Uma imagem com texto, Tipo de letra, captura de ecrã, Gráficos

Descrição gerada automaticamente

Relatório de Projeto

Aplicativo de receitas culinárias “SideChef”

**Desenvolvimento de aplicativos para dispositivos móveis**

**Grupo 10:**

David Postolea

117738

Hugo Bessa 113783

Águeda, 15 de março de 2024

Licenciatura em Tecnologias de Informação 2º Ano – 2º Semestre

Agradecimentos

Este projeto representa o culminar de uma longa jornada e é fruto do esforço individual de ambos. Reconhecemos que não teríamos alcançado este feito sem o apoio e a contribuição inestimável das pessoas que nos rodeiam.

Gostaríamos de expressar um agradecimento especial ao Professor Gonçalo Marques, pela sua orientação perspicaz e dedicação incansável ao longo deste projeto de dispositivo móvel. Seu conhecimento especializado e orientação foram essenciais para moldar nossa abordagem e alcançar nossos objetivos.Sua sabedoria e competência na supervisão foram fundamentais para a concretização deste projeto.

Índice

Contente

[1. Introdução 7](#_Toc161416718)

[1.1. Visão geral do sistema 7](#_Toc161416719)

[1.2. Cliente 7](#_Toc161416720)

[1.3. Metas 7](#_Toc161416721)

[dois. Metodologia 7](#_Toc161416722)

[2.1. Metodologia SDLC usada 7](#_Toc161416723)

[2.2. Tempo gasto 7](#_Toc161416724)

[3. Modelo de Requisitos 8](#_Toc161416725)

[3.1. Requisitos 8](#_Toc161416726)

[3.1.1. Requisitos funcionais 8](#_Toc161416727)

[3.1.2. requisitos não Funcionais 8](#_Toc161416728)

[3.2. Modelo de caso de uso 8](#_Toc161416729)

[3.2.1. Visão geral 8](#_Toc161416730)

[3.2.2. Atores 8](#_Toc161416731)

[3.2.3. Casos de uso 8](#_Toc161416732)

[4. A aplicação 8](#_Toc161416733)

[4.1. Base de dados 8](#_Toc161416734)

[4.1.1. Visão geral 8](#_Toc161416735)

[4.1.2. Diagrama de classes 8](#_Toc161416736)

[4.1.3. Esquema de pronto-socorro 8](#_Toc161416737)

[4.1.4. Roteiro de criação 8](#_Toc161416738)

[4.2. API 9](#_Toc161416739)

[4.3. API REST 9](#_Toc161416740)

[4.4. Aplicativo 9](#_Toc161416741)

[4.4.1. Processo de implantação 9](#_Toc161416742)

[4.4.2. Projeto 9](#_Toc161416743)

[4.4.3. Atividades (páginas de inscrição) 9](#_Toc161416744)

[5. Conclusão 9](#_Toc161416745)

[6. Bibliografia 9](#_Toc161416746)

Índice de tabela

Índice de figura

# Introdução

## Visão geral do sistema

A gastronomia contemporânea atravessa uma revolução digital, impulsionada pela crescente integração da tecnologia no nosso quotidiano. As aplicações móveis desempenham um papel crucial nesta transformação, facilitando o acesso a um vasto leque de receitas, dicas de cozinha e ferramentas práticas que tornam a experiência na cozinha mais agradável e inspiradora. Neste sentido, este relatório propõe analisar e explorar em profundidade a aplicação SideChef, uma plataforma móvel desenhada para satisfazer as necessidades e exigências dos entusiastas da culinária de todo o mundo.

SideChef é mais do que um simples aplicativo de receitas, é um ecossistema culinário completo que combina uma vasta coleção de receitas de alta qualidade com ferramentas intuitivas e recursos personalizados. Ao entrar no aplicativo, o usuário fica imerso em um universo gastronômico, onde pode explorar as mais diversas receitas de todo o mundo, encontrar inspiração para novas criações e aprender técnicas culinárias por meio de instruções detalhadas.

Entre as funcionalidades essenciais do aplicativo SideChef estão o cadastro e login do usuário, permitindo uma experiência personalizada e contínua. A funcionalidade de pesquisa avançada permite que os usuários encontrem nomes de receitas específicas, enquanto a visualização detalhada dos ingredientes e da preparação fornece orientação passo a passo para cada prato. A receita favorita facilita aos usuários a organização e o acesso às suas receitas favoritas.

