Universidade do Minho
Escola de Engenharia

Mestrado em Engenharia Informática
Requisitos e Arquiteturas de Software

Projeto PictuRAS - Fase 2

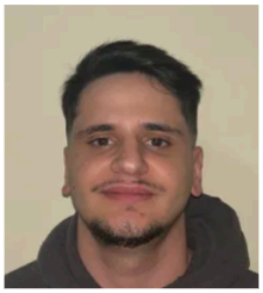
Novembro 2024

pg57511	**Benjamim Rodrigues**
pg57868	**Bruno Gião**
pg57549	**Hugo Gomes**
pg55956	**João Magalhães**
pg57565	**João Ribeiro**
pg55972	**José Pacheco**
pg57884	**Lara Pereira**
pg57885	**Lingyun Zhu**
pg57582	**Luís Ferreira**
pg55992	**Pedro Lopes**

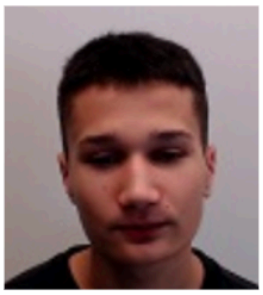
Índice












Número	Membro	Contributo
PG55956	João Luís Ferreira Magalhães	+ 1.5
PG57884	Lara Regina da Silva Pereira	0
PG57868	Bruno Dias Gião	0
PG57549	Hugo Ricardo Macedo Gomes	-1
PG55972	José Pedro Teixeira Pacheco	+ 1.5
PG57511	Benjamim Meleiro Rodrigues	-1.5
PG57565	João Machado Ribeiro	-1
PG57885	Lingyun Zhu	+ 1.5
PG57582	Luís Miguel Moreira Ferreira	-1
PG55992	Pedro Afonso Moreira Lopes	0

1 Introdução e Objetivos

O *PictuRAS* é um projeto destinado ao desenvolvimento de uma aplicação *web* de edição de imagens, com o objetivo de fornecer ao utilizador uma experiência simples, intuitiva e funcional. A aplicação destina-se ao público diversificado, incluindo utilizadores com diferentes níveis de familiaridade em tecnologias de edição.

Esse sistema permite o *upload*, processamento, edição e exportação de imagens por meio de um conjunto de ferramentas, organizadas de forma modular e configurável, de modo a possibilitar a aplicação de edições básicas e avançadas.

O desenvolvimento do *PictuRAS* segue uma abordagem orientada por um conjunto de requisitos funcionais e não funcionais, considerados fundamentais para o sucesso do sistema. Estes requisitos foram elaborados com o intuito de garantir que a aplicação é:

- Intuitiva e fácil de usar para o público geral, proporcionando uma interface amigável;
- Altamente escalável para acomodar um grande número de utilizadores em simultâneo, sem comprometer com o desempenho;
- Flexível e extensível, permitindo a integração de novas ferramentas de edição e de suporte para futuras expansões;
- Otimizada para os custos, ajustando automaticamente o consumo de recursos com a demanda.

1.1 Resumo dos Requisitos

Na fase inicial do desenvolvimento deste projeto, foram definidas um conjunto de funcionalidades essenciais para um editor de imagens, como *login* e registo de utilizador, *upload* de imagens e aplicação de ferramentas de edição a um conjunto de imagens. Podemos vê-las ilustradas na seguinte figura:

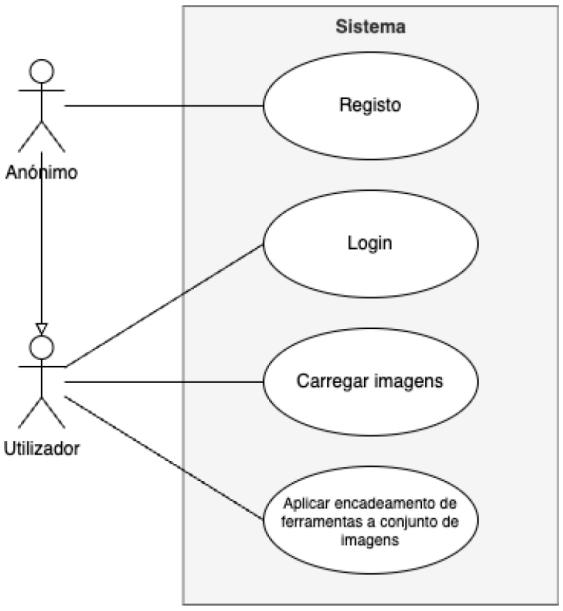


Figura 3: Diagrama de Use Cases

Considerando o documento de requisitos construído na fase de desenvolvimento anterior e as funcionalidades essenciais do sistema, foram destacados os seguintes requisitos funcionais, considerados prioritários para a implementação de uma versão inicial do sistema.

Requisito	**Use cases**	**Descrição**	**Prioridade**
Req1	*Login* de Utilizador	O utilizador autentica-se utilizando as suas credenciais.	Must
Req2	Registo do Utilizador	O utilizador anónimo regista-se.	Must
Req3	Registo do Utilizador	O utilizador escolhe o plano de subscrição Gratuito.	Must
Req4	Registo do Utilizador	O utilizador escolhe o plano de subscrição Premium.	Must
Req6	Carregar imagens	O utilizador cria um projeto.	Must
Req8	Carregar imagens	O utilizador acede à área de edição de um projeto.	Must
Req9	Carregar imagens	O utilizador carrega imagens para um projeto.	Must
Req11	Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador adiciona uma ferramenta de edição ao projeto.	Must
Req12	Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador desencadeia o processamento de um projeto.	Must
Req13	Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador transfere o resultado de um projeto para o dispositivo local.	Must
Req15	Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador altera a ordem das ferramentas de um projeto.	Must
Req16	Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador altera os parâmetros das ferramentas.	Must

Os requisitos não funcionais incluem aspetos críticos de desempenho, segurança, escalabilidade e usabilidade, que são fundamentais para o potencial uso do sistema em grande escala e para competir com aplicações de edição de imagem já existentes no mercado.

1.2 Objetivos de Qualidade

Com base nos requisitos não funcionais identificados, foram estabelecidos cinco principais objetivos de qualidade para orientar a arquitetura do *PictuRAS*.

Estes objetivos, priorizados de acordo com a sua relevância para a aplicação, são:

- **Usabilidade**: o sistema é destinado a um público amplo e não necessariamente técnico. Para que a aplicação se destaque e atenda às expectativas do utilizador, a interface deve ser amigável, intuitiva e fácil de navegar, ou seja, é essencial que as funcionalidades principais sejam facilmente acessíveis, e que possuam um *design* visual que minimize a necessidade de instruções.
- **Escalabilidade e Elasticidade**: é necessário suportar grandes volumes de utilizadores, especialmente em alturas de picos, sem comprometer o desempenho. Para evitar custos desnecessários em períodos de baixa utilização é também necessário que o sistema se ajuste automaticamente o consumo de recursos de acordo com o volume de acesso.
- **Extensibilidade**: dada a constante evolução do mundo de edição de imagens, a aplicação precisa de ser extensível e permitir a adição de novas ferramentas e funcionalidades ao longo do tempo, ou seja, é vantajoso utilizar uma abordagem modular, onde cada funcionalidade de edição possa ser tratada independentemente como um microsserviço.
- **Desempenho e Eficiência de Resposta**: para as operações frequentes, como o carregamento de imagens e a aplicação de ferramentas de edição, é necessário que o tempo de resposta do sistema seja mínimo. É fundamental proporcionar uma experiência fluida, especialmente para operações em lote, onde várias imagens são editadas simultaneamente.
- **Segurança e Conformidade**: é necessário proteger informações de *login*, dados de utilizadores e arquivos de imagem para garantir que apenas utilizadores autorizados tenham acesso aos seus projetos. A conformidade com os regulamentos de proteção de dados é essencial para garantir que as informações pessoais e as imagens carregadas são armazenadas de forma segura e manuseadas com responsabilidade.

2 Restrições

2.1 Restrições da Implementação do Projeto

O desenvolvimento da aplicação *PictuRAS* enfrenta diversas restrições que influenciam tanto o processo de *design* quanto as decisões de implementação. Estas restrições são fundamentais para garantir que a aplicação atende aos requisitos iniciais do projeto, que envolvem compatibilidade, eficiência, segurança e conformidade com os padrões modernos de desenvolvimento.

2.1.1 Restrições Técnicas

- **Acessibilidade através de Navegador Web**: A aplicação deve ser acessível e compatível com os navegadores modernos, como o Google Chrome, o Mozilla Firefox, o Safari e o Microsoft Edge. Esta restrição é obrigatória, pois o *PictuRAS* visa ser uma solução acessível a um público amplo, e a experiência deve ser consistente independentemente do dispositivo utilizado.
- **Desempenho e Escalabilidade Horizontal**: A arquitetura deve ser projetada com a capacidade de se escalar horizontalmente e permitir a adição de novos servidores e instâncias de microsserviços. Este requisito implica o uso de microsserviços e *containers* (por exemplo, Docker e Kubernetes) para possibilitar uma rápida adaptação ao aumento da carga de trabalho sem comprometer com a performance.
- **Uso de Tecnologias para a Modularidade e Integração**: A arquitetura deve ser modular e permitir a que diferentes partes do sistema possam ser mantidas e escaladas independentemente. Para tal, mais uma vez, é essencial a abordagem de microsserviços e integrar tecnologias que suportem a modularidade.
- **Infraestrutura em Nuvem com Ajuste Dinâmico de Recursos**: Para otimizar o uso de recursos e reduzir os custos, o sistema deve operar numa infraestrutura de nuvem que permita ajustar os recursos dinamicamente, adaptando-se automaticamente às flutuações de demanda. Esta elasticidade é essencial para manter os custos baixos sem influenciar negativamente a performance em períodos de uso de pico.
- **Conformidade com Padrões de Segurança e Privacidade**: O sistema deve seguir os padrões de segurança para proteger as informações dos utilizadores e dos dados armazenados (imagens e projetos), bem como seguir diretrizes de proteção de dados para garantir a conformidade com as regulamentações de privacidade (criptografia para o armazenamento de informações sensíveis e na implementação de protocolos de segurança nas interfaces de *API*).

2.1.2 Restrições de Processo

- **Orçamento Limitado para a Infraestrutura e Recursos**: A aplicação deve ser desenvolvida e implementada com o orçamento disponível, o que impõe uma limitação significativa para as decisões de design e para as escolhas tecnológicas.

- **Tecnologias Predeterminadas**: O uso de tecnologias como Go e Node.js para o *backend*, Vue.js para o *frontend* e mongoDB como base de dados foi pré-estabelecido, limitando a equipa de trabalhar com essa *stack* tecnológica específica. Esta escolha visa garantir o desenvolvimento do projeto com ferramentas e *frameworks* amplamente utilizadas e bem documentadas, reduzindo a curva de aprendizagem.
- **Estrutura Modular para Suporte à Extensibilidade**: A estrutura modular impõe uma restrição de design, orientando a equipa a priorizar a modularidade e a flexibilidade na construção dos serviços.

