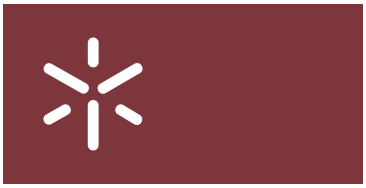

Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Requisitos e Arquiteturas de Software

Ano Letivo de 2024/2025

PictuRAS

Eduardo Cunha
PG55939

Rodrigo Ralha
PG56005

Délio Alves
a94557

Rafael Peixoto
PG55998

Rui Gonçalves
PG56011

November 13, 2024

RAS

Índice

1. Introdução e Objetivos **1**

2. Restrições **2**
2.1. Restrições Temporais 2
2.2. Restrições Técnicas/Solução 2
2.3. Restrições Orçamentais 3
2.4. Restrições de Âmbito 3

3. Requisitos Não Funcionais **5**
3.1. Impacto dos Requisitos no Sistema 7

4. Contexto e Âmbito do Sistema **8**

5. Estratégia da Solução **9**
5.1. Padrão arquitetural 9
5.1.1. Map Reduce 9
5.1.2. Micro-Serviços 10
5.1.3. Publish-Subscribe 11
5.1.4. Peer-to-Peer 11
5.1.5. Client Server and N Tier 11
5.1.6. Layered 11
5.1.7. Batch-Sequential 11
5.1.8. Model-Centered 11
5.1.9. Pipe-and-Filter 11
5.2. Definição da Arquitetura 11
5.2.1. Identificação de microsserviços 11
5.3. Modelos de dados das Bases de Dados 13
5.3.1. Gestão de Projetos DB 13
5.3.2. Users DB 13
5.4. Tecnologias para a implementação 14
5.4.1. Frontend 14
5.4.2. Microsserviços e API Gateway 14
5.4.3. Bases de dados 14
5.5. Decisões organizacionais 14

6. Building Block View **16**

7. Runtime View **17**

8. Deployment View **18**
8.1. Contexto de desenvolvimento e teste 18
8.2. Contexto de produção 18

Lista de Figuras

1. Introdução e Objetivos

Este documento foi concebido no âmbito de uma proposta para desenvolver o sistema *PictuRAS*, que visa oferecer uma plataforma de processamento e edição de imagens, acessível a diversos tipos de utilizadores. A motivação principal para a criação do *PictuRAS* reside na necessidade de uma ferramenta intuitiva e versátil que permita tanto a amadores como a profissionais explorar um vasto conjunto de funcionalidades de edição de imagem de forma eficiente e acessível, suportada por uma arquitetura escalável e moderna na *cloud*.

Nele, descrevem-se diversas etapas relacionadas com o planeamento da implementação do sistema. Inicialmente, abordamos o conjunto de restrições impostas à equipa de desenvolvimento, englobando limitações orçamentais, temporais e relacionadas com a implementação da solução. Adicionalmente, destacamos os benefícios esperados do sistema, incluindo a capacidade de processamento em lote e a extensibilidade para incorporar novas ferramentas.

Seguidamente, procedemos à análise dos requisitos não funcionais apresentados no documento, selecionando aqueles com maior impacto na escolha da arquitetura do sistema a desenvolver, tais como a escalabilidade, a usabilidade e a segurança. Posteriormente, delimitamos o contexto do sistema, construindo uma decomposição funcional para compreender as fronteiras e capacidades das funcionalidades oferecidas.

De seguida, estabelecemos a arquitetura que sustentará o sistema, decidindo-a com base na análise dos requisitos não funcionais e na sua devida priorização, com especial atenção ao suporte a serviços externos e à modularidade.

Por último, descrevemos detalhes relativos à implementação do sistema. Começamos por descrever os componentes principais do sistema e a forma como se comunicam entre si, bem como a implementação de todas as funcionalidades previstas. Esta fase foca-se na tradução dos requisitos em soluções técnicas robustas que garantam a sustentabilidade e a eficiência do sistema.

2. Restrições

Aqui, apresentamos as restrições definidas no documento de requisitos e analisamos de que forma estas irão impactar a nossa solução.

