**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 - 3ª fase

**Duarte Araújo**
A100750

**Francisco Ferreira**
PG55942

**Gonçalo Costa**
PG55944

**João Gomes**
PG55960

**Luís Barros**
PG55978

**Paulo Pinto**
PG55991

**Pedro Sousa**
PG55994

**Pedro Leiras**
PG55995

**Pedro Carvalho**
PG55997

# RAS

20 de janeiro de 2024

# Índice

# 1. Introdução e Objetivos

Este relatório apresenta o desenvolvimento da terceira fase do projeto *PictuRAS*, no âmbito da unidade curricular de Requisitos e Arquitetura de Software. Nesta fase, foi implementado o Minimum Viable Product (MVP) do sistema, seguindo a arquitetura definida nas etapas anteriores e garantindo a cobertura dos requisitos funcionais e não funcionais especificados.

O MVP do *PictuRAS* permite a gestão de projetos de edição de imagem, oferecendo funcionalidades essenciais, tais como a criação e listagem de projetos, o carregamento de imagens, a aplicação de ferramentas de edição e a exportação dos resultados. Além disso, a solução adotada baseia-se numa arquitetura de microsserviços, onde cada ferramenta de edição é disponibilizada como um serviço independente, permitindo escalabilidade e modularidade no sistema.

Durante o processo de implementação, a equipa organizou-se para desenvolver diferentes microsserviços de ferramentas, garantindo a integração do serviço picturas-watermark-tool-ms disponibilizado pela equipa docente. A implementação seguiu os padrões arquiteturais e de comunicação previamente definidos, assegurando a operabilidade entre os componentes.

Este relatório documenta a abordagem adotada na implementação, analisando o cumprimento dos requisitos e avaliando a cobertura funcional da solução. Além disso, são apresentadas reflexões críticas sobre eventuais alterações arquiteturais necessárias durante o desenvolvimento, bem como métricas e considerações que permitem uma análise aprofundada da solução implementada.

# 2. Implementação

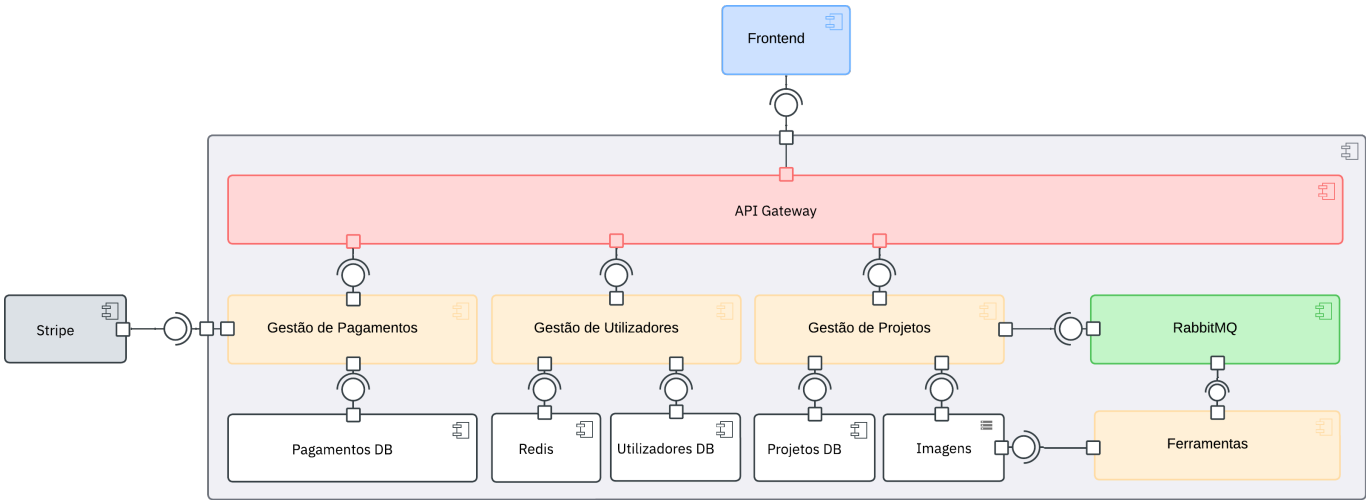


Figura 1: Diagrama de componentes final do *PictuRAS*

## 2.1. API Gateway

O API Gateway foi implementado com uma instância do Caddy, um *reverse proxy*, que apenas redireciona pedidos para microsserviços específicos dependendo do caminho do URL.

Neste caso, como quem irá servir um *websocket* será apenas o microsserviço dos projetos, podemos anular completamente o WS Gateway que estava previsto, fazendo com o que o API Gateway faça esse trabalho.

Este microsserviço não faz qualquer autenticação nem verificação de autorizações, e é responsabilidade dos microsserviços de baixo a fazerem.

```
http://localhost {
  handle /api/v1/users* {
    reverse_proxy http://users-ms:8010
  }

  handle /api/v1/projects* {
    reverse_proxy http://projects-ms:8000
  }

  handle /api/v1/images* {
    reverse_proxy http://projects-ms:8000
  }

  handle /api/v1/projects/*/ws {
    reverse_proxy http://projects-ms:8000 {
      transport http {
        versions h1 h2
      }
    }
  }

  handle * {
    reverse_proxy http://frontend
  }

  respond "Not found" 404
}
```

Listagem 1: Caddyfile - Configuração do Caddy

## 2.2. Microsserviço dos Utilizadores

O microsserviço dos utilizadores é responsável pela autenticação, criação e alteração de contas dos utilizadores.

### 2.2.1. Autenticação

Para a autenticação foi implementado um sistema de *refresh tokens* com *revoke lists*, recorrendo a um Redis a rodar num *container* separado.

Para evitar que todos os pedidos de outros microsserviços tenham de ser autenticados a partir deste microsserviço, são gerados dois JWTs: um *access token* e um *refresh token*. O *access token* tem um tempo de vida curto (15 minutos) e o *refresh token* tem um tempo de vida longo (1 semana). O *refresh token* também é adicionado a uma *allowed list* no Redis[1], para que possa ser revogado a qualquer momento, por exemplo, quando o utilizador troca de *password* ou faz *logout*.

[1] O Redis é utilizado para evitar que este microsserviço guarde estado e possa ser replicado facilmente.

```
{
  "sub": "c9fc3a04-a9c9-4f58-b55b-ff514a5ba6ee",
  "name": "chico",
  "email": "mail6@chicoferreira.com",
  "token_id": "aeeeeb67-f547-4d79-a682-c8af85ea28dd",
  "exp": 1737307708
}
```

Listagem 2: Exemplo do *payload* de um *access token*

```
{
  "sub": "c9fc3a04-a9c9-4f58-b55b-ff514a5ba6ee",
  "token_id": "aeeeeb67-f547-4d79-a682-c8af85ea28dd",
  "exp": 1737910230
}
```

Listagem 3: Exemplo do *payload* de um *refresh token*

```
CREATE TABLE IF NOT EXISTS users
(
    uuid       UUID PRIMARY KEY,
    name       VARCHAR(255)                          NOT NULL,
    email      VARCHAR(255)                          NOT NULL,
    password   VARCHAR(255)                          NOT NULL,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP NOT NULL
);

CREATE INDEX IF NOT EXISTS users_email_idx ON users (email);
```

Listagem 4: Schema da criação da base de dados do utilizador

Estes JWTs recorrem a chaves assimétricas para garantir a sua autenticidade, um par para *refresh tokens* e outro par para *access tokens*. As chaves privadas, que geram os JWTs, são apenas conhecidas por este microsserviço, enquanto as chaves públicas são conhecidas por todos os microsserviços que precisem de verificar a autenticidade dos JWTs. Estas chaves são guardadas e carregadas a partir de *Docker Secrets*.

