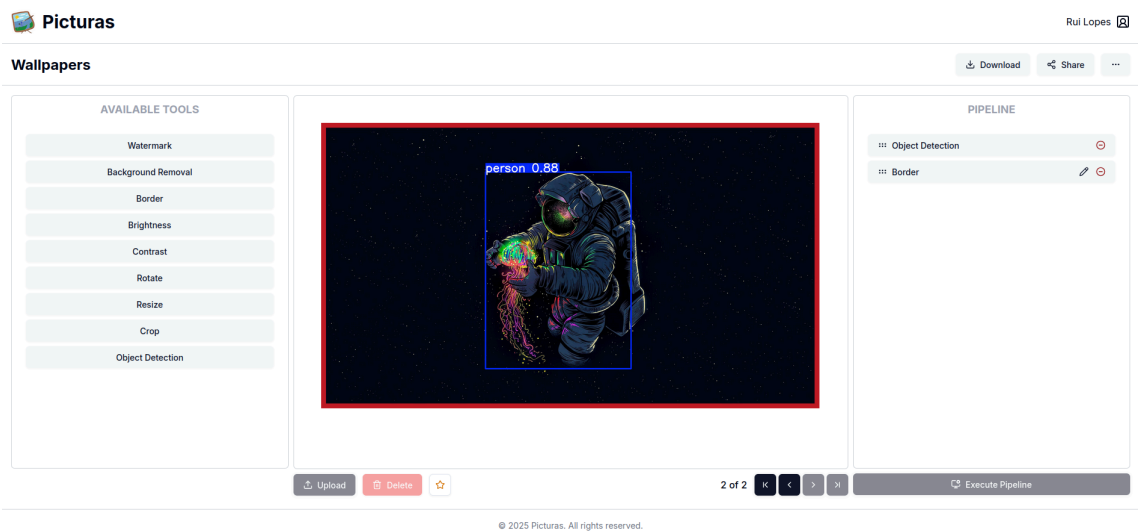
Already have an account? [Login](#)