2.1.3 Restrições de Âmbito

Este projeto deve cumprir requisitos específicos estabelecidos pelos *stakeholders*. As seguintes restrições foram identificadas como fundamentais para o desenvolvimento e operacionalização da aplicação:

- **Perfis de Utilizador:** A aplicação deve contemplar três perfis de utilização com permissões distintas:
  - **Anónimo:** Oferece operações a imagens com restrições de dimensão. Este perfil é indicado para utilizadores sem conta, permitindo o uso básico da aplicação.
  - **Registado:** Limita a cinco operações diárias em imagens, proporcionando o acesso a um conjunto básico de funcionalidades, ainda que, com limitações.
  - **Premium:** Não possui limites em relação ao número de operações ou ao tamanho das imagens, oferecendo o máximo de flexibilidade e acesso às funcionalidades.
- **Ferramentas Básicas:** A aplicação deve incluir um conjunto essencial de ferramentas para o processamento de imagens, tais como binarização, redimensionamento e remoção de fundo. Todas as ferramentas básicas devem permitir que o utilizador ajuste os parâmetros, de forma a atender às necessidades específicas de edição de imagem.
- **Ferramentas Avançadas:** Para além das funções básicas, a aplicação deve disponibilizar funcionalidades avançadas como é o caso de:
  - Contar pessoas
  - Extrair texto presente na imagem (OCR)
  - Identificar objetos
  - Colocar objetos em pessoas
- **Encadeamento de ferramentas e processamento em lote:** A aplicação permite encadear ferramentas, aplicando cada resultado como entrada da próxima. O utilizador pode definir uma sequência de edição automática para várias imagens em lote, carregadas via .zip ou diretoria, desde que as ferramentas sejam compatíveis. Isto facilita edições consistentes e eficientes em grandes quantidades de imagens.

2.2 Requisitos Obrigatórios

Os seguintes requisitos foram considerados obrigatórios para a solução, de forma a garantir que a aplicação cumpre os objetivos definidos:

- **Compatibilidade Web**: A aplicação deve ser acessível através do navegador, garantindo uma interface facilmente acessível sem a necessidade de pré-instalação de software.
- **Autenticação e Autorização Segura**: A segurança no login e nas permissões de acesso é essencial, de modo a garantir que apenas os utilizadores autenticados possam acessar e modificar os seus projetos e imagens.
- **Processamento de Edição Modular e Configurável**: O sistema deve suportar uma sequência configurável de ferramentas de edição, possibilitando ao utilizador de personalizar o fluxo de trabalho de edição de imagem. As ferramentas devem ser gerenciadas de forma modular, permitindo futuras expansões.
- **Armazenamento e Gestão de Projetos e Imagens**: A aplicação deve oferecer suporte para *upload*, armazenamento e organização de imagens e de projetos de forma eficaz. O armazenamento deve ser seguro e acessível, com backup e redundância para evitar a perda de dados.
- **Otimização de Recursos e Eficiência de Custos**: A infraestrutura em nuvem deve ser otimizada para garantir que o sistema ajuste dinamicamente o uso de recursos de acordo com a demanda, minimizando os custos durante períodos de baixa utilização e permitindo expansão rápida em alturas de picos.

2.3 Requisitos Não Funcionais

A equipa de desenvolvimento decidiu priorizar os requisitos de qualidade que exercem maior influência sobre o sistema, orientando assim à escolha da solução arquitetural. Esta decisão procura reforçar as qualidades essenciais da arquitetura para atender aos objetivos principais do projeto.

Requisito #: 21 Tipo: Desempenho *Use cases* #: 4

Descrição	**As ferramentas de edição de imagem devem ser processadas rapidamente.**
Rationale	A velocidade é crucial para garantir uma boa experiência ao utilizador, especialmente em tarefas básicas e repetitivas.
Origem	Engenheiro de Software
Fit criterion	Pelo menos 95% das ferramentas básicas devem executar em até 1 segundo por imagem e as avançadas em até 3 segundos. O tempo de execução nunca deve ultrapassar os 2 e 5 segundos, respetivamente para cada tipo de ferramenta.
Prioridade	Should
Data	2024/10/03

Figura 4: Requisito 21 - Performance

Requisito #: 22	Tipo: Desempenho	*Use cases* #: 4
Descrição	**O sistema deve processar até 100 imagens ao mesmo tempo, sem quebras percetíveis no desempenho.**	
Rationale	Suportar computação paralela é essencial para manter a eficiência e garantir que múltiplos utilizadores ou processamentos encadeados não causem *starvation*.	
Origem	Engenheiro de Software	
Fit criterion	O sistema deve processar 100 imagens simultâneas com uma degradação de desempenho inferior a 20%.	
Prioridade	Must	
Data	2024/10/03	

Figura 5: Requisito 22 - Escalabilidade

Requisito #: 23	Tipo: Desempenho	*Use cases* #: —
Descrição	**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.**	
Rationale	O crescimento do número de utilizadores pode sobrecarregar o sistema. A escalabilidade garante que o sistema continue eficiente com o aumento da carga.	
Origem	Engenheiro de Software	
Fit criterion	O sistema deve suportar crescimentos de 100% dos utilizadores a cada 10 minutos sem quebras de desempenho significativas (menos de 20% de variação no tempo de execução). Recursos são desalocados quando deixam de ser necessários.	
Prioridade	Must	
Data	2024/10/03	

Figura 6: Requisito 23 - Escalabilidade e Elasticidade

Requisito #: 25	Tipo: Operacional e Ambiental	*Use cases* #: —
Descrição	**A aplicação deve ser integrável com outras plataformas e serviços de terceiros.**	
Rationale	A integração com outras plataformas aumenta a utilidade, permitindo que os utilizadores partilhem e acedam às imagens em múltiplos serviços.	
Origem	Engenheiro de Software	
Fit criterion	APIs bem documentadas e que permitam a integração com serviços externos.	
Prioridade	Should	
Data	2024/10/03	

Figura 7: Requisito 25 - Adaptabilidade

Requisito #: 26	Tipo: Operacional e Ambiental	*Use cases* #: 4
Descrição	**A aplicação deve otimizar o uso de recursos computacionais do cliente (e.g., CPU, memória)**	
Rationale	Uma aplicação que consome menos energia em dispositivos móveis prolonga a duração da bateria e melhora a experiência do utilizador.	
Origem	Engenheiro de Software	
Fit criterion	A utilização de memória deve ser minimizada.	
Prioridade	Should	
Data	2024/10/03	

Figura 8: Requisito 26 - Eficiência

Requisito #: 32	Tipo: Manutenibilidade e Suporte	*Use cases* #: —
Descrição	**O sistema deve ser projetado para facilitar a execução de testes.**	
Rationale	A testabilidade assegura que a aplicação seja verificável, mantendo a qualidade e reduzindo o tempo de deteção de erros.	
Origem	Engenheiro de Software	
Fit criterion	Testes de regressão devem ser realizados antes de cada atualização para garantir que as funcionalidades anteriores não sejam afetadas.	
Prioridade	Must	
Data	2024/10/03	

Figura 9: Requisito 32 - Testabilidade

2.4 Impacto dos Requisitos Não Funcionais no Sistema

A performance das ferramentas de edição de imagem (**requisito 21**) é uma prioridade que garante que as imagens sejam processadas rapidamente para proporcionar uma experiência fluida e responsiva ao utilizador. Esse requisito é essencial para evitar atrasos tanto em operações básicas como avançadas, reforçando a agilidade e a usabilidade da aplicação. A arquitetura precisa de ser otimizada para reduzir o tempo de processamento por imagem.

A capacidade de processamento em lote (**requisito 22**) também é fundamental, permite que o sistema processe até 100 imagens ao mesmo tempo, sem falhas perceptíveis no desempenho. Isto exige uma arquitetura que suporte paralelização, aplicando técnicas de distribuição de carga para que múltiplas operações possam ser executadas simultaneamente, mantendo a performance estável.

A escalabilidade horizontal (**requisito 23**) é crucial para suportar um aumento no número de utilizadores e no volume de imagens, mantendo o desempenho consistente. A arquitetura deve ser capaz de expandir conforme a demanda, alocando recursos de forma dinâmica para assegurar que a aplicação continue eficiente mesmo em picos de utilização.

A integração com outras plataformas e serviços de terceiros (**requisito 25**) representa uma necessidade importante para expandir as funcionalidades da aplicação. Para isso, a arquitetura deve permitir integrações externas de forma segura e compatível, possibilitando a incorporação de novos recursos sem comprometer o desempenho.

A otimização do uso de recursos computacionais (**requisito 26**), como CPU e memória, é essencial para melhorar a eficiência e a experiência do utilizador, especialmente em dispositivos com recursos limitados. O sistema deve ser projetado para utilizar esses recursos de maneira inteligente, evitando sobrecargas que possam afetar a performance.

Por fim, a facilidade na execução de testes (**requisito 32**) reforça a importância de uma arquitetura modular e bem organizada. Isso facilita o desenvolvimento e a execução de testes automáticos, permitindo uma validação mais eficaz das funcionalidades e garantindo a confiabilidade do sistema durante todo o ciclo de desenvolvimento.

3 Contexto e Âmbito do Sistema

A aplicação foi concebida para operar num ambiente *web* e visa ser uma solução inovadora e acessível para o processamento e edição de imagens, atendendo a um público diversificado a nível de experiência de *design* gráfico.

Esta secção descreve o escopo do sistema *PictuRAS*, em relação aos sistemas externos e aos atores com os quais interage. Define, ainda, as interfaces externas do sistema, na perspectiva de negócios e na perspetiva técnica.

3.1 Perspetiva de Negócio/Domínio

Do ponto de vista de negócios, o sistema permite aos utilizadores criarem, editarem e organizarem projetos, com a possibilidade de aplicar sequências de ferramentas de edição em lote de várias imagens.

O desenvolvimento do *PictuRAS* surge da necessidade de superar as limitações das ferramentas de edição convencionais, que muitas vezes exigem uma infraestrutura avançada ou conhecimentos técnicos especializados. A plataforma pretende oferecer um ambiente flexível, permitindo a edição de imagens diretamente na *Cloud*, sem que os utilizadores precisem de recursos computacionais avançados.

O principal objetivo do *PictuRAS* é oferecer uma plataforma intuitiva e acessível a qualquer utilizador, independentemente do seu nível de experiência ou das limitações de hardware. Este conceito permite simplificar o acesso às ferramentas de edição de imagens e aos utilizadores de realizar operações complexas naturalmente.

3.2 Perspetiva Técnica

O *PictuRAS* é estruturado numa arquitetura de microsserviços, o que facilita a escalabilidade e a manutenção modular do sistema. Cada serviço é independente e interage com os outros por meio de *APIs* definidos. Isto permite que novas funcionalidades sejam adicionadas sem impacto significativo na estrutura existente.

A arquitetura é baseada nas seguintes principais componentes técnicas:

- ***Frontend* Web**: desenvolvido em Vue.js, interface responsiva que permite interagir com as funcionalidades de edição e gestão de projetos.
- **API Gateway**: ponto de entrada para todas as requisições do *frontend*, redirecionando-as para os respetivos serviços específicos.
- **Serviços Backend**: implementados em Go e Node.js, os serviços incluem gestão de utilizadores, processamento de imagens e gestão de projetos. Cada serviço é independente, seguindo a arquitetura de microsserviços para garantir escalabilidade e modularidade
- **Armazenamento de Dados**: a base de dados MongoDB é utilizada para armazenar dados do utilizador, projetos e configurações de edição, garantindo um acesso rápido e flexível.