Quando um utilizador se regista, o seu nome, email e *password* vão no *body* do pedido. A *password* é *hashed* com Argon2 e guardada na base de dados. Depois disso, o *refresh token* e o *access token* são criados e retornados na resposta em JSON que também incluí o UUID do utilizador criado.

Quando um utilizador faz login, o seu email e *password* são enviados no *body* do pedido. A *password* é verificada, sendo *hashed* com Argon2 e comparada com a que está na base de dados. Se a autenticação for bem-sucedida, um novo *access token* e *refresh token* são gerados e retornados na resposta em JSON, juntamente com o UUID do utilizador. Caso contrário, uma mensagem de erro apropriada é retornada.

### 2.2.2. Atualizar JWTs

Existe também um *endpoint* para fazer *refresh* de um *access token* expirado, que recebe o *refresh token* que estão nas cookies do utilizador e retorna um novo *access token* na resposta em JSON, juntamente com o UUID do utilizador.

### 2.2.3. Atualizar informação do utilizador

Foi também implementado um *endpoint* para alterar a *password* do utilizador, que quando for feito com sucesso, são gerados novos *refresh* e *access* tokens, e os antigos são invalidados. De notar que, o *access token* ainda será válido durante o seu tempo de vida, já que os outros microsserviços não têm como saber que esse *access token* se refere a um *refresh token* que já foi invalidado. Neste caso, quando o *access token* expirar, o utilizador tentará fazer *refresh* do seu *access token*, mas como o *refresh token* que será usado não estará mais na *allowed list* do Redis, não o conseguirá fazer, tendo necessidade de fazer login novamente noutros dispositivos.

### 2.2.4. Logout

O *logout* remove o *access* e o *refresh token* das cookies do utilizador, e remove o *refresh token* da lista de *refresh tokens* válidos no Redis.

### 2.2.5. Lista de *endpoints*

#### 2.2.5.1. `POST /api/v1/users/register`

**Exemplo de Pedido:**

```
{
    "name": "chico",
    "email": "mail@chicoferreira.com",
    "password": "12345678"
}
```

**Exemplo de Resposta:**

```
{
    "uuid": "baa9a612-4cfe-46ed-bea2-e840f3175df0",
    "name": "chico",
    "email": "mail@chicoferreira.com",
    "created_at": "2025-01-19T20:32:43.909171690Z",
    "access_token": "...",
    "refresh_token": "..."
}
```

Os *tokens* também vão em *headers* de **SET_COOKIE**, para que sejam alterados automaticamente no browser do cliente.

#### 2.2.5.2. `POST /api/v1/users/login`

**Exemplo de Pedido:**

```
{
    "email": "mail@chicoferreira.com",
    "password": "12345678"
}
```

**Exemplo de Resposta:**

```
{
    "access_token": "...",
    "refresh_token": "..."
}
```

Os *tokens* também vão em *headers* de **SET_COOKIE**, para que sejam alterados automaticamente no browser do cliente.

#### 2.2.5.3. `POST /api/v1/users/refresh`

**Exemplo de Pedido:**

Body vazio, lê `"refresh_token"` dos *cookies*.

**Exemplo de Resposta:**

```
{
    "access_token": "..."
}
```

O *token* também vai em *headers* de **SET_COOKIE**, para que seja alterado automaticamente no browser do cliente.

#### 2.2.5.4. `GET /api/v1/users/me`

**Exemplo de Pedido:**

Body vazio, lê `"access_token"` dos *cookies*.

**Exemplo de Resposta:**

```
{
    "uuid": "baa9a612-4cfe-46ed-bea2-e840f3175df0",
    "name": "chico",
    "email": "mail@chicoferreira.com",
    "created_at": "2025-01-19T20:32:43.909171690Z",
    "access_token": "...",
    "refresh_token": "..."
}
```

#### 2.2.5.5. `POST /api/v1/users/logout`

**Exemplo de Pedido:**

Body vazio, lê `"refresh_token"` dos *cookies*.

```
Cookie: refresh_token=refresh_token_value
```

**Exemplo de Resposta:**

```
{
    "uuid": "baa9a612-4cfe-46ed-bea2-e840f3175df0",
    "name": "chico",
    "email": "mail@chicoferreira.com",
    "created_at": "2025-01-19T20:32:43.909171Z"
}
```

#### 2.2.5.6. `POST /api/v1/users/changepassword`

**Exemplo de Pedido:**

```
{
    "current_password": "password123",
    "new_password": "newpassword456"
}
```

**Exemplo de Resposta:**

```
{
    "access_token": "...",
    "refresh_token": "..."
}
```

## 2.3. Projetos

O microsserviço dos projetos é responsável por gerir projetos, imagens de projetos, ferramentas aplicadas, versões de imagens e a *queue* de ferramentas para os microsserviços de ferramentas.

Este microsserviço tem várias rotinas como criar, ver e apagar projetos, adicionar ou alterar a lista de ferramentas, fazer upload ou remover imagens, começar o processo de aplicar ferramentas e ver as versões das imagens que são resultado do processo de aplicação de ferramentas.

### 2.3.1. Criação de um projeto

Quando um utilizador envia um pedido de criação de projeto, este microsserviço sabe as informações do utilizador porque requer que ele envie o *access token* gerado pelo microsserviço dos utilizadores. A autenticidade deste *access token* é verificado a partir da chave pública dos *Access tokens*. **Esta verificação é feita em todos os pedidos.**

```
CREATE TABLE IF NOT EXISTS projects
(
    id         UUID PRIMARY KEY,
    name       VARCHAR(255)                          NOT NULL,
    user_id    UUID                                  NOT NULL,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP NOT NULL
);
```

Listagem 5: *Schema* de criação da tabela dos projetos

### 2.3.2. Upload de imagens

A partir de um pedido *multipart*, o microsserviço guarda a imagem que lhe foi enviada num volume partilhado entre este e os microsserviços de aplicar ferramentas num caminho previsível (`/images/<projectid>/<imageid>`). Estas imagens podem ser descarregadas a partir de outro *endpoint* caso o utilizador seja dono do projeto a que elas pertencem.

```
CREATE TABLE IF NOT EXISTS images
(
    id         UUID PRIMARY KEY,
    name       VARCHAR(255) NOT NULL,
    project_id UUID         NOT NULL REFERENCES projects (id)
);
```

Listagem 6: *Schema* de criação da tabela das imagens

### 2.3.3. Adição de ferramentas

O utilizador pode também adicionar ou alterar as ferramentas do projeto, especificando o seu nome de `procedure` e os seus parâmetros. Estes parâmetros só serão verificados quando forem enviados para os microsserviços de ferramentas. Esta alteração **não faz** com que as ferramentas sejam aplicadas logo de seguida.

```
CREATE TABLE IF NOT EXISTS tools
(
    id         UUID PRIMARY KEY,
    project_id UUID         NOT NULL REFERENCES projects (id),
    position   INTEGER      NOT NULL,
    procedure  VARCHAR(255) NOT NULL,
    parameters JSONB        NOT NULL
);
```

Listagem 7: *Schema* de criação da tabela das ferramentas

### 2.3.4. Aplicação de ferramentas

Ao chamar o *endpoint* de aplicar ferramentas, o utilizador pode especificar se quer aplicar a todas as imagens ou apenas a algumas imagens (funcionalidade necessária para o *preview* de ferramentas no site).