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 primeira fase, uma versão preliminar do relatório técnico será entregue até o dia 18 de outubro de 2024. Este será apresentado aos stakeholders entre os dias 21 e 25 de outubro de 2024 para avaliação e feedback.
- As seguintes fases serão submetidas nas seguintes datas:
    - **Entrega da arquitetura da app:** 22 de novembro de 2024, incluindo a definição completa da arquitetura do sistema.
    - **Defesa da arquitetura:** 25 a 29 de novembro de 2024, durante a qual a equipa deverá justificar as escolhas técnicas e receber feedback detalhado.
    - **Entrega da implementação da app:** 17 de janeiro de 2025, incluindo a versão final do software com todas as funcionalidades 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:

- **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), 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 ataques comuns (e.g.,SQL injection, XSS) e políticas adequadas de gestão de dados dos utilizadores, conforme regulamentos de privacidade como 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 utiliza dores ativos ou da carga de processamento.

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.

Estas restrições exigem que a equipa de desenvolvimento esteja atenta a alguns detalhes, tais como: a necessidade de projetar uma arquitetura que suporte uma infraestrutura escalável na cloud. Além disso, a aplicação deve incluir camadas robustas de segurança integradas na arquitetura, garantindo proteção contra ameaças. É igualmente essencial que a arquitetura seja focada em tecnologias web responsivas e compatíveis com diversos dispositivos.

2.3. Restrições Orçamentais

Descrição: O desenvolvimento da aplicação deve respeitar restrições financeiras estabelecidas pelo cliente. A equipa de desenvolvimento deve planear o projeto 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

Descrição: O projeto 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 (e.g.: binarização, alteração de tamanho, remoção de fundo, etc.). 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.

  Estas restrições exigem que a equipa de desenvolvimento preste atenção a alguns aspetos, tais como: a necessidade de uma arquitetura que inclua um sistema de controlo de acesso com perfis distintos e limitações específicas para cada perfil. Além disso, a necessidade de suportar operações encadeadas e processamento em lote aponta para uma arquitetura modular, que permita a combinação de várias ferramentas e o processamento eficiente de grandes volumes de dados.

3. Requisitos Não Funcionais

A equipa de desenvolvimento decidiu focar-se nos requisitos não funcionais que considera ter maior impacto no sistema e, por consequência, influenciar a seleção da solução arquitetural. Esta escolha arquitetural tem como objetivo maximizar as características prioritárias da arquitetura.

Requisito #: RNF21	Tipo: Não funcional	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	

Requisito 1: Performance das ferramentas

Requisito #: RNF22	Tipo: Não funcional	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	

Requisito 2: Múltiplos processamentos em paralelo

Requisito #: RNF23	Tipo: Não funcional	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	

Requisito 3: Escalabilidade e elasticidade

Requisito #: RNF24	Tipo: Não funcional	Use cases #: —
Descrição	**A aplicação deve estar disponível 24 horas por dia, todos os dias.**	
Rationale	Garantir alta disponibilidade é crucial para utilizadores globais que dependem da aplicação para trabalho contínuo, independentemente da zona horária.	
Origem	*Engenheiro de Software*	
Fit criterion	A disponibilidade mínima deve ser de 99,9%, permitindo até 40 minutos de downtime por mês.	
Prioridade	*Should*	
Data	2024/10/03	

Requisito 4: Disponibilidade

Requisito #: RNF26	Tipo: Não funcional	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	

Requisito 5: Eficiência energética

Requisito #: RNF32	Tipo: Não funcional	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	

Requisito 6: Testabilidade

A equipa priorizou os requisitos que oferecem uma experiência de utilizador contínua e eficaz, mesmo sob carga elevada. O impacto é classificado em uso individual e coletivo, sendo que os requisitos de uso coletivo visam garantir uma experiência estável e eficiente para um grande número de utilizadores simultâneos.

3.1. Impacto dos Requisitos no Sistema

Para o uso individual, a ênfase está na performance das ferramentas de edição de imagem (requisito 21). A prioridade é que as ferramentas sejam processadas rapidamente, assegurando uma experiência de utilizador eficiente e responsiva. Este requisito é essencial para garantir que o utilizador não experimente atrasos durante operações básicas e avançadas, mantendo uma sensação de fluidez e usabilidade na aplicação. O impacto disso é que a arquitetura precisa ser otimizada para execução rápida de tarefas e minimizar o tempo de processamento por imagem.