3.3 Diagrama de Contexto

Depois de uma análise ao documento de requisitos disponibilizado, partindo dos *use cases* e dos requisitos funcionais, obtivemos o seguinte diagrama:

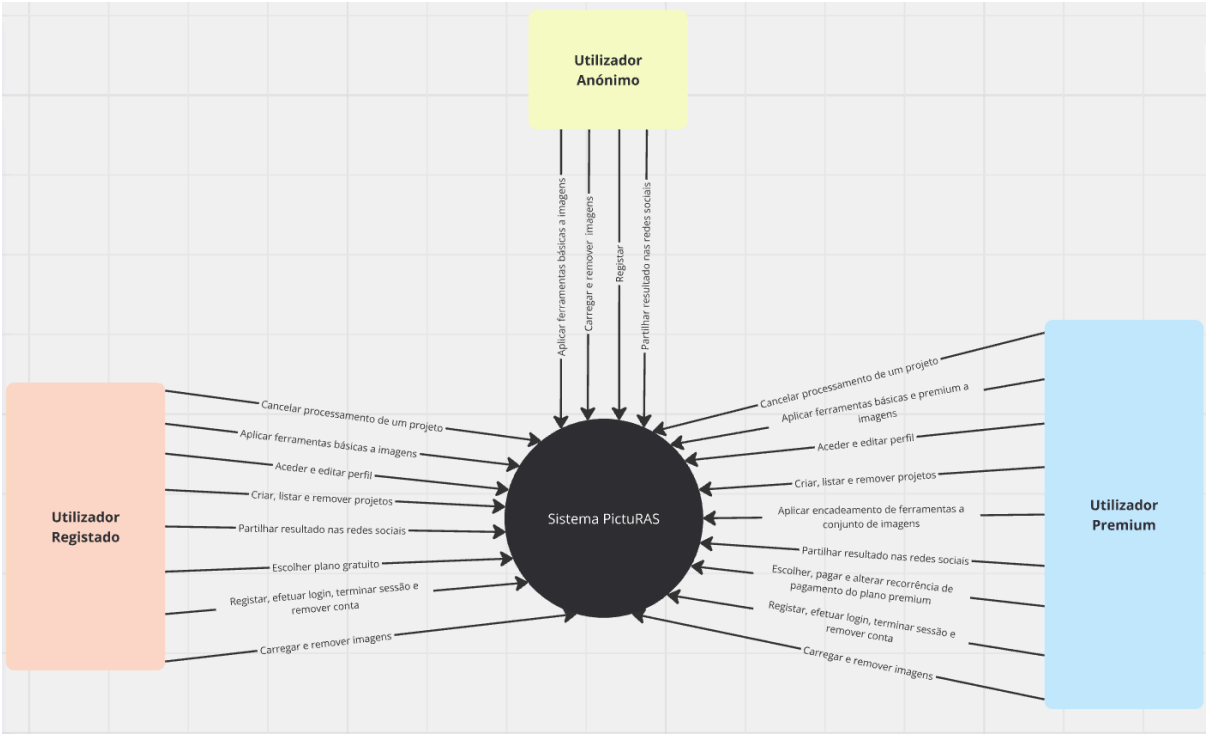


Figura 10: Diagrama de Contexto

O diagrama apresentado tem como objetivo ilustrar as interações entre o sistema e os seus utilizadores, ajudando a definir claramente os limites do sistema e as responsabilidades atribuídas a cada ator envolvido. Assim, fica claro que diferentes utilizadores terão acesso a funcionalidades distintas, dependendo do seu perfil.

4 Estratégias da Solução

A arquitetura do projeto *PictuRAS* foi desenvolvida para atender a requisitos de escalabilidade, usabilidade e extensibilidade. Esta secção fornece uma visão geral das decisões fundamentais e das estratégias de solução que moldam a arquitetura, incluindo, a decomposição do sistema de acordo com a sua funcionalidade, os padrões arquiteturais selecionados para alcançar os objetivos de qualidade e as tecnologias adotadas. A escolha de uma arquitetura modular e flexível visa não apenas satisfazer as necessidades atuais dos utilizadores, mas também facilitar a evolução da aplicação em resposta a demandas futuras.

4.1 Decomposição Funcional

A decomposição do sistema *PictuRAS* foi organizada num diagrama de componentes, onde cada funcionalidade é dividida numa área de responsabilidade, simplificando a arquitetura da aplicação. Para tal, foi necessário analisar detalhadamente os requisitos funcionais, o que resultou no seguinte diagrama:

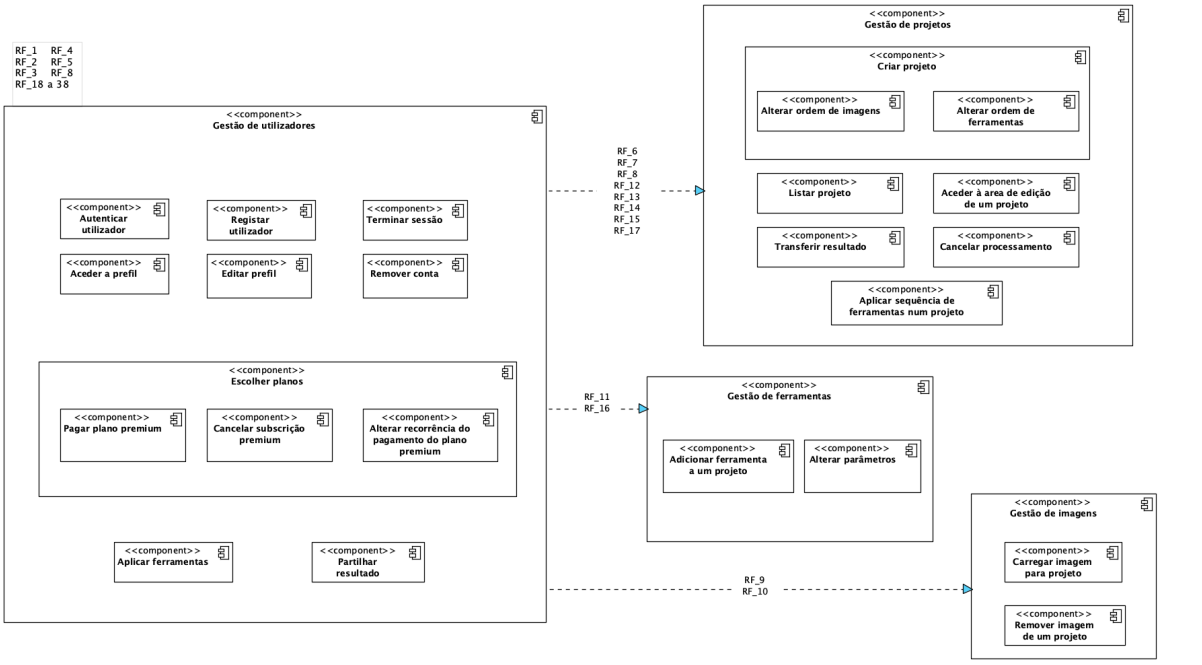


Figura 11: Diagrama de blocos - Decomposição Funcional

Como podemos observar no diagrama anterior, a decomposição funcional resultou em quatro blocos referentes à gestão das entidades principais do sistema: Gestão de utilizadores, Gestão de ferramentas, Gestão de projetos e Gestão de imagens.

Analisando o exemplo da Gestão de utilizadores, podemos verificar que esta secção agrupa todos os componentes responsáveis pelo ciclo de vida do utilizador, incluindo a autenticação, registo, edição e cancelar a conta. Além disso, possui um subgrupo para a gestão de planos, permitindo que o utilizador registado adquira um plano *Premium*, cancele ou ajuste a recorrência do

pagamento. Ao separar essas operações, o sistema garante que as funcionalidades de gestão de utilizadores são facilmente escaláveis e geridas de forma independente.

A partir do diagrama, constata-se que uma grande parte da complexidade do sistema está concentrada na Gestão de projetos. Este cenário sugere a necessidade de intensificar o esforço da equipa de desenvolvimento, especialmente na implementação detalhada das funcionalidades projetadas. Além disso, o diagrama evidencia as interações estabelecidas pelos requisitos funcionais entre os blocos funcionais, indicando possíveis dependências e cooperação entre eles.

4.2 Padrão arquitetural

Com o objetivo de escolher o padrão arquitetural mais adequado para o sistema *PictuRAS*, começamos identificar as arquiteturas que poderiam adequar-se à solução deste projeto:

- Layered;
- Pipe & Filter;
- Client-Server;
- Peer-to-Peer (P2P);
- Microsserviços;
- Arquitetura Orientada a Serviços (SOA).

Estas arquiteturas foram comparadas e avaliadas de acordo com a sua adequação ao cumprimento dos seguintes atributos arquiteturais, retirados da Secção dos Requisitos Não Funcionais: Operabilidade, Segurança, Resiliência e Performance. Como resultado obtivemos a seguinte tabela de comparação, onde os padrões arquiteturais estão classificados de 1 a 5, atendendo aos atributos escolhidos.

Atributos	**Layered**	**Pipe & Filter**	**Client--Server**	**P2P**	**Micros-serviços**	**SOA**
Operabilidade	★★	★	★★★★★	★★	★★★★★	★★★★★
Segurança	★★★★★	★★★★★	★★★★	★	★★★★	★★★
Resiliência	★★	★	★★★	★★★★★	★★★★★	★★★★★
Performance	★★★★★	★★★★★	★★★★	★★★	★★★★	★★★★

Tabela 1: Comparação dos Padrões Arquiteturais

Apesar do desempenho exemplar das arquiteturas monolíticas, estas têm problemas consideráveis na operabilidade, na resiliência e na escalabilidade. O que nos leva a optar por uma **arquitetura distribuída**.

Dentro das arquiteturas distribuídas, já devemos analisar as classificações a grão mais fino, por exemplo, uma arquitetura **Peer-to-peer** depende de implementações específicas em cada par, o que reduz a nossa operabilidade consideravelmente. Reduz ainda mais a segurança, sendo que os dados são replicados e/ou distribuídos ao longo dos vários pontos, o que resultaria em utilizadores com fragmentos ou até mesmo dados que não lhes pertencem.

No outro lado do espetro, também podemos considerar uma arquitetura **Cliente-Servidor**, que sofre na resiliência, uma vez que terá um único ponto de falha, o servidor.

Restam, assim, duas abordagens para considerar: **Microsserviços** e **Arquitetura Orientada a Serviços**. Ambas as arquiteturas têm classificações muito semelhantes, exceto ao nível da segurança, onde Microsserviços têm uma ligeira vantagem, devido ao facto de cada microsserviço ser independente dos outros, podendo ter a sua própria base de dados. Também é de notar que muitos dos *use cases* do *PictuRAS* são independentes uns dos outro, modelando conceitualmente uma arquitetura que permita que certos serviços e certas funcionalidades sejam mesmo independentes umas das outras.

Devido a estas razões, a equipa de desenvolvimento acordou com o padrão arquitetural de **Microsserviços** para a concretização do sistema *PictuRAS*.

4.3 Microsserviços

A arquitetura de Microsserviços oferece vantagens em termos de escalabilidade, resiliência e flexibilidade, permitindo o uso de diferentes tecnologias adaptadas às necessidades específicas de cada microsserviço.

Os requisitos não funcionais mais significantes para esta escolha, conforme detalhado na Secção 3, foram ordenados por classificação da seguinte forma:

Requisito	**Rate**	**Justificação**
RNF_32	★★★★★	A arquitetura de microsserviços facilita o desenvolvimento de testes unitários e de integração, permitindo testar serviços individualmente e melhorar a detecção de falhas.
RNF_26	★★★★★	Microsserviços otimizam o uso de CPU e memória para cada funcionalidade, mas podem ter overhead se a comunicação entre serviços não for bem gerida.
RNF_23	★★★★★	A escalabilidade horizontal é viabilizada, permitindo aumentar instâncias de serviços específicos sem replicar toda a aplicação, o que melhora custo e desempenho.
RNF_22	★★★★★	Microsserviços permitem processar múltiplas imagens simultaneamente e escalar de forma independente, mantendo o desempenho.
RNF_25	★★★★	A modularidade dos microsserviços facilita a integração com serviços externos, como armazenamento em nuvem e redes sociais, sem interferir em outros serviços.
RNF_21	★★★★	A edição de imagem pode ser isolada em microsserviços especializados, otimizando o desempenho, embora a latência das chamadas entre serviços possa reduzir ligeiramente a velocidade.

Tabela 2: Rating dos Requisitos Não Funcionais para Microsserviços

4.3.1 Identificação de Microsserviços

Na arquitetura de microsserviços, cada funcionalidade do sistema é implementada como um serviço independente. Isto permite uma escalabilidade modular, flexibilidade e facilita a manutenção e a evolução do sistema, visto que desta forma cada componente pode ser desenvolvido, atualizado e escalado individualmente, promovendo uma arquitetura robusta e extensível.