Após isso, o microsserviço calcula as ferramentas que serão necessárias para cada imagem (no caso de encadeamento) e monta os pedidos no formato recomendado pelos docentes e envia para a *queue* do RabbitMQ criada para essa transformação. Aí, os microsserviços que são compatíveis com essa ferramenta, serão consumidores dessa *queue* e então o processo de realmente aplicar a ferramenta se despoletará. Os microsserviços de ferramentas serão responsáveis por colocar a imagem resultado no caminho de output que lhes foi passado, que tem o padrão de `/images/<projectid>/output/<originalimageid>/<imageversionid>`.

Quando o microsserviço das ferramentas retorna o resultado de sucesso para a *queue* de resultado do RabbitMQ, o microsserviço dos projetos é consumidor desse, portanto lê e guarda o resultado na base de dados, notifica clientes relevantes ligados via *web socket* da nova imagem, e verifica se é necessário que mais ferramentas sejam aplicadas naquela imagem resultado (no caso de encadeamento).

```sql
CREATE TABLE IF NOT EXISTS image_versions
(
    id                UUID PRIMARY KEY,
    project_id         UUID                                    NOT NULL REFERENCES
projects (id),
    original_image_id UUID                                    NOT NULL REFERENCES
images (id),
    tool_id            UUID                                   NOT NULL REFERENCES
tools (id),
    text_result       TEXT, -- OCR results, if the tool is OCR, or any textual output
    created_at        TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP NOT NULL
);
```

Listagem 8: *Schema* de criação da tabela das versões de imagens

Como as ferramentas que ainda precisam de ser aplicadas a certa imagem não são enviadas para o RabbitMQ, estas são guardadas em memória, associando-as ao `messageId` que vai no pedido para o RabbitMQ. Ora isto faz com que este microsserviço não seja replicável atualmente, mas futuramente podemos fazer com que esta informação esteja guardada, por exemplo, numa instância de Redis fazendo com que o microsserviço seja facilmente escalável.

#### 2.3.4.1. Websocket para resultados imediatos

O utilizador pode conectar-se via *web socket* para obter resultados instântaneos da aplicação de ferramentas. Quando uma ferramenta é concluída e o microsserviço dos projetos recebe esse resultado, também notifica todos os clientes ligados que sejam relevantes ao projeto da imagem.

```json
{
  "id": "123e4567-e89b-12d3-a456-426614174005",
  "original_image_id": "123e4567-e89b-12d3-a456-426614174002",
  "project_id": "123e4567-e89b-12d3-a456-426614174000",
  "tool_id": "123e4567-e89b-12d3-a456-426614174003",
  "text_result": null,
  "created_at": "2023-10-01T12:34:56Z",
  "url": "..."
}
```

Listagem 9: Exemplo de *MESSAGE* do *web socket*

#### 2.3.4.2. Descarregar resultados em *batch*

Também é possível descarregar imagens resultantes de aplicação de ferramentas, acedendo a um *endpoint* e especificando a que ferramenta se referem (para que seja possível descarregar passos intermédios). O microsserviço monta um ficheiro zipado com todas as imagens e envia para o cliente.

### 2.3.5. Lista de Endpoints

#### 2.3.5.1. `POST /api/v1/projects`

**Exemplo de Pedido:**

```
{
  "name": "My Project"
}
```

**Exemplo de Resposta:**

```
[
    {
        "id": "123e4567-e89b-12d3-a456-426614174000",
        "name": "My Project",
        "user_id": "123e4567-e89b-12d3-a456-426614174001",
        "created_at": "2023-10-01T12:34:56Z",
        "updated_at": "2023-10-01T12:34:56Z"
    }
]
```

#### 2.3.5.2. `GET /api/v1/projects`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de Resposta:**

```
[
    {
        "id": "123e4567-e89b-12d3-a456-426614174000",
        "name": "My Project",
        "user_id": "123e4567-e89b-12d3-a456-426614174001",
        "created_at": "2023-10-01T12:34:56Z",
        "updated_at": "2023-10-01T12:34:56Z"
    }
]
```

#### 2.3.5.3. `GET /api/v1/projects/{project_id}`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de Resposta:**

```
{
    "id": "123e4567-e89b-12d3-a456-426614174000",
    "name": "My Project",
    "user_id": "123e4567-e89b-12d3-a456-426614174001",
    "created_at": "2023-10-01T12:34:56Z",
    "updated_at": "2023-10-01T12:34:56Z"
}
```

#### 2.3.5.4. `DELETE /api/v1/projects/{project_id}`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de Resposta:**

```
{
    "id": "123e4567-e89b-12d3-a456-426614174000",
    "name": "My Project",
    "user_id": "123e4567-e89b-12d3-a456-426614174001",
    "created_at": "2023-10-01T12:34:56Z",
    "updated_at": "2023-10-01T12:34:56Z"
}
```

#### 2.3.5.5. `POST /api/v1/projects/{project_id}/images`

**Exemplo de Pedido:**

Body multipart com a imagem, “access_token” nos cookies.

**Exemplo de Resposta:**

```
[
    {
        "id": "123e4567-e89b-12d3-a456-426614174002",
        "name": "image.jpg",
        "project_id": "123e4567-e89b-12d3-a456-426614174000"
    }
]
```

#### 2.3.5.6. `GET /api/v1/projects/{project_id}/images`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de Resposta:**

```
[
    {
        "id": "123e4567-e89b-12d3-a456-426614174002",
        "name": "image.jpg",
        "project_id": "123e4567-e89b-12d3-a456-426614174000"
    }
]
```

#### 2.3.5.7. `GET /api/v1/projects/{project_id}/images/{image_id}`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de Resposta:**

Bytes da imagem no body.