No uso coletivo, a capacidade de processamento em massa (requisito 22) tem grande importância. A possibilidade de processar até 100 imagens ao mesmo tempo, sem quebras significativas de desempenho, reflete a necessidade de uma arquitetura capaz de suportar computação paralela eficiente. Esse requisito implica que a aplicação utilize técnicas de distribuição de carga e processamento simultâneo para garantir que múltiplos utilizadores possam executar operações em paralelo sem comprometer a performance.

A escalabilidade e elasticidade (requisito 23) são críticas para suportar o crescimento da base de utilizadores e picos de carga, mantendo o desempenho e controlando os custos operacionais. A arquitetura precisa de estar preparada para escalar horizontalmente e ajustar a alocação de recursos em tempo real, o que garante que a aplicação se mantenha eficiente mesmo com um aumento abrupto de utilizadores ou operações.

A disponibilidade contínua (requisito 24) é um fator que reforça a necessidade de manter a aplicação acessível 24/7. Para os utilizadores globais, isso significa que o sistema deve ser resistente a falhas e ter uma alta taxa de uptime. A infraestrutura deve ser robusta, com redundâncias que assegurem que o serviço continue operacional mesmo em casos de falhas técnicas.

Por último, a eficiência energética (requisito 26) destaca-se, especialmente em dispositivos móveis. Otimizar o uso de CPU e memória reduz o consumo de energia, prolongando a duração da bateria e melhorando a experiência do utilizador. Este impacto implica que a aplicação deve ser projetada para usar os recursos de forma inteligente, evitando sobrecarga de processamento.

4. Contexto e Âmbito do Sistema

O sistema proposto, denominado "*PictuRAS*", é uma solução inovadora para facilitar o processamento e a edição de imagens, voltada para diferentes tipos de utilizadores, desde amadores até profissionais de design gráfico.

O desenvolvimento do *PictuRAS* é motivado pela necessidade de superar as limitações das ferramentas de edição tradicionais, que muitas vezes requerem infraestrutura robusta ou conhecimentos técnicos avançados. A plataforma busca oferecer um ambiente acessível e flexível, permitindo a edição de imagens diretamente na nuvem, o que elimina a necessidade de recursos computacionais avançados por parte dos utilizadores.

O objetivo principal do *PictuRAS* é fornecer uma plataforma de fácil utilização, acessível a qualquer utilizador, independentemente de suas limitações de recursos ou experiência técnica. Este conceito promove a democratização do acesso a ferramentas de edição de imagens, possibilitando que o utilizador realize operações complexas com poucos cliques.

