Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Unidade Curricular de Requisitos e Arquitetura de Software

Ano Letivo de 2024/2025
Grupo F

Duarte Araújo
A100750

Francisco Ferreira
PG55942

Gonçalo Costa
PG55944

João Gomes
PG55960

Luís Barros
PG55978

Marta Pereira
PG55981

Paulo Pinto
PG55991

Pedro Sousa
PG55994

Pedro Leiras
PG55995

Pedro Carvalho
PG55997

RAS

22 de Novembro de 2024

Índice

1. Introdução e Objetivos

Este documento explicita todos os planos, decisões e passos tomados pelo **grupo F** para conceber uma solução arquitetural para a aplicação ***PictuRAS***, uma aplicação web de edição de imagens.

Inicialmente, neste relatório, iremos apresentar toda a análise do documento de requisitos providenciado pelos docentes para a aplicação, tendo sempre em conta as restrições e limitações impostas pela equipa que concebeu os requisitos. A partir da análise dos requisitos, iremos realizar uma filtragem de acordo com a prioridade dos requisitos não funcionais, para que a escolha da arquitetura final seja robusta e o mais adequada possível consoante os mesmos. Justificaremos a escolha da arquitetura escolhida, comparando-a com a análise e avaliação de outras candidatas.

Posteriormente, delimitamos o contexto da aplicação, realizando uma decomposição funcional do mesmo para compreender as fronteiras do funcionamento do sistema.

Concebida a arquitetura da solução a ser usada, iremos descrever detalhadamente o plano da implementação da mesma, referindo como os componentes interagem entre si e como cada funcionalidade do sistema é sustentada para cada **use case**.

O objetivo desta segunda fase do processo de desenvolvimento da aplicação *PictuRAS* é que tenhamos essencialmente a arquitetura definida e que os vários cenários se enquadrem na mesma, para que a implementação seja posteriormente (fase 3) feita de forma suave, clara e direta.

1.1. Resumo dos requisitos funcionais

Na fase inicial foi desenvolvido um documento de requisitos para servir de base para todas as próximas fases. Nesse documento estão descritas as funcionalidades do sistema contando com os devidos atores.

Nesta secção descrevemos os requisitos de maior prioridade para que a escolha da nossa arquitetura futura tenha em conta os mínimos requeridos.

Os requisitos funcionais derivam dos **Use Cases** da aplicação, estes são descritos pelo seguinte diagrama:

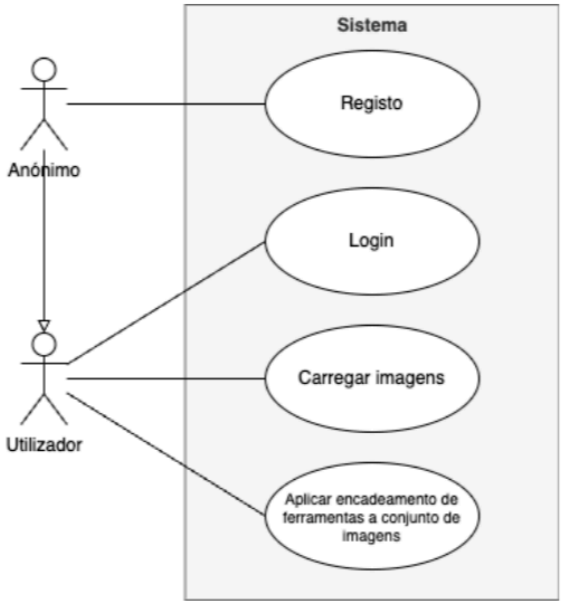

Figura 1: Diagrama de Use Cases

Para que a solução final tenha todas as funcionalidades pretendidas apresentamos aqui os requisitos funcionais com maior prioridade para que tenhamos uma versão da aplicação com pelo menos o mínimo de funcionalidades pedidas.

Requisito	Use cases	Descrição	Prioridade
RF1	2 - Login do utilizador	O utilizador autentica-se utilizando as suas credenciais.	Must
RF2	1 - Registo do utilizador	O utilizador anónimo regista-se.	Must
RF3	1 - Registo do utilizador	O utilizador escolhe o plano de subscrição Gratuito	Must
RF4	1 - Registo do utilizador	O utilizador escolhe o plano de subscrição Premium.	Must
RF6	3 - Carregar imagens	O utilizador cria um projeto.	Must
RF7	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador lista os seus projetos.	Must
RF8	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador acede à área de edição de um projeto.	Must
RF9	3 - Carregar imagens	O utilizador carrega imagens para um projeto.	Must

Requisito	Use cases	Descrição	Prioridade
RF10	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador remove uma imagem do projeto.	Must
RF11	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador adiciona uma ferramenta de edição ao projeto.	Must
RF12	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador desencadeia o processamento de um projeto.	Must
RF13	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador transfere o resultado de um projeto para o dispositivo local.	Must
RF15	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador altera a ordem das ferramentas de um projeto.	Must
RF16	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador altera os parâmetros das ferramentas.	Must
RF22	2 - Login de utilizador	O utilizador registado termina a sua sessão.	Must
RF23	1 - Registo do utilizador	O utilizador registado remove a sua conta e dados.	Must
RF25	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador recorta manualmente imagens.	Must
RF26	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	utilizador escala imagens para dimensões específicas.	Must
RF27	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador adiciona borda a imagens.	Must
RF29	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador ajusta o brilho a imagens.	Must
RF30	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador ajusta o contraste a imagens.	Must
RF32	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador roda imagens.	Must
RF33	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador aplica um algoritmo de recorte automático de imagens com base no seu conteúdo.	Must
RF35	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador remove o fundo da imagem, mantendo apenas o objeto principal.	Must
RF37	4 - Aplicar encadeamento de ferramentas a conjunto de imagens	O utilizador aplica um algoritmo de reconhecimento de objetos em imagens.	Must

1.2. Objetivos de qualidade

Nesta secção o grupo selecionou de acordo com os seus critérios, os 5 requisitos não funcionais de qualidade que devem à partida ter maior prioridade, e que serão, portanto essenciais para definir a nossa arquitetura de solução.

Para cada tipo de requisito atribuímos um peso consoante a sua característica arquitetural que contribuirá para a escolha da arquitetura a ser utilizada. Esta avaliação permitirá sermos o mais objetivo possível na nossa decisão final de arquitetura:

Característica arquitetural	Peso atribuído
Manutenção e Suporte	10%
Escalabilidade	15%
Desempenho	20%
Extensibilidade	15%
Segurança	10%
Operacional	15%
Usabilidade	15%

Atribuímos ao desempenho a maior prioridade, já que queremos que o cliente tenha uma experiência fluida, rápida e satisfatória ao utilizar a nossa solução, minimizando atrasos e maximizando a eficiência do sistema. Como o objetivo principal da nossa aplicação é editar imagens, achamos que se justifica o maior peso deste tipo de requisitos para a avaliação quantitativa final.

Os restantes atributos têm praticamente o mesmo grau de importância, à exceção da segurança, manutenção e suporte visto que numa primeira versão da aplicação não são questões fulcrais a ter em consideração.

Os 5 requisitos não funcionais que iremos ter em conta para a escolha da nossa arquitetura segundo a sua prioridade são:

Requisito não funcional	Descrição	Prioridade
RNF12	A aplicação deve ser capaz de escalar horizontalmente para suportar o aumento de utilizadores e volume de imagens, mantendo o desempenho	Must
RNF10	Operações avançadas devem ser concluídas em até 5 segundos por imagem. Nos processos encadeados, pode haver uma pequena redução de desempenho.	Must
RNF22	A aplicação deve realizar backups automáticos dos dados e imagens dos utilizadores, garantindo a sua recuperação em caso de falhas.	Must
RNF5	A aplicação deve apresentar o resultado da ferramenta em tempo real	Must
RNF11	O sistema deve processar até 100 imagens ao mesmo tempo, sem quedas perceptíveis no desempenho.	Must

2. Restrições

Para além de todas as funcionalidades e qualidades da solução que temos de ter em conta, devemos analisar cada restrição imposta para que a equipa de trabalho desenvolva o projeto dentro dos parâmetros propostos.

Nesta secção vamos apresentar e analisar cada restrição do sistema.

2.1. Restrições temporais

Descrição: O projeto deve ser entregue nas datas mencionadas, incluindo as principais funcionalidades implementadas, a arquitetura da solução e um relatório técnico detalhado que descreva o processo de desenvolvimento.

Cada entrega será avaliada para garantir que o projeto está no caminho certo e cumpre os requisitos iniciais estabelecidos pelos clientes. As seguintes restrições temporais aplicam-se:

- Numa segunda fase, o relatório técnico será entregue até o dia 15 de novembro de 2024. Este será apresentado aos *stakeholders* entre os dias 18 e 22 de novembro de 2024 para avaliação e *feedback*.

As seguintes fases serão submetidas nas seguintes datas:

- **Entrega da arquitetura da aplicação:** 15 de novembro de 2024, incluindo a definição completa da arquitetura do sistema.
- **Defesa da arquitetura:** 18 a 22 de novembro de 2024, durante a qual a equipa deverá justificar as escolhas técnicas e receber feedback detalhado.
- **Entrega da implementação da aplicação:** 17 de janeiro de 2025, incluindo a versão final do software com todas as funcionalidades principais implementadas.
- **Defesa da implementação:** 27 a 31 de janeiro de 2025, focando na avaliação da qualidade da implementação e na identificação de melhorias finais.

Implicações: O cumprimento deste prazo é fundamental para garantir tempo suficiente para o feedback dos stakeholders e para os ajustes necessários antes das fases subsequentes. Atrasos na entrega comprometem as atividades futuras do projeto.

2.2. Restrições Técnicas/Solução

Descrição: A aplicação deverá ser projetada para funcionar na Cloud, garantindo acessibilidade global a qualquer utilizador com uma conexão à internet. As seguintes restrições técnicas devem ser consideradas para que a escolha da arquitetura nesta fase seja a mais adequada:

- **Plataforma Web:** A aplicação deve ser exclusivamente desenvolvida como uma aplicação web.
- **Hospedagem:** A aplicação deverá estar preparada para ser disponibilizada numa infraestrutura Cloud (AWS, Azure, Google Cloud), considerando aspetos como escalabilidade, segurança e otimização de custos.
- **Segurança e Privacidade:** É obrigatório garantir que a plataforma implemente medidas robustas de segurança, incluindo a proteção contra ataques comuns (e.g., SQL injection, XSS) e políticas adequadas de gestão de dados dos utilizadores, conforme regulamentos de privacidade como o RGPD.
- **Escalabilidade:** 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.
- **Extensibilidade**: A arquitetura de aplicação deve ser desenvolvida de modo que tenha em consideração a capacidade de implementar de modo relativamente fácil novas ferramenta e/ou novas funcionalidades.

Implicações: Estas restrições técnicas são fundamentais para assegurar que o desenvolvimento da aplicação ocorra eficientemente. Falhas no cumprimento destas restrições podem comprometer a usabilidade, segurança e acessibilidade da aplicação, resultando num produto que não corresponde às expectativas dos stakeholders.

2.3. Restrições Orçamentais

Descrição: O desenvolvimento da aplicação deve respeitar restrições financeiras estabelecidas pelo cliente. A equipa de desenvolvimento deve escolher a arquitetura tendo em conta os custos de infraestrutura, desenvolvimento e manutenção. As principais restrições orçamentais incluem:

- **Salários dos elementos da equipa de desenvolvimento:** Os salários devem ser claramente definidos e controlados no orçamento.
- **Custos de hospedagem na nuvem:** A aplicação será disponibilizada na nuvem, com custos a serem controlados relativos a:
  - **Armazenamento:** Os custos de armazenamento devem ser monitorizados e estimados com precisão.
  - **Escalabilidade:** Deve-se considerar os custos adicionais para escalar a infraestrutura em resposta a picos de uso.
  - **Manutenção e suporte:** A manutenção contínua da aplicação deve ser planeada financeiramente, garantindo a sustentabilidade do projeto a longo prazo.

Implicações: O não cumprimento dessas restrições financeiras pode comprometer a viabilidade do projeto, limitando a escalabilidade e o suporte aos utilizadores.

2.4. Restrições de Âmbito

A arquitetura deve seguir requisitos específicos definidos pelos *stakeholders*.