#### 2.3.5.8. `DELETE /api/v1/projects/{project_id}/images/{image_id}`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de Resposta:**

```
{
    "id": "123e4567-e89b-12d3-a456-426614174002",
    "name": "image.jpg",
    "project_id": "123e4567-e89b-12d3-a456-426614174000"
}
```

### 2.3.5.9. `GET /api/v1/projects/{project_id}/tools`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de Resposta:**

```
[
    {
        "id": "123e4567-e89b-12d3-a456-426614174003",
        "project_id": "123e4567-e89b-12d3-a456-426614174000",
        "position": 1,
        "procedure": "blur",
        "parameters": {
            "radius": 10
        }
    }
]
```

### 2.3.5.10. `POST /api/v1/projects/{project_id}/tools`

**Exemplo de Pedido:**

“access_token” nos cookies.

```
{
    "procedure": "blur",
    "parameters": {
        "radius": 10
    }
}
```

**Exemplo de Resposta:**

```
{
    "id": "123e4567-e89b-12d3-a456-426614174003",
    "project_id": "123e4567-e89b-12d3-a456-426614174000",
    "position": 1,
    "procedure": "blur",
    "parameters": {
        "radius": 10
    }
}
```

#### 2.3.5.11. `POST /api/v1/projects/{project_id}/tools/apply`

**Exemplo de Pedido:**

“access_token” nos cookies.

```
{
    "filter_images": ["123e4567-e89b-12d3-a456-426614174002"]
}
```

**Exemplo de Resposta:**

```
{
    "image_ids": ["123e4567-e89b-12d3-a456-426614174002"],
    "message": "Hook to websocket to get realtime results"
}
```

#### 2.3.5.12. `PUT /api/v1/projects/{project_id}/tools`

**Exemplo de Pedido:**

“access_token” nos cookies.

```
[
    {
        "procedure": "blur",
        "parameters": {
            "radius": 10
        }
    },
    {
        "procedure": "rotate",
        "parameters": {
            "angle": 90
        }
    }
]
```

**Exemplo de Resposta:**

```
[
    {
        "id": "123e4567-e89b-12d3-a456-426614174003",
        "project_id": "123e4567-e89b-12d3-a456-426614174000",
        "position": 1,
        "procedure": "blur",
        "parameters": {
            "radius": 10
        }
    },
    {
        "id": "123e4567-e89b-12d3-a456-426614174004",
        "project_id": "123e4567-e89b-12d3-a456-426614174000",
        "position": 2,
        "procedure": "rotate",
        "parameters": {
            "angle": 90
        }
    }
]
```


#### 2.3.5.13. `GET /api/v1/projects/{project_id}/tools/images`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de Resposta:**


```
[
    {
        "id": "123e4567-e89b-12d3-a456-426614174005",
        "original_image_id": "123e4567-e89b-12d3-a456-426614174002",
        "project_id": "123e4567-e89b-12d3-a456-426614174000",
        "tool_id": "123e4567-e89b-12d3-a456-426614174003",
        "text_result": null,
        "created_at": "2023-10-01T12:34:56Z",
        "url": "https://f.primecog.com/api/v1/projects/123e4567-e89b-12d3-a456-
426614174000/tools/images/123e4567-e89b-12d3-a456-426614174005"
    }
]
```


#### 2.3.5.14. `GET /api/v1/projects/{project_id}/tools/images/{img_version_id}`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de Resposta:**

Retorna os bytes da imagem

#### 2.3.5.15. `GET /api/v1/projects/{project_id}/tools/imageszip`

**Exemplo de Pedido:**

“access_token” nos cookies.

```
{
    "tool_id": "123e4567-e89b-12d3-a456-426614174003"
}
```

**Exemplo de Resposta:**

Retorna um arquivo ZIP com as imagens

#### 2.3.5.16. `WEBSOCKET /api/v1/projects/{project_id}/ws`

**Exemplo de Pedido:**

Body vazio, “access_token” nos cookies.

**Exemplo de mensagens recebidas:**

```
{
    "id": "123e4567-e89b-12d3-a456-426614174005",
    "original_image_id": "123e4567-e89b-12d3-a456-426614174002",
    "project_id": "123e4567-e89b-12d3-a456-426614174000",
    "tool_id": "123e4567-e89b-12d3-a456-426614174003",
    "text_result": null,
    "created_at": "2023-10-01T12:34:56Z",
    "url": "https://f.primecog.com/api/v1/projects/123e4567-e89b-12d3-a456-
426614174000/tools/images/123e4567-e89b-12d3-a456-426614174005"
}
```

## 2.4. Subscrições

O microsserviço de subscrições trata de lidar com o processo de subscrição do utilizador e do pagamento associado, através de uma conexão ao serviço Stripe.

### 2.4.1. Conexão ao Stripe

A ligação do microsserviço com o Stripe depende exclusivamente do uso de “API keys” que servem para autenticar os pedidos feitos à API do Stripe e garantir a autenticidade dos dados enviados pelo Stripe. A “stripe secret key” permite validar os requests do nosso sistema, enquanto o “webhook signing secret” serve o propósito de confirmar a origem dos events emitidos pelo Stripe. Estas chaves e asseguram o bom funcionamento do microsserviço das subscrições, sendo elas definidas como “secrets” no docker-compose.yml, para evitar possíveis quebras de segurança.

### 2.4.2. Criação de uma checkout session

Para a criação de uma nova subscrição, é feita uma verificação do cookie “access_token” presente no request feito ao endpoint especificado para tal, que dita se o usuário está autorizado a efetuar esta ação e ainda é extraído o user_id desse mesmo cookie. Isto é executado em resposta a todos os pedidos do utilizador. De seguida faz-se uma chamada à API do Stripe para que seja criada uma “checkout session”, com o modo subscrição, que aceite pagamento com cartão e ainda redirecione o user para outras páginas em caso de sucesso ou de falha.

```
session = stripe.checkout.Session.create(
            payment_method_types=['card'],
            mode='subscription',
            line_items=[{
                'price': PRICE_OBJECT,
                'quantity': 1,
            }],
            success_url= SUCCESS_URL,
            cancel_url= CANCEL_URL,
        )
```

Listagem 10: Chamada à API do Stripe

No seguimento desta API call, é criado um registo de subscrição a ser adicionado à base de dados do microsserviço, assinalado com o status de "inativa" e com os ids de sessão e subscrição relevantes para a futura confirmação do pagamento. É então retornado um objeto JSON com o url da página de pagamento para a qual o utilizador será redirecionado.

```
class Subscription(Base):
    __tablename__ = 'subscriptions'
    id = Column(Integer, primary_key=True)
    user_id = Column(UUID, nullable=False)
    price = Column(Float, nullable=False)
    start_date = Column(DateTime, nullable=False)
    end_date = Column(DateTime, nullable=False)
    status = Column(String(20), nullable=False, default='inactive')
    session_id = Column(String, nullable=False)
    stripe_subscription_id = Column(String)
```

Listagem 11: Model da tabela de Subscrições

### 2.4.3. Confirmação do pagamento

Assim que o utilizador tiver introduzido os seus dados e confirmado a sua ação, este será redirecionado para o SUCCESS_URL (ou CANCEL_URL) e o Stripe enviará, para um endpoint acordado, um event que permitirá a alteração da subscrição do cliente de inativa para ativa. Consoante a informação enviada pelo Stripe, podemos determinar se a subscrição foi criada, ou renovada com sucesso, e recuperar o registo correspondente na base de dados, usando o id da subscrição. Após as confirmações e alterações, o utilizador é notificado pelo serviço através de um post a um endpoint configurável, contendo um JSON com a role atual do user e a data de validade, no caso de ser premium.