As principais funcionalidades identificadas e estruturadas como microsserviços são as seguintes:

Gestão de utilizadores
- **Função:** Gerir todas as operações relacionadas aos dados dos utilizadores, garantindo uma experiência segura.
- **Responsabilidades**: *login*, *logout*, registar utilizador, editar perfil, alterar plano e partilhar resultado de projetos.

Gestão de ferramentas
- **Função:** Gerir as ferramentas de edição disponíveis no sistema, oferecendo flexibilidade para adicionar e para remover.
- **Responsabilidades**: adicionar filtro e remover filtro.

Gestão de projetos
- **Função:** Organizar e gerir projetos de edição de imagem.
- **Responsabilidades**: criar projeto, aplicação de ferramentas e transferir projeto

Gestão de imagens
- **Função:** *Upload*, armazenamento e organização das imagens nos projetos.
- **Responsabilidades**: carregar imagem e remover imagem.

4.4 Modelos de dados das Bases de Dados dos Microsserviços

Nesta secção, vamos apresentar os dicionários de dados correspondentes a cada uma das Bases de Dados definidas na arquitetura.

4.4.1 Gestão de Projetos

Na Gestão de Projetos, iremos utilizar uma base de dados MongoDB, na qual teremos uma tabela para armazenar as informações relativas aos projetos.

Campo	Tipo	Descrição
id	Integer	Número identificativo único do projeto
name	String	Nome identificativo do projeto
images	Array (Imagem)	Array de imagens que compõem o projeto
historical	Array (String)	Array de strings (alterações) feitas no projeto
tools	Array (Ferramenta)	Array de ferramentas associadas ao projeto
date	Date	Data de criação do projeto

Tabela 3: Informação armazenada dos projetos

4.4.2 Gestão de Utilizadores

Na Gestão de Utilizadores, iremos utilizar uma base de dados MongoDB, na qual teremos uma tabela para armazenar as informações relativas aos utilizadores.

Campo	Tipo	Descrição
id	Integer	Número identificativo único do utilizador
name	String	Nome identificativo do utilizador
email	String	E-mail identificativo do utilizador
password	String	Conjunto de caracteres que compõem a senha do utilizador
plan	String	Identificação do plano do utilizador (“Premium”, “Registado”, “Anonimo”)
Date_payment	Date	Data do próximo pagamento para a renovação da subscrição (apenas para plan = “Premium”)
results	Array (Imagem)	Array de imagens resultado das suas operações dentro da aplicação prontas a serem publicadas

Tabela 4: Informação armazenada dos utilizadores

4.4.3 Gestão de Imagem

Na Gestão de Imagens, iremos utilizar uma base de dados MongoDB, na qual teremos uma tabela para armazenar as informações relativas às imagens.

Campo	Tipo	Descrição
id	Integer	Número identificativo único da imagem
name	String	Nome identificativo do imagem
email	String	E-mail identificativo do utilizador

Tabela 5: Informação armazenada das imagens

4.4.4 Gestão de Ferramentas

Na Gestão de Ferramentas, iremos utilizar uma base de dados MongoDB, na qual teremos uma tabela para armazenar as informações relativas às ferramentas.

Campo	Tipo	Descrição
id	Integer	Número identificativo único da ferramenta
name	String	Nome identificativo da ferramenta
options	Array(Strings)	Array de opções da ferramenta

Tabela 6: Informação armazenada das ferramentas

4.5 Tecnologias para a implementação

Nesta secção, iremos apresentar as tecnologias escolhidas pela equipa de desenvolvimento para a construção do sistema.

- **Microserviços e API Gateway:** Nas aplicações que compõem os microsserviços, serão utilizadas as linguagens de programação Go e Node.js. Para efetuar a comunicação com o servidor da base de dados, recorreremos a uma API RESTful, que permitirá realizar as operações necessárias e garantir a interação com as bases de dados. No caso de Go, utilizaremos a biblioteca GIN[1] e, no caso de Node.js, utilizaremos MongooseJS[2].

  Relativamente à API Gateway, esta será desenvolvida em Node.js e Express.js. No entanto, não utilizará Bases de Dados, sendo que só reencaminha pedidos para microsserviços.

- **Base de Dados:** As equipas responsáveis por cada microsserviço, após análise detalhada dos requisitos de armazenamento de dados, decidiram adotar o MongoDB como Sistema de Gestão de Bases de Dados (SGBD) para todos os microsserviços, optando por um modelo de dados não relacional orientado a documentos.

- **Frontend:** Para o desenvolvimento do *frontend*, utilizaremos Vue.js para proporcionar uma interface de utilizador dinâmica e interativa. No *design* e personalização da interface visual, serão utilizadas tecnologias como CSS3.

4.6 Decisões Organizacionais

A equipa de desenvolvimento, composta por 10 elementos, decidiu delegar a responsabilidade dos Microsserviços de forma a otimizar a eficiência e a qualidade do desenvolvimento. Os microsserviços a serem desenvolvidos incluem: Gestão de utilizadores, Gestão de projetos, Gestão de imagens e Gestão de ferramentas. Neste sentido, atribuímos uma sub-equipa a cada microsserviço, com o número de elementos baseado na sua complexidade:

[1] https://gin-gonic.com/docs/
[2] https://mongoosejs.com/docs/guide.html

Microsserviços	**Número de elementos**
Gestão de utilizadores	2
Gestão de projetos	4
Gestão de imagens	2
Gestão de ferramentas	2

Tabela 7: Distribuição do número de elementos pelos microsserviços

Além disto, cada equipa responsável por um microsserviço também ficará encarregue de desenvolver o *frontend* e o API Gateway associados ao microsserviço que está a gerir. Desta forma, a gestão completa do microsserviço, incluindo a camada de apresentação e a interface de comunicação, será feita pela mesma equipa, garantindo maior coesão e eficiência no desenvolvimento.

5 *Building Block View*

A Visão de Blocos de Construção oferece uma decomposição estática do sistema, onde é possível focar nas componentes principais e nas interações. Esta secção apresenta os principais blocos de construção que constituem esta arquitetura de microsserviços, detalhando a organização hierárquica dos componentes e das suas funções.

5.1 Blocos Lógicos das Funcionalidades do Sistema

Para realizar a decomposição do sistema em blocos que descrevem as suas principais componentes e as respetivas interações, começamos por mapear cada requisito funcional a um conjunto de blocos lógicos que descrevem a sua funcionalidade. Com estes blocos, pudemos definir um diagrama de atividade, que descreve o comportamento do respetivo requisito funcional.

Apresentamos, de seguida, alguns dos requisitos funcionais e o respetivo mapeamento em blocos lógicos:

5.1.1 Recortar Imagens

Requisito #: 9	Tipo: Funcional	*Use cases* #: 3
Descrição	**O utilizador carrega imagens para um projeto.**	
Rationale	Permitir o *upload* de uma ou mais imagens para posterior edição.	
Origem	Engenheiro de Software	
Fit criterion	Utilizador consegue adicionar uma ou mais imagens a um projeto.	
Prioridade	Must	
Data	2024/10/03	

Figura 12: RF9 - *Upload* de imagens para um projeto

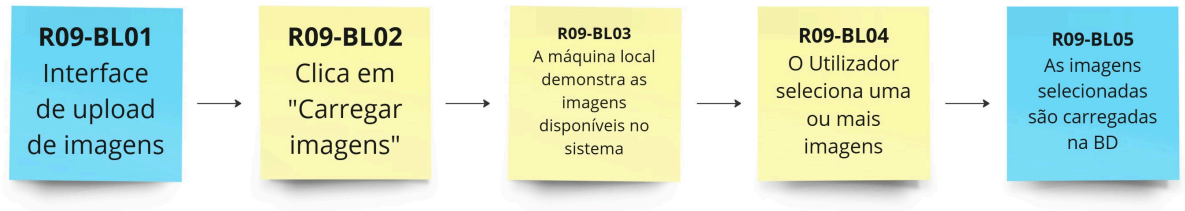


Figura 13: RF9 - Blocos Lógicos

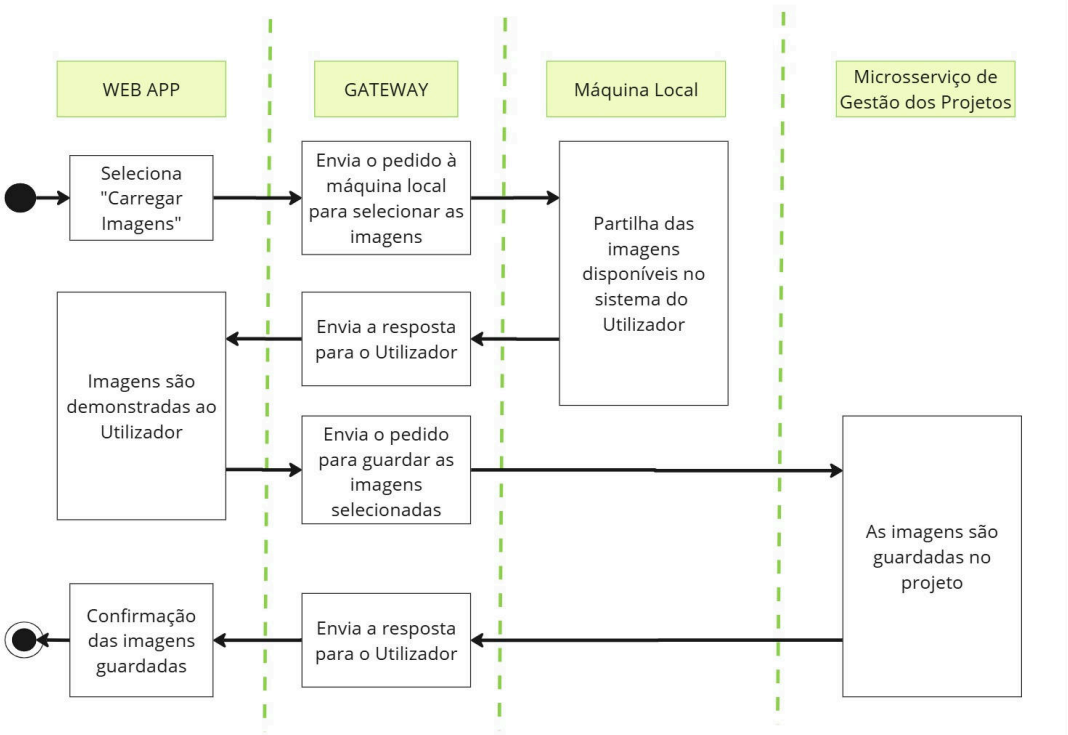


Figura 14: RF9 - Diagrama de Atividades

5.1.2 Alteração de Parâmetros

Requisito #: 16	Tipo: Funcional *Use cases* #: 4
Descrição	**O utilizador altera os parâmetros das ferramentas.**
Rationale	Para possibilitar personalizações detalhadas das operações de edição, atendendo às necessidades específicas de cada utilizador.
Origem	Engenheiro de Software
Fit criterion	O utilizador consegue ajustar os parâmetros das ferramentas de edição.
Prioridade	Must
Data	2024/10/03