As seguintes restrições de âmbito são aplicáveis:

- **Perfis de utilizador:** A aplicação terá os seguintes perfis:
  - **Anónimo:** Permite operações em fotografias com dimensões limitadas.
  - **Registado (gratuito):** Permite até cinco operações por dia em fotografias.
  - **Premium:** Sem limites de operações ou dimensões.
- **Ferramentas básicas:** A aplicação deve incluir um conjunto básico de ferramentas de processamento de imagens. As ferramentas devem aceitar parâmetros ajustáveis pelo utilizador.
- **Ferramentas avançadas:** Para além das ferramentas básicas, a aplicação deve oferecer ferramentas avançadas, geralmente apoiadas em IA, tais como:
- **Contagem de pessoas.**
- **Extração de texto (OCR).**
- **Identificação de objetos.**
- **Alterações específicas** (e.g.: colocar óculos em pessoas).
- **Encadeamento de ferramentas e processamento em lote:** A aplicação deve permitir encadear ferramentas, onde o output de uma ferramenta serve como input para a próxima. Deve ser possível aplicar sequências de ferramentas automaticamente a um conjunto de imagens, utilizando arquivos `.zip` ou diretorias. A funcionalidade deve ser restrita a ferramentas compatíveis, que aceitem o tipo de output produzidos pela ferramenta anterior.

Implicações: O não cumprimento destas restrições de âmbito, pode resultar em funcionalidades inadequadas ou incompletas, comprometendo a experiência do utilizador e dos objetivos do projeto.

3. Contexto

Este sistema é uma aplicação de edição de imagens web que permite aos utilizadores realizarem modificações básicas e avançadas nas suas imagens de forma intuitiva, diretamente no navegador, sem necessidade de instalar *software*.

O escopo de um sistema define o que o sistema faz e o que ele não faz. Em outras palavras, é um limite que separa as funcionalidades que vamos implementar no nosso sistema (dentro do escopo) das que não vamos implementar (fora do escopo)

Incluídos no nosso escopo do sistema estão por exemplo as funcionalidades mais óbvias e retratas nos **Use Cases**, mais diretamente nos requisitos funcionais:

- **Ferramentas de edição:** Oferecer funcionalidades básicas e avançadas de edição de imagens.
- **Visualização em tempo real:** Apresentar as edições feitas na imagem instantaneamente, para que o utilizador veja o resultado em tempo real.
- **Download da imagem editada:** Permitir que o utilizador salve a imagem editada no seu dispositivo.

Não tão salientes são as funcionalidades do sistema que não são diretamente visualizáveis pelo utilizador, e que temos de ter em conta na nossa arquitetura de solução, devem estar no nosso escopo, como por exemplo:

- **Armazenamento de projetos:** Permitir que os utilizadores façam upload de imagens de seus dispositivos para as editar na aplicação e que no final da sua edição o projeto seja armazenado para que esteja no histórico de projetos do utilizador e seja acessível novamente.
- **Processamento de ferramentas na imagem:** O sistema deve permitir que os utilizadores consigam aplicar ferramentas, mas esta mesma aplicação deverá ser feita pelo sistema em causa e não localmente.
- **Haver protocolos de segurança mínimos:** O sistema deve, dada a arquitetura e as várias tecnologias a implementar, estabelecer no processamento dos vários cenários de uso princípios de segurança e de manutenção.

3.1. Contexto de negócio

Editar fotos não é uma atividade muito recorrente na vida dos utilizadores mais casuais. Assim, a aplicação web *PictuRAS*, convida estes, que querem uma experiência pontual, simples e rápida, assim como os utilizadores mais sérios e que precisam de ferramentas mais complexas a dar asas à sua imaginação ao editar as suas imagens.

Para lidar com todos estes tipos de utilizadores, a aplicação contará com perfis **anónimos**, que não poderão beneficiar de todas as funcionalidades existentes, mas que servirá como primeiro passo ao uso da *PictuRAS* e de uma eventual adesão a serviços mais complexos, sendo estes os perfis **registado** e **premium**, sendo então providenciado um serviço para cada tipo de utilizador.

Os perfis **registados** são uma oferta mais viável para aqueles que querem manter um histórico das suas edições, assim como trabalhar em projetos maiores, fazendo uso apenas de ferramentas básicas.

O perfil **premium** é projetado para utilizadores mais investidos nas suas edições e que será a principal fonte de rendimento da aplicação. Este perfil permite o uso de toda e qualquer ferramenta, desde o simples recorte até à identificação de objetos nas imagens, sem um limite de utilização ou upload.

A *PictuRAS* foi projetada com foco em escalabilidade e desempenho, o que garante uma experiência de edição fluida mesmo com elevada demanda de utilização.

Como aplicação web na nuvem, oferece flexibilidade e acessibilidade, assim, as pessoas têm acesso às ferramentas de qualquer dispositivo com conexão à internet, sem necessidade de instalações locais. Além disso, a possibilidade de partilhar edições diretamente nas redes sociais ou de gerar links para projetos, facilita a colaboração e a divulgação dos trabalhos realizados, ampliando o alcance da aplicação para um público mais vasto.

O sistema está limitado a funcionalidades de edição de imagem, a todas as que se relacionam com a gestão das mesmas e funcionalidades no que toca a gestão de contas, para além disso a tudo o que possa contribuir para o bom e seguro funcionamento do sistema.

Para melhor entender o funcionamento do sistema e a relação entre entidades apresentamos o diagrama de contexto na perspetiva de negócio, referenciando apenas a entidade externa **Serviço de pagamentos** para perceber como são feitos os pagamentos das subscrições premium (futuramente iremos mencionar todas as entidades externas que atuam no funcionamento da aplicação *PictuRAS*):

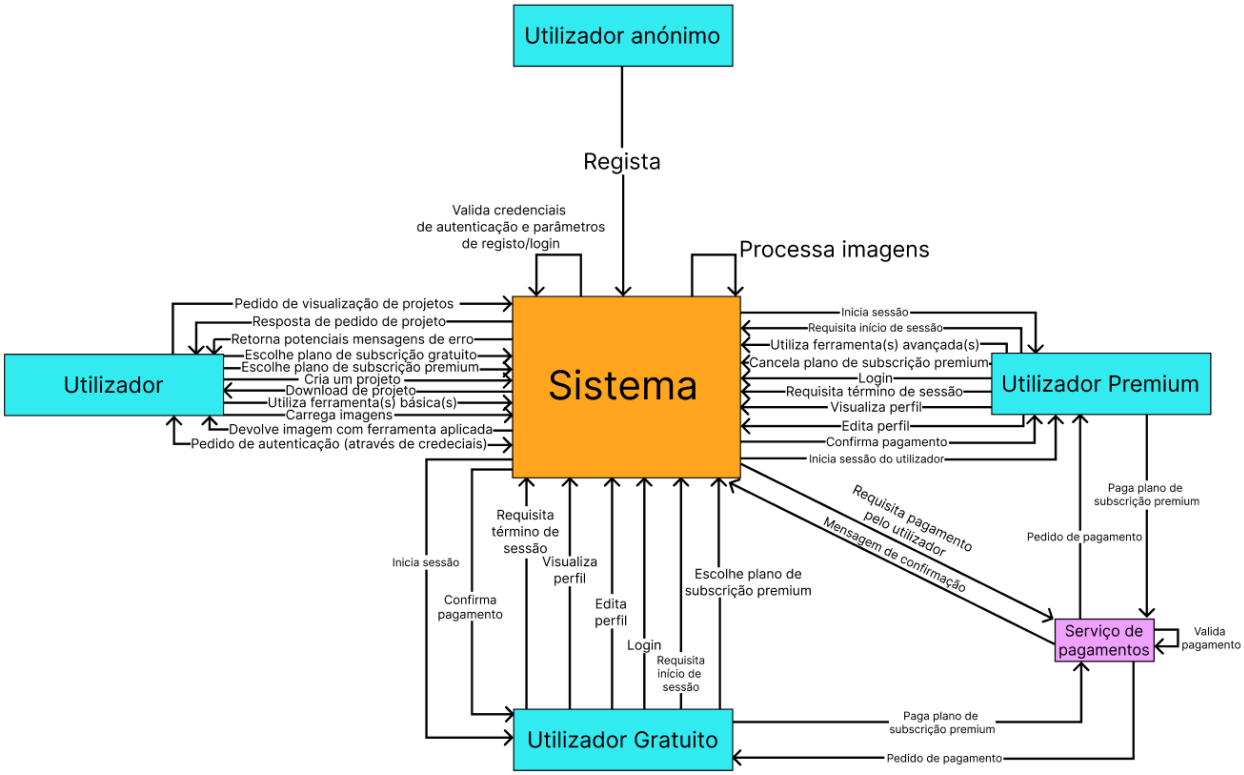

Figura 2: Diagrama de contexto (perspetiva de negócio)

Analisando o gráfico de contexto podemos esperar que o sistema tenha de suportar várias funcionalidades para diferentes tipos de utilizadores. O essencial é garantir sobretudo que as ações de cada utilizador sejam úteis e funcionais em qualquer momento no seu ciclo de vida e que este sistema seja prático, escalável e extensível para futuras novas implementações.

3.2. Contexto técnico

Para assegurar o cumprimento dos requisitos funcionais da aplicação, será necessária a integração com diversos serviços externos. Por exemplo, um serviço de e-mail será utilizado para automatizar tarefas como o envio de confirmações de registo, recuperação de palavras-passe e emissão de faturas de pagamento. Além disso, outros serviços serão integrados para atender funcionalidades específicas, como armazenamento de imagens e processamento de pagamentos.

O modelo completo está descrito no seguinte diagrama:

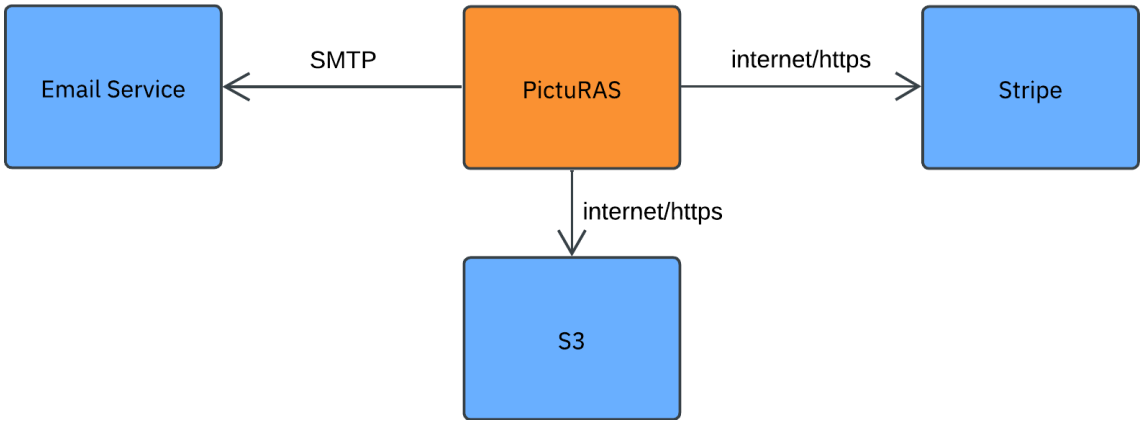

Figura 3: Diagrama de Contexto Técnico

Os principais nós envolvidos e as suas respetivas descrições estão detalhados na tabela a seguir:

Nó	Descrição
PictuRAS	Responsável pelo projeto principal.
Email Service	Serviço dedicado ao envio de e-mails para utilizadores, abrangendo confirmação de registo, redefinição de senha, envio de faturas de pagamento, entre outros.
S3	Serviço de armazenamento e distribuição de imagens, onde a aplicação armazena e disponibiliza os recursos visuais de maneira segura e eficiente.
Stripe	Integração para processar pagamentos da versão premium da aplicação, garantindo suporte a múltiplos métodos de pagamento com segurança e conformidade.

4. Estratégia da Solução

O objetivo desta secção é de nos informar mais profundamente no âmbito das arquiteturas disponíveis dado o contexto em que nos encontramos, dos nossos requisitos e dos aspetos em que cada arquitetura se demonstra mais apta, para que a escolha da arquitetura do sistema seja informada e a mais versátil face aos objetivos que nos foram impostos.

Iremos analisar os requisitos não funcionais anteriormente mencionados neste relatório e a partir dessa mesma análise selecionaremos o padrão arquitetural mais adequado. Após abordado esse tema,vamos decompor funcionalmente o sistema, para saber como enquadrar as funcionalidades na arquitetura escolhida.

4.1. Padrão arquitetural

Para tomarmos uma decisão robusta para a arquitetura do sistema vamos fazer uma avaliação segundo as característica arquiteturais, isto para perceber qual a arquitetura mais capaz de satisfazer as necessidades do sistema, segundo os requisitos não funcionais mencionados.