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

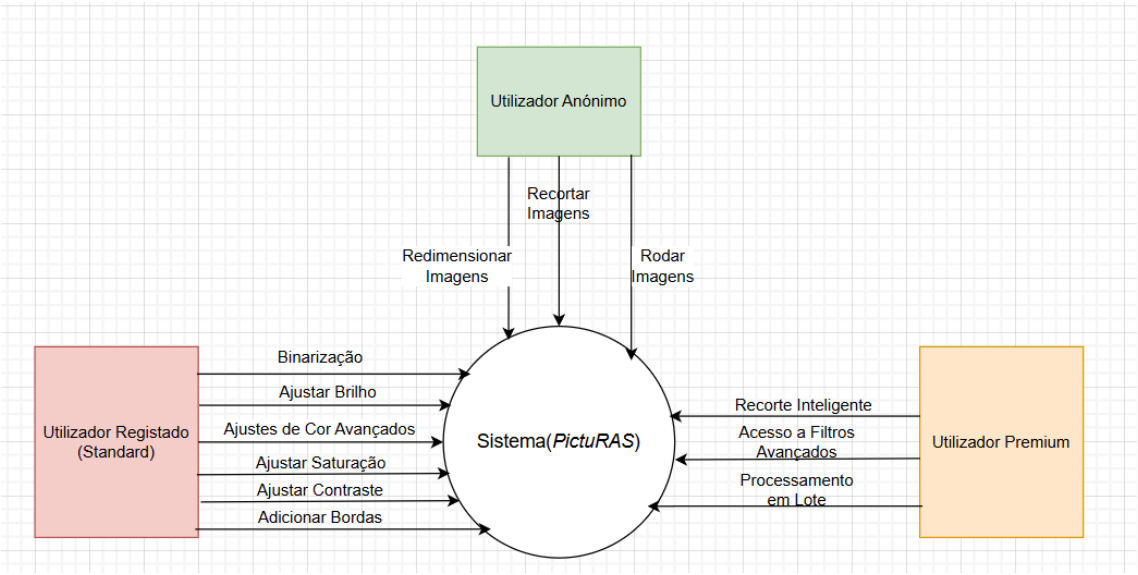

Figura X - Diagrama de contexto

O diagrama apresentado visa ilustrar as interações entre o sistema e os atores envolvidos. Com este diagrama, pretende-se definir de forma precisa os limites do sistema, especificando o que está incluído no âmbito do projeto e clarificando as responsabilidades e funcionalidades atribuídas ao sistema. Além disso, permite concluir que cada utilizador dispõe de funcionalidades distintas. Por exemplo, um utilizador anónimo possui acesso limitado às ferramentas básicas, enquanto um utilizador premium dispõe de todas as funcionalidades, incluindo processamento em lote e uso das ferramentas avançadas.

5. Estratégia da Solução

Começaremos por escolher a arquitetura com base na análise de arquiteturas já existentes. Em seguida, iremos definir os vários componentes da arquitetura selecionada, adaptando-a para responder às necessidades específicas deste projeto, e detalhando as tecnologias e decisões adotadas.

5.1. Padrão arquitetural

Nesta secção, procederemos à escolha do padrão arquitetural mais adequado, analisando algumas arquiteturas já existentes. Tendo em conta os requisitos definidos na secção anterior, avaliaremos a adequação de cada arquitetura ao caso específico em estudo.

5.1.1. Map Reduce

Requisito Não Funcional	***Rate***	**Justificação**
[24] A aplicação deve estar disponível 24 horas por dia, todos os dias	5/5	A arquitetura MapReduce possui tolerância a falhas robusta, com dados distribuídos e redundância que garantem a continuidade do processamento mesmo em caso de falhas de hardware. Isso contribui para uma alta disponibilidade e robustez, ideal para um sistema que exige operação contínua sem interrupções.
[23] 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.	4/5	O MapReduce é muito adequado para escalabilidade horizontal, uma vez que foi projetado para operar em *clusters* e adicionar nós para lidar com maiores volumes de dados. No entanto, essa escalabilidade não resolve a latência individual de cada tarefa. Portanto, é uma boa escolha para lidar com crescimento de volume e usuários, mas não elimina a limitação de latência.
[22] O sistema deve processar até 100 imagens ao mesmo tempo, sem quedas percetíveis no desempenho	3/5	Embora o MapReduce consiga distribuir o processamento entre vários nós, o sistema não foi otimizado para tarefas de baixa latência ou para um número de pequenas tarefas paralelas, como o processamento de imagens simultâneas em tempo real.
[26] A aplicação deve otimizar o uso de recursos computacionais do cliente (e.g., CPU, memória).	3/5	O MapReduce é relativamente eficiente em termos de uso de recursos em grandes processamentos de dados em lote. No entanto, ele não é tão eficaz para otimização de recursos em tarefas menores e de baixa latência, como o processamento individual de imagens em tempo real, o que pode levar a uso sub-ótimo de CPU e memória.
[32] O sistema deve ser projetado para facilitar a execução de testes	2/5	Testar uma aplicação baseada em MapReduce pode ser complexo devido ao seu processamento distribuído, que exige um ambiente e dados representativos da escala completa do sistema para garantir testes confiáveis.

[21] As ferramentas de edição de imagem devem ser processadas rapidamente.	2/5	MapReduce é ideal para o processamento em lote de grandes volumes de dados, mas apresenta limitações quando é necessário baixa latência e respostas rápidas para tarefas de menor porte, como edição de imagem em tempo real.