Figura 15: RF16 - Alteração de Parâmetros

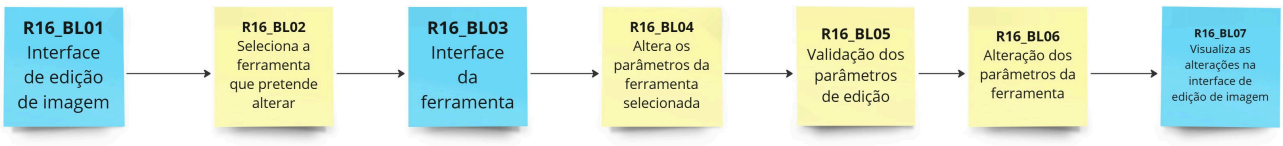


Figura 16: RF16 - Blocos Lógicos

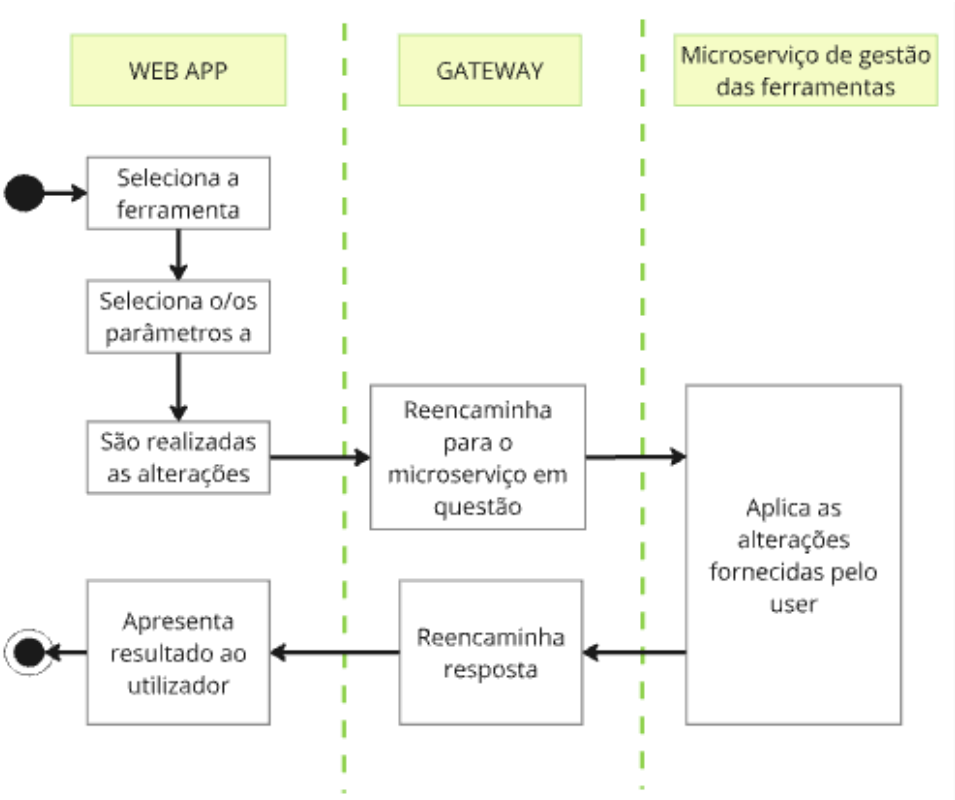


Figura 17: RF16 - Diagrama de Atividades

5.1.3 Alteração da Recorrência de Pagamento

Requisito #: 20	Tipo: Funcional *Use cases* #: 1
Descrição	**O utilizador com subscrição *Premium* altera a recorrência de pagamento entre mensal ou anual.**
Rationale	O utilizador pode modificar o plano de subscrição, permitindo atualizações conforme as suas precisa.
Origem	Modelo de Negócio
Fit criterion	O utilizador consegue alterar o tipo de subscrição através do perfil.
Prioridade	Could
Data	2024/10/03

Figura 18: RF20 - Alteração da Recorrência de Pagamento

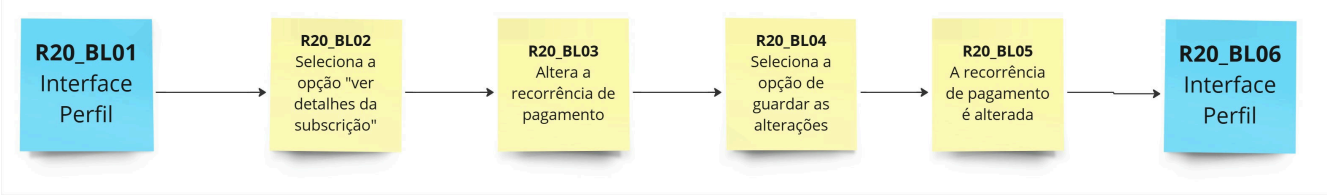


Figura 19: RF20 - Blocos Lógicos

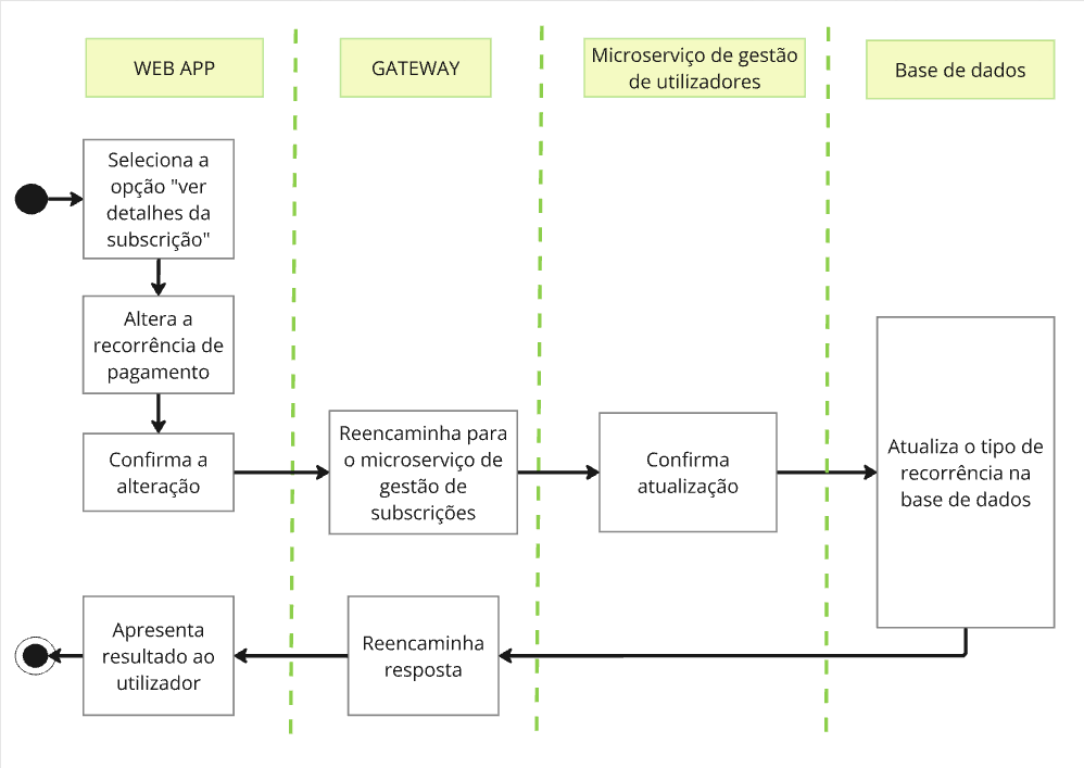


Figura 20: RF20 - Diagrama de Atividades

5.1.4 Recortar Imagens

Requisito #: 25	Tipo: Funcional		*Use cases* #: 4
Descrição	**O utilizador recorta manualmente imagens.**		
Rationale	Permitir ao utilizador remover partes indesejadas das imagens.		
Origem	Engenheiro de Software		
Fit criterion	O projeto passa a conter a ferramenta de corte de imagens na sua sequência de ferramentas.		
Prioridade	Must		
Data	2024/10/03		

Figura 21: RF25 - Ferramenta Básica: Recorte de Imagens

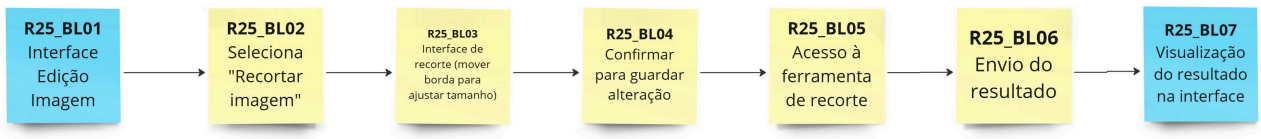


Figura 22: RF25 - Blocos Lógicos

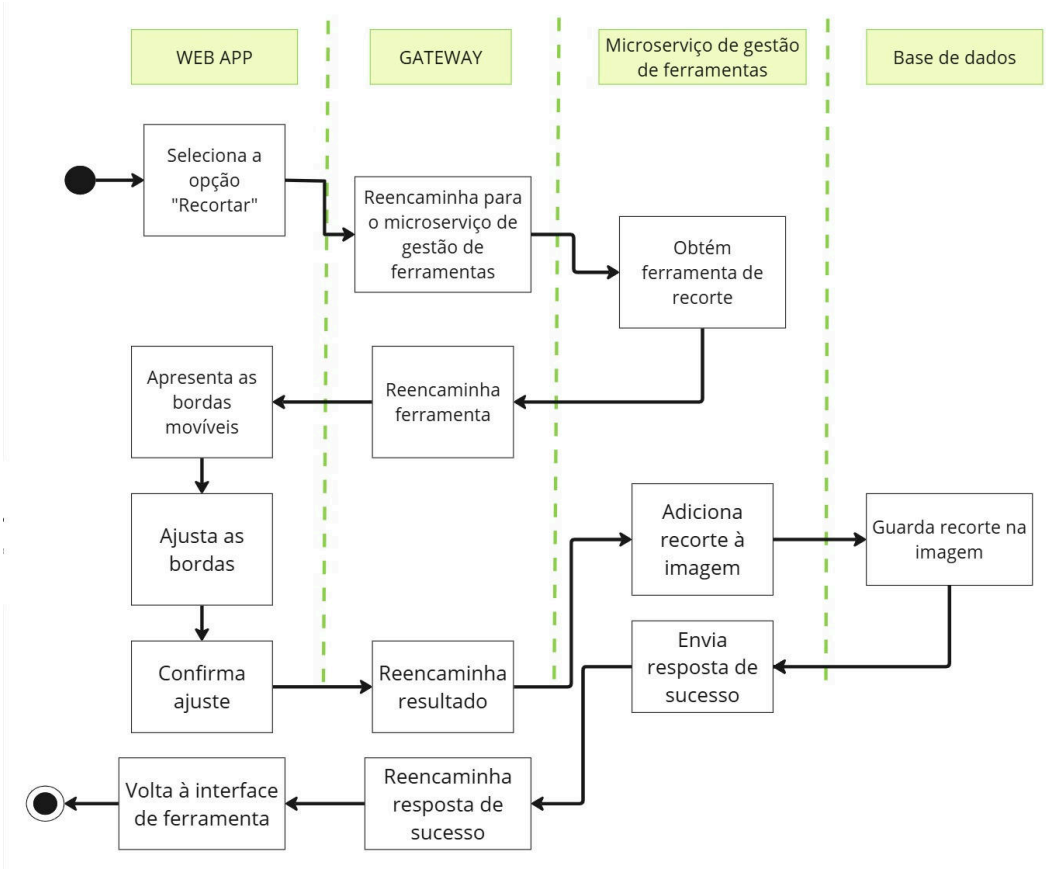


Figura 23: RF25 - Diagrama de Atividades

5.2 Diagrama de Componentes

O Diagrama de Componentes do *PictuRAS* representa a estrutura dos microsserviços como um conjunto de caixas brancas, cada uma correspondendo a um serviço independente.

Cada serviço é subdividido em blocos funcionais que representam módulos e classes específicas. Esta organização reflete as responsabilidades de cada serviço, promovendo a independência e o encapsulamento dos blocos de construção do sistema.

