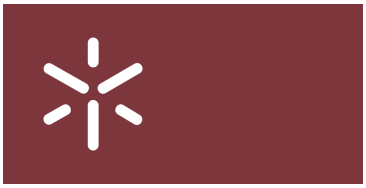

Universidade do Minho
Mestrado em Engenharia Informática

Unidade Curricular de Requisitos e Arquiteturas de Software

Ano Letivo de 2024/2025

Solução Arquitetural

PictuRAS

Diogo A. Costa e Paulo R. Sousa

Documento baseado no trabalho realizado por:

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 planeamento, modelação, 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 segunda etapa, o foco incide no planeamento da implementação do sistema a desenvolver, nomeadamente abordando aspetos como a decomposição funcional do mesmo e a forma como os diferentes componentes se interligam do ponto de vista composicional e interativo.

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, Aplicações para a Web progressivas, Bases de Dados, RabbitMQ, Vue.js, Python, Elixir, Structured Query Language.

Índice

1. Preâmbulo

1.1. Requisitos Funcionais

Durante a fase inicial deste projeto, foi elaborado um documento de requisitos[1] cujo foco incidiu sobre o domínio do problema. Neste documento, foram definidas funcionalidades do sistema relativamente aos atores correspondentes, servindo como base para extrair os requisitos funcionais do sistema.

De seguida, apresentamos as funcionalidades descritas no documento de requisitos.

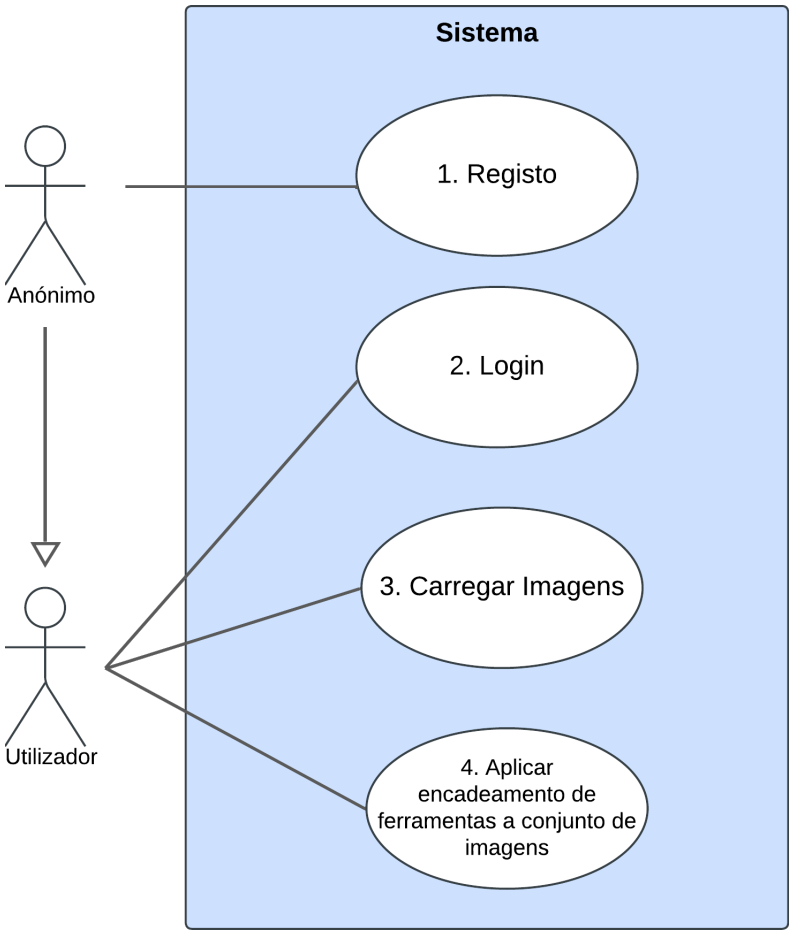

Figura 1: Diagrama de *use cases* do PictuRAS

Tendo em conta o sistema que se pretende desenvolver, a equipa de trabalho decidiu estabelecer, de antemão, um conjunto de requisitos funcionais que considera essenciais para poder implementar, no mínimo, uma versão estável e funcional do PictuRAS[2].

Deste modo, estabeleceu-se a seguinte lista de requisitos funcionais como os mais prioritários:

[1] A equipa docente forneceu, posteriormente, um documento comum a todas as equipas de trabalho.
[2] Uma espécie de MVP, *Minimum Viable Product*.

Requisito	Descrição	*Use Case*	Prioridade
Req1	O utilizador autentica-se utilizando as suas credenciais.	2	Must
Req2	O utilizador anónimo regista-se.	1	Must
Req3 e Req4	O utilizador escolhe o plano de subscrição (Gratuito ou *Premium*).	1	Must
Req6	O utilizador cria um projeto.	3	Must
Req7	O utilizador lista os seus projetos.	3	Must
Req8	O utilizador acede à área de edição de um projeto.	3	Must
Req9	O utilizador carrega imagens para um projeto.	3	Must
Req11	O utilizador adiciona uma ferramenta de edição ao projeto.	4	Must
Req12	O utilizador desencadeia o processamento de um projeto.	4	Must
Req13	O utilizador transfere o resultado de um projeto para o dispositivo local	4	Must
Req18	O utilizador registado acede às suas informações de perfil.	1	Should
Req21	O utilizador *premium* cancela a sua subscrição *premium*.	1	Should
Req22	O utilizador registado termina a sua sessão.	2	Must
Req25	O utilizador recorta manualmente imagens.	4	Must
Req26	O utilizador escala imagens para dimensões específicas.	4	Must
Req27	O utilizador adiciona borda a imagens.	4	Must
Req29	O utilizador ajusta o brilho a imagens.	4	Must
Req30	O utilizador ajusta o contraste a imagens.	4	Must
Req32	O utilizador roda imagens.	4	Must
Req35	O utilizador aplica um algoritmo de recorte automático de imagens com base no seu conteúdo.	4	Must

Tabela 1: Lista tabular dos requisitos funcionais prioritários

Não obstante, os seguintes requisitos podem ser considerados ótimas adições a uma iteração seguinte da plataforma.

Requisito	Descrição	*Use Case*	Prioridade
Req15	O utilizador altera a ordem das ferramentas de um projeto.	4	Must
Req16	O utilizador altera os parâmetros das ferramentas.	4	Must
Req17	O utilizador cancela o processamento de um projeto durante a sua execução.	4	Should
Req19	O utilizador edita o seu perfil.	2	Should
Req36	O utilizador extrai texto de imagens.	4	Should
Req37	O utilizador aplica um algoritmo de reconhecimento de objetos em imagens.	4	Must
Req38	O utilizador aplica um algoritmo de contagem de pessoas em imagens.	4	Should

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

1.2. Requisitos Não Funcionais

A equipa de trabalho decidiu selecionar os requisitos não funcionais que considera ter maior impacto no sistema, analisando-os de cinco perspetivas. Mais tarde, a escolha arquitetural visará maximizar algumas das características não funcionais selecionadas nesta etapa.

Usabilidade
Uma aplicação *web* de uso genérico, que enfrenta concorrência com muitas outras aplicações semelhantes, só terá sucesso se a sua utilização for simples, intuitiva e amigável.

A interface deve ser concebida para oferecer uma experiência de utilizador fluida, com uma navegação clara e acessível, minimizando a necessidade de instruções complexas.

Escalabilidade e Elasticidade
Uma aplicação *web* de uso genérico, se for bem sucedida, poderá enfrentar picos de utilização muito elevados. Por isso, é fundamental que esteja preparada para escalar de forma elástica e automática, garantindo que o seu crescimento não seja comprometido pelo próprio sucesso.

A arquitetura deve ser capaz de suportar um aumento significativo de utilizadores em simultâneo sem causar falhas ou interrupções no serviço.

Simultaneamente, é essencial evitar custos desnecessários durante períodos de baixa utilização.

Extensibilidade
O número de possíveis ferramentas de processamento de imagem que podem ser integradas na aplicação é quase ilimitado. Assim, a aplicação deve ser fácil de estender.

A extensão do âmbito da aplicação não deve estar limitada a uma única tecnologia. A aplicação deve promover um contexto de desenvolvimento que conviva com um nível elevado de heterogeneidade tecnológica. Isto significa que diferentes partes da aplicação podem ser desenvolvidas com tecnologias, linguagens e padrões diferentes.

Manutenção
Para garantir que uma aplicação *web* de uso genérico seja duradoura e evolua conforme as necessidades dos seus utilizadores, é fundamental que seja de fácil manutenção. A aplicação deve ser projetada de

forma modular, permitindo a adição, modificação ou remoção de funcionalidades sem causar alterações significativas no sistema como um todo.

Adicionalmente, a aplicação deve estar preparada para atualizações contínuas, monitorização de desempenho e testes regulares das funcionalidades.

Com estas características delineadas como prioridade, do ponto de vista do desenvolvimento da aplicação, a equipa selecionou os seguintes requisitos não funcionais entre os demais como os prioritários na construção da solução arquitetural.

Requisito	Descrição	Contexto	Prioridade
RNF5	A página de um projeto distingue visualmente entre as ferramentas básicas e avançadas.	**Usabilidade**	Must
RNF7	O utilizador cria um projeto implicitamente ao arrastar ficheiros para o *dashboard* da aplicação.	**Usabilidade**	Must
RNF8	O utilizador carrega imagens arquivadas num único ficheiro `.zip`.	**Usabilidade**	Must
RNF9	O utilizador é mantido informado acerca do estado de processamento de um projeto em tempo real.	**Usabilidade**	Must
RNF14	A visualização de imagens grandes ou pequenas é auxiliada pelo utilitário zoom.	**Usabilidade**	Must
RNF15	A visualização de imagens permite que se navegue em todas as duas direções arrastando o rato.	**Usabilidade**	Must
RNF18	A aplicação é compatível com diferentes plataformas e browsers, incluindo de dispositivos móveis e *desktop*.	**Usabilidade**	Must
RNF19	A aplicação fornece *feedback* visual claro e imediato ao utilizador em caso de erro ou falha durante um procedimento demorado.	**Usabilidade**	Should
RNF22	O sistema deve processar até 100 imagens ao mesmo tempo, sem quebras percetíveis no desempenho.	**Escalabilidade e Elasticidade**	Must
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
RNF25	A aplicação deve ser integrável com outras plataformas e serviços de terceiros.	**Extensibilidade**	Must
RNF28	A aplicação deve ser facilmente estendida com novas ferramentas de edição.	**Extensibilidade**	Must
RNF32	O sistema deve ser projetado para facilitar a execução de testes.	**Manutenção**	Must

Requisito	Descrição	Contexto	Prioridade
RNF33	A aplicação deve realizar *backups* automáticos dos dados e imagens dos utilizadores.	**Manutenção**	Should

Tabela 3: Lista tabular dos requisitos não funcionais prioritários

1.3. Restrições

Esta secção visa enunciar as restrições impostas pelo documento de requisitos a par de uma análise sobre o seu impacto na solução final, se tal existir.

Restrição Técnica 1
A aplicação deve ser exclusivamente desenvolvida como uma aplicação *web*.

Análise
Partindo da análise dos requisitos não funcionais, anteriormente feita, é possível observar que existe a intenção da aplicação final poder ser utilizada e disponibilizada em dispositivos *mobile*. Assim, e uma vez que é imposto que a mesma seja desenvolvida como uma aplicação *web*, a equipa de trabalho decide optar por uma solução técnica que permita a extração de uma aplicação *web* progressiva[3] capaz de ser instalada e, até disponibilizada, em dispositivos *mobile*.

Restrição Técnica 2
A aplicação deverá estar preparada para ser disponibilizada numa infraestrutura Cloud (AWS, Azure, Google Cloud), considerando aspetos como escalabilidade, segurança e otimização de custos.

Análise
Esta restrição não constitui um problema real. A utilização de qualquer tecnologia considerada recente terá o seu suporte assegurado pelas três grandes empresas que disponibilizam infraestrutura Cloud.