5.1.2. Micro-Serviços

Requisito Não Funcional	***Rate***	**Justificação**
[22] O sistema deve processar até 100 imagens ao mesmo tempo, sem quedas percetíveis no desempenho	5/5	Microsserviços são projetados para escalar horizontalmente e distribuir carga, tornando-os ideais para lidar com grandes volumes de requisições simultâneas. Ao escalar cada serviço de forma independente, o sistema pode facilmente processar várias imagens ao mesmo tempo, mantendo o desempenho.
[23] 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.	5/5	A arquitetura de microsserviços foi desenvolvida exatamente para este tipo de escalabilidade. Serviços independentes permitem aumentar ou diminuir a quantidade de instâncias de cada serviço conforme a demanda. Isso permite que o sistema suporte grandes volumes de tráfego sem afetar o desempenho geral.
[32] O sistema deve ser projetado para facilitar a execução de testes.	5/5	Microsserviços, sendo modulares e independentes, facilitam o desenvolvimento de testes unitários e de integração para cada serviço individualmente. Isso torna o processo de teste muito mais fácil e eficaz, já que cada serviço pode ser testado em isolamento, melhorando a deteção de falhas e facilitando a automação de testes.
[26] A aplicação deve otimizar o uso de recursos computacionais do cliente (e.g., CPU, memória).	4/5	Cada microserviço pode ser desenvolvido e ajustado de forma independente, facilitando a otimização do uso de CPU e memória para cada funcionalidade específica. Essa flexibilidade permite uma otimização eficiente; no entanto, o consumo de recursos pode aumentar se os serviços não forem projetados e monitorizados adequadamente.
[24] A aplicação deve estar disponivel 24 horas por dia, todos os dias	4/5	A arquitetura de microserviços aumenta a disponibilidade ao isolar falhas, garantindo que uma falha em um serviço específico não afete todo o sistema. No entanto, a maior complexidade de monitorização e o aumento no número de pontos de falha requerem maior atenção.
[21] As ferramentas de edição de imagem devem ser processadas rapidamente.	4/5	A arquitetura de microserviços possibilita a execução de operações em tempo real graças à independência e flexibilidade de cada serviço. No entanto, a latência pode aumentar devido à necessidade de comunicação entre os serviços e ao overhead de rede.

5.1.3. Publish-Subscribe

5.1.4. Peer-to-Peer

5.1.5. Client Server and N Tier

5.1.6. Layered

5.1.7. Batch-Sequential

5.1.8. Model-Centered

5.1.9. Pipe-and-Filter

5.2. Definição da Arquitetura

Após a análise de várias arquiteturas realizada anteriormente, a equipa de desenvolvimento avaliou que, para este tipo de sistema, a melhor arquitetura a adotar é o padrão de **microsserviços**. Esta arquitetura oferece grande escalabilidade, desempenho, resiliência (com falhas isoladas), facilidade de manutenção e atualização, bem como maior simplicidade nos testes. Além disso, permite o uso de tecnologias específicas para cada microserviço. A escolha foi feita por consenso geral.

5.2.1. Identificação de microsserviços

Através do estudo do projeto *PictuRAS*, foram definidos os seguintes microsserviços:

1. **Microserviço de Recorte de Imagens**
   - **Responsabilidades**: Realizar o recorte manual de imagens com base nas coordenadas fornecidas pelo utilizador.
   - **Entrada**: Imagem original e parâmetros de recorte
   - **Saída**: Imagem recortada.
   - **Tecnologias**: OpenCV, PIL (Python Imaging Library).
2. **Microserviço de Redimensionamento de Imagens**
   - **Responsabilidades**: Redimensionar imagens para dimensões específicas.
   - **Entrada**: Imagem original, nova largura e altura.
   - **Saída**: Imagem redimensionada.
   - **Tecnologias**: OpenCV, ImageMagick.
3. **Microserviço de Ajuste de Saturação**
   - **Responsabilidades**: Ajustar a saturação das cores de uma imagem.
   - **Entrada**: Imagem original e nível de saturação desejado.
   - **Saída**: Imagem com saturação ajustada.
   - **Tecnologias**: OpenCV, PIL.
4. **Microserviço de Ajuste de Brilho**
   - **Responsabilidades**: Modificar o brilho de uma imagem.
   - **Entrada**: Imagem original e nível de brilho desejado.
   - **Saída**: Imagem com brilho ajustado.

- **Tecnologias**: OpenCV, PIL.