### 2.4.4. Detalhes de subscrição

Este microsserviço compromete-se ainda a fornecer dados relativos à subscrição de um utilizador, assim com a role atual dele no sistema, em resposta a pedidos dos próprios.

## 2.4.5. Lista de Endpoints

### 2.4.5.1. `POST /api/v1/subscriptions/create-checkout-session`

“access token” nos cookies

**Exemplo de mensagens recebidas:**

```
[
    {
        "url": "https://checkout-session/"
    }
]
```

### 2.4.5.2. `POST /api/v1/subscriptions/webhook`

**Exemplo de pedido:**

```
{
  "id": "in_1MtHbELkdIwHu7ixl4OzzPMv",
  "object": "invoice",
  "account_country": "US",
  "account_name": "Stripe Docs",
  ...
  "billing_reason": "manual",
  ...
  "paid": false,
  "paid_out_of_band": false,
  "payment_intent": null,
  "payment_settings": {
    "default_mandate": null,
    "payment_method_options": null,
    "payment_method_types": null
  },
  "subscription": null,
  "subtotal": 0,
  "subtotal_excluding_tax": 0,
  "tax": null,
  "test_clock": null,
  "total": 0,
  "total_discount_amounts": [],
  "total_excluding_tax": 0,
  "total_tax_amounts": [],
  "transfer_data": null,
  "webhooks_delivered_at": 1680644467
}
```

**Exemplo de mensagens recebidas:**

```
[
    {
      "status": 'success',
    }
]
```

#### 2.4.5.3. `GET /api/v1/subscriptions/subscription-details`

“access token” nos cookies

**Exemplo de mensagens recebidas:**

```
[
    {
      "sub_id": sub.id,
      "sub_price": sub.price,
      "start_date": sub.start_date,
      "end_date": sub.end_date,
      "status": sub.status,
      "user_role": 'premium' if sub.status == 'active' else 'default'
    }
]
```

## 2.5. Microsserviços de Ferramentas

Estes microsserviços são responsáveis por consumir tarefas da fila de pedidos, aplicar as ferramentas, guardar a imagem resultante, e enviar informações do resultado de volta.

### 2.5.1. Microsserviço das Ferramentas Gerais

Este microsserviço recorre à biblioteca photon-rs para efeitos simples e ao Leptess para a ferramenta de OCR.

A lista de efeitos que este microsserviço é responsável é a seguinte:
- “crop”: Alterar as dimensões da imagem;
- “scale”: Esticar a imagem para uma dimensão maior ou menor;
- “addBorder”: Adicionar uma borda sólida com tamanho personalizável;
- “adjustBrightness”: Ajustar brilho da imagem;
- “adjustContrast”: Ajustar contraste da imagem;
- “rotate”: Rodar imagem;
- “blur”: Desforcar imagem;
- “ocr”: Reconhecer caracteres na imagem;
- “binarize”: Binarização da imagem;
- “grayscale”: Colocar a imagem a preto e branco.

Dentro desta lista, é possível determinar quais ferramentas uma instância deste microsserviço pode executar, fazendo com que seja possível escalar ferramentas individualmente, criando apenas instâncias. Na inicialização, o programa regista-se como consumidor nas *queues* das ferramentas que foram especificadas e então começa a receber pedidos apenas dessas ferramentas.

Como este microsserviço não tem de inicializar nenhum programa externo, nem carregar grandes quantidades de dados, como modelos de inferência, etc., não haverá nenhuma penalidade no uso de recursos a que todos estes efeitos estejam associados a um só microsserviço.

No entanto, no nosso *deployment* atual apenas temos uma instância deste microsserviço a servir todas as transformações.

O formato das mensagens que este recebe e envia seguem o formato enviado pelos docentes.

### 2.5.2. Microsserviço de marca de água dos docentes

Como adotamos o formato das mensagens dos docentes, integrar o microsserviço de marca de água dos docentes foi trivial, apenas necessitando de o conectar à instância do RabbitMQ.

### 2.5.3. Microsserviço de remover fundo

Este microserviço foi feito em *python* recorrendo à biblioteca rembg para remover o fundo de uma imagem. Utilizando modelos de deep learning para entregar um resultado o mais perto do desejável possível.

O facto de a ferramenta estar num microserviço próprio garante que a utilização de recursos é feita exclusivamente por este programa, o que é ideal já que utiliza um modelo bastante pesado.

## 2.6. Frontend

Para o desenvolvimento do *frontend*, recorremos à framework *Vue.js* para a componente interativa da interface de utilizador e de *Tailwind CSS* para componente visual, ambas tecnologias já familiarizadas pelos alunos e de fácil integração.

Optamos também por utilizar *shadcn* pois proporciona uma coleção de componentes pré-desenhados e acessíveis, alinhados com as boas práticas de design e acessibilidade. Estes componentes integram-se perfeitamente com *Tailwind CSS*, permitindo-nos criar interfaces modernas e consistentes com maior rapidez e menor esforço.

A comunicação entre o *frontend* e o *backend* é realizada através do *gateway*, que atua como um ponto intermediário essencial, mediando todas as interações entre os clientes e a API. Esta arquitetura foi desenhada com base numa abordagem centrada em *APIs REST*, proporcionando uma interface padronizada e consistente para a troca de informações entre as diferentes camadas do sistema.

O gateway não só facilita a abstração da complexidade do backend, como também assegura a implementação de mecanismos essenciais, como autenticação, autorização e validação de pedidos, garantindo segurança e integridade na comunicação.