Restrição Técnica 3
É obrigatório garantir que a plataforma implemente medidas robustas de segurança, incluindo a proteção contra ataques comuns (e.g., SQL injection, XSS).

Análise
Esta restrição pode ser assegurada, sem grande investimento, pela utilização de tecnologias recentes, amplamente utilizadas e empregues no mercado existente.

Restrição Técnica 4
A arquitetura da aplicação deve ser elástica para suportar múltiplos utilizadores simultâneos, e o desempenho do sistema deve ser otimizado para garantir uma experiência fluida, independentemente do número de utilizadores ativos ou da carga de processamento.

Análise
Esta restrição é também muito importante e será abordada aquando da escolha da arquitetura para a implementação da solução final.

Restrição Legal 1
É obrigatório garantir que a plataforma implemente políticas adequadas de gestão de dados dos utilizadores, conforme regulamentos de privacidade como o RGPD.

Análise
Esta restrição incluí tópicos como a aplicação de políticas adequadas e gestão de dados dos utilizadores *compliant*. Um exemplo prático, mais próximo da realidade, é a obrigatoriedade da palavra-passe de um utilizador ser persistida de forma cifrada.

[3] Consultar mais sobre aplicações *web* progressivas aqui.

Restrição de Negócio 1
Dependendo do tipo de utilizador existem limites associados ao número de processamento diários, de imagens num projeto, ao tipo de ferramentas que podem ser utilizadas e à dimensão das imagens carregadas.

Análise
Esta restrição implica que o sistema final deverá possuir uma forma de contabilizar e validar, para cada utilizador e projeto, cada um dos parâmetros referidos. De preferência, a solução deverá ter persistência, por exemplo, num sistema de dados.

Restrição Temporais 1
O documento presente, referente à arquitetura da aplicação, deve ser entregue até ao dia 22 de novembro de 2024, incluindo a arquitetura completa do sistema.

Análise
Esta restrição é de extrema importância, para que possa ser feita uma avaliação do estado do projeto atempadamente antes do desenvolvimento do sistema concebido.

Restrição Temporais 2
A entrega da implementação da aplicação deve ser realizada até 17 de janeiro de 2025, incluindo a versão final do *software* com todas as funcionalidades principais implementadas.

Análise
Esta restrição é importante, visto que procuramos alinhar o âmbito do projeto com a capacidade de entrega da equipa, assegurando que as funcionalidades principais sejam implementadas a tempo de cumprir o prazo estabelecido.

2. Contexto e Âmbito do Sistema

O sistema proposto, denominado PictuRAS, surge como uma solução inovadora para facilitar o processamento e edição de imagens básico e avançado.

A motivação para o desenvolvimento deste projeto surge na necessidade crescente de uma solução prática e eficiente, oferecida como uma aplicação *web* disponibilizada na nuvem para o efeito já descrito.

Este conceito abre caminho à flexibilidade e acessibilidade, independentemente das condições específicas de cada utilizador. A proposta do PictuRAS é adaptar-se a diferentes contextos, promovendo uma utilização igualitária e eficiente mesmo por parte de utilizadores com pouco conhecimento do domínio.

De acordo com os *use cases* e os requisitos funcionais previamente estabelecidos no documento de requisitos, foi possível elaborar o seguinte diagrama de contexto do sistema.

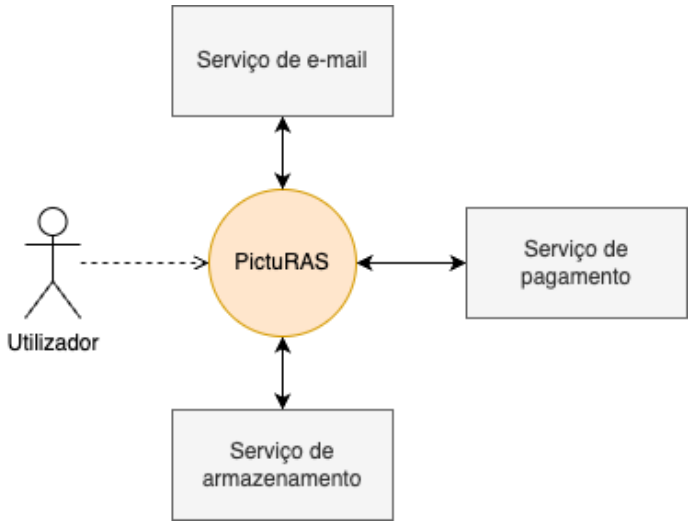

Figura 2: Diagrama de contexto de negócio do sistema

O diagrama representado abaixo, ilustra as interações entre o sistema PictuRAS e os sistemas externos, assim como as interfaces técnicas necessárias para cada um dos casos.

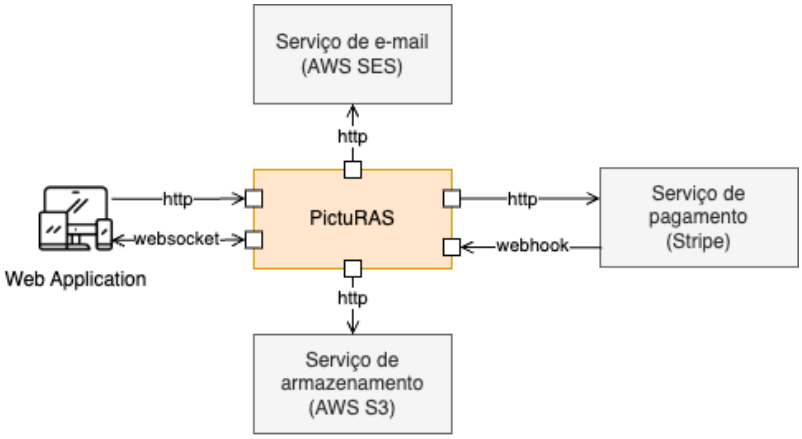

Figura 3: Diagrama de contexto técnico do PictuRAS

3. Estratégia da Solução

Nesta secção, iremos abordar a decomposição funcional do PictuRAS, a escolha do padrão arquitetural baseado nos requisitos não funcionais delineados previamente, o modelo de dados do sistema e as tecnologias que serão aplicadas. Por fim, apresentaremos a organização da equipa de desenvolvimento.

3.1. Decomposição Funcional

Após examinarmos minuciosamente os requisitos funcionais e os relacionados casos de uso, procedemos à decomposição funcional do sistema. Podendo esta decomposição ser descrita num diagrama que demonstra a divisão das responsabilidades do sistema a vários níveis de profundidade. Estes subsistemas oferecem uma visão mais clara do sistema, que numa fase subsequente, facilitará o desenvolvimento da solução final.

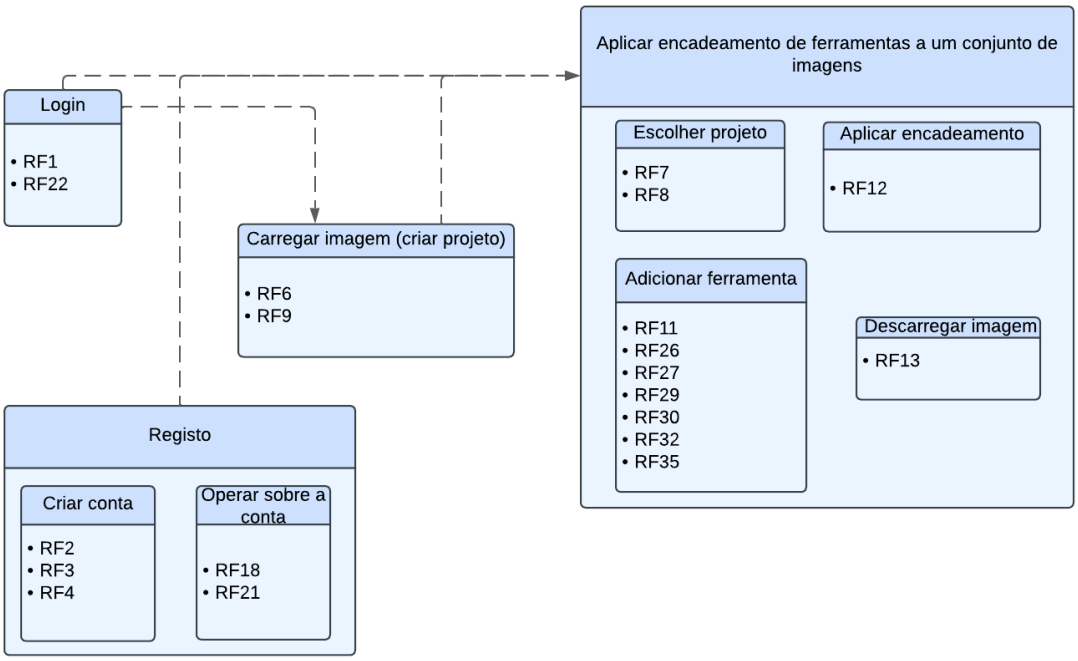

Figura 4: Diagrama de decomposição funcional do PictuRAS

A partir do diagrama exposto, é possível deduzir que uma grande parte da complexidade do sistema concentra-se no encadeamento de ferramentas, que representa o núcleo das interações e dependências entre subsistemas. Este encadeamento exige a integração de diferentes módulos, garantindo a interoperabilidade, consistência e desempenho, pelo que deverá existir um esforço redobrado, por parte da equipa de trabalho, na implementação destas funcionalidades.

3.2. Padrões Arquiteturais

A escolha do padrão arquitetural a aplicar durante o desenvolvimento do sistema é muito importante, visto que, será este padrão a ditar de que forma o sistema se compromete perante os requisitos não funcionais anteriormente mencionados. De facto, são os requisitos não funcionais que ditam a escolha de um padrão arquitetural. Como tal, a equipa de trabalho decidiu abordar uma série de padrões, conhecidos na indústria, avaliando a sua resposta perante os requisitos impostos.

Em baixo, encontra-se uma lista com os seis requisitos não funcionais, e as correspondentes características, que a equipa de trabalho considerou os mais importantes para a seleção da arquitetura.

Requisito	Descrição	Contexto
RNF22	O sistema deve processar até 100 imagens ao mesmo tempo, sem quebras percetíveis no desempenho.	**Escalabilidade e Elasticidade**
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**
RNF25	A aplicação deve ser integrável com outras plataformas e serviços de terceiros	**Extensibilidade**
RNF28	A aplicação deve ser facilmente estendida com novas ferramentas de edição.	**Extensibilidade**
RNF32	O sistema deve ser projetado para facilitar a execução de testes.	**Manutenção**
RNF33	A aplicação deve realizar *backups* automáticos dos dados e imagens dos utilizadores.	**Manutenção**

Tabela 4: Lista tabular dos requisitos não funcionais com impacto significativo na escolha arquitetural

De seguida, encontra-se uma breve apresentação sobre cada uma das arquiteturas cogitadas. Esta apresentação está acompanhada de uma classificação[4] (de zero a cinco estrelas) da arquitetura, em relação a cada um dos requisitos mencionados imediatamente acima.

Layered

A arquitetura *Layered*, ou em camadas, organiza a aplicação em diferentes níveis, cada um com responsabilidades diferentes. As camadas incluem, por norma, uma camada de apresentação, onde ocorre a interação com o utilizador, e uma camada de lógica de negócio, onde são processadas as regras e operações centrais da aplicação. Esta separação permite que cada camada funcione de forma independente, facilitando a manutenção e o desenvolvimento.

Requisito	Classificação
RNF22	★★
RNF23	★
RNF25	★★★

[4] Classificação baseada no trabalho realizado por Mark Richards e Neal Ford em *Fundamentals of Software Architecture - An Engineering Approach*.

Requisito	Classificação
RNF28	★
RNF32	★★
RNF33	★★★★
Média	★★

Tabela 5: Classificação da arquitetura Layered perante alguns requisitos não funcionais

Event-driven

A arquitetura *event-driven* (orientada a eventos) baseia-se na emissão e captura de eventos para desencadear ações dentro do sistema. Nesta, componentes independentes utilizam comunicação assíncrona, isto é, quando um evento ocorre (como uma ação do utilizador ou uma mudança no estado do sistema), este é emitido para os componentes que estão à "escuta".