## Cliente

Ao desenvolver a aplicação SideChef é fundamental compreender os utilizadores que interagem com este sistema culinário digital. Esses usuários são representados por entusiastas da culinária, desde cozinheiros amadores até chefs profissionais, que recorrem ao aplicativo em busca de inspiração, aprendizado e organização de suas experiências gastronômicas. A aplicação procura oferecer uma interação intuitiva e cativante, respondendo às necessidades dos utilizadores, desde a disponibilização de uma vasta seleção de receitas de qualidade até à oferta de ferramentas intuitivas de organização e acesso rápido a informações sobre pratos.

## Objetivos

Os objetivos da aplicação SideChef centram-se em proporcionar uma experiência culinária completa e enriquecedora aos seus utilizadores. Esses objetivos vão desde oferecer inspiração culinária por meio de uma ampla seleção de receitas, até facilitar o processo de aprendizagem por meio de instruções detalhadas e vídeos explicativos. Além disso, o aplicativo busca disponibilizar recursos intuitivos que facilitem a navegação e a personalização da experiência do usuário. Pretende também incentivar a interactividade e a partilha entre os utilizadores, criando uma comunidade culinária onde se possam inspirar e partilhar as suas experiências gastronómicas.

# Metodologia

## Metodologia SDLC

Na implementação da aplicação SideChef optámos por seguir a metodologia SDLC, nomeadamente o modelo Waterfall. Este modelo caracteriza-se por uma abordagem sequencial, onde cada fase de desenvolvimento – como análise, design, implementação, testes e manutenção – é realizada de forma linear e sequencial, sem sobreposição significativa entre as etapas. Dessa forma, cada fase é concluída antes de passar para a próxima, proporcionando uma estrutura clara e definida para o processo de desenvolvimento de software. A escolha deste modelo baseou-se nas características específicas do projeto e nas necessidades identificadas na fase inicial de planeamento.

## Tempo gasto

10-03 = 7h

17-03 = 9h

24-03 = 22h

31-03 = 9h

08-04 = 3h

14-04 = 3h

21-04 = 4h

28-04 = 13h

# Modelo de Requisitos

## Requisitos

### Requisitos funcionais

|  |  |  |  |
| --- | --- | --- | --- |
| Não. | Nome | Descrição | Prioridade |
| 1 | Cadastro e login | Permita que os usuários se registrem e façam login no aplicativo SideChef. | Alto |
| 2 | Pesquisa avançada de receitas | Fornece funcionalidade de pesquisa avançada para que os usuários encontrem receitas específicas. | Alto |
| 3 | Visualização detalhada da receita | Apresentar informações detalhadas sobre os ingredientes, instruções de preparo e nutrição das receitas. | Média |
| 4 | Adicionar aos favoritos | Permita que os usuários adicionem receitas favoritas a uma lista de favoritos. | Média |

### requisitos não Funcionais

|  |  |  |  |
| --- | --- | --- | --- |
| Não. | Nome | Descrição | Prioridade |
| 1 | Usabilidade | O aplicativo deve ser fácil de usar e intuitivo para garantir uma boa experiência ao usuário. | Alto |
| dois | Desempenho | Garanta que o aplicativo responda rapidamente e sem atrasos, mesmo em situações de alta carga. | Alto |
| 3 | Segurança | Proteja os dados do usuário e garanta a privacidade das informações pessoais. | Alto |
| 4 | Compatibilidade | Certifique-se de que o aplicativo seja compatível com uma variedade de dispositivos e sistemas operacionais. | Média |
| 5 | Confiabilidade | Certifique-se de que a aplicação esteja estável e não apresente falhas ou erros frequentes. | Alto |
| 6 | Eficiência | Utilize efetivamente os recursos do dispositivo para garantir um desempenho otimizado. | Média |
| 7 | Capacidade de manutenção | Facilite a manutenção e atualizações de aplicativos para resolver problemas e adicionar novos recursos. | Média |
| 8 | Escalabilidade | Certifique-se de que o aplicativo seja capaz de lidar com um aumento no número de usuários e dados. | Alto |