Para não estar a abordar arquiteturas em demasia a nossa avaliação vai ter em conta 3 arquiteturas, estas sendo o mais distintas possível. A avaliação quantitativa é de valor mínimo 1 estrela e valor máximo 5 estrelas e cada avaliação acompanha uma explicação breve do porquê da mesma:

Característica arquitetural	Microsserviços	Master-worker	Layered
Manutenção e suporte	★★★★★ (Ao utilizar microserviços evitamos uma paragem total do sistema quando queremos fazer manutenção da app. Podemos simplesmente ir diretos ao problema em questão enquanto o resto das funcionalidades continua a funcionar de forma independente)	★★★★ (Esta arquitetura facilita a manutenção, pois o mestre gere as tarefas e distribuí-las para os trabalhadores, que são atualizados e mantidos igual e individualmente. O único senão é se o mestre falhar, o que faz com que o sistema como um todo falhe.)	★★★ (A modularidade promovida por este padrão de design arquitetural permite que sejam identificados e corrigidos problemas do sistema de forma eficiente. Podemos até testar cada camada individualmente, devido ao isolamento do código. Apesar disso, é necessário ter em conta que existe dependência entre camadas, o que torna o processo de alteração do sistema mais trabalhoso.)

Característica arquitetural	Microsserviços	Master-worker	Layered
Escalabilidade	★★★★★ (Ao modularizar um sistema tornamos a sua escalabilidade mais fácil, direta e económica, uma vez que otimizamos o serviço que tiver maior demanda e/ou necessidade, o que beneficia o sistema como um todo)	★★★★ (A escalabilidade nesta arquitetura depende da capacidade deste para distribuir as tarefas pelos seus trabalhadores. O sistema escala facilmente através da criação de mais trabalhadores. O desempenho pode ser limitado pela capacidade do mestre de gerir os seus súbditos.)	★★★ (É possível acomodar novas funcionalidades e alterações nas camadas que necessitam delas, consoante a demanda associada, no entanto, a complexidade do sistema e a conectividade entre as camadas podem aumentar a dificuldade dessa tarefa.)
Desempenho	★★★★★ (Os microsserviços podem trazer vantagens, podemos focar nos pontos que mais pesam no sistema de modo que ofereçam menos limitações, como menos gargalos, sendo ainda possível distribuir a carga por diferentes servidores. No entanto, existe o risco de problemas na sincronização entre microsserviços,o que obriga a uma gestão da base de dados mais controlada.	★★★★★ (As várias tarefas da aplicação são divididas pelos vários trabalhadores, nunca afetando o desempenho da aplicação, pois é tudo executado paralelamente.)	★★★★ (A utilização das camadas na arquitetura do sistema faz com que este esteja dependente da comunicação entre elas, e, dependendo do número de camadas presentes e da sua complexidade, a execução de uma operação pode ser relativamente custosa, e vir com uma latência associada mais elevada. Contudo, é possível tomar decisões relativas ao desempenho do sistema de forma mais focada e precisa, devido à sua modularidade.)

Característica arquitetural	Microsserviços	Master-worker	Layered
Extensabilidade	★★★★★ (A arquitetura de microsserviços é muito extensível, pois cada serviço é independente e pode ser atualizado, escalado ou substituído sem impactar outros serviços. Essa independência permite adicionar novas funcionalidades sem alterar as existentes.)	★★★ (É possível adicionar novos trabalhadores para melhorar a capacidade de processamento, mas, geralmente, esses trabalhadores executam tarefas similares definidas pelo mestre. Extensões para adicionar novos tipos de tarefas podem exigir alterações significativas na lógica do mestre.)	★★★★ (A extensibilidade depende de como as camadas são estruturadas e de como as interfaces entre camadas estão definidas. É possível adicionar novas camadas ou modificar camadas individuais, mas mudanças maiores podem necessitar ajustes em camadas adjacentes, o que em princípio não será o caso.)
Segurança	★★★★ (Como os serviços são isolados, as vulnerabilidades de um serviço não são as mesmas de outro, assim, qualquer ataque afeta apenas aquilo que atacou. O sistema é ainda mais seguro por causa desta arquitetura, já que cada serviço tem acesso a dados e permissões especificas. Como requerido anteriormente esta arquitetura permite utilizar MFA em cada ponto de comunicação.)	★★★ (Um pouco limitada pelo facto de, se o mestre é comprometido, toda a aplicação é comprometida, devido ao facto de que é ele quem comanda a operação. Uma das vantagens é a individualização das tarefas, pois se um dos trabalhadores for comprometido, só aquela tarefa específica é que é afetada.)	★★★★ (Dado que o sistema está dividido em camadas, diferentes medidas de segurança podem ser implementadas em cada uma delas, tendo em atenção as necessidades das mesmas. No entanto, uma falha de segurança numa camada pode colocar o resto do sistema em risco, já que existe dependência das camadas inferiores.)

Característica arquitetural	Microsserviços	Master-worker	Layered
Operacional	★★★★ (Através do bom uso desta arquitetura, se não forem cometidos erros, esta proporciona um ambiente excelente para desenvolvimento e funcionamento da mesma. No entanto, é muito difícil gerir o funcionamento de algo tão granular, o que pode ser um desafio.)	★★★★ (Uma aplicação desenvolvida neste estilo arquitetural consegue assegurar um ambiente de desenvolvimento organizado e modular, mas caso haja descuidos na projeção do sistema, corre-se o risco deste se transformar numa estrutura monolítica e complexa, difícil de manter, ou adaptar.)	★★★★ (Nesta arquitetura, os trabalhadores são adicionados, removidos ou substituídos sem interromperem o funcionamento da aplicação. Sendo tudo feito automaticamente, o único elemento a ser inicializado ou terminado é o mestre.)
Usabilidade	★★★★ Como os microsserviços são pequenos e especializados, podem ser projetados para fornecer API's claras e específicas, facilitando o uso para os clientes e desenvolvedores. No entanto, a interação entre múltiplos serviços pode gerar complexidade no uso e na manutenção.	★★★ (A simplicidade na comunicação (mestre delega tarefas para trabalhadores) pode facilitar o uso em alguns cenários, mas pode tornar-se limitada quando se precisa de flexibilidade e escalabilidade de tarefas mais complexas.)	★★★★★ (A arquitetura em camadas é organizada de forma que cada camada tenha uma responsabilidade bem definida, o que facilita a compreensão e o uso dos componentes pelos desenvolvedores. A divisão clara das camadas também simplifica o uso para utilizadores finais, pois cada camada serve um propósito específico e bem delimitado.=

Baseada na tabela acima avaliamos então cada requisito não funcional de prioridade Must:

Requisito	Descrição	Característica arquitetural	Microsserviços	Master-worker	Layered
RNF12	A aplicação deve ser capaz de escalar horizontalmente para suportar o aumento de utilizadores e volume de imagens, mantendo o desempenho	Desempenho/ Escalabilidade	★★★★★	★★★★	★★
RNF10	Operações avançadas devem ser concluídas em até 5 segundos por imagem. Nos processos encadeados, pode haver uma pequena redução de desempenho.	Desempenho	★★★★★	★★★★★	★★
RNF22	A aplicação deve realizar backups automáticos dos dados e imagens dos utilizadores, garantindo a sua recuperação em caso de falhas.	Segurança	★★★★★	★★★★	★★★★
RNF5	A aplicação deve apresentar o resultado da ferramenta em tempo real	Usabilidade/ Desempenho	★★★	★★★★	★★★
RNF11	O sistema deve processar até 100 imagens ao mesmo tempo, sem quedas percetíveis no desempenho.	Desempenho	★★★★★	★★★★★	★★

Requisito	Descrição	Característica arquitetural	Microsserviços	Master-worker	Layered
RNF23	O sistema deve usar MFA para aumentar a segurança dos utilizadores premium e registados.	Segurança	★★★★★	★★★	★★★★

Para obtermos um valor objetivo da nossa avaliação, usando os pesos definidos para cada característica arquitetural no capítulo 2, construímos uma tabela de médias finais das três arquiteturas:

	Microsserviços	Master-Worker	Layered
Média final (Características arquiteturais)	4.6 ★	3.8 ★	3.75 ★
Média final (Requisitos não funcionais)	~4.67 ★	~4.167 ★	~2.83 ★

Através desta avaliação, o grupo tomou a decisão de implementar o sistema com base na arquitetura de **microsserviços**, na sua generalidade, e para além disso, para haver uma melhor gestão e extensibilidade das ferramentas de edição de imagem, o microsserviço de gestão de ferramentas irá conter em si uma arquitetura **Master-Worker** que será devidamente descrita futuramente neste relatório.

4.2. Decomposição funcional

A decomposição funcional é um processo essencial no *design* e no planeamento de sistemas complexos, como é o caso da nossa aplicação. Este processo permite dividir o sistema em partes menores e mais fáceis de gerir, cada uma responsável por uma função ou conjunto de funções específicas. Ao identificar e segmentar funcionalidades chave, é possível alcançar uma melhor organização e clareza no desenvolvimento, além de facilitar a manutenção e evolução futura da aplicação.

Neste contexto, a decomposição funcional desempenha um papel crítico ao permitir uma arquitetura modular, onde cada componente pode ser desenvolvido, testado e implementado de forma independente. Isso é especialmente relevante para aplicações que envolvem múltiplos microsserviços, como gestão de ferramentas de edição de imagens, gestão de projetos e gestão de contas, garantindo que cada funcionalidade atenda às necessidades específicas dos utilizadores e ofereça uma experiência eficiente e intuitiva.

Além disso, ao decompor as funcionalidades, a equipa de desenvolvimento pode priorizar os recursos mais importantes, identificar dependências entre os módulos e distribuir tarefas de maneira eficiente. Este capítulo detalhará o processo de decomposição funcional da aplicação, destacando as razões pelas quais adotamos essa abordagem e como ela contribui para o alcance dos objetivos de desempenho, escalabilidade e qualidade do software.

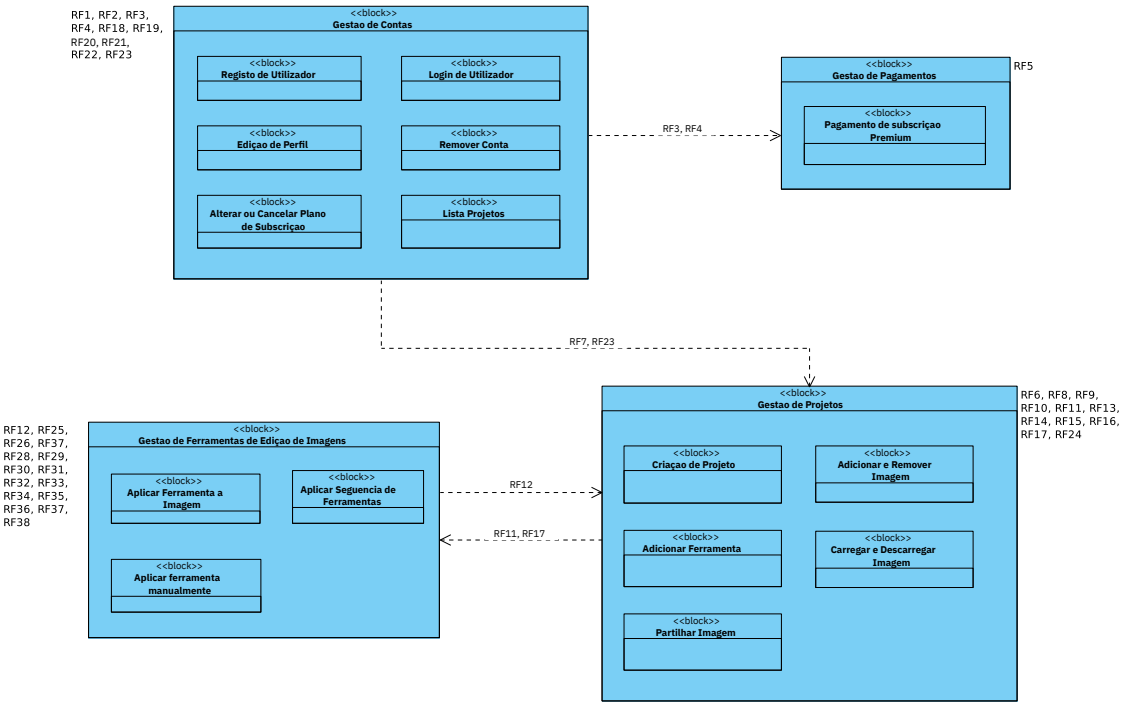

Figura 4: Diagrama de Decomposição Funcional

4.3. Decisões Organizacionais

Neste capítulo, detalhamos a organização da equipa e as decisões tomadas para garantir uma divisão eficiente de tarefas e responsabilidades no projeto *PictuRAS*. A tabela abaixo apresenta a distribuição de elementos por microsserviços, refletindo o esforço para balancear cargas de trabalho e assegurar o cumprimento dos prazos.