Requisito	Classificação
RNF22	★★★★
RNF23	★★★
RNF25	★★★★
RNF28	★★★★
RNF32	★★★
RNF33	★★★
Média	★★★★

Tabela 6: Classificação da arquitetura Event-driven perante alguns requisitos não funcionais

Space-based

A arquitetura *space-based* (ou baseada em espaço) está projetada para suportar altos volumes de dados, especialmente em aplicações com picos de utilização inesperados. Nesta abordagem, os dados e processos são distribuídos em partições ou nós independentes que compartilham um "espaço" comum, normalmente implementado através de sistemas de *caching* distribuídos. Assim, é possível eliminar pontos únicos de falha e reduzir a sobrecarga de acesso a uma base de dados centralizada, garantindo uma resposta mais rápida do sistema.

Requisito	Classificação
RNF22	★★★★★
RNF23	★★★★
RNF25	★★
RNF28	★★
RNF32	★★

Requisito	Classificação
RNF33	★★★★★
Média	★★★

Tabela 7: Classificação da arquitetura Space-based perante alguns requisitos não funcionais

Microsserviços
A arquitetura de microsserviços organiza uma aplicação como um conjunto de serviços independentes, cada um responsável por uma função específica. Cada microsserviço é executado como uma unidade autónoma, com o seu próprio ciclo de desenvolvimento e escala, comunicando com outros serviços geralmente via APIs. Esta abordagem permite uma grande modularidade, onde cada serviço pode ser desenvolvido, atualizado ou substituído de forma isolada, sem afetar diretamente o funcionamento dos outros.

Requisito	Classificação
RNF22	★★★★★
RNF23	★★★★★
RNF25	★★★★★
RNF28	★★★★★
RNF32	★★★★
RNF33	★★★
Média	★★★★★

Tabela 8: Classificação da arquitetura Microsserviços perante alguns requisitos não funcionais

3.3. Arquitetura Idealizada

De acordo com a análise apresentada é possível depreender que a arquitetura ideal para a solução final é a de **Microsserviços**, com uma classificação final média de cinco estrelas. Esta, destaca-se positivamente, e, na maioria das vezes, excecionalmente, em todos os parâmetros desejados.

Além disso, a equipa de trabalho decidiu incorporar algumas nuances presentes numa arquitetura ***Event-driven***, por via de uma fila de mensagens.

Utilizaremos, portanto, uma mistura destas duas arquiteturas.

3.3.1. Decomposição em Microserviços

Tendo em conta as funcionalidades principais do PictuRAS e, também, todo o trabalho refinado e enumerado até agora, a equipa de trabalho chegou à seguinte lista de microsserviços:

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 divida 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 de Projetos
Este microsserviço será responsável pela gestão dos projetos dos utilizadores. Desde a criação, edição e remoção até ao manuseamento das imagens.

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, eventualmente, utilizando 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.

3.3.2. Comunicação entre Microsserviços

A decomposição funcional apresentada foca-se em garantir um elevado nível de desacoplamento entre os vários microsserviços, sendo que a comunicação entre eles está, na sua maioria, circunscrita à dinâmica de aplicação de ferramentas de edição. Por dotar a aplicação de beneficiar características como a escalabilidade, performance e redundância, esta comunicação deverá ser feita através de um sistema assíncrono, baseado em passagem de mensagens (*event-driven*).

Neste contexto há dois eventos que importa destacar:

1. Pedido para aplicação de uma ferramenta

Este evento é utilizado para solicitar a execução de uma transformação por parte de um microserviço. O pedido inclui informações sobre o tipo de transformação a realizar (e.g., rotação, redimensionamento, contagem de pessoas) e os parâmetros necessários para a sua execução. Este evento é enviado para a fila apropriada no RabbitMQ para ser processado pelo microserviço responsável.

Exemplo:

```
{
   "messageId": "request-2",
   "timestamp": "2024-11-01T12:00:00Z",
   "procedure": "rotate",
   "parameters": {
      "inputImageURI": "file:///images/request-1-out.jpg",
      "outputImageURI": "file:///images/request-2-out.jpg",
      "degrees": -90
   }
}
```

2. Conclusão da aplicação de uma ferramenta

Este evento é gerado quando um dado microserviço responsável termina uma transformação, indicando se o processo foi concluído com sucesso ou se, pelo contrário, ocorreu um erro. O evento deve incluir

o estado genérico do processamento, informações sobre o ficheiro ou resultado produzido (caso tenha sucesso) ou detalhes do erro (caso tenha falhado) e, ainda, um identificador que permita determinar a que pedido dá resposta. Este evento é utilizado para informar os sistemas interessados sobre o desfecho da operação.

Exemplo, em caso de sucesso:

```
{
   "messageId": "completion-2",
   "correlationId": "request-2",
   "timestamp": "2024-11-01T12:00:01Z",
   "status": "success",
   "output": {
      "type": "image",
      "imageURI": "file:///images/request-2-out.jpg"
   },
   "metadata": {
      "processingTime": 0.2,
      "microservice": "picturas-rotate-tool-ms"
   }
}
```

3.4. 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.

O 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 os diferentes dados do sistema.

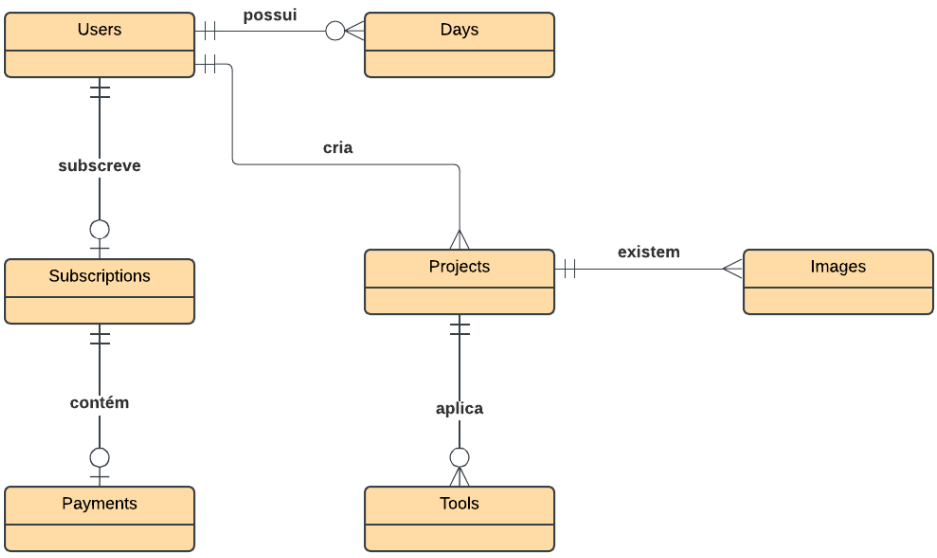

Figura 5: Diagrama ER

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.

Note-se que esta é apenas uma proposta. Outras abordagens, como técnicas NoSQL, podem ser igualmente válidas ou até preferíveis em algumas situações (e.g., para representar mais facilmente a ordem das ferramentas num projeto).

3.4.1. Subscrições

O microsserviço de subscrições dará origem a duas tabelas distintas, mas com uma relação direta: uma para armazenar informação sobre as subscrições ativas dos utilizadores e outra para guardar informação sobre os pagamentos realizados por essas subscrições.

Subscriptions		
Campo	**Tipo**	**Descrição**
id	`uuid`	Identificador único de uma subscrição
user_id	`uuid`	Identificador do utilizador a que se refere o subscrição
type	`enum`	Tipo de subscrição: mensal ou anual
state	`enum`	Estado da subscrição: ativa (*default*) ou inativa
inserted_at	`datetime`	Data e hora em que a subscrição se tornou ativa

Tabela 9: Modelo de dados de uma subscrição

Campo	**Valor**
id	090e0ca9-b2eb-4962-87bb-7517aff67c6c
user_id	0e6d0ce7-08c1-4de9-b7ff-82554bad32d8
type	`monthly`
state	`active`
inserted_at	`2024-11-21T16:32:12.810367Z`

Tabela 10: Exemplo de uma entrada da tabela das subscrições

Payments		
Campo	**Tipo**	**Descrição**
id	`uuid`	Identificador único de um pagamento
subscription_id	`uuid`	Identificador da subscrição a que se refere o pagamento
extra	`map`	Informação relativa ao sistema externo Stripe

Tabela 11: Modelo de dados de um pagamento

Campo	Valor
id	090e0ca9-b2eb-4962-87bb-7517aff67c6c
user_id	0e6d0ce7-08c1-4de9-b7ff-82554bad32d8
extra	`{"transaction_id": "...", "receipt_url": "...", ...}`

Tabela 12: Exemplo de uma entrada da tabela dos pagamentos

3.4.2. Utilizadores

Neste caso, existirá apenas uma tabela dedicada aos utilizadores do sistema. De realçar que esta tabela deverá suportar a existência de utilizadores anónimos. Adicionalmente, 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`
	`nullable`
	`citext`
		Email do utilizador

password_hash	`nullable` `varchar(72)`	Hash da palavra-passe do utilizador
type	`enum`	Tipo de utilizador: anónimo, gratuito (*default*) ou *premium*

Tabela 13: Modelo de dados de um utilizador

Campo	Valor
id	0e6d0ce7-08c1-4de9-b7ff-82554bad32d8
name	Rui Lopes
email	mail@ruilopesm.com
password_hash	$2a12M.P4DK7okN/GuDhIoUh6B.Qorkv0p6tJ3rrcdVx1loaDs8MUB9NPy
type	`premium`

Tabela 14: Exemplo de uma entrada da tabela dos utilizadores

Days		
Campo	**Tipo**	**Descrição**
day	date	Um dos dias em que o utilizador realizou processamentos
processed	int	Processamentos realizados nesse dia
user_id	`uuid`	Utilizador ao qual a contabilização de processamentos neste dia se refere

Tabela 15: Modelo de dados de um dia

Campo	**Valor**
day	`2024-11-21`
processed	2
user_id	0e6d0ce7-08c1-4de9-b7ff-82554bad32d8

Tabela 16: Exemplo de uma entrada da tabela dos dias

3.4.3. Projetos

No que toca aos projetos existirão três tabelas. Primeiramente, uma tabela principal para dar conta da informação direta do projeto. Em seguida, uma tabela para guardar informação sobre os *paths* das imagens, dado que que um projeto pode possuir várias imagens. Finalmente, uma tabela de ferramentas para guardar, de forma ordenada, as ferramentas aplicadas no projeto, juntamente com os respetivos parâmetros.

Projects		
Campo	**Tipo**	**Descrição**
id	`uuid`	Identificador único de um projeto
name	`varchar(200)`	Nome do projeto
user_id	`uuid`	Utilizador que criou o projeto

Tabela 17: Modelo de dados de um projeto

Campo	**Valor**
id	516d3862-2a87-4e04-baf8-5bf640b94838
name	Fotografias de astronomia
user_id	0e6d0ce7-08c1-4de9-b7ff-82554bad32d8

Tabela 18: Exemplo de uma entrada na tabela dos projetos

Images		
Campo	**Tipo**	**Descrição**
id	`uuid`	Identificador único de uma imagem
project_id	`uuid`	Projeto ao qual a imagem pertence
uri	`text`	Path/link para a imagem

Tabela 19: Modelo de dados de uma imagem

Campo	**Valor**
id	d3011aad-7c20-4f4a-8191-7749325a49ae
project_id	516d3862-2a87-4e04-baf8-5bf640b94838
uri	https://s3.picturas.com/d3011aad-7c20-4f4a-8191-7749325a49ae.jpeg

Tabela 20: Exemplo de entrada na tabela das imagens

Tools		
Campo	**Tipo**	**Descrição**
id	`uuid`	Identificador único da ferramenta
position	int	Posição na lista virtual de ferramentas
procedure	`varchar(200)`	Tipo de transformação a aplicar (e.g., ‘rotate’, ‘peopleCount’)
parameters	`map`	Mapeamento do nome de um parâmetro para o valor do mesmo
project_id	`uuid`	Projeto ao qual a ferramenta pertence

Tabela 21: Modelo de dados de uma ferramenta

Campo	**Valor**
id	d3011aad-7c20-4f4a-8191-7749325a49ae
position	3
procedure	rotate
parameters	`{"degrees": -90}`
project_id	516d3862-2a87-4e04-baf8-5bf640b94838

Tabela 22: Exemplo de entrada na tabela das ferramentas

4. Building Block View

Nesta secção iremos abordar o PictuRAS do ponto de vista da sua decomposição em diversos componentes. Um componente pode ser entendido como um subsistema, módulo, classe, interface, pacote, biblioteca, etc.