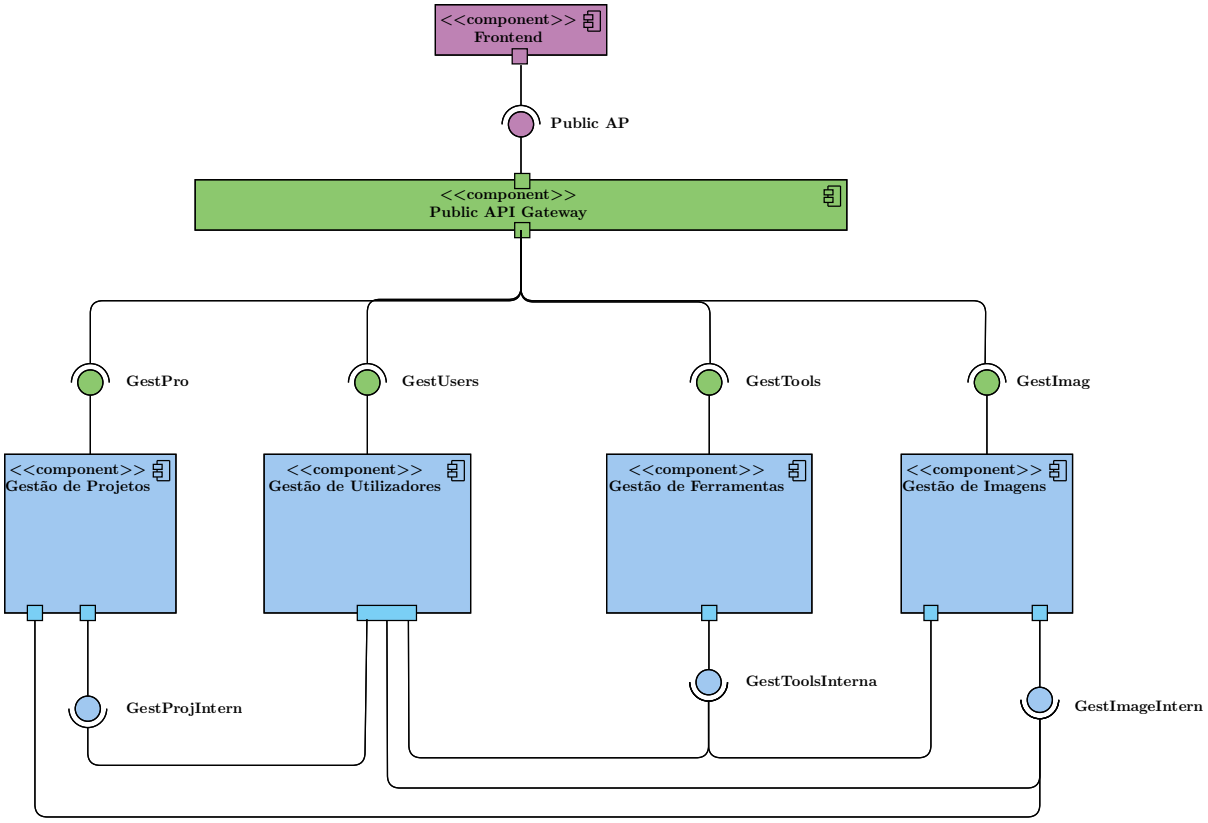


Figura 24: Diagrama de Componentes

Nos diagramas seguintes, serão apresentados os detalhes de cada componente do sistema, especificamente, os componentes dos microsserviços.

5.2.1 Gestão de Utilizadores

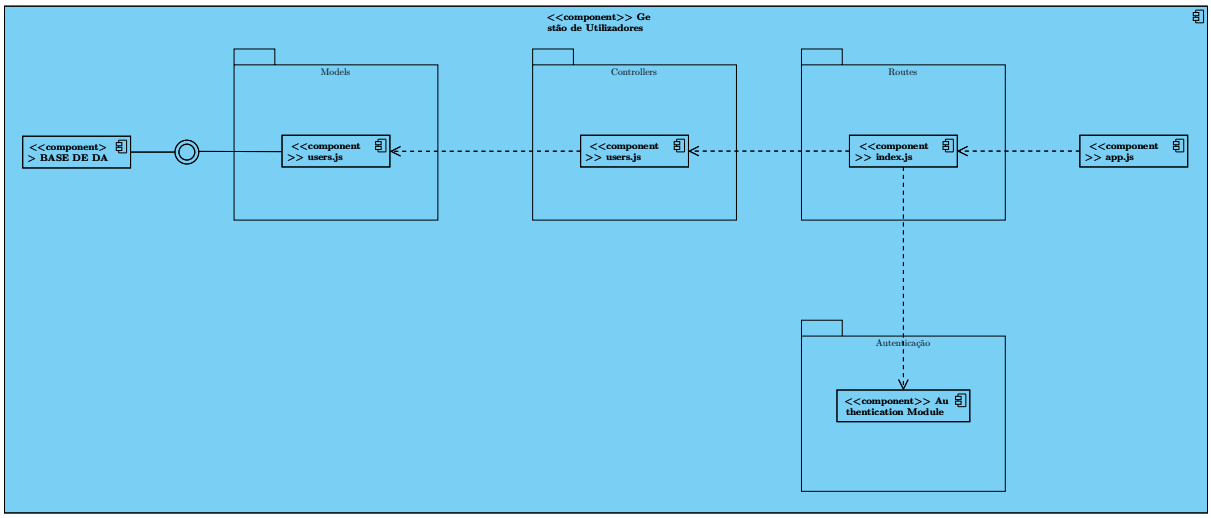


Figura 25: Diagrama de Componentes - Gestão de Utilizadores

Este diagrama foca a Gestão de utilizadores do sistema e possui seis componentes principais, distribuídos em diferentes pacotes:

- **Componente da Base de Dados:** Representa a conexão com a base de dados, onde estão armazenadas todas as informações dos utilizadores.
- **Package Models:** Contém o componente *users.js*, responsável por definir o modelo de dados dos utilizadores. Este componente mapeia a estrutura e os atributos de um utilizador, que serão utilizados ao interagir com a base de dados.
- **Package Controllers:** Inclui o componente *users.js*, que implementa a lógica de controlo da aplicação, gerindo a criação, atualização, exclusão e consulta de dados de utilizadores. É onde a lógica do sistema lida diretamente com o modelo de utilizadores.
- **Package Routes:** Contém o componente *index.js*, responsável por definir as rotas de acesso para operações de gestão de utilizadores. Este componente mapeia endpoints para funções específicas no controlador de utilizadores e fornece o ponto de entrada para as requisições do cliente.
- **Autenticação:** Inclui o módulo Authentication Module, que lida com os processos de autenticação e autorização dos utilizadores. Este módulo assegura que apenas utilizadores autenticados possam acessar certos recursos do sistema.
- **Componente *app.js*:** O ponto de entrada para a aplicação, que inicializa e configura as rotas e funcionalidades do sistema de gestão de utilizadores.

5.2.2 Gestão de Ferramentas

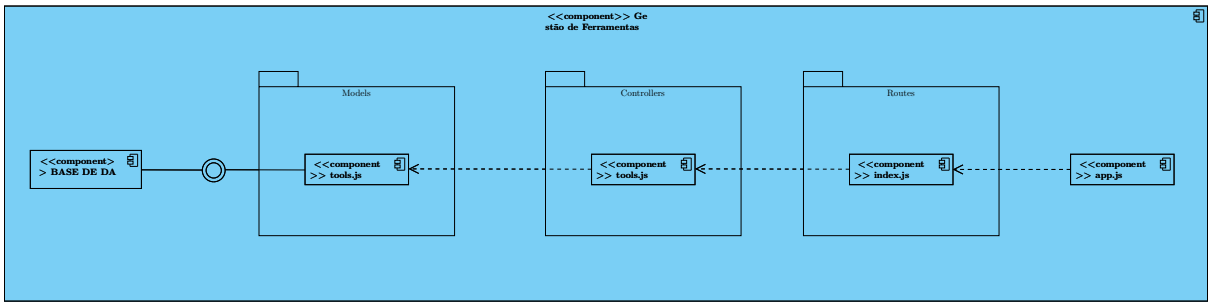


Figura 26: Diagrama de Componentes - Gestão de Ferramentas

Este diagrama foca a Gestão de ferramentas do sistema e possui cinco componentes principais, distribuídos em diferentes pacotes:

- **Componente da Base de Dados:** Este componente representa o acesso à base de dados onde as informações das ferramentas estão armazenadas.
- **Package Models:** Contém o componente *tools.js*, que define o modelo de dados das ferramentas. Este componente descreve a estrutura e os atributos das ferramentas, funcionando como uma interface entre o banco de dados e a aplicação.
- **Package Controllers:** Inclui o componente *tools.js*, que implementa a lógica da aplicação para operações com as ferramentas.

- **Package Routes:** Inclui o componente *index.js*, que define as rotas para as operações relacionadas com ferramentas. Este componente conecta as rotas com as funções do controlador de ferramentas, permitindo que o cliente faça requisições para interagir com as ferramentas.
- **Componente *app.js*:** O ponto de entrada para a aplicação, que inicializa e configura as rotas e funcionalidades do sistema de gestão de ferramentas.

5.2.3 Gestão de Projetos

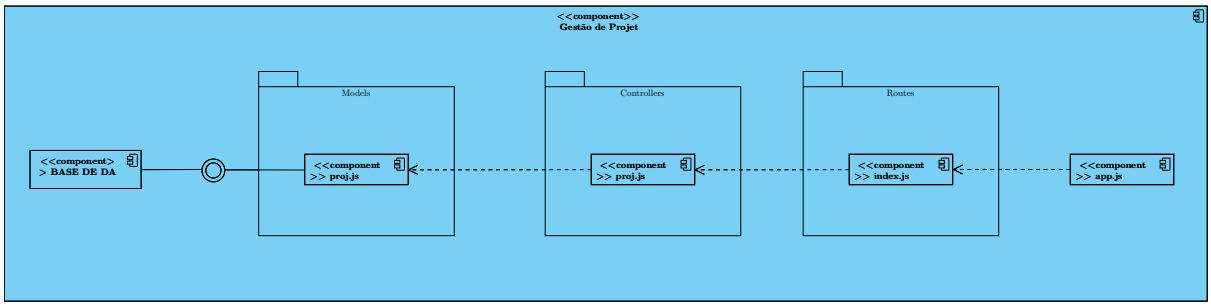


Figura 27: Diagrama de Componentes - Gestão de Projetos

Este diagrama foca a Gestão de projetos do sistema e possui cinco componentes principais, distribuídos em diferentes pacotes:

- **Componente da Base de Dados:** Representa o componente de conexão com a base de dados para armazenamento de dados relacionados com os projetos.
- **Package Models** Inclui o componente *projs.js*, que define o modelo de dados dos projetos, especificando atributos e estruturas que representam cada projeto no sistema.
- **Package Controllers:** Contém o componente *projs.js*, que implementa a lógica do sistema para a gestão de projetos.
- **Package Routes** Contém o componente *index.js*, responsável por definir as rotas de acesso para gestão de projetos, mapeando os endpoints para as operações suportadas pelo controlador de projetos.
- **Componente *app.js*:** O ponto de entrada para a aplicação, que inicializa e configura as rotas e funcionalidades do sistema de gestão de projetos.