Microsserviço	Número de elementos
Frontend	4
API Gateway	1
Gestão de Pagamentos	1
Gestão de Contas	1
Gestão de Projetos	1
Ferramentas	2

Tabela 6: Distribuição de tarefas pela equipa

A lógica subjacente a esta distribuição fundamenta-se na análise das necessidades técnicas e operacionais de cada componente:

- **Frontend (4 elementos):** Devido à importância central da interface do utilizador na *PictuRAS*, quatro membros da equipa foram alocados ao desenvolvimento do frontend. Isto permite dividir tarefas como design de interface, desenvolvimento de componentes Vue.js, testes de usabilidade e otimização de desempenho, garantindo uma experiência intuitiva e responsiva para o utilizador dentro dos prazos estabelecidos.
- **API Gateway (1 elemento):** Atuando como ponto central entre o frontend e os microsserviços, a API Gateway exige elevado conhecimento técnico em protocolos de rede e segurança. Um especialista foi designado para garantir que seja eficiente, segura e capaz de lidar com múltiplas requisições simultâneas, cumprindo os requisitos de desempenho e escalabilidade.
- **Gestão de Pagamentos (1 elemento):** Este módulo lida com transações financeiras e dados sensíveis, requerendo foco especial em segurança e conformidade com normas financeiras. Um membro dedicado irá implementar processamento de pagamentos, gestão de subscrições premium e emissão de faturas.
- **Gestão de Contas (1 elemento):** Responsável por registo de utilizadores, autenticação e gestão de perfis. Apesar de fundamental, é modular e estável após a implementação inicial. Um elemento pode gerir eficientemente este microsserviço, focando-se na integração com a API Gateway.
- **Gestão de Projetos (1 elemento):** Encarga-se de armazenar e organizar os projetos dos utilizadores. Sendo mais estático e menos complexo, um elemento é suficiente para garantir a integridade dos dados, eficiência no acesso e armazenamento, e coordenação com o microsserviço de Ferramentas para processamento de imagens.
- **Ferramentas (2 elementos):** Núcleo da aplicação, envolve operações complexas e processamento intensivo. Dois membros foram alocados para desenvolver, testar e otimizar as diversas ferramentas de edição, garantindo que atendam aos requisitos de desempenho e possam ser facilmente estendidas para futuras funcionalidades.

Estas decisões refletem uma estratégia orientada para a eficiência, priorizando áreas de maior impacto sem negligenciar as funcionalidades de suporte.

4.3.1. Adoção da Metodologia Ágil

A equipa decidiu adotar a metodologia Ágil para gerir o desenvolvimento do projeto *PictuRAS*. Esta abordagem oferece maior flexibilidade e adaptabilidade às mudanças nos requisitos, permitindo entregas incrementais e frequentes. A metodologia Ágil facilitará a colaboração entre os membros da equipa, garantindo que o desenvolvimento se mantenha alinhado com as expectativas dos stakeholders e promovendo uma comunicação eficaz durante todo o processo.

5. View de Building Block

Nesta secção, abordaremos os diversos componentes que compõem o sistema. No próximo capítulo apresentaremos o diagrama de componentes geral, englobando todos os microsserviços. O objetivo deste diagrama é identificar os componentes e a comunicação entre eles. Nos próximos diagramas, descemos o nível de abstração e especificámos cada componente detalhadamente.

No exercício de decomposição funcional, identificamos os seguintes building blocks:

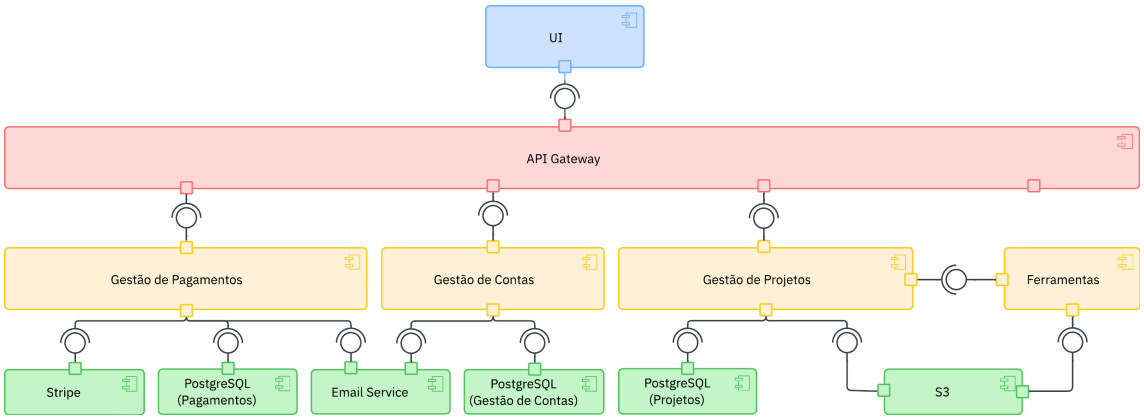

Figura 5: Diagrama de componentes - Diagrama Arquitetural

5.1. Tecnologias para Implementação

Para o desenvolvimento da aplicação *PictuRAS*, selecionámos tecnologias modernas que atendem aos requisitos de desempenho, segurança e escalabilidade. Para o backend (microsserviços por baixo da camada da API Gateway), escolhemos Rust em conjunto com o *framework* Axum. Rust é uma linguagem de programação rápida e segura, que em combinação com o Axum (*framework* para criação de *web servers*), permite-nos desenvolver um back-end extremamente eficiente, sem grande esforço de desenvolvimento. Esta eficiência é de extrema importância para cumprir os requisitos de performance do projeto.

Para *frontend*, optámos pelo Vue.js, uma *framework* JavaScript que permite criar interfaces de utilizador interativas. O Vue.js é simples de usar para desenvolvedores familiarizados com HTML, CSS e JavaScript, atualiza automaticamente a interface quando os dados mudam e promove um código organizado e reutilizável. Outra razão na escolha foi a familiaridade do grupo na tecnologia, que é maior em comparação a outras.

Para a base de dados, nos componentes em que a necessitam, escolhemos o PostgreSQL, um sistema de gestão de bases de dados robusto e fiável. O PostgreSQL é capaz de lidar com grandes volumes de dados, suporta transações seguras e tipos de dados complexos e é altamente escalável.

Na implementação das funcionalidades de edição e processamento de imagem, utilizamos bibliotecas de Rust como o Photon-rs, OpenCV e Tesseract. O Photon-rs é utilizado para ajustes de cor, filtros e transformações básicas; o OpenCV fornece funcionalidades avançadas de visão computacional, como reconhecimento de objetos. E o Tesseract é usado para reconhecimento de texto em imagens (OCR).

Para os pagamentos, utilizámos o Stripe que garante uma solução segura, fiável e eficiente de pagamentos, subscrições e reembolsos de forma simples e rápida, alinhada com os requisitos de desempenho e escalabilidade do projeto.

Além disso, devido à escolha dos serviços web AWS, para o armazenamento das imagens associadas aos projetos, utilizámos o S3, que proporciona um armazenamento escalável, com custos reduzidos, assegurando uma gestão eficiente dos ficheiros da aplicação.

Esta combinação tecnológica permite-nos desenvolver uma aplicação web que é rápida, segura, escalável e amigável ao utilizador. Conseguimos processar imagens de forma eficiente, prevenir erros, escalar a aplicação conforme aumenta o número de utilizadores e oferecer uma interface intuitiva e responsiva. Assim, garantimos que a *PictuRAS* proporciona uma experiência fluida e rápida aos utilizadores, cumprindo os requisitos definidos para o projeto.

5.2. Descrição dos Microsserviços

A aplicação web de edição de imagens *PictuRAS* foi projetada com uma arquitetura de microsserviços para garantir escalabilidade, flexibilidade e uma experiência integrada ao utilizador. Os principais componentes incluem como demonstrado anteriormente:

UI (Frontend): Responsável por renderizar o HTML que irá ser mostrado no lado do cliente, fazendo pedidos à API Gateway e comunicando com websockets com o cliente.

API Gateway: Atua como o ponto de entrada único para todos os pedidos da UI aos microsserviços. Centraliza e coordena a comunicação entre o frontend e os diversos serviços backend com interação segura e eficiente. Este componente é responsável por funcionalidades essenciais como autenticação e autorização de utilizadores, balanceamento de carga e roteamento de pedidos para os microsserviços apropriados.

Microsserviço de Gestão de Contas: Responsável por gerir o registo e login de utilizadores, autenticação, controlo de permissões e dados de perfil, garantindo acessos personalizados à plataforma e às funcionalidades que tem para oferecer.

Microsserviço de Gestão de Pagamentos: Lida com o processamento de transações e histórico de pagamentos, oferecendo um ambiente seguro para transações financeiras e várias opções de pagamento.

Microsserviço de Gestão de Ferramentas de Edição de Imagens: Fornece as funcionalidades de edição definidas nos requisitos. Decidimos agrupar as ferramentas num só microsserviço pois assim permite que sejam acessadas de maneira a facilitar o encadeamento de ferramentas e que estender/escalar estas ferramentas seja simples e direto, diferente do método de fazer um novo microsserviço para cada ferramenta fazendo com que a complexidade e quantidade de microsserviços atinja, a partir de certo ponto, um nível incontrolável de gestão.

Microsserviço de Gestão de Projetos: Organiza e armazena os projetos dos utilizadores, gerindo as imagens nele contidas e o seu histórico de edições, facilitando o acesso e o controlo das alterações feitas no mesmo. Também reencaminha pedidos de transformação para o microsserviço de Ferramentas.

Esta arquitetura baseada em microsserviços permite operarem de forma independente, otimizando recursos e facilitando a manutenção, atualização e processo de escalabilidade/extensibilidade do sistema.

5.2.1. Frontend

- **UI (Frontend):** Este serviço fornece a interface para utilizador comunicar-se com o sistema.

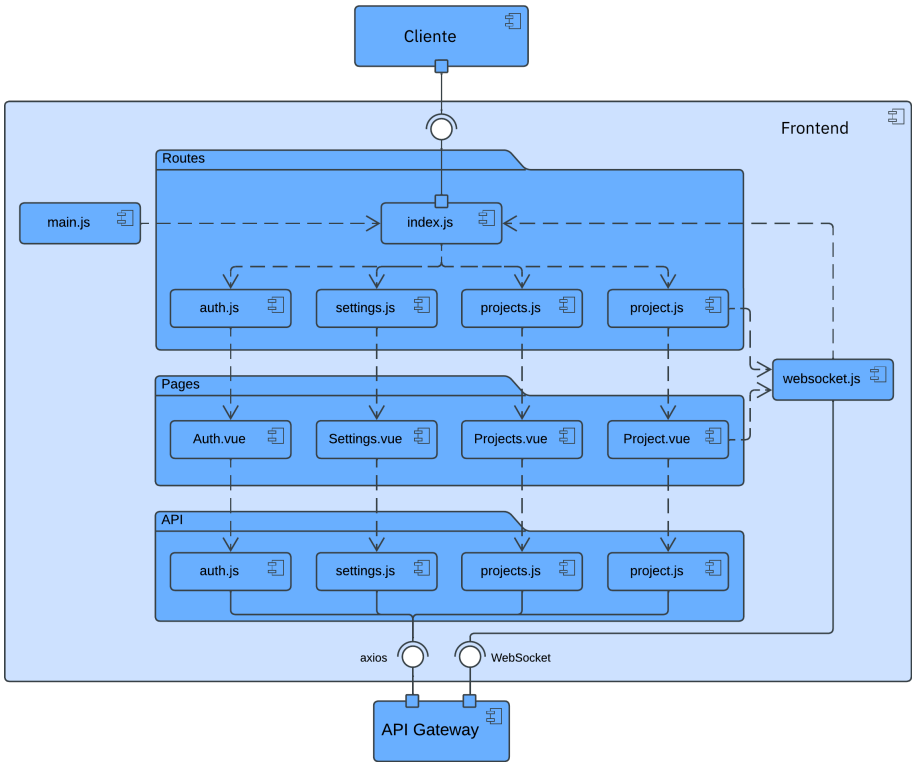

Figura 6: Diagrama de Componentes - Frontend

Descrevendo os componentes:

- **Routes (index.js e arquivos de rota):** O sistema de rotas organiza a navegação da aplicação, conectando URLs específicas às páginas correspondentes. Ele é composto pelos seguintes submódulos:
  - **index.js:** O arquivo principal de configuração das rotas.
  - **auth.js:** Define as rotas relacionadas à autenticação do utilizador (login, registo, recuperação de password).
  - **settings.js:** Gerencia as rotas associadas às definições do utilizador.
  - **projects.js:** Responsável pelas rotas
  - **project.js:** Responsável pelas rotas de visualização de um projeto.
  - **websocket.js:** Responsável por comunicação em tempo real com a API Gateway, para, por exemplo, atualização de progresso, resultado de ferramentas, etc.