5. **Microserviço de Ajuste de Contraste**
    - **Responsabilidades**: Ajustar o contraste de uma imagem.
    - **Entrada**: Imagem original e nível de contraste desejado.
    - **Saída**: Imagem com contraste ajustado.
    - **Tecnologias**: OpenCV, PIL.
6. **Microserviço de Binarização**
    - **Responsabilidades**: Converter a imagem em preto e branco utilizando um valor de *threshold* específico.
    - **Entrada**: Imagem original e valor de *threshold*.
    - **Saída**: Imagem binarizada.
    - **Tecnologias**: OpenCV.
7. **Microserviço de Rotação de Imagens**
    - **Responsabilidades**: Rodar imagem num ângulo especificado.
    - **Entrada**: Imagem original e ângulo de rotação.
    - **Saída**: Imagem rotacionada.
    - **Tecnologias**: OpenCV, ImageMagick.
8. **Microserviço de Recorte Inteligente**
    - **Responsabilidades**: Realizar um recorte automático com base no conteúdo da imagem.
    - **Entrada**: Imagem original.
    - **Saída**: Imagem recortada automaticamente.
    - **Tecnologias**: Modelos de IA, OpenCV.
9. **Microserviço de Ajuste Automático**
    - **Responsabilidades**: Ajustar automaticamente brilho, contraste e saturação com base no conteúdo da imagem.
    - **Entrada**: Imagem original.
    - **Saída**: Imagem ajustada automaticamente.
    - **Tecnologias**: Algoritmos de otimização de imagem.
10. **Microserviço de Remoção de Fundo**
    - **Responsabilidades**: Remover o fundo de uma imagem, deixando apenas o objeto principal.
    - **Entrada**: Imagem original.
    - **Saída**: Imagem com fundo removido.
    - **Tecnologias**: Modelos de segmentação de imagem, OpenCV.
11. **Microserviço de OCR (Reconhecimento Óptico de Caracteres)**
    - **Responsabilidades**: Extrair texto de imagens.
    - **Entrada**: Imagem original.
    - **Saída**: Texto extraído.
    - **Tecnologias**: Tesseract OCR, OpenCV.
12. **Microserviço de Reconhecimento de Objetos**
    - **Responsabilidades**: Identificar e etiquetar objetos numa imagem.
    - **Entrada**: Imagem original.
    - **Saída**: Lista de objetos identificados e as suas localizações.
    - **Tecnologias**: Modelos de deep learning (e.g., YOLO, TensorFlow).
13. **Microserviço de Contagem de Pessoas**
    - **Responsabilidades**: Contar o número de pessoas presentes numa imagem.
    - **Entrada**: Imagem original.
    - **Saída**: Número de pessoas identificadas.
    - **Tecnologias**: Modelos de deteção de pessoas, OpenCV.
14. **Microserviço de Adição de Bordas**

- **Responsabilidades**: Adicionar bordas coloridas às imagens.
- **Entrada**: Imagem original e cor da borda desejada.
- **Saída**: Imagem com borda adicionada.
- **Tecnologias**: OpenCV, PIL.

15. **Microserviço de Gestão de Projeto**
    - **Responsabilidades**: Centralizar e gerir as operações do projeto, integrando os vários microsserviços e facilitando a comunicação entre eles.
    - **Entrada**: Parâmetros e dados para integração dos microsserviços.
    - **Saída**: Respostas e coordenação de operações entre microsserviços.
    - **Tecnologias**: Ferramentas de orquestração de microsserviços, APIs de comunicação interna.

Decidimos definir um microserviço para cada funcionalidade do sistema, em vez de agrupar várias funcionalidades, de forma a evitar a sobrecarga de um microserviço no caso de agrupamento. Se as funcionalidades estivessem agrupadas no mesmo microserviço, uma funcionalidade poderia não conseguir ser utilizada devido à sobrecarga das funcionalidades do mesmo microserviço. Com a abordagem adotada, este problema não ocorre, embora haja um aumento na latência de comunicação entre os microserviços. No entanto, a equipa de desenvolvimento considerou que esta era a melhor solução a seguir.

5.3. Modelos de dados das Bases de Dados

A equipa de desenvolvimento decidiu que a maioria dos microserviços, por se dedicarem exclusivamente ao processamento de imagens, não necessita de uma base de dados integrada. No entanto, o microserviço de gestão de projetos requer uma base de dados, e haverá também uma base de dados separada, não associada a um microserviço específico, para armazenar os utilizadores.

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

5.3.1. Gestão de Projetos DB

Para a gestão de projetos, iremos utilizar uma base de dados ***PostgreSQL***, na qual teremos uma tabela para armazenar as informações relativas aos projetos.