5.2.4 Gestão de Imagens

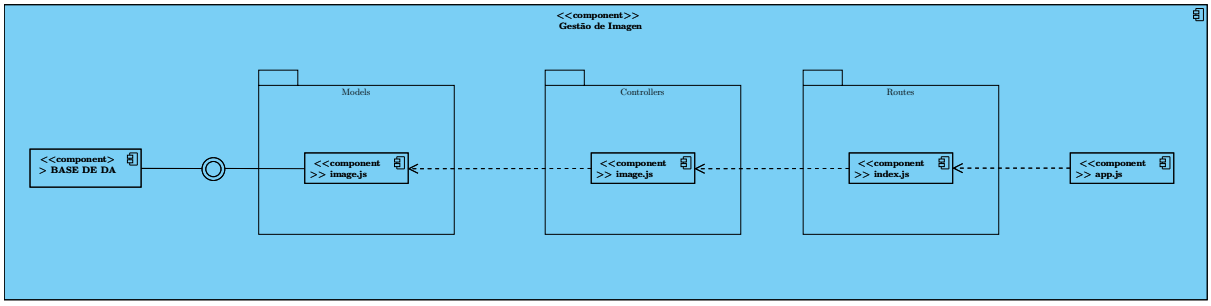


Figura 28: Diagrama de Componentes - Gestão de Imagens

Este diagrama é focado na gestão de imagens do sistema e possui cinco componentes principais, distribuídos em diferentes pacotes:

- **Componente da Base de Dados:** Este componente é responsável pela conexão com a base de dados, onde as informações sobre as imagens são armazenadas.
- **Package Models:** Inclui o componente *image.js* que define o modelo de dados das imagens, mapeando os atributos e propriedades das imagens, como nome, tipo, tamanho. Este componente é a base para interagir com a base de dados e armazenar as imagens.
- **Package Controllers:** Contém o componente *image.js*, este componente contém a lógica do sistema relacionada às operações de imagens.
- **Package Routes:** Contém o componente *index.js*, este componente define as rotas do sistema relacionadas às imagens, mapeando as URLs para as funções do controlador *image.js*, permitindo que os clientes façam requisições específicas para manipular imagens.
- **Componente app.js:** O ponto de entrada para a aplicação, que inicializa e configura as rotas e funcionalidades do sistema de gestão de imagens.

6 *Runtime View*

Nesta secção, apresentamos o funcionamento da solução proposta utilizando diagramas de sequência elaborados com base nos *use cases* do sistema. Estes diagramas ilustram as interações entre os atores e os diversos componentes do sistema, destacando também situações de exceção ou possíveis erros, além das estratégias adotadas pelo sistema para lidar com essas situações.

6.1 Diagrama de sequência

Um diagrama de sequência é um tipo de diagrama UML (Unified Modeling Language) usado para representar a interação entre os atores e os componentes de um sistema ao longo do tempo. Foca-se na troca de mensagens e chamadas de métodos entre objetos ou entidades para cobrir uma funcionalidade específica, geralmente associada a um *use case*.

6.1.1 Registo

Este diagrama de sequência descreve o processo de registo de um utilizador no sistema. O fluxo começa com o ator a inserir email e palavra-passe, que são enviados ao Frontend e depois validados pelo serviço de Gestão de Utilizadores.

Se o email ou a palavra-passe forem inválidos, serão exibidas mensagens de erro. Caso sejam válidos, é gerado um *token* e o utilizador escolhe entre o plano gratuito ou pago. Para o plano gratuito, o *token* é atualizado, e o acesso básico à aplicação é concedido. Já para o plano pago, os dados do cartão são verificados e, em caso de sucesso, o pagamento é processado, o plano *Premium* é ativado, e o utilizador recebe acesso completo à aplicação.

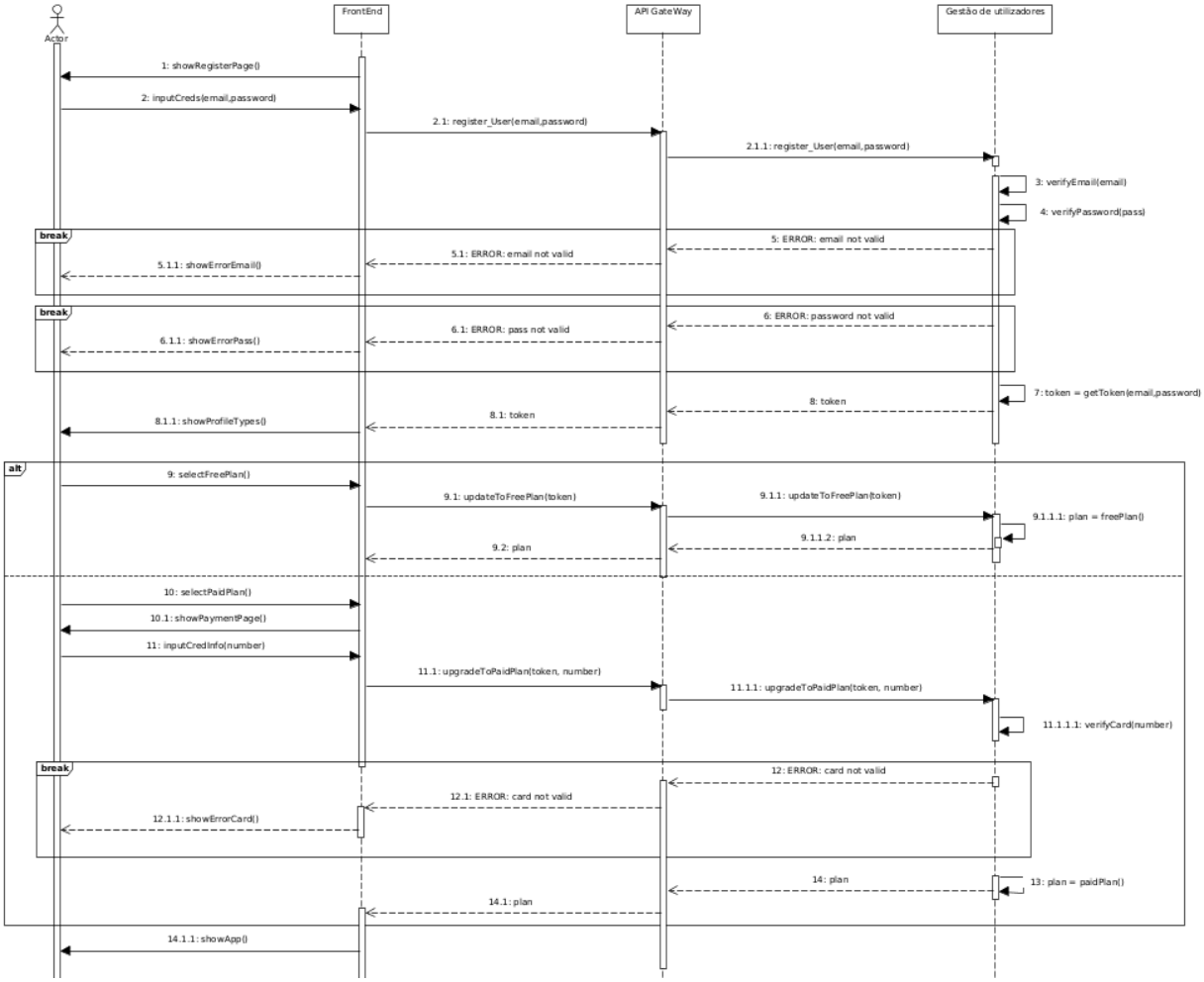


Figura 29: Diagrama de Sequência - Registo

6.1.2 Autenticação

Este diagrama de sequência mostra o processo do *use case* de autenticação de um utilizador no sistema. O fluxo inicia com o ator acedendo à página de *login* e inserindo as suas credenciais, que são enviadas ao Frontend. De seguida, o Frontend passa as credenciais ao API Gateway, que encaminha para o serviço de Gestão de Utilizadores para a respetiva validação.

Primeiro, o email é verificado. Se não for encontrado, uma mensagem de erro é retornada e exibida. Se o email for válido, a palavra-passe é verificada. Se estiver incorreto, ocorre outra mensagem de erro. Caso ambas as credenciais estejam corretas, um *token* de autenticação é gerado e devolvido ao utilizador, concedendo-lhe acesso à aplicação.

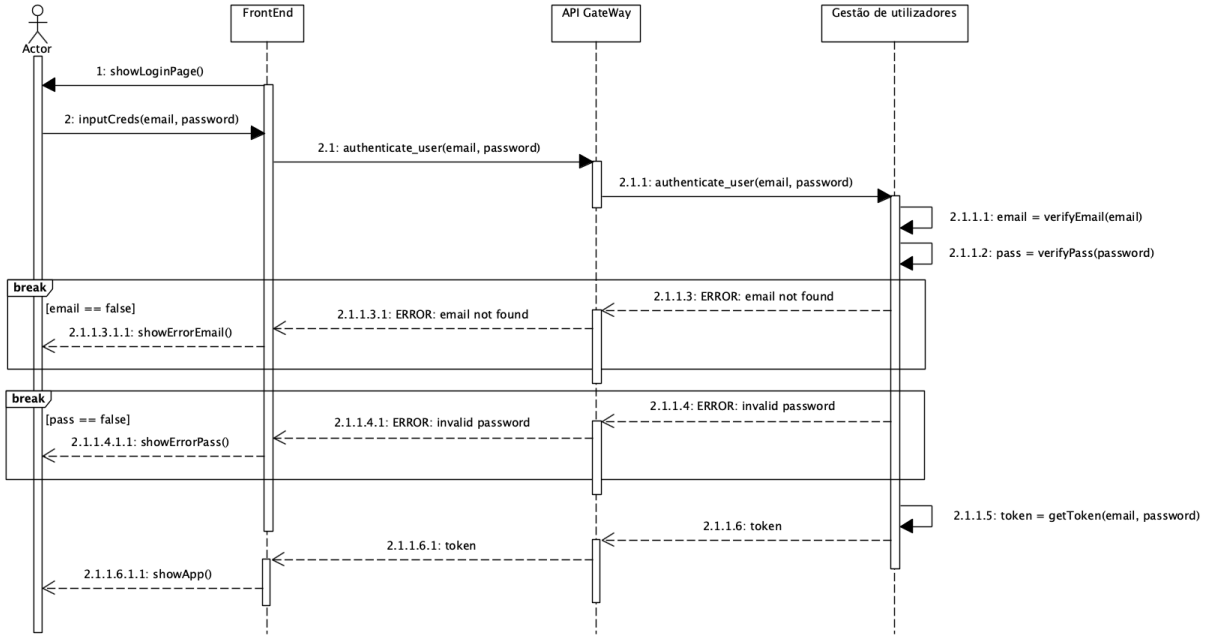


Figura 30: Diagrama de Sequência - Autenticação

6.1.3 *Upload* de imagem

Este diagrama de sequência descreve o *use case* de *upload* de uma imagem associada a um projeto, abordando tanto a validação dos dados quanto o tratamento de erros. O fluxo inicia com o ator a selecionar a opção para carregar uma imagem, o que aciona a interface de *upload* no Frontend. Após o envio da imagem, o Frontend comunica com o API Gateway, que intermedia a verificação dos dados nos serviços de Gestão de Imagens e Gestão de Projetos. Se o ID da imagem for inválido, o sistema informa o erro ao utilizador. Caso o projeto não exista, é criado automaticamente. Com todos os dados validados, a imagem é associada ao projeto e uma mensagem de confirmação é enviada ao responsável.