- **Pages:** Este módulo contém os componentes do Vue.js responsáveis por renderizar as páginas da aplicação:
  - **Auth.vue:** Página dedicada à autenticação do utilizador.
  - **Settings.vue:** Interface para gerir as definições do utilizador.
  - **Projects.vue:** Página para listar e gerir os projetos.
  - **Project.vue:** Página para visualizar os detalhes de um projeto específico.

- **API:** O módulo de API gerencia as chamadas à API Gateway através da biblioteca Axios, encapsulando a lógica para interação com o backend:
  - **auth.js:** Gerencia chamadas relacionadas à autenticação (login, logout, registo).
  - **settings.js:** Envia e recebe dados relacionados às configurações do utilizador.
  - **projects.js:** Responsável por operações de criação, leitura, atualização e eliminação (CRUD) relacionadas a projetos.
  - **project.js:** Responsável por operações relacionadas a um projeto, por exemplo, adição ou remoção de novas imagens ou ferramentas.

- **WebSocket:** Necessário para manter o utilizador atualizado do progresso da aplicação de ferramentas.
  - **websocket.js:** Usado como API para comunicação de Web Sockets entre o cliente e a API Gateway.

Esta arquitetura permite uma separação clara entre as camadas de interface, navegação e integração com o backend, promovendo uma aplicação frontend eficiente, escalável e de fácil manutenção.

5.2.2. API Gateway

- **API Gateway**: A API Gateway centraliza as comunicações entre o frontend e os diferentes microsserviços. Esta abordagem oferece uma camada adicional de segurança e abstração, facilitando a gestão de chamadas HTTP e o balanceamento de carga. Também permite implementar autenticação e autorização num único ponto de entrada.

Para melhor descrever e entender as dependências de componentes apresentamos o diagrama de componentes do software da API Gateway a ser implementada:

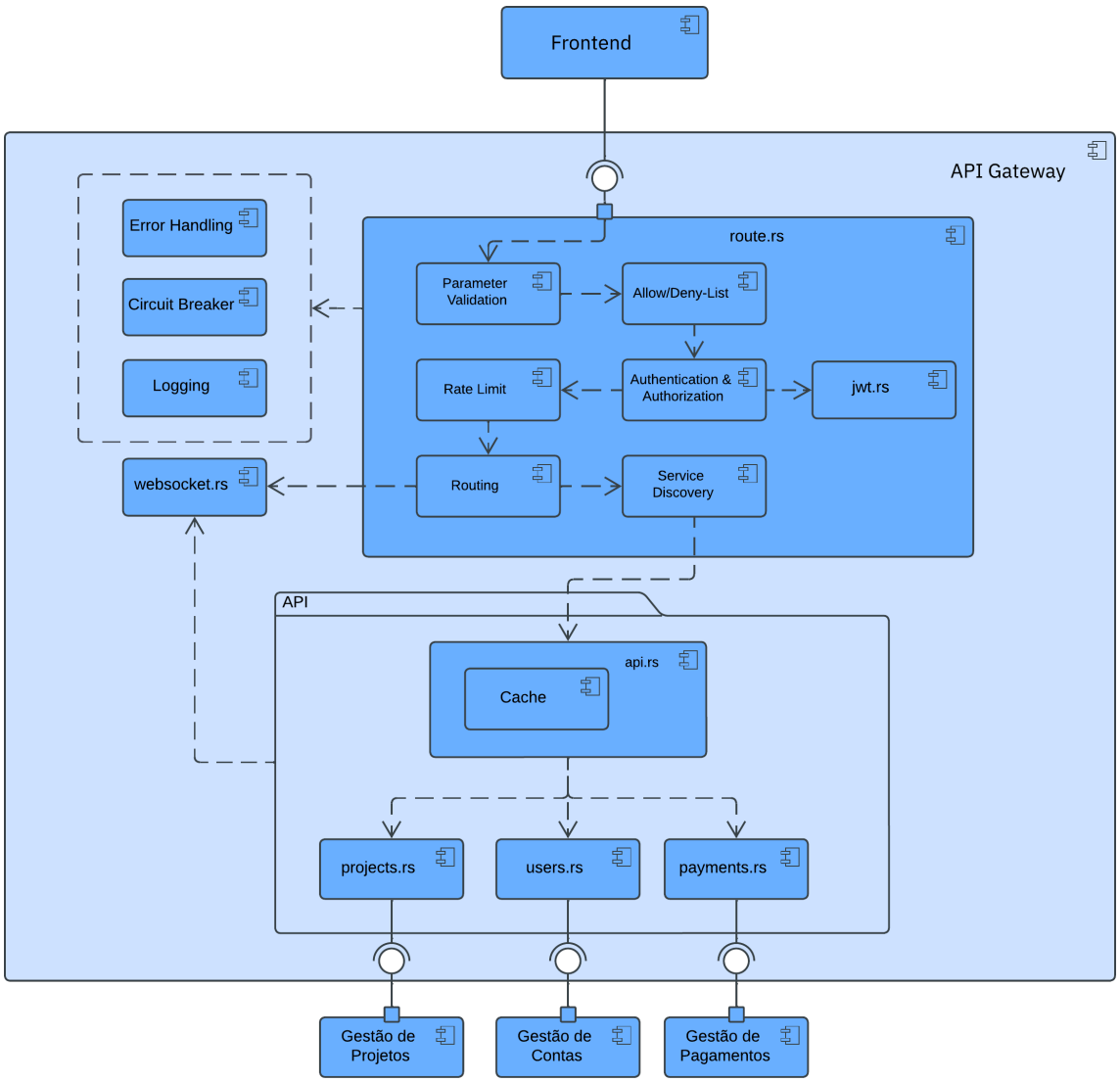

Figura 7: Diagrama de componentes - API Gateway

Componentes de gestão e segurança

- **Parameter Validation**: Verifica se os parâmetros das solicitações estão no formato correto e dentro dos valores permitidos. Isso ajuda a garantir que apenas dados válidos e formatados corretamente entrem no sistema.
- **Rate Limit:** Controla a quantidade de solicitações que um cliente pode fazer em um determinado período para assim evitar sobrecarga de pedidos.
- **Routing:** Direciona as solicitações para o microsserviço correto com base nas regras configuradas.
- **Allow-list / Deny-list:** Define regras para permitir ou negar acesso a clientes específicos com base em critérios como IP, localização ou outros identificadores.
- **Authentication & Authorization:** Verifica a identidade dos utilizadores ou sistemas que fazem requisições (autenticação) e valida se eles têm permissão para aceder os recursos solicitados (autorização). Este componente é integrado com o componente jwt.rs para gestão de tokens.
- **Service Discovery:** Ajuda o Gateway a localizar dinamicamente as instâncias dos microsserviços para que ele possa redirecionar as solicitações corretamente, mesmo que as instâncias dos serviços mudem frequentemente.
- **Protocol Conversion:** Converte protocolos entre diferentes tipos de comunicação. Isso permite que serviços que usam diferentes protocolos de comunicação cooperem sem problemas.

Componentes de Suporte e Resiliência

- **Error Handling:** Gere e regista erros que ocorrem durante o processamento de solicitações. Este componente ajuda a garantir que os clientes recebam mensagens de erro apropriadas e que erros sejam monitorados.
- **Circuit Breaker:** Monitoriza as conexões para os microsserviços e, se um microsserviço começar a falhar repetidamente, o circuito abre, bloqueando novas tentativas de conexão temporariamente. Isso ajuda a evitar sobrecargas em microsserviços problemáticos.
- **Logging & Monitoring:** Regista informações sobre as solicitações e respostas, ajudando na análise e na monitorização do desempenho do sistema.
- **Cache:** Armazena temporariamente respostas a solicitações comuns para reduzir a carga sobre os serviços e melhorar a latência.

Componentes de Integração

- **jwt.rs:** Este módulo é responsável pela validação e descodificação de *tokens JWT* para autenticação e autorização segura.
- **websocket.rs:** Módulo necessário para gestão e comunicação de Web Sockets clientes.
- **Microsserviços Backend:** O API Gateway integra-se aos módulos *backend*, incluindo Ferramentas, Gestão de Projetos, Gestão de Contas e Gestão de Pagamentos. Cada módulo é representado por um componente específico (`ferramentas.rs`, `projects.rs`, `users.rs`, `payments.rs`) dentro do componente `api.rs`.

5.2.3. Gestão de Contas

Gestão de Contas: Este microsserviço é responsável pela gestão das contas de utilizador, incluindo registo, autenticação e manutenção de perfis. Isso assegura que as operações de conta possam ser realizadas de maneira segura e independente de outras funcionalidades.

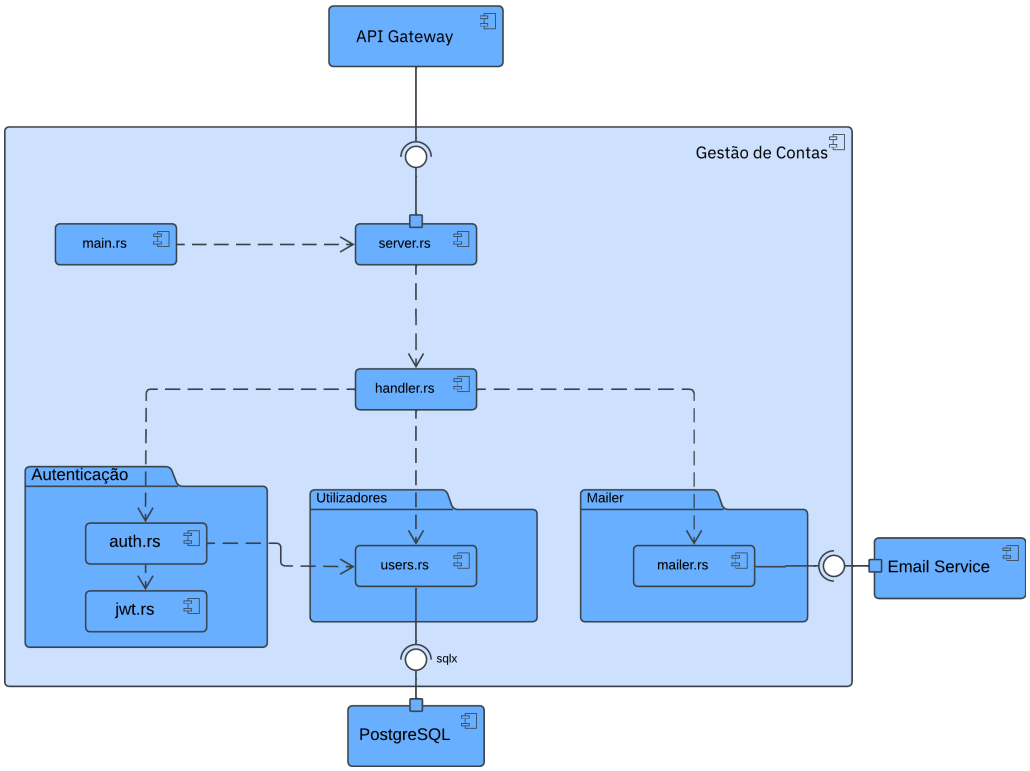

Figura 8: Diagrama de Componentes - Gestão de Contas

Segue-se a descrição dos seus componentes:

Routes (routes.rs): Este módulo define as rotas associadas à Gestão de Contas. Fornece os *endpoints* necessários para as operações relacionadas com autenticação, gestão de utilizadores e envio de notificações, servindo como ponte entre o exterior e o restante sistema.

Handler (handler.rs): Centraliza a lógica de processamento de pedidos, encaminhando-as para os módulos adequados (como autenticação ou gestão de utilizadores).

Autenticação (auth.rs): O módulo de autenticação implementa a lógica necessária para validar e gerir as credenciais dos utilizadores e verificação de permissões. Este módulo é essencial para garantir a segurança do sistema.

- **JWT (jwt.rs):** Responsável por fazer a criação de JWTs e manter a chave privada.

Utilizadores (users.rs): Este módulo gere os dados e as operações relacionadas com os utilizadores, incluindo o registo, atualização de perfis e consulta de informações. Estas operações incidem sobre uma base de dados PostgreSQL.