Com este primeiro diagrama apresentamos uma vista mais geral, e com um nível de abstração alto, do sistema. Esta vista engloba todos os microsserviços já descritos, um API Gateway, uma fila de mensagens e diferentes tipos de Frontend. Outros componentes externos ao sistema também são relatados, pela sua interação com o PictuRAS.

Nos capítulos consequentes segue-se, em maior detalhe, cada um dos componentes presentes.

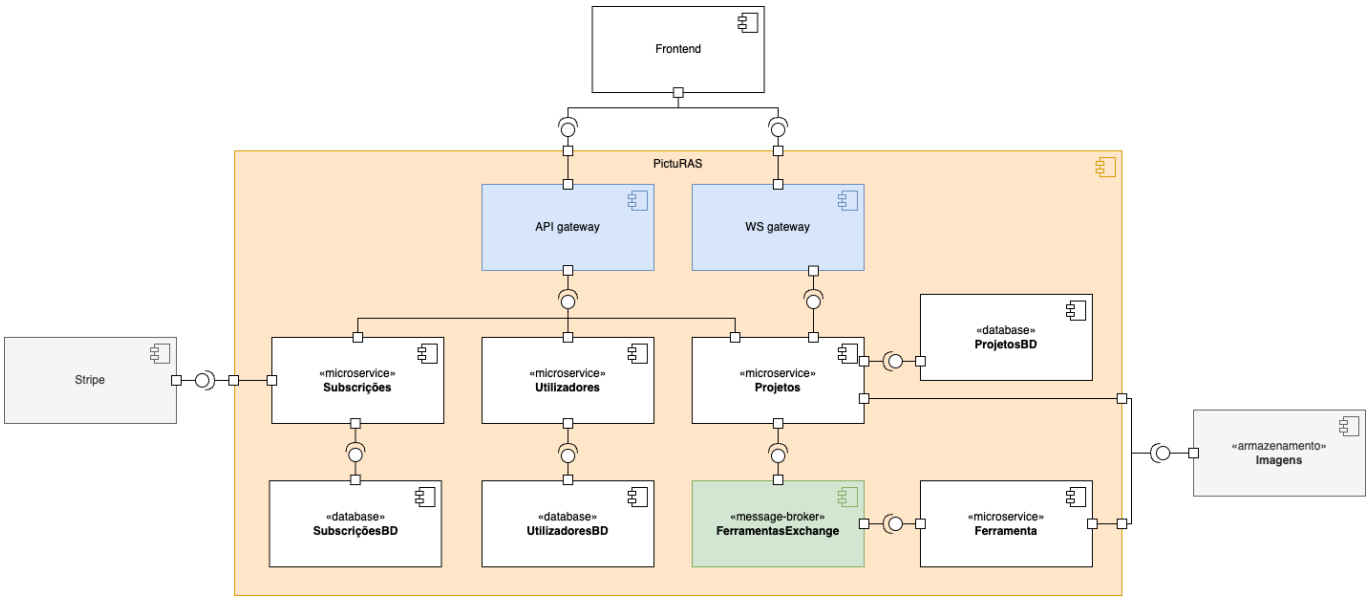

Figura 6: Diagrama de componentes do PictuRAS

4.1. Frontend

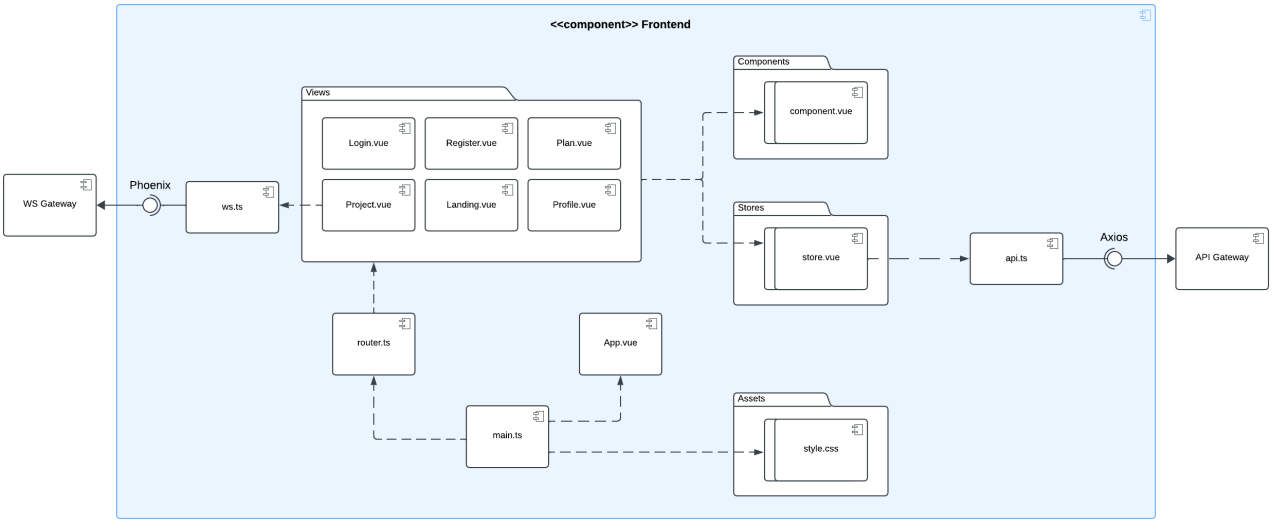

Figura 7: Diagrama de componentes do Frontend

Este diagrama abrange todos os pacotes, módulos e interações com outros componentes do Frontend.

Essencialmente, existe um ponto de entrada para o programa, presente no módulo `main.ts`, que delega a execução para um dos componentes presentes no pacote `Views`, com base na página requisitada pelo utilizador.

A página é renderizada (mostrada ao utilizador) através de um conjunto de diferentes componentes, presentes no pacote `Components`. Além disso, cada uma destas páginas possui, conforme a necessidade, acesso a um conjunto de `stores` que, por sua vez, fazem uso de uma interface com a biblioteca `Axios` para comunicar com o API Gateway.

De notar, ainda, que caso a página a renderizar esteja relacionada ao processamento de um projeto (`Project.vue`) irá existir outro tipo de comunicação com o API Gateway, por via de *web* sockets, através de uma interface com a biblioteca `Phoenix`.

Por fim, cabe mencionar a existência de um módulo `App.vue` responsável pela correta configuração da *framework* utilizada, Vue.js, e do módulo `style.css` responsável por aplicar alguns estilos CSS genéricos a todo o Frontend.

4.2. API Gateway

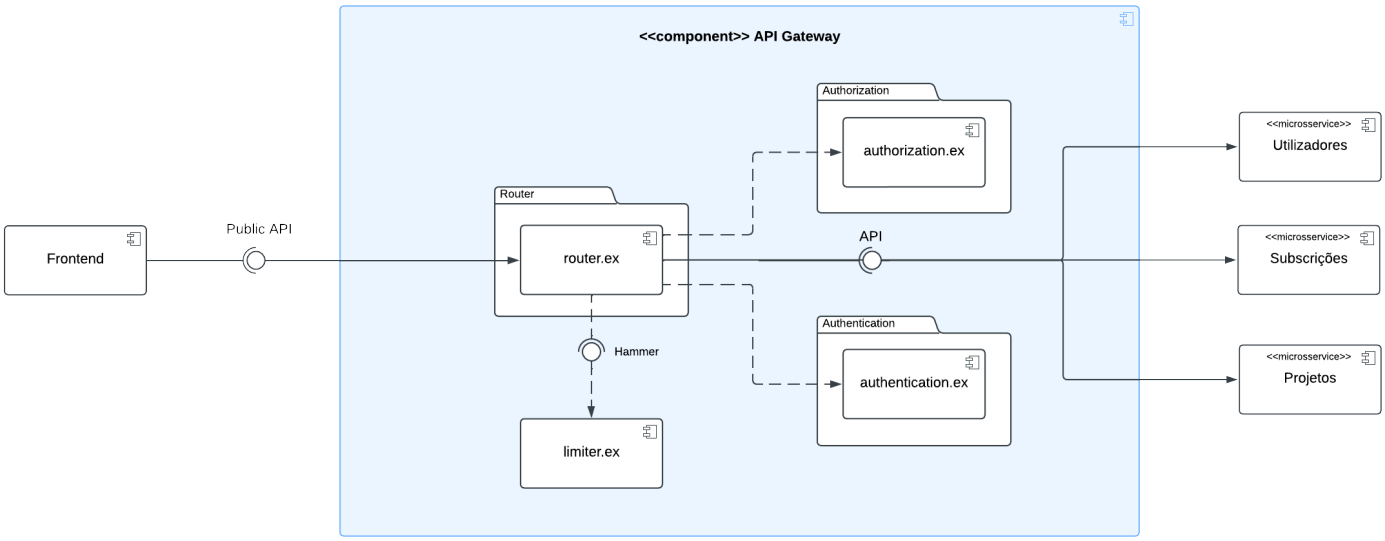

Figura 8: Diagrama de componentes do API Gateway

Este diagrama representa os componentes, módulos e interações que compõem o API Gateway, bem como a sua comunicação com outros sistemas.

No centro da API Gateway encontra-se o módulo `router.ex`, que atua como ponto de entrada principal, sendo responsável por encaminhar os pedidos recebidos através da Public API para os diferentes micros-serviços associados.

Para garantir o controlo e segurança do sistema, o módulo `limiter.ex`, em conjunto com a biblioteca `Hammer`, é utilizado para implementar mecanismos de limitação de pedidos, evitando sobrecargas e acessos abusivos à API.

No que diz respeito à gestão de autenticação, a API Gateway está dividida em dois módulos principais:

- O módulo `authentication.ex`, que se encarrega de validar as credenciais e gerir sessões de utilizadores, garantindo que apenas acessos autenticados são permitidos.
- O módulo `authorization.ex`, que verifica permissões específicas para assegurar que os utilizadores têm os acessos necessários para obter determinados recursos.

Ambas estas nuances foram já explicadas em detalhe nos capítulos anteriores.

4.3. WS Gateway

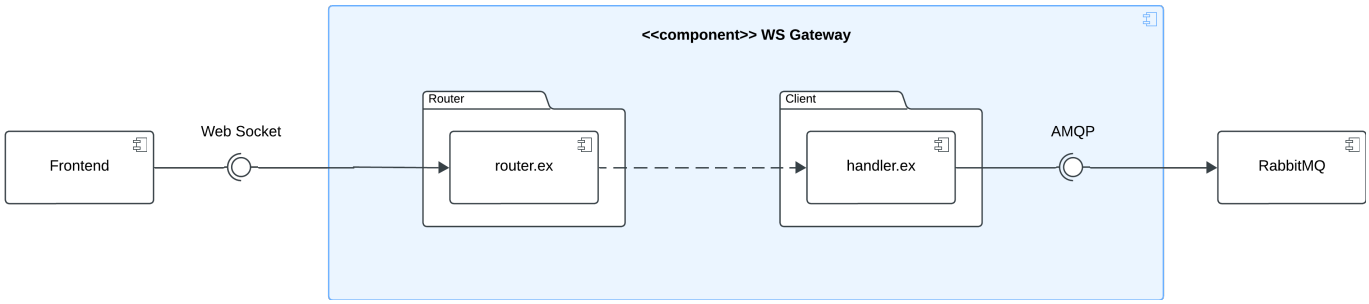

Figura 9: Diagrama de componentes do WS Gateway

Este diagrama representa os componentes, módulos e interações que compõem o WS Gateway, bem como a sua comunicação com outros sistemas.