## 4.3. Página Principal

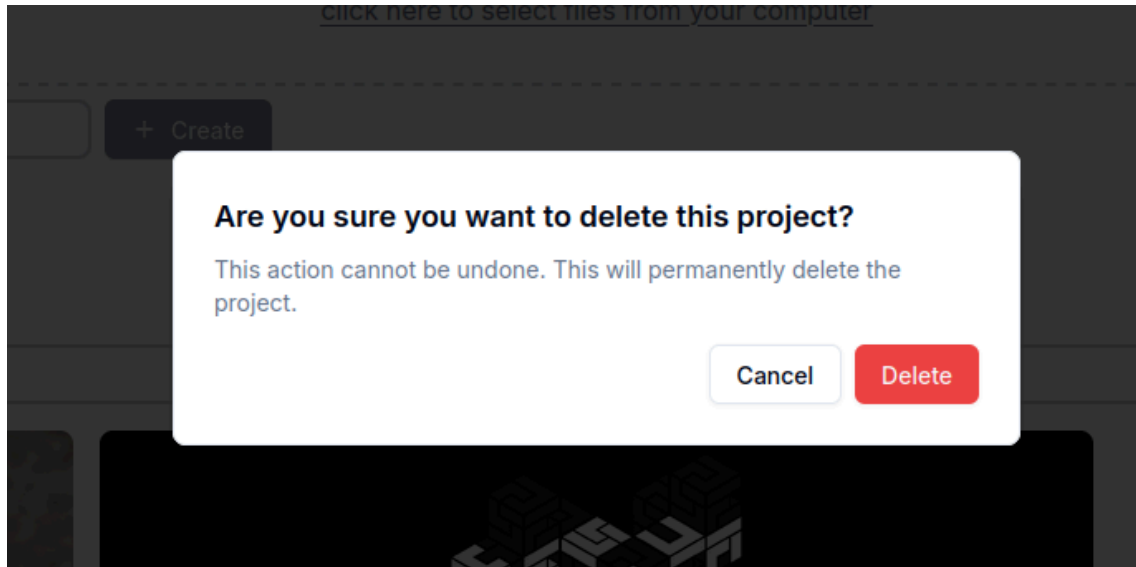


## 4.4. Edição de Projeto

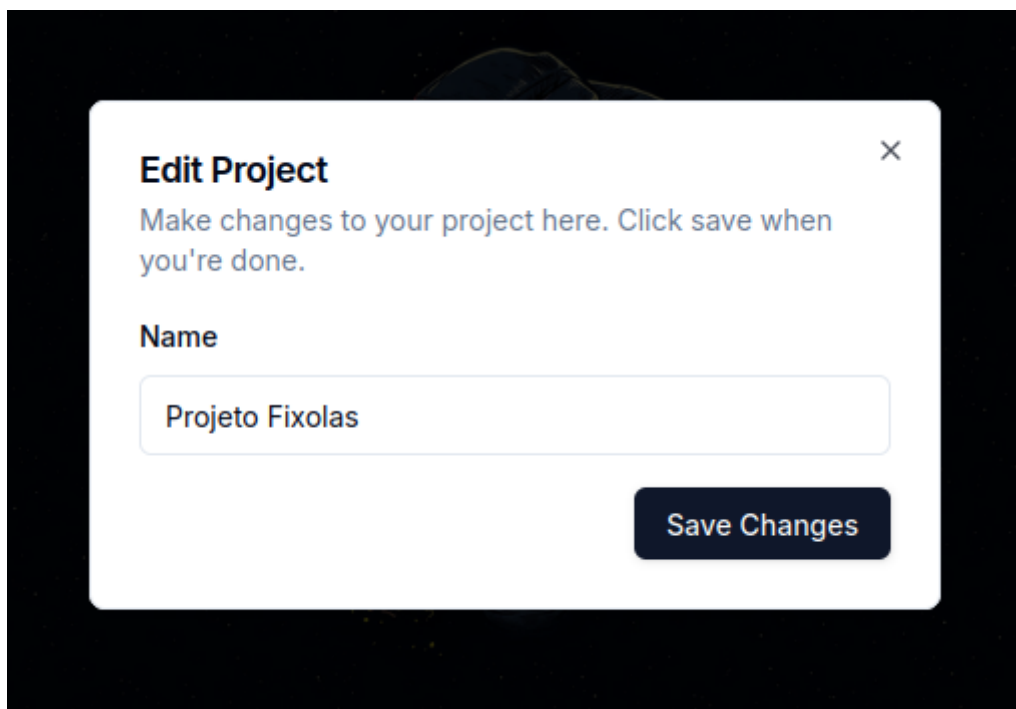


## 4.5. Modals

### 4.5.1. Apagar Projeto



### 4.5.2. Renomear Projeto



### 4.5.3. Partilhar Projeto

Share Project

×


Share your project with others by putting their email addresses here.

Email

mail@picturas.com

People with access

mail@ricardosilva.com




Share

## 4.6. Mobile



# Picturas

Rui Lopes 

## Create Project



Drag and drop your pictures  
here to create a new project  
or

[click here to select files from  
your computer](#)

+ Create

## My Projects



## 5. Organização da Equipa

Nesta secção apresentamos como foi distribuído o trabalho entre os membros da equipa.

A equipa de desenvolvimento é formada por 9 membros, tendo sido decidido que cada elemento ficasse associado à implementação de, pelo menos, uma ferramenta. Contudo, esta divisão não foi executada de maneira uniforme, já que alguns existiram mais ferramentas do que elementos e existiram outros componentes do PictuRAS que apresentam um maior nível de complexidade.

Microserviço	Contributos
PG55940	<b>Ferramenta implementada:</b> Ajustar contraste
PG55974	<b>Ferramentas implementadas:</b> Ajustar brilho, Adicionar borda
PG57895	<b>Ferramentas implementadas:</b> Extrair texto, Remover fundo
PG55967	<b>Ferramentas implementadas:</b> Inversão de cores, Ajuste HUE <b>Páginas Frontend:</b> Página Principal, Página de Edição
PG57886	<b>Ferramenta implementada:</b> Ajustar saturação <b>Páginas Frontend:</b> Página Principal, Página de Edição
PG55982	<b>Ferramentas implementadas:</b> Rodar imagem, Ajuste de nitidez <b>Páginas Frontend:</b> Página de Login, Página de Registo
PG56000	<b>Ferramenta implementada:</b> Redimensionar imagem <b>Páginas Frontend:</b> Página Principal, Página de Edição
PG55626	<b>Ferramenta implementada:</b> Recortar imagem <b>Microserviços envolvido:</b> Utilizadores, Projetos, API Gateway
PG56009	<b>Ferramentas implementadas:</b> Reconhecer objetos, Blur de imagem <b>Microserviços envolvido:</b> Utilizadores, Projetos, API Gateway <b>Páginas Frontend:</b> Página Principal, Página de Edição

Tabela 17: Distribuição da equipa de desenvolvimento

É ainda possível referir que como metodologia de desenvolvimento optamos por Agile com Kanban, uma vez que tivemos um prazo de entrega bastante limitado.