Mailer (mailer.rs): O módulo responsável por enviar emails transacionais relacionados às contas, como confirmações de registo, alterações de password e notificações de atividades relevantes.

Esta arquitetura modular permite o desenvolvimento, manutenção e extensão dos serviços de forma independente, enquanto assegura robustez e confiabilidade na gestão de contas dos utilizadores.

5.2.4. Gestão de Pagamentos

O componente de Gestão de Pagamentos tem como objetivo principal gerir a lógica relacionada com transações financeiras.

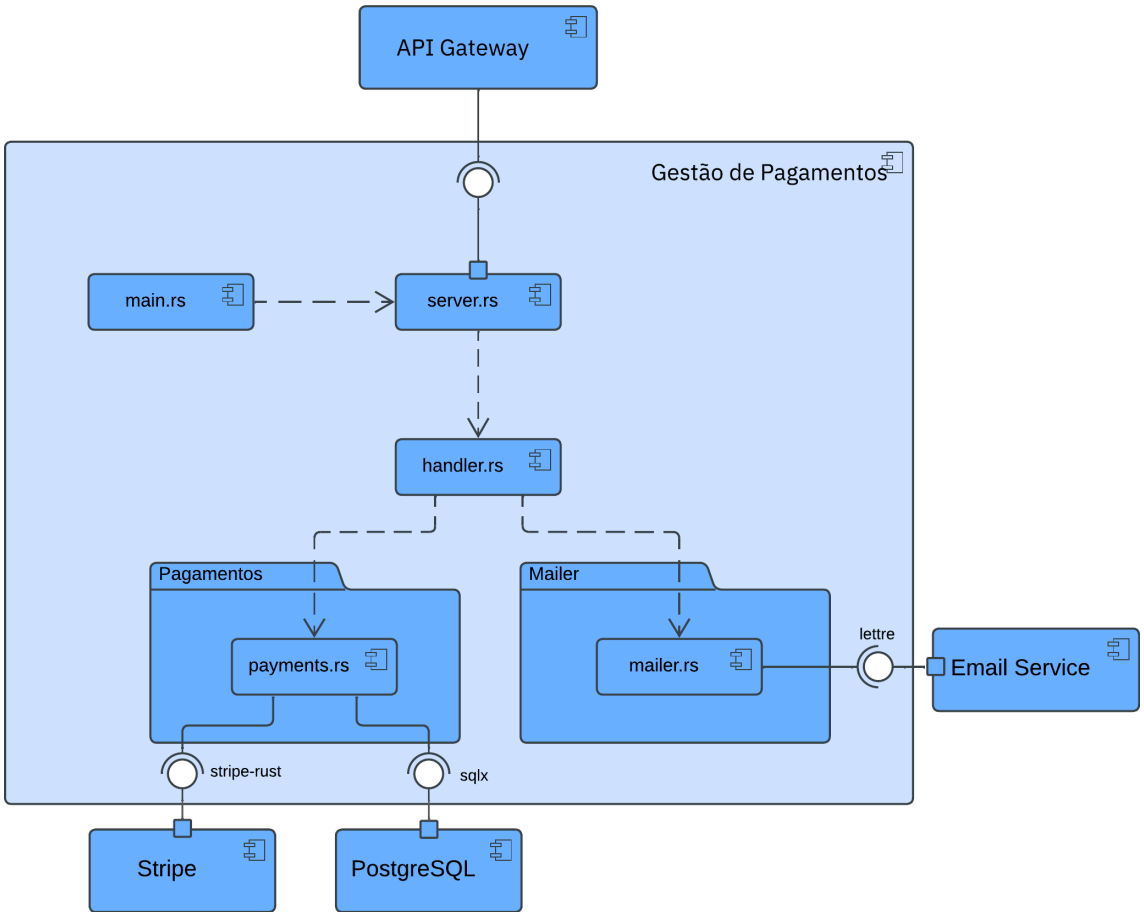

Figura 9: Diagrama de Componentes - Gestão de Pagamentos

Fazendo uma descrição breve de cada um dos seus componentes:

Server (server.rs): Este módulo define as rotas do serviço, organizando os endpoints disponíveis para a interação com o sistema. Serve como ponto inicial para a gestão de pedidos antes de serem processadas pelos *handlers*.

Handler (handler.rs): O handler.rs centraliza a lógica de processamento dos pedidos, fazendo a ligação entre as rotas e os componentes internos, como o módulo de Pagamentos ou o módulo de Mailer.

Pagamentos (payments.rs): Este módulo implementa a lógica de negócios relacionada com transações financeiras. Está integrado com a biblioteca stripe-rust para interagir com a API da Stripe e realizar operações como autorizações, capturas e reembolsos. Além disso, com a biblioteca *sqlx*, guarda os dados relacionados numa base de dados PostgreSQL.

Mailer (mailer.rs): O mailer.rs é responsável pela lógica de envio de emails transacionais, como notificações de pagamento bem-sucedido, juntamente com a fatura, ou falhas em transações.

5.2.5. Gestão de Projetos

- **Gestão de Projetos**: Este serviço organiza e armazena as informações dos projetos de edição de imagens dos utilizadores. Ele armazena o histórico das edições, imagens carregadas e configurações de cada projeto, permitindo que os utilizadores revisitem e modifiquem projetos antigos a qualquer momento.

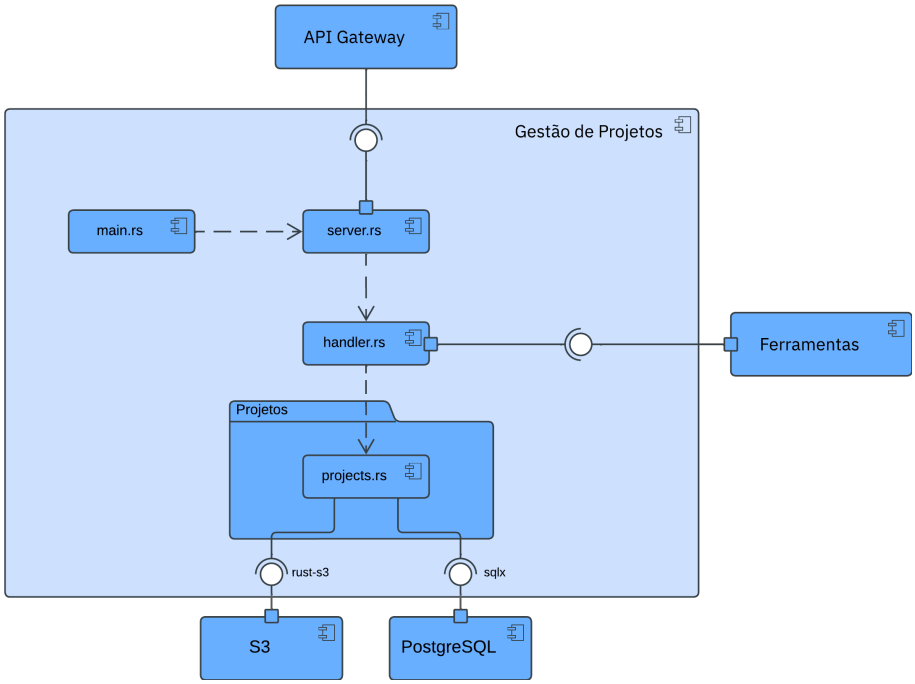

Figura 10: Diagrama de Componentes - Gestão de Projetos

Seguindo para a descrição de cada um dos componentes:

- **Server (server.rs):** Este módulo define os *endpoints* expostos pelo sistema, permitindo a interação com as funcionalidades de gestão de projetos. Mantem as rotas e encaminha as requisições recebidas para o *handler* apropriado.
- **Handler (handler.rs):** Centraliza a lógica de processamento das requisições relacionadas aos projetos, ligando os *endpoints* às operações específicas implementadas no módulo `projects.rs`. Atua como intermediário entre as requisições e as operações de negócio. Pedidos de transformação a imagens serão reencaminhados para o microsserviço de ferramentas, juntamente com o URL da imagem, que será obtido a partir da base de dados.
- **Projetos (projects.rs):** Este módulo contém a lógica de negócios para gerir os dados dos projetos. Suas responsabilidades incluem:
  - **Criação e atualização de projetos:** Guardar o estado atual do projeto (ferramentas aplicadas, histórico e URLs de imagens) na base de dados PostgreSQL.
  - **Persistência de Imagens:** Utilizando bibliotecas como `rust-s3` para interagir com o serviço de armazenamento Amazon S3, guarda imagens referentes ao projeto. A tabela dos projetos na base de dados PostgreSQL terá um identificador onde especificará a localização das imagens referentes ao projeto no S3.

Esta arquitetura modular oferece flexibilidade e escalabilidade para gerir eficientemente os projetos, ao mesmo tempo em que garante a integridade dos dados e a facilidade de integração com outros serviços do sistema.

5.2.6. Ferramentas

- **Ferramentas (com arquitetura Master-Slave)**: Este microsserviço lida com as ferramentas de edição de imagem, com uma arquitetura Master-Slave para atender ao requisito de escalabilidade nas operações de processamento de imagem. Nesta estrutura, o serviço ***Master*** coordena a distribuição das tarefas de processamento para múltiplos ***Slaves***, garantindo que as operações pesadas possam ser processadas em paralelo, otimizando o tempo de resposta para o utilizador.

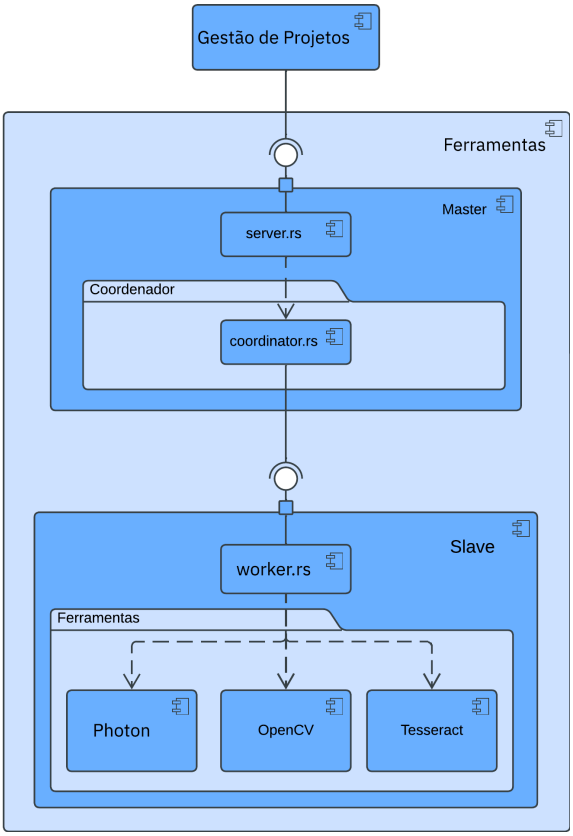

Figura 11: Diagrama de Componentes - Ferramentas

Descrevendo agora cada um dos componentes:

Componente Master:

- **server.rs:** Responsável por encaminhar um pedido de aplicar ferramenta para o **coordinator.rs**.
- **coordinator.rs:** Reencaminha o pedido de aplicar ferramenta para o *slave* (mais livre por exemplo) que irá resultar num maior aproveitamento de recursos.

Componente Slave:

- **worker.rs:** Responsável por encaminhar o pedido vindo do **Master** para cada uma das bibliotecas de edição de imagem, dependendo do pedido, e atualizar o progresso da aplicação da transformação ao **Master** (que esse irá reencaminhar para a **Gestão de Projetos**).
- **Photon:** Aplica efeitos de transformação básicos, como aumentar saturação ou contraste, alterar tamanho da imagem, etc.
- **OpenCV:** Aplica efeitos de transformação avançados, como detetar pessoas ou caras.
- **Tesseract:** Reconhecimento de caracteres.