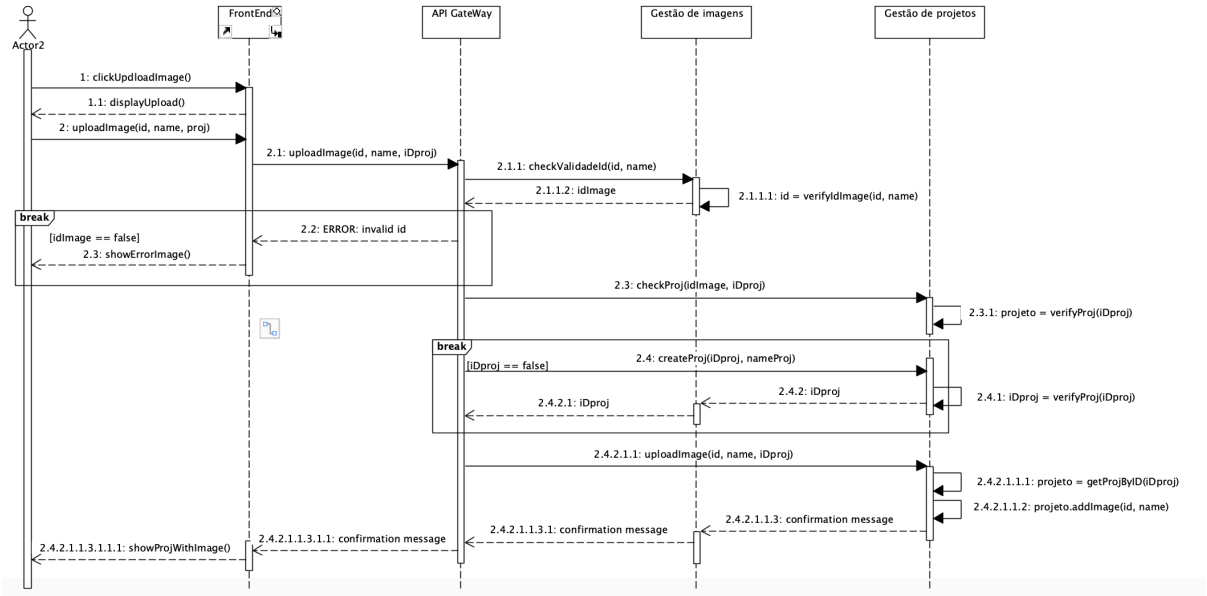


Figura 31: Diagrama de Sequência - Carregar imagens

6.1.4 Aplicação de Ferramentas num projeto

Este diagrama de sequência descreve o *use case* de *Aplicar encadeamento de ferramentas a conjunto de imagens*. O fluxo inicia com o ator a selecionar a opção de uma ferramenta qualquer disponível no menu de ferramentas, que estará sempre ativo durante a edição do projeto. Ao escolher a ferramenta, esta pode ou não ter opções, o que vai levar a outra escolha de opção. Caso contrário, a escolha da ferramenta fica feita. Depois de ser feita a busca da ferramenta à gestão de Ferramentas, provocada pela API, a ferramenta fica disponível para uso. Finalmente, quando o utilizador aplicar uma alteração às imagens com a ferramenta, vai ser chamada a interface *useTool*, que vai pedir à API para avisar o microsserviço de Gestão de Projetos para atualizar o projeto atual com as mudanças feitas às suas imagens, adicionando assim a ferramenta e as edições ao projeto. Com isto tudo feito, o projeto fica atualizado e as mudanças da ferramenta são aplicadas às imagens.

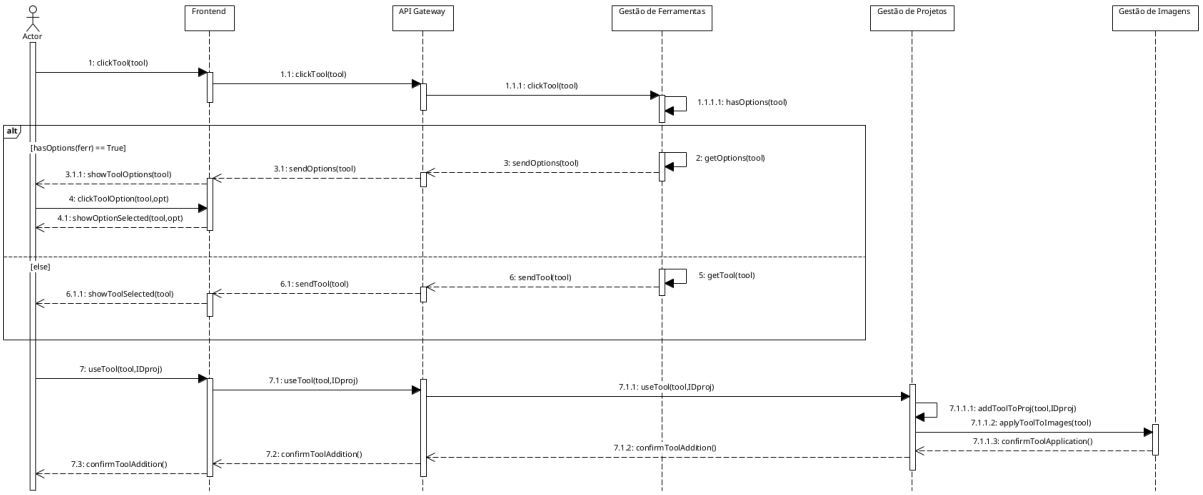


Figura 32: Diagrama de Sequência - Aplicar Ferramentas

7 *Deployment View*

A Visão de *Deployment* do projeto *PictuRAS* descreve a infraestrutura técnica necessária para a execução do sistema, incluindo os ambientes, os servidores, os processadores e a topologia. Esta secção apresenta a forma como os blocos de construção são mapeados para os elementos específicos da infraestrutura, detalhando como cada componente será instalado.

7.1 Infraestrutura Técnica

A arquitetura é baseada numa infraestrutura de nuvem para aproveitar a flexibilidade e a escalabilidade de ambientes de *containers*. Esta escolha permite o escalonamento automático dos microsserviços, otimizando a utilização de recursos e assegurando a sua disponibilidade. A infraestrutura técnica do sistema é organizada em três ambientes principais: Desenvolvimento, Teste e Produção.

- **Ambiente de Desenvolvimento**: Este ambiente é utilizado pela equipa de desenvolvimento para testar e validar novas funcionalidades do sistema e possui uma configuração própria para permitir atualizações e alterações rápidas, sem comprometer com o ambiente de produção.
- **Ambiente de Teste**: Este ambiente serve para realizar testes de integração, de desempenho e de segurança, simulando o ambiente de produção para garantir consistência e precisão nos testes.
- **Ambiente de Produção**: Este é o ambiente principal para os utilizadores acederem ao *PictuRAS*, é configurado para suportar uma grande carga de utilizadores e implementado numa infraestrutura de nuvem pública em múltiplas zonas para assegurar a resiliência do sistema.

7.2 Diagrama de *Deployment* de Produção

O Diagrama de Instalação do Ambiente de Produção apresenta a distribuição dos componentes de software na infraestrutura. Cada microsserviço é executado em *docker containers* e é gerido por *Amazon Elastic Compute Cloud* que aloca os recursos conforme a demanda.

O processo de *deployment* envolve a criação de imagens Docker para cada microsserviço, que são armazenadas em *containers*. Nesta esta aquitetura, todas as requisições dos utilizadores são encaminhadas para o API Gateway, local em que reencaminha o pedido para o serviço correspondente.

Esta arquitetura foi projetada para maximizar a escalabilidade e a disponibilidade (aumentando a elasticidade) através da *Amazon EC2 VM* que se ajustará de acordo com o nível de demanda. A segmentação em *containers* permite isolar cada microserviço, que por sua vez capacita a modificação de um serviço sem influenciar outros, aumentando a segurança e a flexibilidade.

No diagrama representado abaixo, está especificado as características mínimas necessárias para o sistema funcionar corretamente.

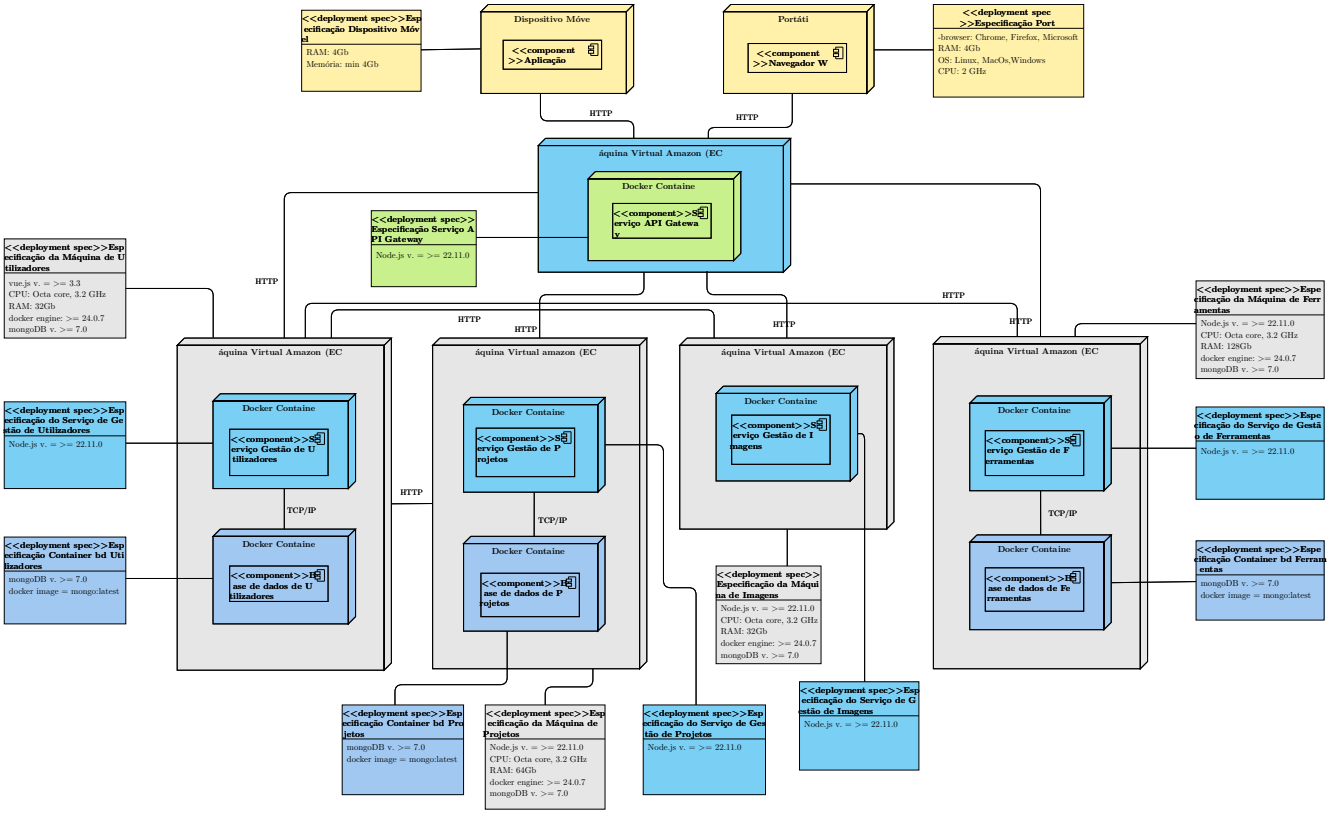


Figura 33: Diagrama de *Deployment* - Ambiente de Produção

8 Conclusões e Considerações

Embora a proposta arquitetural apresentada seja sólida e robusta, é importante destacar que algumas decisões podem ser ajustadas ou revistas conforme o desenvolvimento avance e novas necessidades surjam.

Entre as possíveis alterações, destacamos:

- **Escolha de bibliotecas e frameworks**: pode ser necessário substituir ou adicionar bibliotecas para a gestão de APIs, dependendo das necessidades de implementação.
- **Configuração dos microsserviços**: a quantidade e a organização dos microsserviços podem ser modificadas, seja pela realocação de funcionalidades, seja pela adição de novas.
  - Por exemplo, o **microsserviço de gestão de utilizadores** poderia ser integrado diretamente à lógica de negócios do API Gateway. Neste caso, o Gateway também passaria a ter acesso a uma base de dados dedicada.

Estas adaptações destacam a importância de uma arquitetura flexível, capaz de evoluir sem comprometer a eficiência nem o alinhamento com os requisitos e restrições do projeto. Além disso, reforçam que, embora seja uma solução sólida, a proposta também se adapta com naturalidade às mudanças que fazem parte do processo de desenvolvimento.