## 6. Deployment

O deployment do sistema como um todo foi realizado com recurso a Docker, mais especificamente Docker Compose. O mesmo foi realizado na máquina virtual cedida pelos docentes e pode ser executado utilizando o comando `docker compose up --build -d`, que irá construir as imagens (se não disponíveis) e executar os respetivos containers na ordem correta.

De facto, um sistema de deployment única e exclusivamente baseado em containers Docker não é, por si só, muito resiliente. Assim sendo, teria sido mais benéfica a implementação de um sistema baseado em Kubernetes, por exemplo - até com sistemas de monitorização mais avançados e específicos, escalando conforme as necessidades.

De qualquer forma, o sistema encontra-se capaz de integrar um deployment feito em Kubernetes com grande facilidade.

## 7. Reflexão

O desenvolvimento do PictuRAS revelou-se um projeto enriquecedor e desafiador. Esta secção aborda os pontos mais positivos e negativos do sistema, bem como perspectivas para o trabalho futuro.

### 7.1. Pontos Positivos

#### **Arquitetura Modular**

A implementação de uma arquitetura de microsserviços assegurou escalabilidade, flexibilidade, e isolamento de falhas. Este modelo permitiu um desenvolvimento eficiente, permitindo que diferentes equipas trabalhassem em paralelo.

#### **Processamento de Imagens**

O sistema mostrou-se eficaz no cumprimento de requisitos funcionais relacionados com a edição de imagens, recorrendo a bibliotecas como OpenCV, YOLO e PIL. Em que esta capacidade de aplicar ferramentas personalizadas a projetos garantiu uma melhor experiência ao utilizador.

#### **Compatibilidade e Design**

A aplicação foi projetada para funcionar em múltiplas plataformas, incluindo dispositivos móveis, cumprindo os requisitos de usabilidade e acessibilidade.

### 7.2. Pontos Negativos

#### **Cobertura Incompleta dos Requisitos**

Nem todos os requisitos funcionais e não funcionais foram implementados, como a gestão de subscrições.

#### **Falta de Escalabilidade**

A escalabilidade horizontal e elástica, crucial para o aumento de utilizadores e carga do sistema, não foi concluída, limitando a capacidade de expansão.

#### **Monitorização Incompleta**

Apesar de terem existido propostas como Elasticsearch e Prometheus, a monitorização não foi totalmente implementada, o que pode dificultar a deteção e resolução de problemas ocorrentes.

### 7.3. Trabalho Futuro

#### **Expansão de Funcionalidades**

Adicionar novas ferramentas de edição e explorar algoritmos mais avançados de processamento de imagens.

#### **Deployment em Kubernetes**

Realizar um deployment em Kubernetes, possibilitando uma configuração capaz de escalar mais facilmente e de acordo com as necessidades existentes da aplicação.

**Gestão de Subscrições**

Completar o microserviço de subscrições para oferecer suporte a planos pagos e gratuitos, com integração com sistemas de pagamento externos como Stripe.

**Armazenamento Externo**

Substituir o armazenamento local por um serviço compatível com S3, para um maior crescimento aplicativo.

## 7.4. Conclusão

Este projeto, realizado no âmbito da unidade curricular de Requisitos e Arquiteturas de Software, provou-se ideal para a consolidação dos conhecimentos lecionados ao longo do semestre.

O objetivo essencial desta fase consistia em proporcionar um *minimum viable product* (MVP) do sistema especificado e arquitetado nas fases anteriores. Consideramos que alcançamos esse objetivo, com exceção de uma funcionalidade que acabou por ser excluída: a gestão de subscrições. De resto, todas as funcionalidades indicadas foram implementadas e ainda acrescentamos várias outras para complementar o trabalho realizado.

Contudo, embora reconheçamos que há espaço para melhorias no futuro, conforme mencionado no ponto anterior, consideramos que alcançamos os objetivos propostos na unidade curricular com a realização do trabalho prático. Este processo permitiu-nos aprofundar o conhecimento sobre especificação de requisitos, conceção de modelos e diagramas para o desenvolvimento de soluções arquitetadas, bem como a implementação dessas mesmas soluções.