5.3. Padrões e Estratégias para a Qualidade

A arquitetura de microsserviços foi escolhida para cumprir os seguintes atributos de qualidade:

- **Escalabilidade Horizontal**: A escalabilidade horizontal da aplicação *PictuRAS* é suportada pela possível replicação independente de cada microsserviço, que, por não precisar compartilhar estado, pode ser escalado facilmente. As bases de dados de cada microsserviço (que necessitam dela) podem usar ***sharding*** para distribuir dados em vários servidores. A API Gateway poderá realizar balanceamento de carga para cada microsserviço, otimizando a distribuição de solicitações, enquanto o microsserviço de Ferramentas, com arquitetura Master-Slave, ajusta o número de "slaves" conforme a carga, garantindo um desempenho consistente mesmo sob alta demanda. O "Master" deste microsserviço se necessário também poderá ser replicado facilmente pelas mesmas razões apresentadas.
- **Desempenho**: A independência dos microsserviços reduz o impacto no desempenho em caso de aumento de carga em um serviço específico. Além disso, a distribuição de tarefas de processamento de imagem contribui para a redução da latência.
- **Segurança**: A gestão centralizada de autenticação e autorização no API Gateway, utilizando tokens JWT, garante que apenas utilizadores autenticados e autorizados acedem os recursos. Este design protege contra acessos não autorizados e simplifica o cumprimento das regulamentações de proteção de dados.
- **Extensibilidade e Manutenção**: Cada microsserviço pode ser modificado ou substituído independentemente, promovendo uma evolução contínua e uma manutenção simplificada sem impacto no sistema global.
- **Armazenamento Escalável e Disponível:** Para armazenamento de ficheiros, como imagens associadas aos projetos, a aplicação utiliza o S3, que oferece alta disponibilidade e escalabilidade. Isso permite que os utilizadores armazenem e acedem ficheiros de forma eficiente, mesmo com um grande volume de dados.

6. Runtime View

Nesta secção é exposto o funcionamento do sistema com recurso a diagramas de componentes, que mostram como as funcionalidades serão incorporadas em cada componente dos microsserviços, e diagramas de sequência que descrevem cada Use Case definido para entendermos como cada elemento do sistema participa nas várias interações entre utilizador e sistema.

6.1. Runtime (Use Case - Registo do utilizador)

O use case **Registo do utilizador** descreve o registo de um utilizador que ainda não tem conta no *PictuRAS*. O fluxo alternativo surge do tipo de conta que o utilizador deseja subscrever e os casos de erro acontecem quando o sistema deteta que os dados inseridos são inválidos ou o utilizador não conclua o pagamento.
Para melhor entender este processo apresentamos o diagrama de atividade e de sequência para este Use Case:

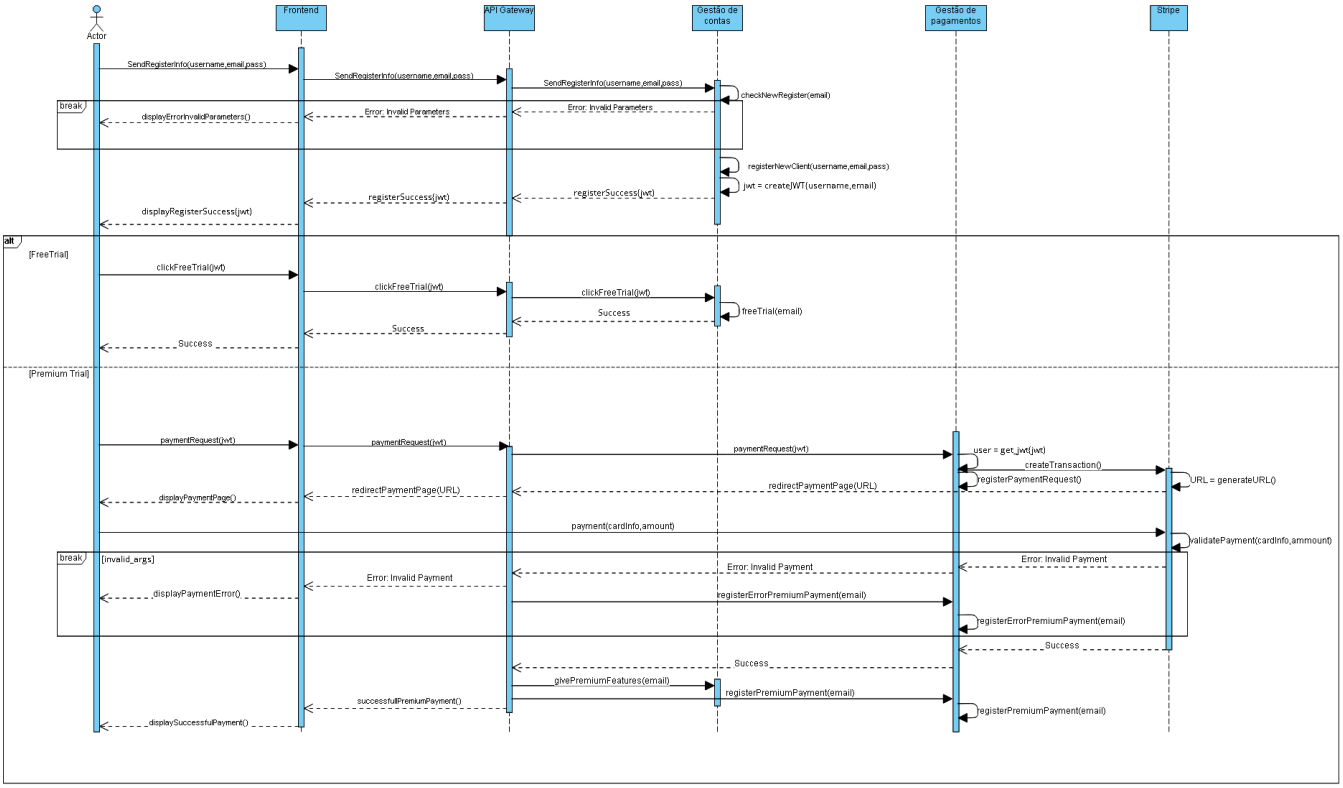

Figura 12: Diagrama de sequência - Registo de utilizador

6.2. Runtime (Use Case - Login do utilizador)

O use case **Login do utilizador** consiste na autenticação de um utilizador presente no sistema, sendo o seu login feito através dos seguintes parâmetros: email/username e password.
Para melhor entender este processo apresentamos o diagrama de atividade e de sequência para este Use Case:

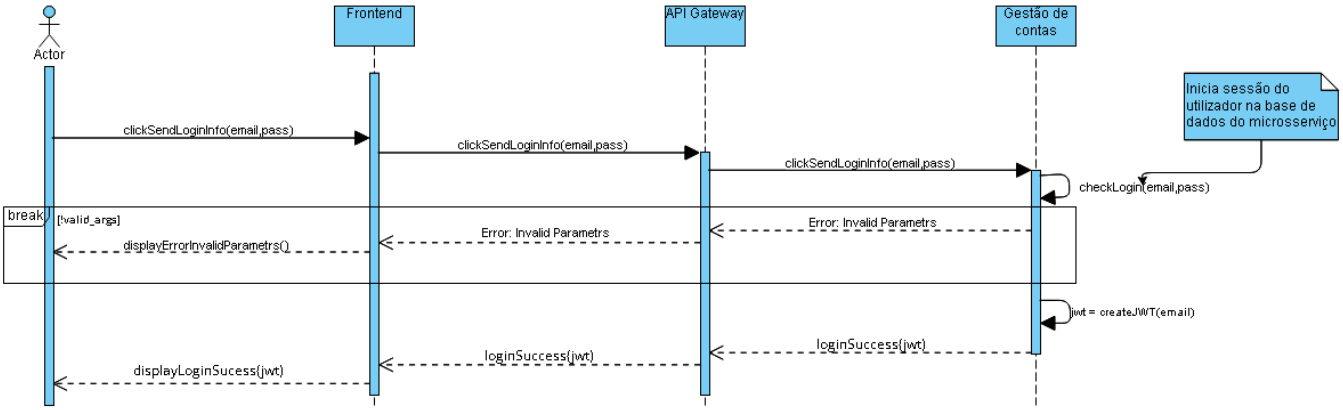

Figura 13: Diagrama de sequência - Login de utilizador

6.3. Runtime (Use Case - Carregamento de imagens)

O use case **Carregar imagens** descreve o procedimento de upload de imagens para um dado projeto.
Para melhor entender este processo apresentamos o diagrama de atividade e de sequência para este Use Case:

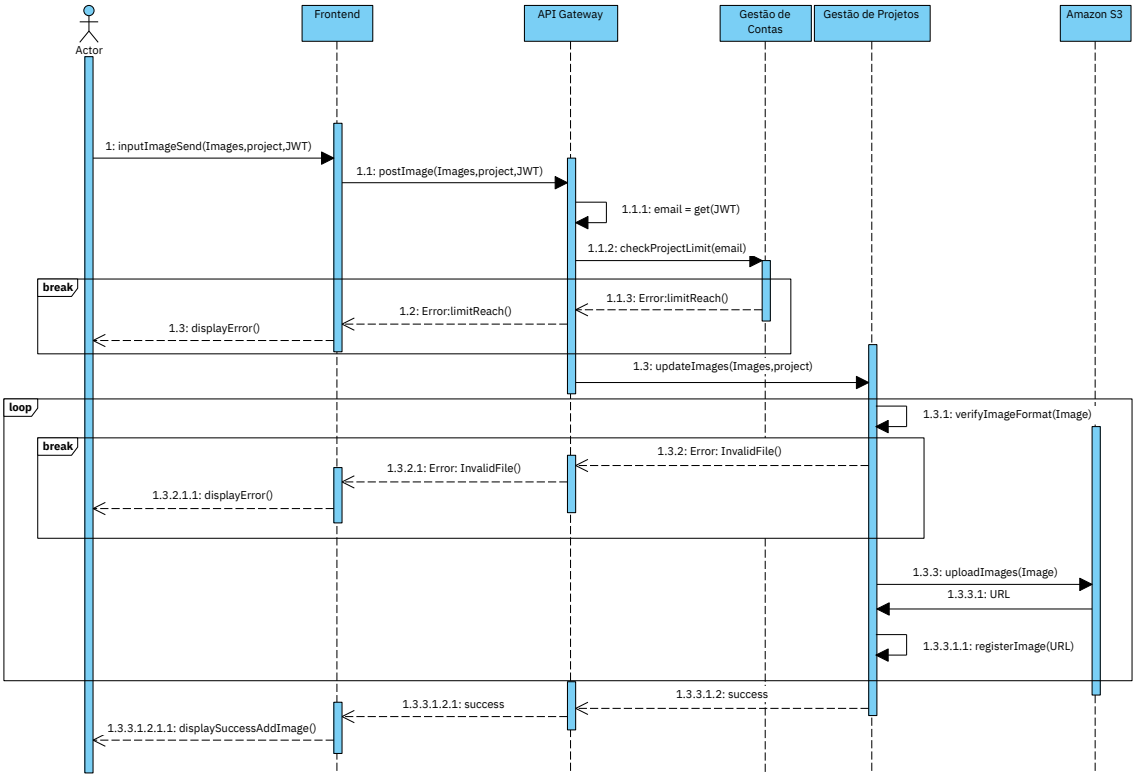

Figura 14: Diagrama de sequência - Carregamento de Imagens

6.4. Runtime (Use Case - Aplicação de encadeamento de ferramentas num conjunto de imagens)

O Use Case aplicar encadeamento de ferramentas a conjunto de imagens descreve o procedimento de selecionar e parametrizar ferramentas num projeto, e processá-lo (i.e., aplicar a sequência de ferramentas a todas as suas imagens).
Para melhor entender este processo apresentamos o diagrama de atividade e de sequência para este Use Case:

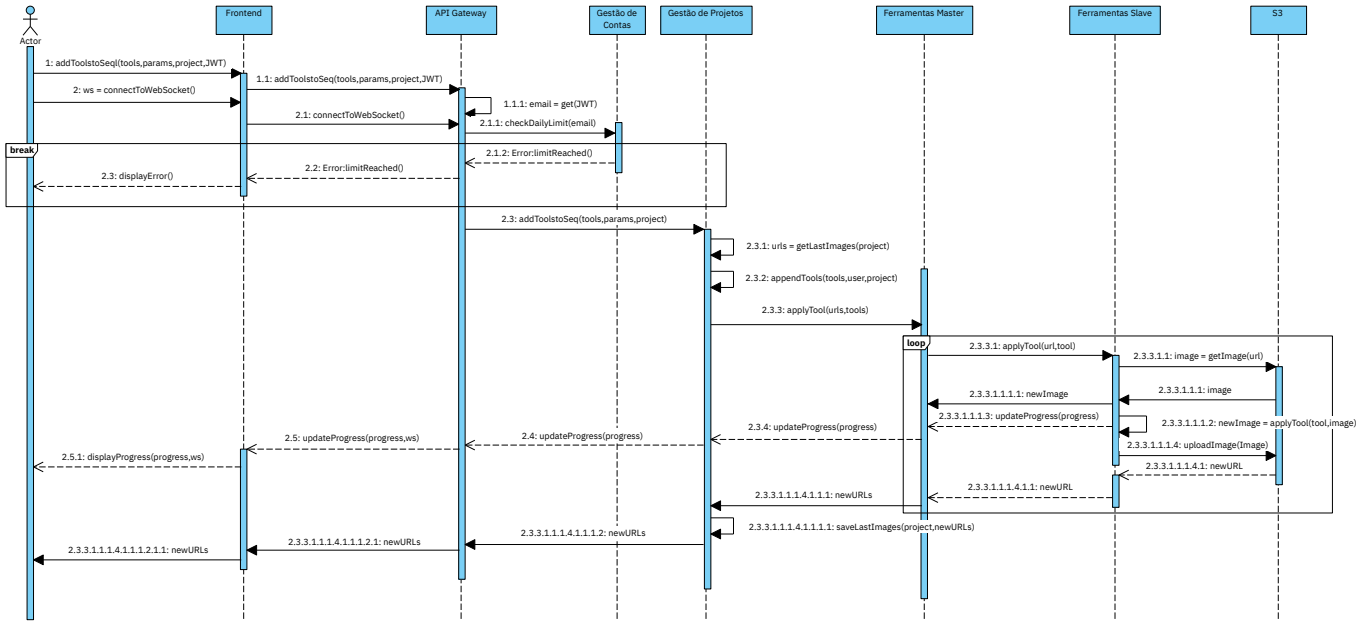

Figura 15: Diagrama de sequência - Aplicação de encadeamento de ferramentas num conjunto de imagens

7. Deployment View

O ***Deployment View*** é uma perspetiva arquitetural essencial para compreender como os componentes de software de uma aplicação são mapeados para os recursos físicos e virtuais do ambiente de execução. Este descreve a organização e a distribuição do sistema em termos de servidores, containers, redes e outros elementos de infraestrutura, garantindo que a solução atenda a requisitos não funcionais como desempenho, escalabilidade, confiabilidade e segurança.