Nome do campo	**Tipo**	**Descrição**
id	Integer	Identificação única do Projeto na base de dados.
nome	String	Nome do Projeto.
user_id	Integer	ID do utilizador responsável pelo projeto.
imagens	Array(Imagem)	Lista de imagens associadas ao projeto. Cada imagem é um objeto.
alteracoes	Array(String)	Histórico de alterações feitas no projeto ou nas imagens associadas.
data_criacao	String	Data em que o projeto foi criado (formato: YYYY-MM-DD).
last_change	String	Data e hora da última atualização do projeto (formato: YYYY--MM-DD HH:mm).

5.3.2. Users DB

Para a gestão de utilizadores, vamos utilizar uma base de dados ***PostgreSQL***, na qual será criada uma tabela para armazenar as informações relativas aos mesmos.

Nome do campo	Tipo	Descrição
id	Integer	Identificação única do usuário na base de dados.
nome	String	Nome do Usuário
email	String	Endereço de e-mail do usuário (único).
password	String	Senha criptografada do usuário.
alteracoes	Array(String)	Histórico de alterações feitas no projeto ou nas imagens associadas.
status	String	Status do usuário (ex.: "Anónimo", "Registado", "Premium").
fim_de_premium	String	Data do final do plano Premium caso o utilizador possua este estatuto (formato: YYYY-MM-DD).

5.4. Tecnologias para a implementação

Nesta secção, iremos descrever as tecnologias selecionadas pela equipa de desenvolvimento para a implementação do sistema.

5.4.1. Frontend

Para o desenvolvimento do frontend, utilizaremos a linguagem JavaScript com o apoio da biblioteca ***React*** para a componente interativa da interface do utilizador.

5.4.2. Microsserviços e API Gateway

Nas aplicações dos microsserviços, será utilizada a linguagem de programação ***JavaScript***, juntamente com o ambiente de execução ***Node.js*** e a framework ***Express***.

5.4.3. Bases de dados

A tecnologia a utilizar para a persistência de dados será o ***PostgreSQL***, um sistema de gestão de base de dados relacional. O ***PostgreSQL*** foi escolhido pela sua robustez, escalabilidade o que o torna adequado para a arquitetura de microsserviços do sistema.

5.5. Decisões organizacionais

A equipa de desenvolvimento é composta por 8 elementos, e decidiu-se que cada um ficará responsável por uma parte do código. A distribuição dos elementos pelos diferentes componentes do sistema será feita da seguinte forma:

1. **Responsáveis pelos Microsserviços:**

A equipa será dividida em 5 elementos, que ficarão responsáveis pelo desenvolvimento, manutenção e comunicação dos 15 microsserviços.

2. **Responsável pela Gestão de Utilizadores:**

Um elemento da equipa ficará responsável pelos microsserviços relacionados com a gestão de utilizadores, incluindo funcionalidades como login, registo de utilizadores, gestão de permissões e uso de projetos.

3. **Responsáveis pelo Frontend:**

Dois elementos da equipa serão responsáveis pelo desenvolvimento do frontend, criando as interfaces de utilizador e garantindo a integração com os microsserviços correspondentes.

6. Building Block View

Nesta secção, abordaremos os diversos componentes que compõem o sistema, assim como as interfaces fornecidas e requeridas de cada um, demonstrando a forma como comunicam entre si para satisfazer cada requisito funcional.

******** o que estamos a fazer agr no draw.io

7. Runtime View

Nesta secção, apresentamos o funcionamento da solução proposta através de diagramas de sequência dos use cases do sistema. Estes diagramas cobrem uma variedade de cenários, ilustrando as interações entre os atores e os diversos componentes do sistema, incluindo situações de exceção ou erro e a forma como o sistema as gere.

******** aula da próxima semana

8. Deployment View

Nesta secção, iremos apresentar dois diagramas de deployment do sistema *PictuRAS*. Primeiramente, apresentaremos um diagrama em que toda a aplicação (Microsserviços, API Gateway e aplicação cliente) é executada num único computador, representando o cenário de execução para efeitos de teste e desenvolvimento.

Em seguida, apresentaremos um diagrama que ilustra o deployment ideal do sistema, ou seja, como a aplicação seria instalada e executada em um ambiente de produção, garantindo alta disponibilidade, escalabilidade e desempenho.

8.1. Contexto de desenvolvimento e teste

8.2. Contexto de produção