No centro do WS Gateway encontra-se o módulo `router.ex`, que atua como ponto de entrada principal, sendo responsável simplesmente por encaminhar os pedidos recebidos através do *web socket* para o RabbitMQ. É, também, responsável por subscrever a tópicos de interesse no RabbitMQ.

4.4. Subscrições

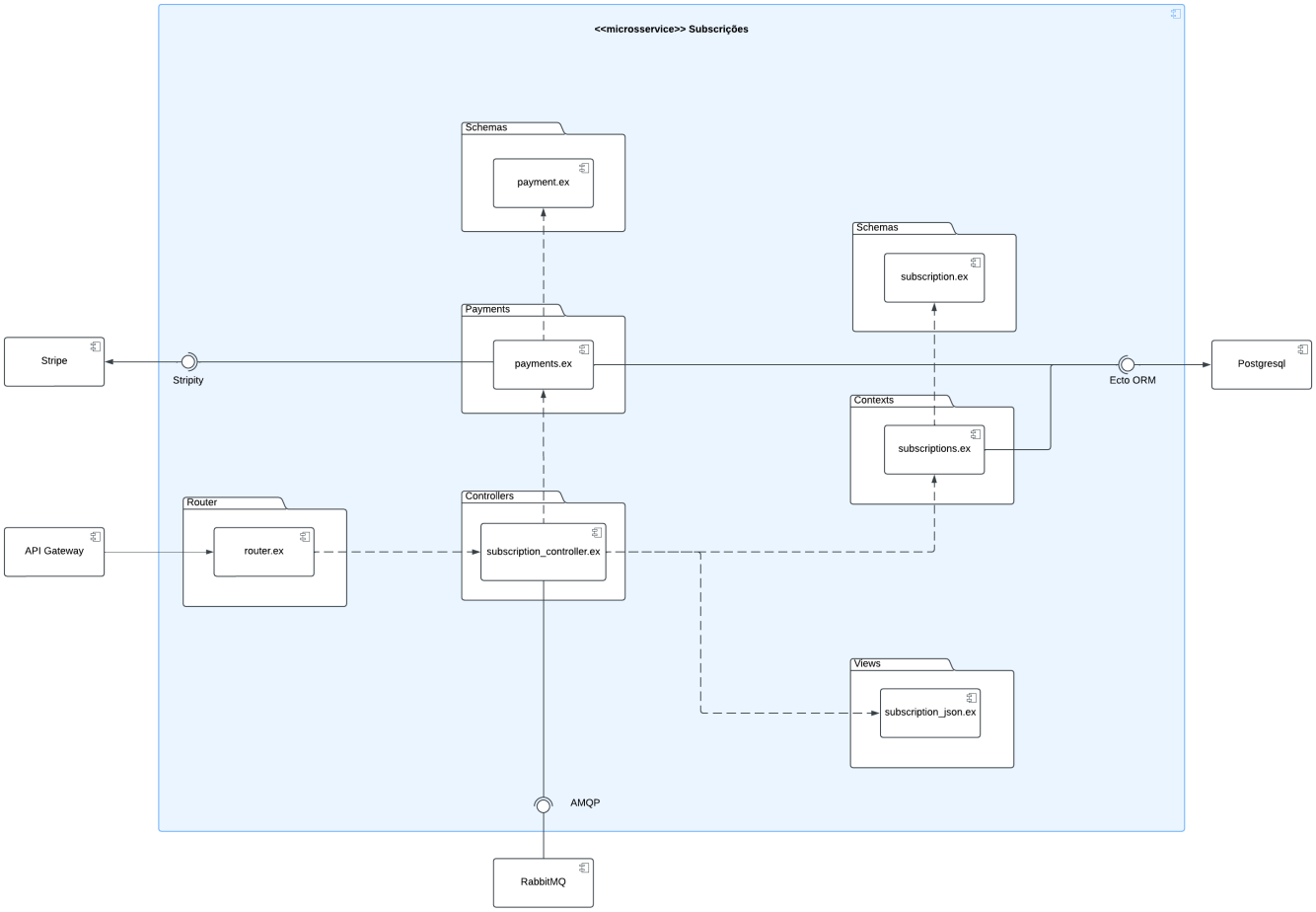

Figura 10: Diagrama de componentes do Microsserviço de Subscrições

Este diagrama abrange todos os pacotes, módulos e interações entre os componentes do microsserviço de subscrições. É importante destacar que, internamente, este microsserviço adota uma arquitetura MVC[5] , e, consequentemente, as interações entre os módulos seguem esta lógica. Uma arquitetura MVC permite uma separação clara entre as responsabilidades de cada componente, o que contribui significativamente para a organização e a extensibilidade do microsserviço.

Essencialmente, existe um ponto de entrada para o programa, presente no módulo `router.ex`, que trata de delegar a execução para o controlador, `subscription_controller.ex`, com base no pedido feito. No estado atual da arquitetura, o `router.ex` apenas redireciona para este controlador. No entanto, a equipa de trabalho considera que a existência de um roteador facilita a expansão futura da aplicação, exigindo apenas uma configuração muito básica do novo controlador no módulo do roteador.

O `subscription_controller.ex`, módulo essencial e crítico do nosso microsserviço, com base no pedido recebido pelo `router.ex`, poderá fazer uso de dois outros módulos: `payments.ex` e `subscriptions.ex`. Além disso, este também faz uso do componente `RabbitMQ`, com o auxílio do protocolo AMQP.

[5] *Model-View-Controller*.

Os módulos `payments.ex` e `subscriptions.ex` são, na *framework* escolhida, Phoenix, chamados de contextos. Este tipo de módulos é responsável por expor funcionalidades de acesso ao sistema de dados, de forma encapsulada e com recurso a um ORM[6] (neste caso, Ecto). Os mesmos podem, ainda, expor funcionalidades de acesso a APIs externas, como é o caso do Stripe para o módulo `payments.ex`.

Existem também os módulos `payment.ex` e `subscription.ex` que pretendem isolar a definição da estrutura de dados que irá ser utilizada em memória e como resultado do mapeamento do sistema de dados para o Elixir, decorrente da utilização de um ORM.

4.5. Utilizadores

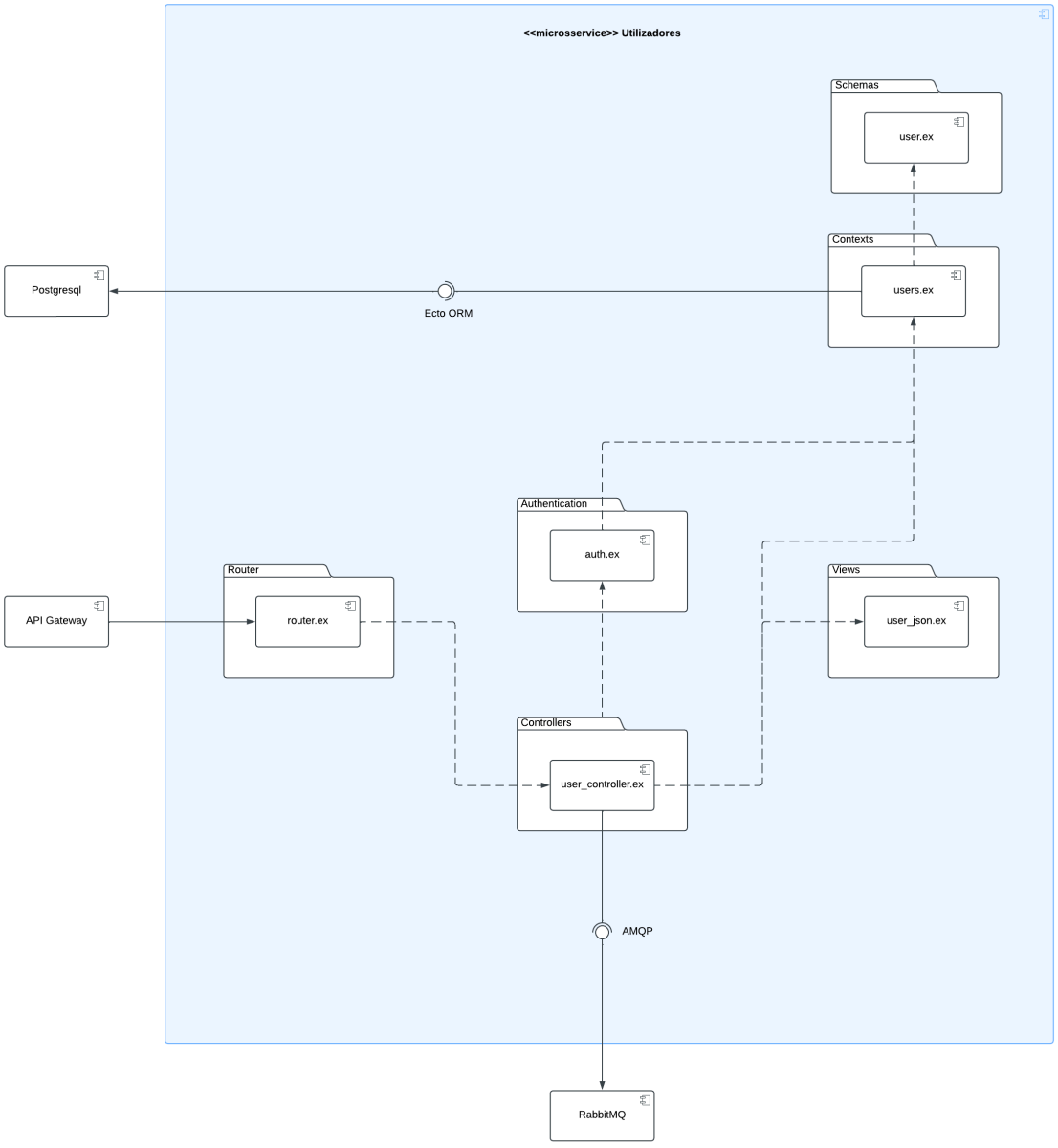

Figura 11: Diagrama de componentes do Microsserviço de Utilizadores

Este diagrama abrange todos os pacotes, módulos e interações entre os componentes do microsserviço de subscrições. Semelhante ao microsserviço das Subscrições, este adota uma arquitetura MVC[7], permitindo

[6] Uma vez que Elixir não é uma linguagem orientada aos objetos, mas sim funcional, o nome correto seria simplesmente Data Mapper.

[7] *Model-View-Controller*.

uma clara separação de responsabilidades e contribuindo para a escalabilidade e organização geral do sistema.

O ponto de entrada para o programa encontra-se no módulo `router.ex`, e está responsável por delegar a execução para o controlador, `user_controller.ex`, com base no pedido feito. Na arquitetura atual, o `router.ex` delega exclusivamente para este controlador. Contuda, a equipa de trabalho considera que a inclusão de um roteador contribui para uma expansão mais ágil da aplicação, requerendo apenas uma configuração simples de novos controladores no módulo do roteador.

O módulo `user_controller.ex`, elemento central do microsserviço, processa os pedidos encaminhados pelo `router.ex` e pode utilizar os módulos `users.ex` e `auth.ex` para responder adequadamente às solicitações. Adicionalmente, este controlador interage com o componente `RabbitMQ` para troca de mensagens, com o auxílio do protocolo AMQP.

O módulo `users.ex` é, na *framework* escolhida, Phoenix, chamado de contexto. Este tipo de módulo é responsável por expor funcionalidades de acesso ao sistema de dados, de forma encapsulada e com recurso a um ORM[6] (neste caso, Ecto).

Além disso, o módulo `user.ex` é responsável por definir a estrutura de dados utilizada em memória, bem como pelo mapeamento do sistema de dados para o Elixir, resultado da adoção de um ORM.

Além disso, por se tratar de um microsserviço que tem de lidar com a autenticação dos utilizadores, existe um módulo `auth.ex`. Este módulo é responsável por gerar e validar *tokens* de autenticação JWT, utilizadas, mais tarde, pelo Gateway (como explicado em capítulos anteriores).