## 2.6.1. Resultados

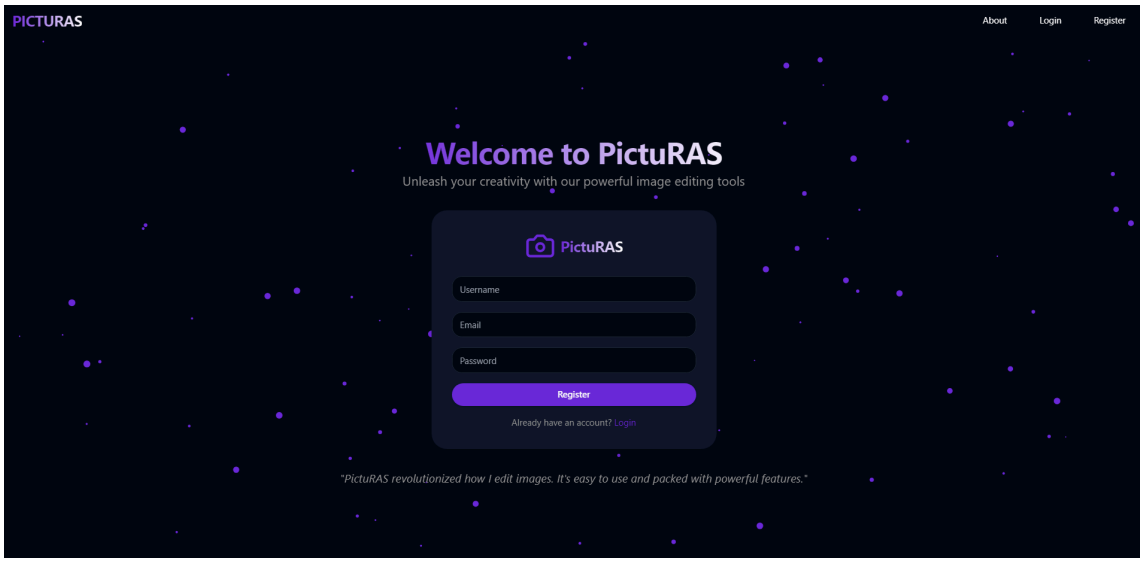


Figura 2: Landing Page

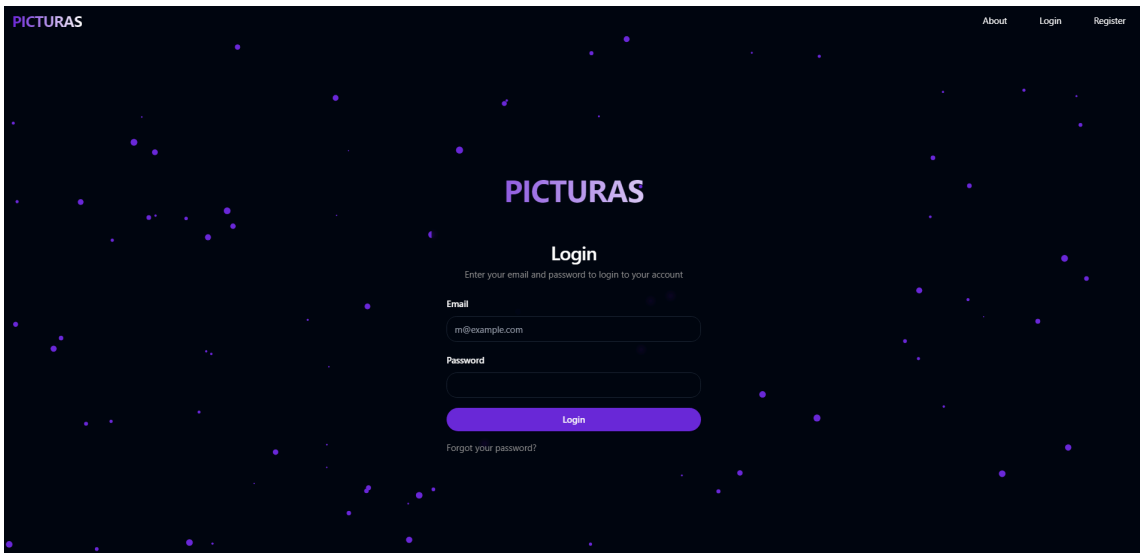


Figura 3: Login Page

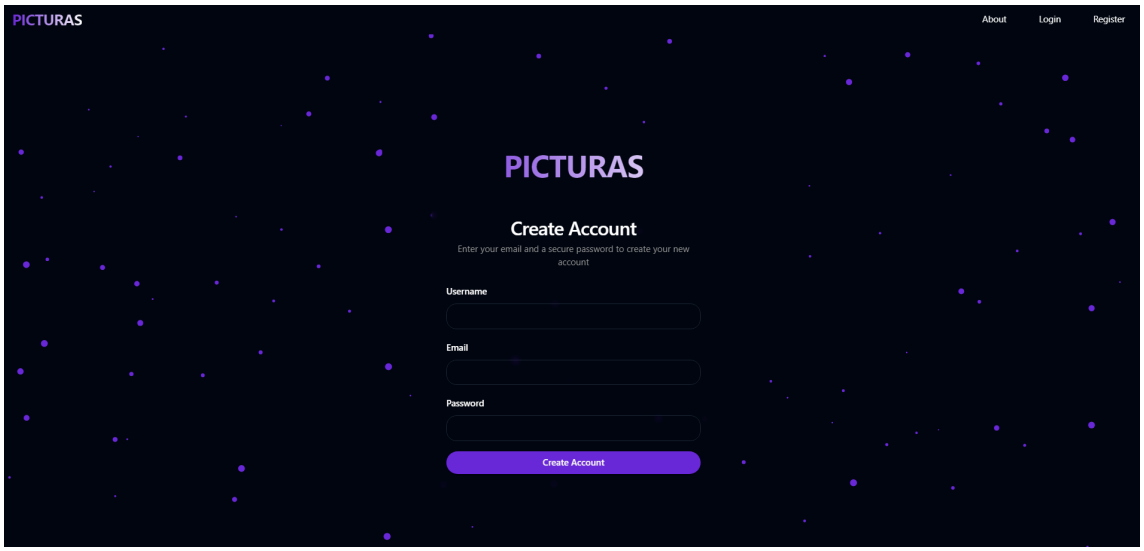


Figura 4: Register Page

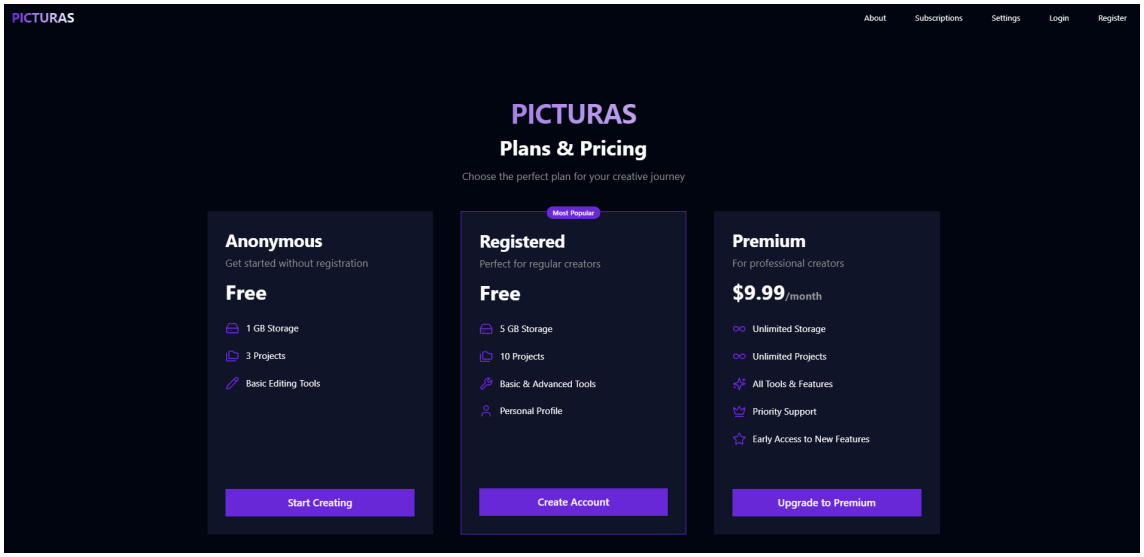


Figura 5: Subscription Plans Page

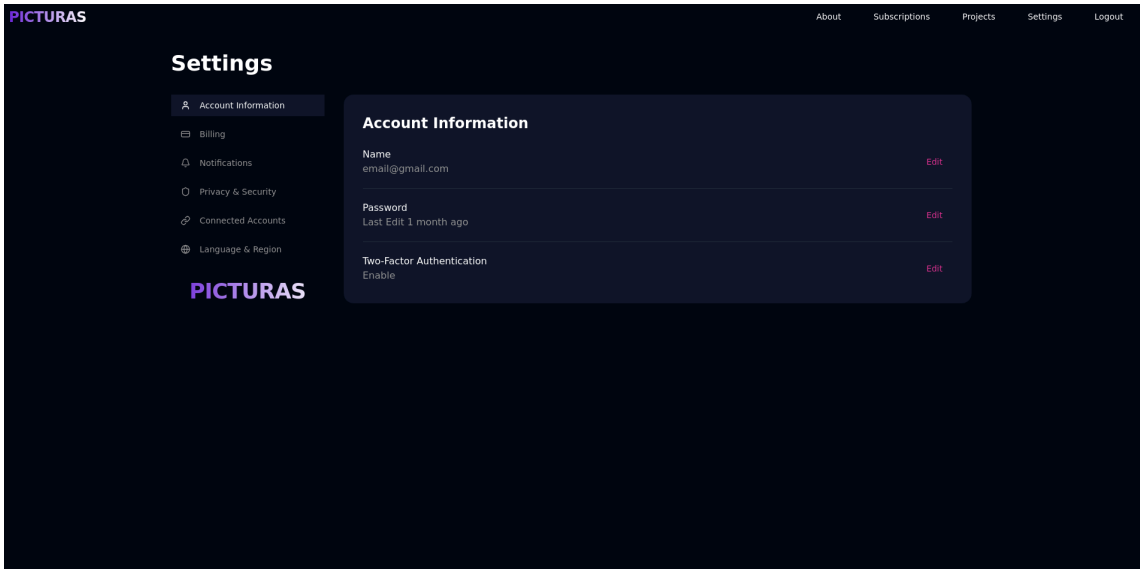


Figura 6: Settings Page Example

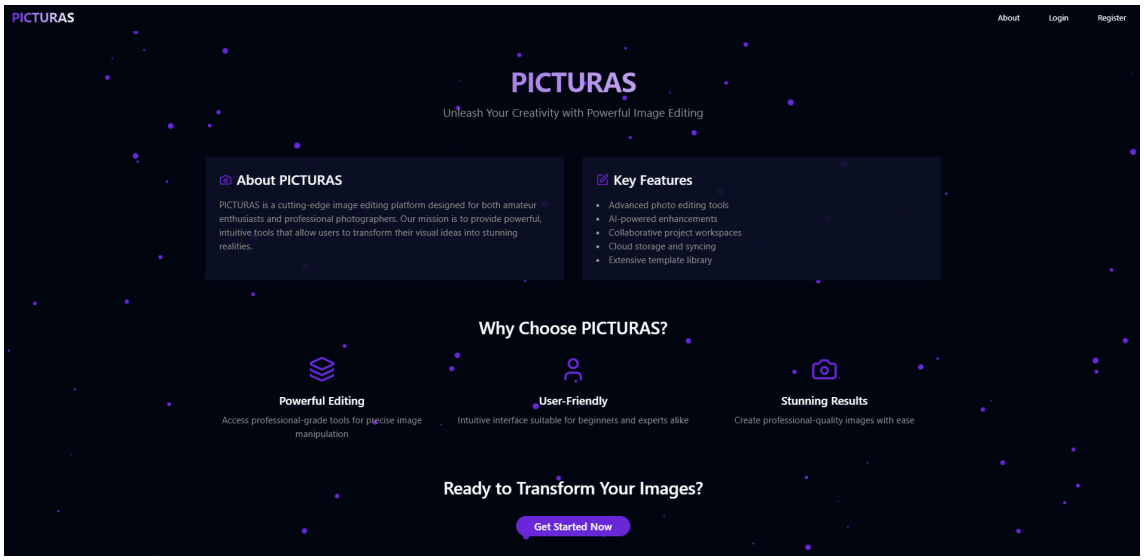


Figura 7: About Page

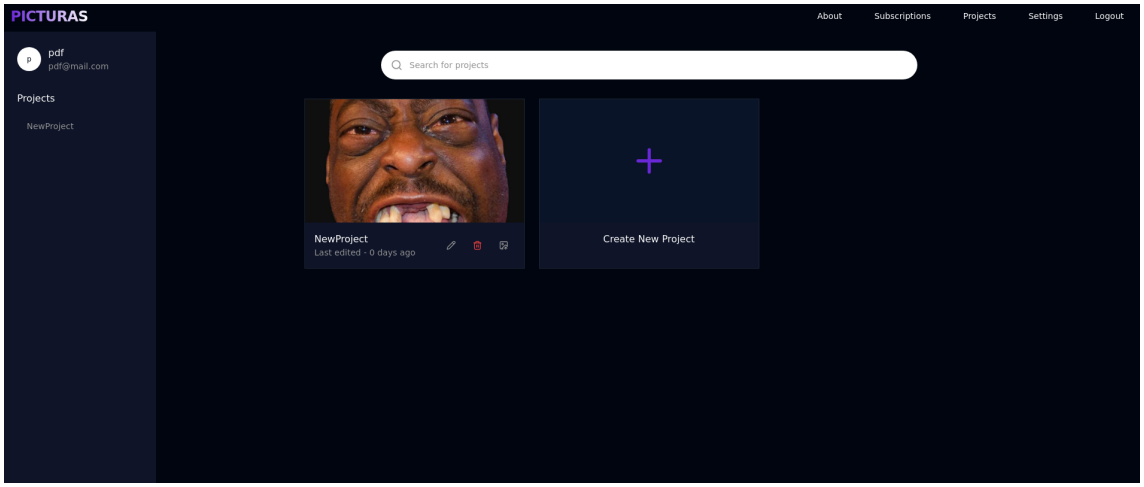


Figura 8: Projects Page

## 2.7. Deployment final

Com todos os microsserviços descritos, podemos passar para o *deployment*. Como todos os microsserviços estão containerizados com um respetivo `docker-compose.yml`, esta parte tornou-se incrivelmente fácil, bastando que haja um `docker-compose.yml` que agrega todos os microsserviços num só, com uma *network* privada entre eles, um disco partilhado para microsserviços que precisem de aceder/escrever em imagens, e segredos partilhados como chaves públicas e privadas de *tokens*.

```yaml
include:
  - tools-ms/docker-compose.yml
  - projects-ms/docker-compose.yml
  - users-ms/docker-compose.yml
  - bg-remover-tool-ms/docker-compose.yml
  - watermark-tool-ms/docker-compose.yml
  - api-gateway/docker-compose.yml
  - frontend/docker-compose.yml

networks:
  picturas-network:

volumes:
  images:

secrets:
  refresh_token_private_key:
    file: ./keys/refresh.key
  refresh_token_public_key:
    file: ./keys/refresh.key.pub
  access_token_private_key:
    file: ./keys/access.key
  access_token_public_key:
    file: ./keys/access.key.pub
```