No contexto da nossa aplicação de edição de imagens, o Deployment View é particularmente relevante devido à necessidade de suportar uma arquitetura baseada em microsserviços. Cada serviço, como a gestão de contas, gestão de projetos, ferramentas de edição e gestão de pagamentos, deve ser implementado de forma eficiente para garantir comunicação rápida e integração fluida. Além disso, a distribuição correta dos serviços minimiza "*choke points*", otimiza o uso de recursos e facilita a gestão da aplicação em ambientes de produção.

Este capítulo explora a importância do Deployment View na estruturação de nossa solução, detalhando como ele nos auxilia a tomar decisões sobre alocação de recursos e configuração de redes. Com esta visão, asseguramos que a aplicação seja robusta e escalável, entregando a melhor experiência possível ao utilizador final.

Iremos apresentar dois diagramas de deployment do nosso sistema. O primeiro diagrama ilustra a aplicação a ser executada numa única máquina, com o objetivo de suportar o desenvolvimento e a realização de testes. O segundo diagrama representa o deployment do sistema em ambiente de produção.

7.1. Contexto de testes e desenvolvimento

O nosso sistema baseia-se numa arquitetura de microsserviços, onde todos os componentes estão isolados em ambientes de virtualização e para efeitos de testes iremos utilizar o Docker. Cada serviço, com exceção do microsserviço das ferramentas, terá a sua própria base de dados. A comunicação entre o front-end, a gateway e os microsserviços será realizada através de HTTP. Entre os microsserviços e as bases de dados, a comunicação será feita por TCP, incluindo a interação entre o master do microsserviço das ferramentas e os seus slaves. Assim, o diagrama de deployment é o seguinte:

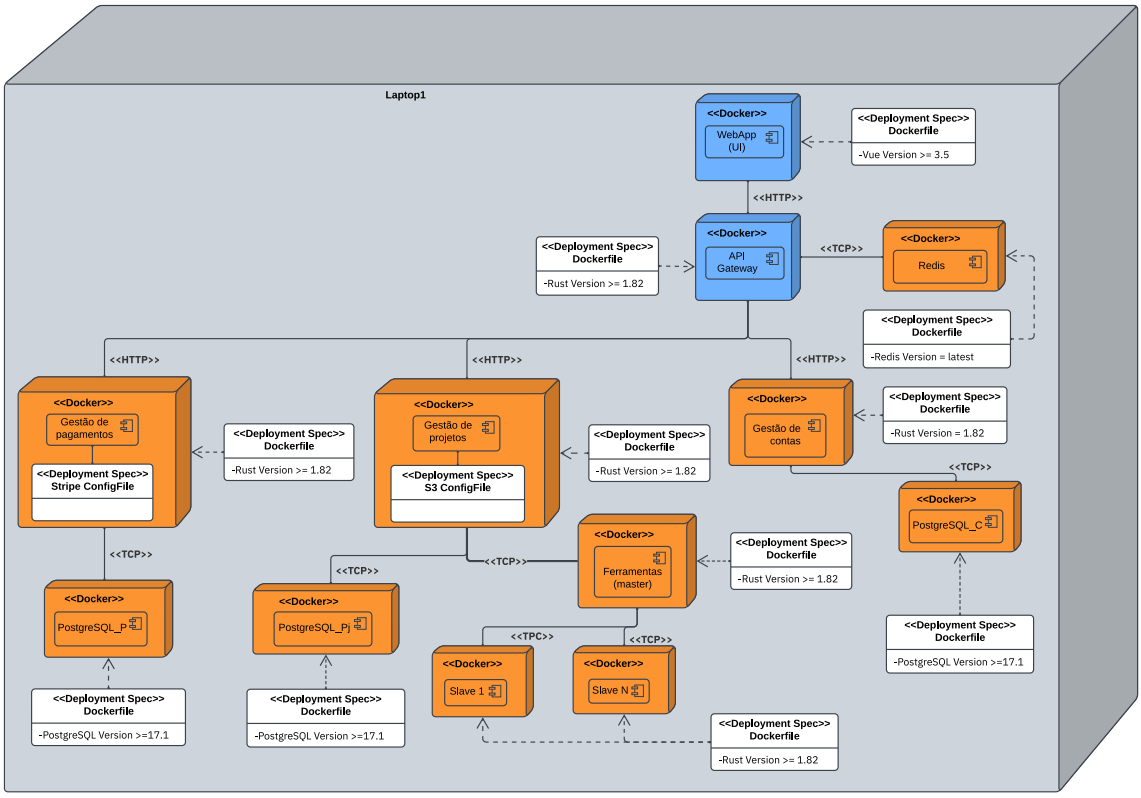

Figura 16: Diagrama de deployment - Testes e desenvolvimento

7.2. Contexto de produção

Como ilustrado na figura abaixo, a nossa arquitetura será baseada num sistema distribuído, implementado num cluster **EKS (Amazon Elastic Kubernetes Service)** que irá controlar todos os pods. Cada microsserviço, por usa vez, será encapsulado em pods, que estarão isolados em containers Docker, garantindo portabilidade e eficiência. Além disso, cada pod será executado em máquinas distintas, proporcionando maior isolamento, escalabilidade e performance. Embora a arquitetura seja bastante semelhante à utilizada nos ambientes de testes e desenvolvimento, em produção a maior parte das comunicações serão realizadas exclusivamente através do HTTPS, assegurando maior segurança na troca de dados.

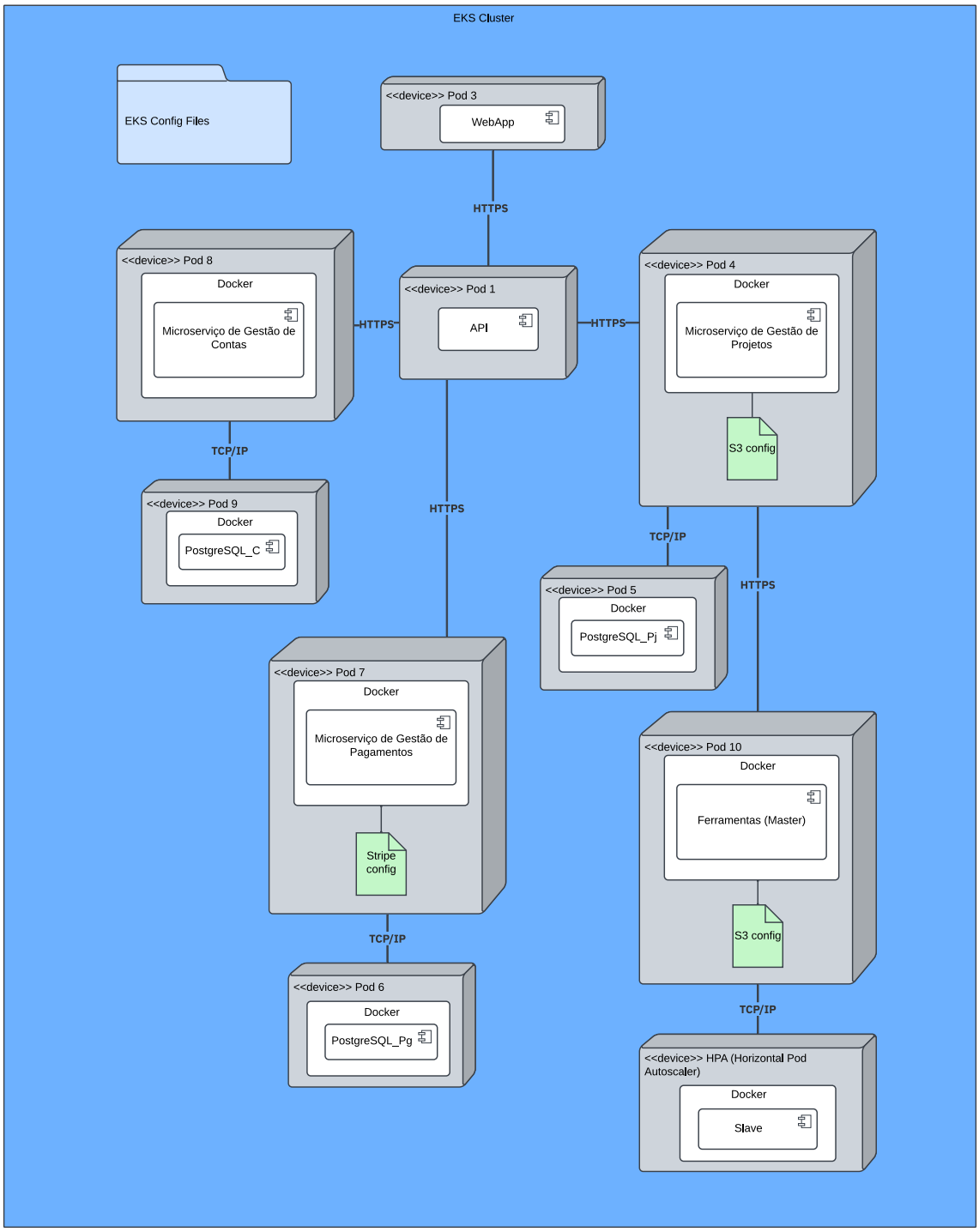

Figura 17: Diagrama de deployment - Produção

8. Conclusão

A segunda fase do desenvolvimento da aplicação de edição de imagens *PictuRAS* da unidade curricular de "Requisitos e Arquitetura de Software" concentrou-se no desenho da arquitetura da solução final. Com base nos requisitos previamente identificados, esta etapa priorizou a definição de uma estrutura robusta e escalável, assegurando a coerência entre as restrições técnicas, orçamentais e temporais.

O trabalho incluiu os seguintes pontos que o grupo achou essenciais:

- **Identificação das restrições**, que guiaram as decisões arquiteturais, considerando limitações técnicas, prazos e âmbito do projeto.
- **Análise detalhada do contexto e escopo**, destacando entidades externas, contexto técnico e de negócio, que fundamentam as funcionalidades e a operação do sistema.
- **Definição da estratégia da solução**, onde foram especificados os padrões arquiteturais adotados e a decomposição funcional que organiza os componentes do sistema.
- **Elaboração do View de Building Block**, no qual foram detalhados os microsserviços que compõem a solução, suas tecnologias de implementação, bem como estratégias de qualidade para garantir um desempenho consistente.
- **Exploração da Runtime View**, que demonstrou o comportamento do sistema em cenários específicos, nomeadamente registo de utilizadores, login, carregamento de imagens e aplicação de ferramentas de edição em conjunto de imagens.
- **Planeamento do View de Deployment**, incluindo a definição dos níveis de infraestrutura e a configuração necessária para o correto funcionamento da aplicação.

O resultado desta fase reflete um progresso significativo rumo à materialização do sistema proposto. Através da estruturação detalhada da arquitetura, foram estabelecidas as bases para a próxima etapa, que envolverá a implementação e validação das funcionalidades descritas. Este esforço reforça a importância de uma visão arquitetural sólida, essencial para garantir a escalabilidade e manutenção de uma aplicação a longo prazo.