4.6. Projetos

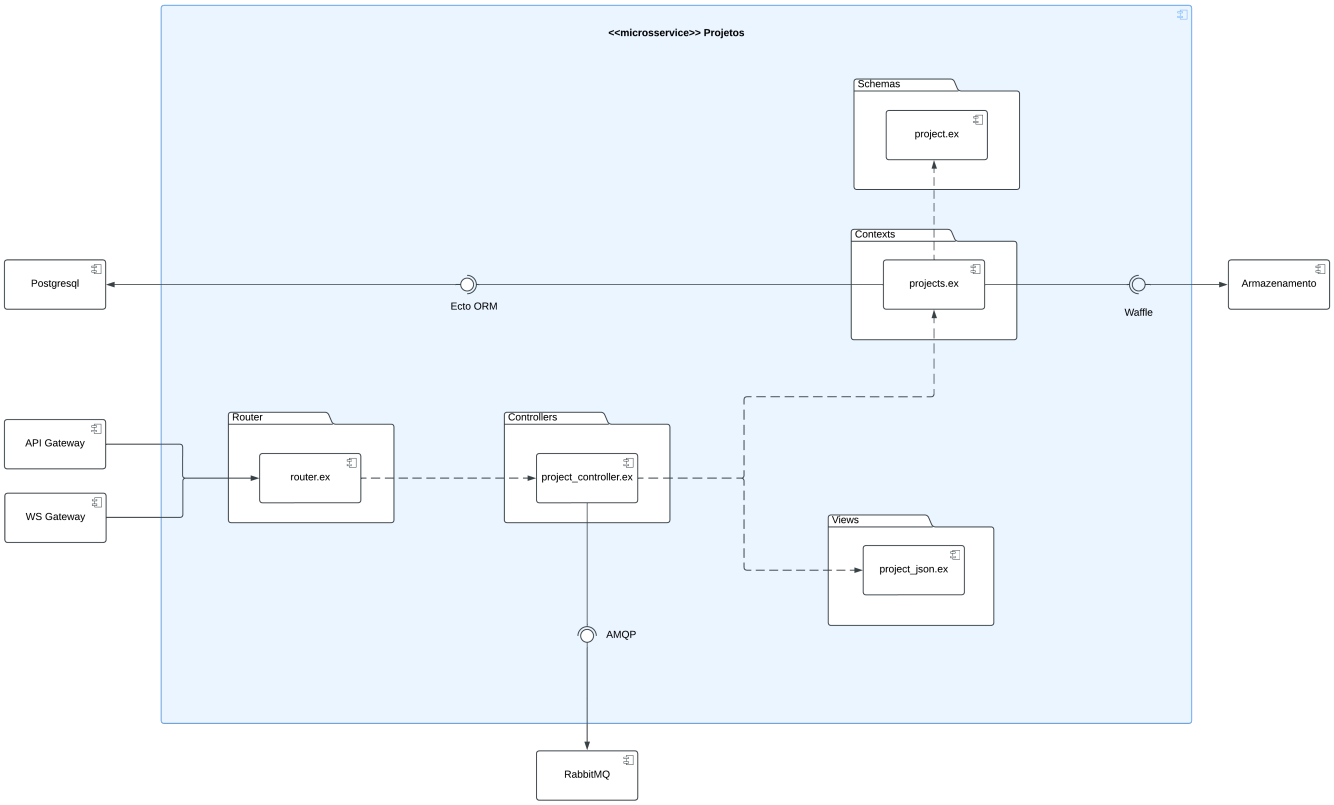

Figura 12: Diagrama de componentes do Microsserviço de Projetos

Este diagrama detalha todos os pacotes, módulos e interações entre os componentes que compõem o microsserviço de projetos. Internamente, este microsserviço utiliza a arquitetura MVC[5],que estrutura as

interações entre os módulos de acordo com a separação de responsabilidades. Este modelo arquitetural contribui para uma organização mais clara e facilita a manutenção e evolução do microsserviço.

O programa possui um ponto de entrada localizado no módulo `router.ex` que encaminha, com base no pedido recebido, a execução para o controlador `project_controller.ex`. Na configuração atual, o `router.ex` apenas redireciona para este controlador. No entanto, a inclusão de um roteador foi uma decisão estratégica para facilitar futuras expansões da aplicação, sendo apenas necessária uma configuração muito básica do novo controlador no módulo do roteador.

O módulo `project_controller.ex`, é um dos principais componentes do microsserviço, faz uso de outro módulo: `projects.ex`. Adicionalmente, este faz, também, uso do componente `RabbitMQ`, com o auxílio do protocolo AMQP.

Na *framework* escolhida, Phoenix, o módulo `projects.ex` desempenha o papel de contexto. Este módulo encapsula o acesso aos dados do sistema, expondo funcionalidades de forma estruturada e utilizando o ORM[6] (neste caso, Ecto).

Existe, ainda, o módulo `project.ex` que pretende isolar a definição da estrutura de dados que irá ser utilizada em memória e como resultado do mapeamento do sistema de dados para o Elixir, um processo realizado com o auxílio de um ORM.

Finalmente, e, uma vez que existe a necessidade de guardar imagens, necessitamos de uma solução compliant com S3 (tal como descrito em capítulos anteriores). Neste caso, iremos utilizar a biblioteca Waffle, cujo papel se centra em estabelecer uma API com serviços deste tipo e, ainda, se integrar com a tabela de dados onde constam os links para as imagens.

4.7. Ferramenta

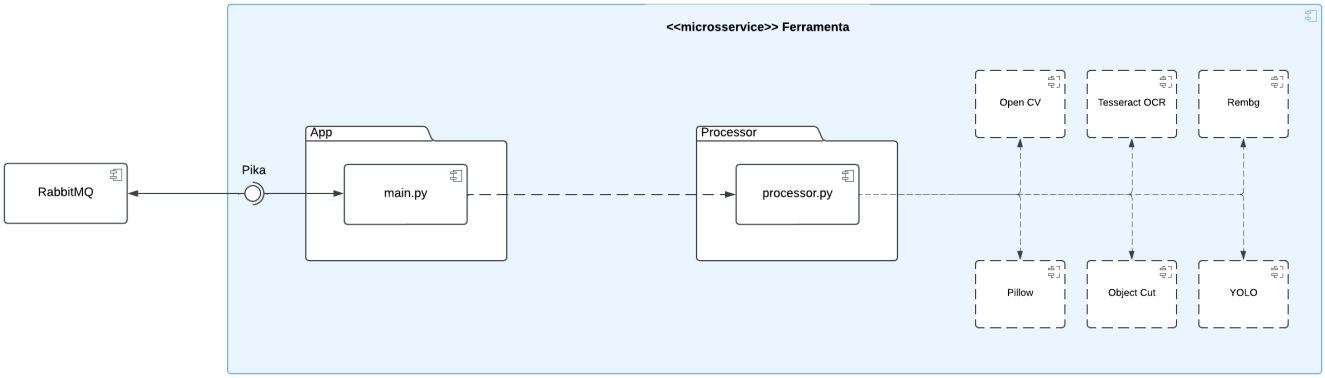

Figura 13: Diagrama de componentes do Microsserviço de uma ferramenta

O diagrama apresentado abrange a estrutura de pacotes, módulos e as principais interações do microsserviço de Ferramentas. Este microsserviço é um microsserviço de computação e processamento e como tal segue uma lógica ligeiramente diferente dos, anteriormente, mencionados.

As informações entram neste microsserviço através da publicação de mensagens no `RabbitMQ`. O módulo `main.py`, auxiliando-se da biblioteca `Pika`, é responsável por processar as mensagens recebidas e encaminhá-las para o módulo `processor.py`.

Este módulo, `processor.py`, é responsável exclusivamente pelo processamento das imagens, aplicando as alterações solicitadas com base no pedido recebido. Para tal (e como descrito em capítulos anteriores), este utilizada diferentes bibliotecas presentes no ecossistema de Python, tais como: `Open CV`, `Pillow`, `Tesseract OCR`, `Object Cut`, `Rembg` e `YOLO`.

Note-se que existirão diferentes instâncias deste tipo de microsserviço, especializadas numa dada ferramenta. Esta característica facilita escalar horizontalmente e de forma atómica uma ferramenta específica; e estender a aplicação com novas ferramentas sem que hajam restrições tecnológicas de suma importância.

5. Runtime View

Na secção abaixo será expresso o comportamento dinâmico do sistema através de diagramas de sequência relativos aos *use cases* criados. Estes irão focar-se nas interações principais entre os componentes do sistema.

De notar que, durante o desenvolvimento dos diagramas de sequência, foi utilizada uma notação especial para mensagens assíncronas (setas a tracejado). Ainda, a receção destas mensagens é, obviamente, feita de forma assíncrona, no entanto, e por simplicidade, decidiu-se que a sua receção iria ser descrita diretamente no diagrama de sequência onde a sua publicação acontece. Além disso, para representar interações com o Frontend, optou-se por utilizar expressões explicativas em vez de chamadas diretas às funções, por serem mais adequadas na nossa solução e clarificarem o funcionamento geral das mesmas.

5.1. Registo

Este *use case* descreve o registo de um utilizador que ainda não tem conta no PictuRAS. O processo inicia-se com o utilizador a inserir os dados necessários, seguido de uma validação e autenticação por parte do sistema.

De seguida, são apresentados os vários perfis disponíveis: gratuito, mensal e anual. Para os dois perfis pagos, é necessário que o utilizador insira os dados para o pagamento e o confirme, ficando, no final, corretamente registado no sistema.

Para toda a inserção de dados por parte do utilizador, ocorre uma verificação adequada dos mesmos, sendo apresentadas mensagens de erro, caso aplicável.

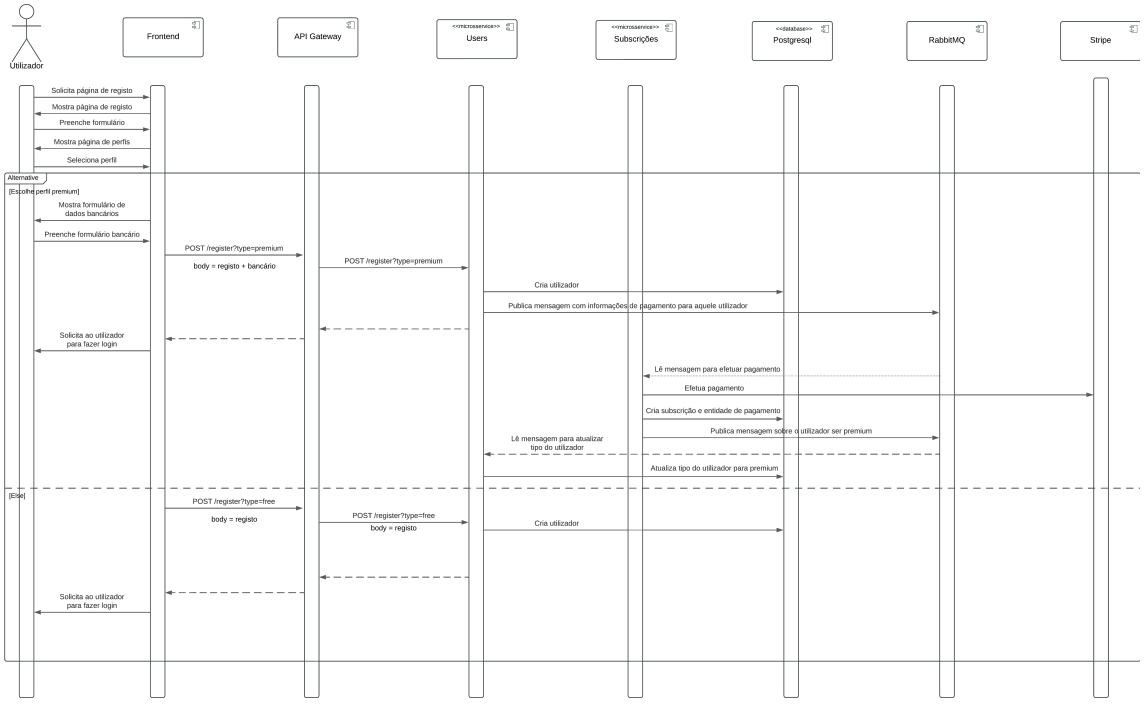

Figura 14: Diagrama de sequência referente ao registo

Neste diagrama, tomamos a liberdade de assumir algumas interações como garantidas:

- Caso o utilizador escolha ser *premium*, existe a necessidade de interagir com outro microsserviço (o de subscrições) que, por sua vez, interage com um sistema externo, o Stripe. De facto, a criação do utilizador e o respetivo pagamento deveria ser uma operação atómica. No entanto, estando perante uma arquitetura distribuída, a equipa de trabalho assume que inevitavelmente se irá atingir uma consistência geral do sistema. Desta forma, quando o microsserviço de subscrições lê uma mensagem relativa a efetuar um pagamento a mesma é lida sem confirmação de *acknowledgement* (ou seja, fica na fila para mais tarde ser, finalmente, *acknowledged*). Mais tarde, e após o pagamento ter sido efetivamente aceite, a mensagem é retirada da fila. Isto garante que o microsserviço de subscrições possui uma tolerância a falhas sem que comprometa a integridade do sistema como um todo.