Listagem 12: `docker-compose.yml` final

# 3. Características funcionais implementadas

| Requisito | Descrição | Implementação |
|---|---|---|
| RF1 | O utilizador autentica-se utilizando as suas credenciais. | Implementado |
| RF2 | O utilizador anónimo regista-se. | Não implementado |
| RF3<br>RF4 | O utilizador escolhe o plano de subscrição (Gratuito ou *Premium*) | Implementado no backend |
| RF6 | O utilizador cria um projeto. | Implementado |
| RF7 | O utilizador lista os seus projetos. | Implementado |
| RF8 | O utilizador acede à área de edição de um projeto. | Implementado |
| RF9 | O utilizador carrega imagens para um projeto. | Implementado |
| RF11 | O utilizador adiciona uma ferramenta de edição ao projeto. | Implementado |
| RF12 | O utilizador desencadeia o processamento de um projeto. | Implementado |
| RF13 | O utilizador transfere o resultado de um projeto para o dispositivo local. | Implementado no backend |
| RF18 | O utilizador registado acede às suas informações de perfil | Implementado |
| RF21 | O utilizador *premium* cancela a sua subscrição *premium* | Implementado no backend |
| RF22 | O utilizador registado termina a sua sessão. | Implementado |
| RF25 | O utilizador recorta manualmente imagens. | Implementado |
| RF26 | O utilizador escala imagens para dimensões específicas. | Implementado |
| RF27 | O utilizador adiciona borda a imagens. | Implementado |
| RF29 | O utilizador ajusta o brilho a imagens. | Implementado |
| RF30 | O utilizador ajusta o contraste a imagens. | Implementado |
| RF32 | O utilizador roda imagens. | Implementado |
| RF35 | O utilizador remove o fundo da imagem, mantendo apenas o objeto principal. | Implementado |

Para além disso mencionamos também as implementações para os requisitos funcionais não prioritários:

| Requisito | Descrição | Implementação |
|---|---|---|
| RF15 | O utilizador altera a ordem das ferramentas de um projeto | Implementado no backend |
| RF16 | O utilizador altera os parâmetros das ferramentas. | Implementado |
| RF17 | O utilizador cancela o processamento de um projeto durante a sua execução. | Não implementado |
| RF19 | O utilizador edita o seu perfil. | Implementado |
| RF36 | O utilizador extrai texto de imagens | Implementado no backend |
| RF37 | O utilizador aplica um algoritmo de reconhecimento de objetos em imagens. | Não implementado |
| RF38 | O utilizador aplicar um algoritmo de contagem de pessoas em imagens. | Não implementado |

Tabela 1: Lista tabular dos requisitos funcionais não prioritários - Implementação

# 4. Características não funcionais implementadas

No desenvolvimento da aplicação web de edição de imagens ***PictuRAS***, seguimos rigorosamente todas as restrições técnicas, de negócio e temporais estabelecidas no documento de requisitos. Além disso, dedicamo-nos para que fosse realizada corretamente a implementação de todos os requisitos não funcionais, reconhecendo o seu papel essencial na arquitetura de software da aplicação sendo estes fundamentais para garantir a qualidade, escalabilidade, segurança e usabilidade do sistema, influenciando diretamente a experiência do utilizador e a robustez da solução.

A seguir, detalhamos as principais características não funcionais que foram implementadas:

| Requisito | Descrição | Implementação |
|---|---|---|
| RNF5 | A página de um projeto distingue visualmente entre as ferramentas básicas e avançadas. | Não implementado |
| RNF7 | O utilizador cria um projeto implicitamente ao arrastar ficheiros para o dashboard da aplicação. | Não implementado |
| RNF8 | O utilizador carrega imagens arquivadas num único ficheiro *.zip*. | Implementado |
| RNF9 | O utilizador é mantido informado acerca do estado de processamento de um projeto em tempo real. | Implementado |
| RNF14 | A visualização de imagens grandes ou pequenas é auxiliada pelo utilitário zoom. | Implementado |
| RNF15 | A visualização de imagens permite que se navegue em todas as duas direções arrastando o rato. | Implementado |
| RNF18 | A aplicação é compatível com diferentes plataformas e browsers, incluindo de dispositivos móveis e *desktop*. | Implementado |
| RNF19 | A aplicação fornece feedback visual claro e imediato ao utilizador em caso de erro ou falha durante um procedimento demorado. | Não implementado |
| RNF22 | O sistema deve processar até 100 imagens ao mesmo tempo, sem quebras percetíveis no desempenho | Implementado |
| RNF23 | A aplicação deve ser capaz de escalar horizontalmente de forma elástica para suportar o aumento de utilizadores e volume de processamentos, mantendo o desempenho mas também os custos controlados. | Implementado[2] |
| RNF25 | A aplicação deve ser integrável com outras plataformas e serviços de terceiros. | Implementado via API pública |
| RNF28 | A aplicação deve ser facilmente estendida com novas ferramentas de edição. | Implementado |

[2]Apenas referindo o pequeno problema do microsserviços de projeto que guarda estado, mas que pode ser facilmente movido para uma instância Redis.

| Requisito | Descrição | Implementação |
|---|---|---|
| RNF32 | O sistema deve ser projetado para facilitar a execução de testes. | Implementado |
| RNF33 | A aplicação deve realizar backups automáticos dos dados e imagens dos utilizadores. | Não implementado |

# 5. Conclusão

Após a realização deste projeto, o grupo sentiu que conseguiu consolidar de forma significativa o conhecimento adquirido ao longo das aulas, tanto práticas como teóricas, ao longo do semestre.

Consideramos que atingimos os objetivos propostos, desenvolvendo uma solução que cumpre os requisitos definidos para o MVP (*Minimum Viable Product*) e apresenta uma arquitetura sólida e eficiente, capaz de ser escalada eficientemente, processamento em tempo real, distribuição de mensagens e *Load Balancing*, monitorização e, acima de tudo, bastante resiliente. O progresso do desenvolvimento decorreu de forma alinhada com o planeado, garantindo uma aplicação funcional, bem estruturada e coerente com as boas práticas aprendidas.

Esta experiência permitiu-nos não só aplicar os conceitos abordados em aula, como também aprimorar as nossas competências técnicas e de trabalho em equipa, resultando num produto que reflete o esforço e dedicação investidos ao longo do semestre.

No entanto, reconhecemos que há espaço para possíveis melhorias e complementos. Por exemplo, algumas áreas do *frontend* não foram completamente finalizadas ou poderiam beneficiar de um acabamento mais refinado. Além disso, o microsserviço de subscrições não foi totalmente integrado à aplicação, o que representa uma oportunidade para aprimoramento.

Independentemente destes pontos, consideramos que entregamos um projeto sólido, com bases bem estruturadas e alinhadas, proporcionando um excelente ponto de partida para futuras expansões e melhorias. Esta experiência reforça o potencial do sistema e destaca o valor do trabalho realizado pelo grupo.