5.2. Login

O *use case* relativo ao *login* refere-se à autenticação de um utilizador previamente registado no sistema. O utilizador insere as suas credenciais, que são submetidas para validação.

Caso estas estejam incorretas, o sistema informa o utilizador do erro e permite a reinserção dos dados. Uma vez validadas as credenciais corretamente, o utilizador inicia sessão na sua conta.

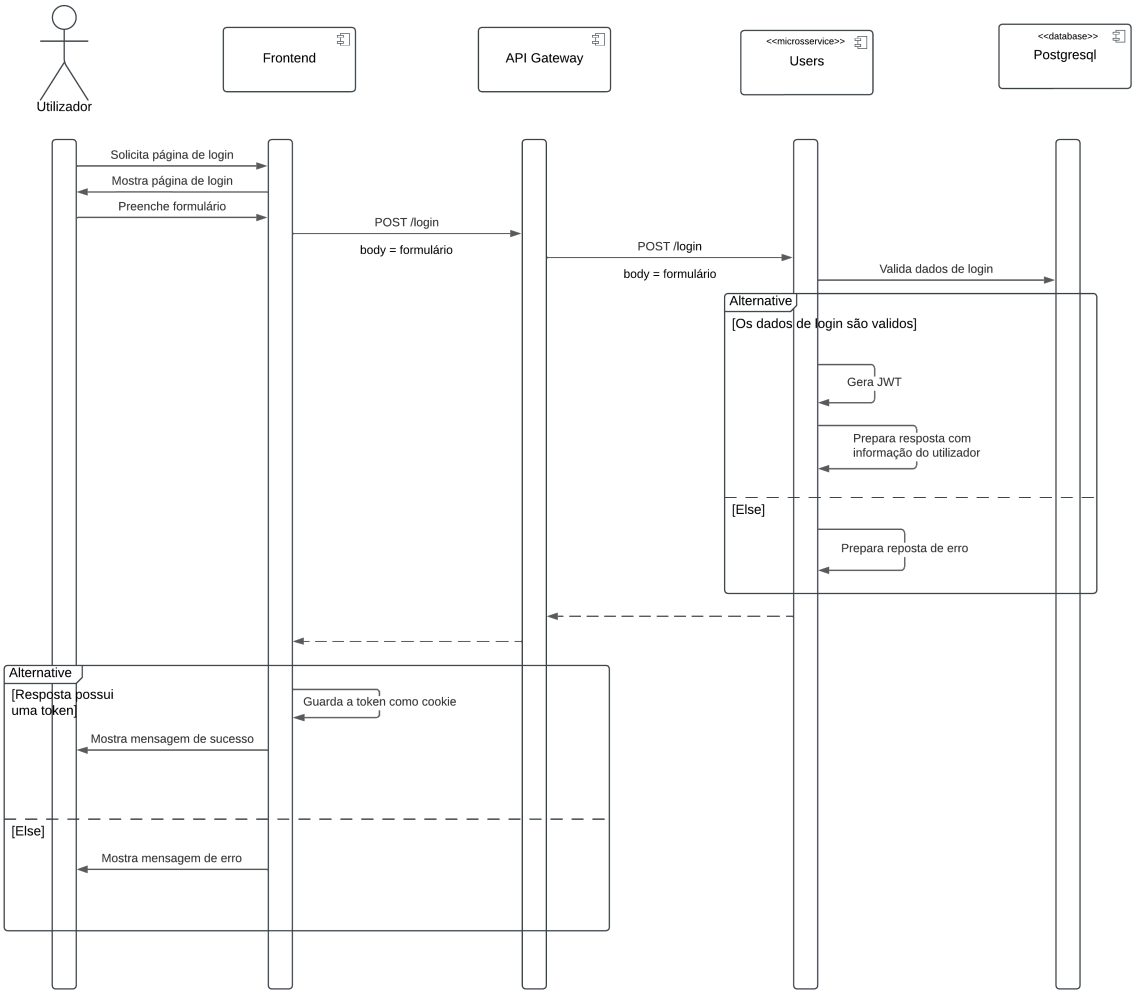

Figura 15: Diagrama de sequência referente ao *login*

5.3. Carregar imagens

O *use case* de carregar imagens refere-se ao ato de o utilizador carregar imagens para um determinado projeto ou para um novo.

Após carregar as imagens para o sistema, é apresentada a possibilidade de criar um novo projeto, sendo necessário, para isso, introduzir um nome para o mesmo. O utilizador pode, também, simplesmente selecionar o projeto desejado. Depois de escolhido o projeto, as imagens são adicionadas ao mesmo.

Tal como em *use cases* anteriores, são realizadas validações quanto ao tamanho das imagens ou ao número de imagens carregadas, sendo sempre apresentadas mensagens relativas aos erros detetados.

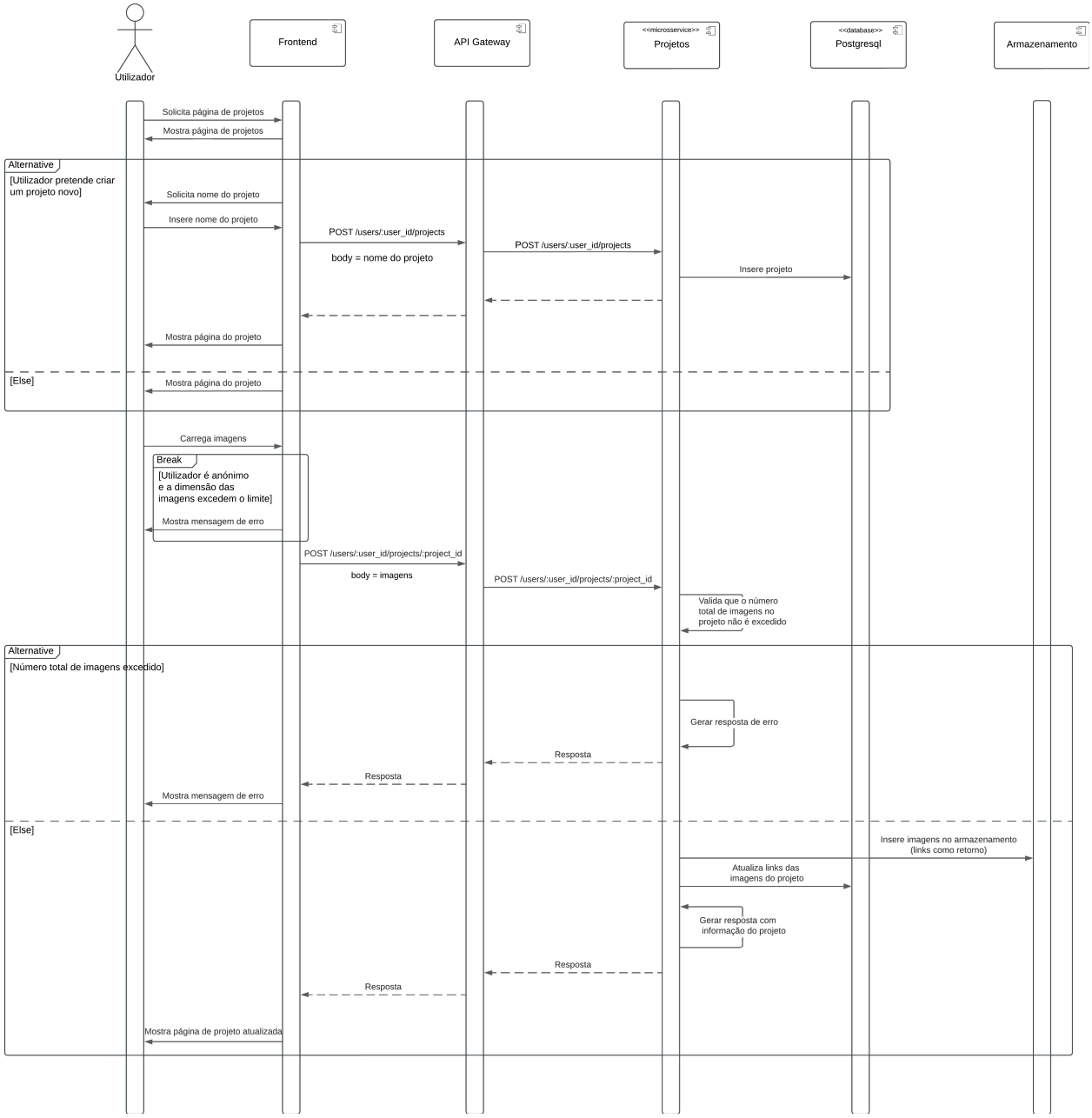

Figura 16: Diagrama de sequência referente ao carregamento de imagens

Neste diagrama, tomamos a liberdade de assumir algumas interações como garantidas:

- Assume-se que antes do diagrama de sequência “ocorrer” foi realizado, com sucesso, um pedido do tipo `GET` para `/users/:user_id/projects`, cuja resposta contém uma lista extensa de todos os projetos daquele utilizador.

5.4. Aplicar encandeamento de ferramentas a conjunto de imagens

Este *use case* refere-se à funcionalidade de aplicar ferramentas (com diversos parâmetros) a um projeto.

Após selecionar o projeto desejado, o utilizador, de forma interativa, escolhe e parametriza as ferramentas pretendidas. De seguida, ocorre o processamento das imagens por parte do sistema. Caso o utilizador esteja satisfeito, poderá, no final, descarregar as imagens processadas.

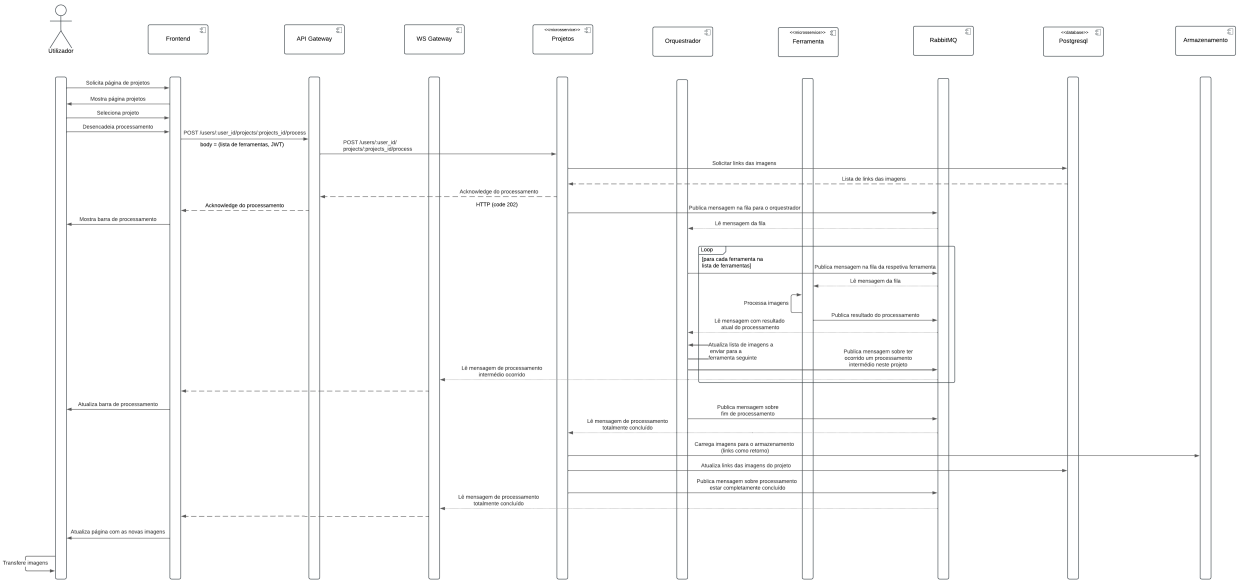

Figura 17: Diagrama de sequência referente à aplicação de um encadeamento de ferramentas a conjunto de imagens

Neste diagrama, tomamos a liberdade de assumir algumas interações como garantidas:

- A conexão por via de *web* sockets com o WS Gateway (e, consequentemente, todo o manuseio de estado interno deste componente) já se encontra corretamente estabelecida.
- Assume-se também que a escolha ordenada das ferramentas a aplicar no projeto foi realizada através da UI.
- Outros pedidos, como o de listar os projetos (mesmo `GET` relatado anteriormente) ou as ferramentas disponíveis (consoante o tipo de utilizador) são também assunções.
- Durante a interação com o API Gateway a verificação de autenticação e autorização é também corretamente realizada (como explicado em capítulos anteriores).

A decisão de não colocar estas partes da interação centra-se no facto de considerarmos que não são muito importantes para o *use case* em si, uma vez que se desviam do foco principal do mesmo: o encandeamento das ferramentas.

6. Deployment View

Nesta secção vamos apresentar o diagrama de *deployment* do PictuRAS, que descreve como é que a aplicação seria instalada e executada num cenário de produção.

6.1. Ambiente de Produção

A aplicação requer uma infraestrutura robusta para atender todas as necessidades dos utilizadores. Assim, e de forma a alcançar esse objetivo, sempre preservando o desempenho e a escalabilidade necessários, a equipa de desenvolvimento decidiu que cada componente do sistema deverá, sempre que possível, executar num ambiente dedicado e isolado.

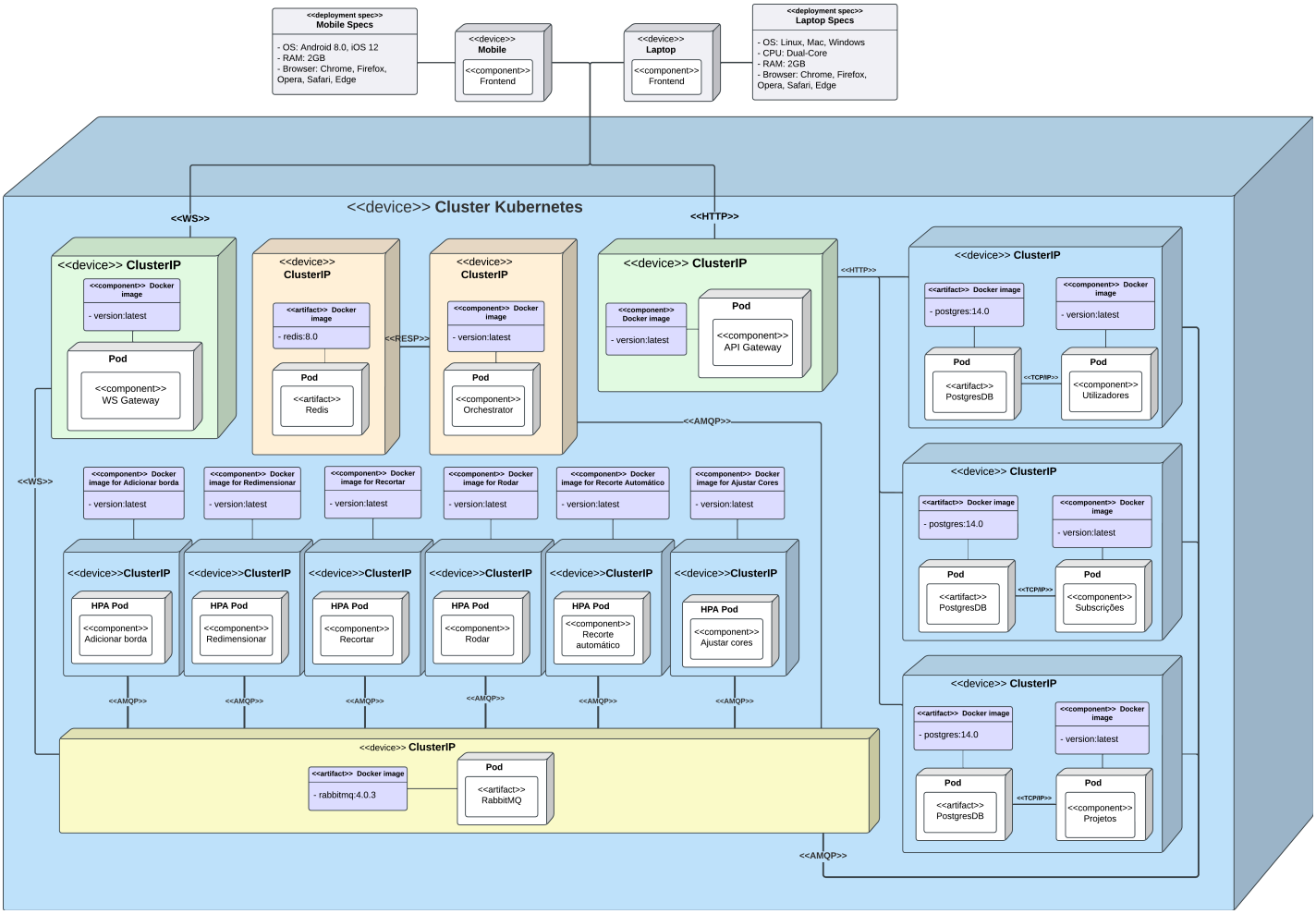

Figura 18: Diagrama de deployment do PictuRAS

6.2. Monitorização

Para garantir a fiabilidade, disponibilidade e desempenho do PictuRAS, é fundamental implementar um sistema de monitorização eficiente. Para isso, devemos utilizar ferramentas robustas e amplamente reconhecidas no mercado.

Assim, propomos a utilização de serviços externos de monitorização, como Elastic Stack e Prometheus.

O Elastic Stack (também conhecido como ELK) oferece uma solução abrangente para a recolha, análise e visualização de *logs*. Através do Elasticsearch, Logstash e Kibana, é possível monitorizar métricas de desempenho, detetar anomalias e gerar alertas em tempo real. Isto permite que problemas sejam identificados rapidamente, evitando interrupções nos diferentes serviços.

O Prometheus será utilizado para a recolha de métricas baseadas em séries temporais, sendo ideal para a monitorização da infraestrutura e dos microsserviços. Com suporte para alertas configuráveis e integração nativa com Kubernetes, torna-se uma solução essencial para acompanhar o desempenho e o estado dos componentes do sistema. Um exemplo de utilização para este componente seria a medição constante do número de mensagens presentes nas filas do RabbitMQ para perceber de que forma se deveria manusear o número de réplicas consumidoras.

7. Apêndice

7.1. *Schema* e exemplos de eventos (RabbitMQ)

Nesta secção, são propostos os *schemas* e alguns exemplos dos eventos apresentados de comunicação com os vários microsserviços de Ferramentas. Estes eventos são fundamentais para a execução das transformações e para a monitorização do seu estado, permitindo criar um fluxo de execução eficiente e estruturado.

7.1.1. Pedido para aplicação de uma ferramenta

JSON Schema

```
{
   "type": "object",
   "additionalProperties": false,
   "properties": {
      "messageId": {
         "type": "string",
         "description": "A unique identifier for the message to allow tracing and logging."
      },
      "timestamp": {
         "type": "string",
         "description": "The time when the message was created.",
         "format": "date-time"
      },
      "procedure": {
         "type": "string",
         "description": "The type of transformation to apply (e.g., 'peopleCount')."
      },
      "parameters": {
         "type": "object",
         "description": "Parameters specific to the procedure.",
         "additionalProperties": true
      }
   },
   "required": [
      "messageId",
      "timestamp",
      "procedure",
      "parameters"
   ]
}
```

Exemplo

```
{
   "messageId": "request-2",
   "timestamp": "2024-11-01T12:00:00Z",
   "procedure": "rotate",
```

```
   "parameters": {
      "inputImageURI": "file:///images/request-1-out.jpg",
      "outputImageURI": "file:///images/request-2-out.jpg",
      "degrees": -90
   }
}
```

7.1.2. Conclusão da aplicação de uma ferramenta

JSON Schema

```
{
   "type": "object",
   "additionalProperties": false,
   "properties": {
      "messageId": {
         "type": "string",
         "description": "A unique identifier for the completion message."
      },
      "correlationId": {
         "type": "string",
         "description": "The ID of the original request message this one relates to."
      },
      "timestamp": {
         "type": "string",
         "description": "The time when the process finished.",
         "format": "date-time"
      },
      "status": {
         "type": "string",
         "description": "The result of the processing.",
         "enum": [
            "success",
            "error"
         ]
      },
      "output": {
         "type": [
            "object",
            "null"
         ],
         "description": "The output produced by the process.",
         "additionalProperties": false,
         "properties": {
            "type": {
               "type": "string",
               "description": "The type of output produced.",
               "enum": [
                  "image",
                  "number",
                  "json",
                  "text",
                  "other"
               ]
            },
            "imageURI": {
               "type": [
                  "string",
                  "null"
               ],
               "description": "The URI or path of the output file, if applicable.",
```

```
               "format": "uri"
            },
            "value": {
               "type": [
                  "string",
                  "number",
                  "object",
                  "null"
               ],
                 "description": "The output value, if not a file. Can be JSON, text, or
a number."
            }
         },
         "required": [
            "type"
         ]
      },
      "error": {
         "type": [
            "object",
            "null"
         ],
         "description": "Details about the error, if the process failed.",
         "additionalProperties": false,
         "properties": {
            "code": {
               "type": "string",
               "description": "A short code identifying the type of error."
            },
            "message": {
               "type": "string",
               "description": "A detailed error message explaining what went wrong."
            },
            "details": {
               "type": "object",
               "description": "Additional error-specific details.",
               "additionalProperties": true
            }
         },
         "required": [
            "code"
         ]
      },
      "metadata": {
         "type": "object",
         "description": "Optional additional information about the process.",
         "additionalProperties": true,
         "properties": {
            "processingTime": {
               "type": "number",
               "description": "The time in seconds it took to process the request."
            },
            "microservice": {
               "type": "string",
               "description": "The name of the microservice that processed the request."
            }
         }
      }
   },
   "required": [
      "messageId",
      "correlationId",
```

```
      "timestamp",
      "status"
   ]
}
```

Exemplo, em caso de sucesso

```
{
   "messageId": "completion-2",
   "correlationId": "request-2",
   "timestamp": "2024-11-01T12:00:01Z",
   "status": "success",
   "output": {
      "type": "image",
      "imageURI": "file:///images/request-2-out.jpg"
   },
   "metadata": {
      "processingTime": 0.2,
      "microservice": "picturas-rotate-tool-ms"
   }
}
```

Exemplo, em caso de erro

```
{
   "messageId": "completion-20",
   "correlationId": "request-20",
   "timestamp": "2024-11-01T12:01:00Z",
   "status": "error",
   "error": {
      "code": "INVALID_INPUT",
      "message": "The input file format is not supported.",
      "details": {
         "inputFileURI": "file:///images/unsupported-file.xyz"
      }
   },
   "metadata": {
      "processingTime": 0.1,
      "microservice": "picturas-crop-tool-ms"
   }
}
```