## Modelo de caso de uso

### Visão geral

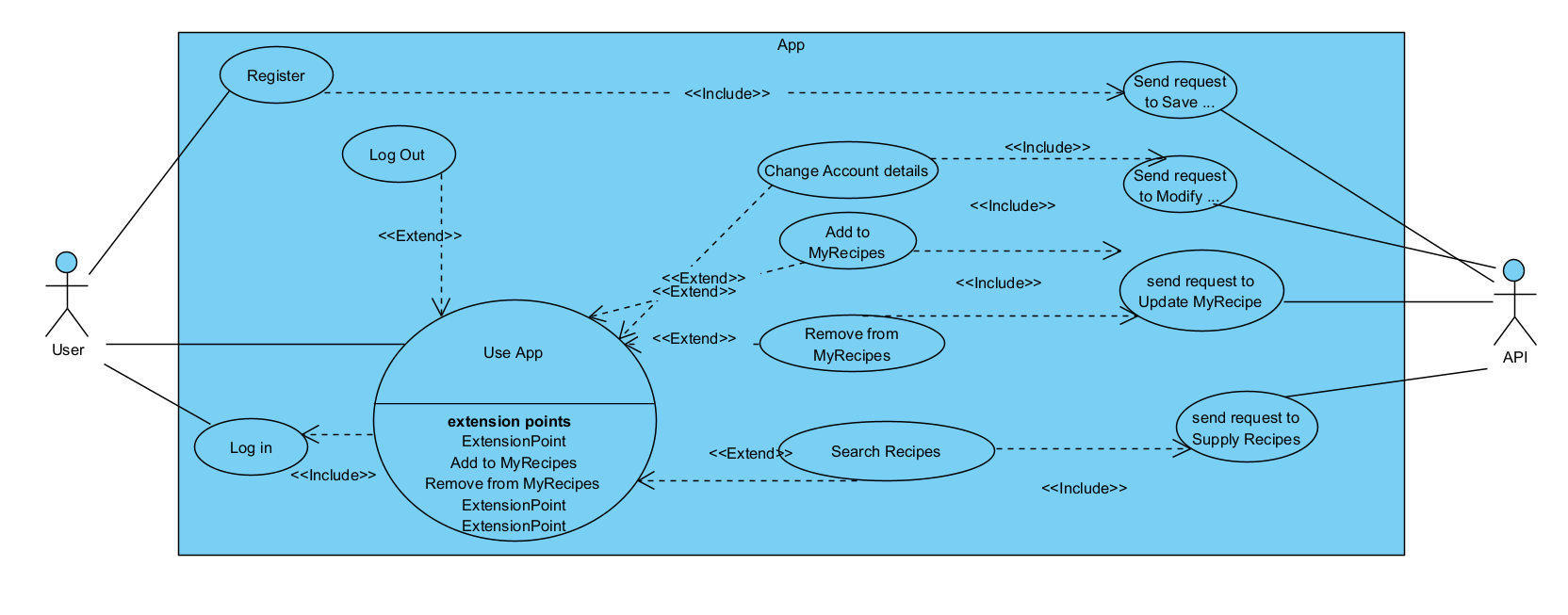
Neste relatório, será apresentado um diagrama de casos de uso para o SideChef. Este diagrama visa oferecer uma representação visual das interações entre os usuários e o sistema, destacando os principais atores envolvidos e os cenários de uso suportados. Ao examinar este diagrama, será possível compreender de forma clara e concisa como os usuários interagem com o sistema e quais funcionalidades estão disponíveis para eles. O objetivo deste documento é fornecer uma visão abrangente do funcionamento do sistema, concentrando-se nas perspectivas dos usuários e nos diferentes papéis que desempenham durante suas interações com o sistema.

### Atores

O ator "Cliente" representa o utilizador final que interage diretamente com a aplicação SideChef. Este utilizador pode aceder à aplicação através de uma interface de utilizador, seja uma aplicação web ou móvel. O "Cliente" é responsável por utilizar as funcionalidades oferecidas pela aplicação SideChef para atingir os seus objetivos culinários, como encontrar receitas e seguir instruções de culinária passo a passo.

O ator "API" representa a interface de programação de aplicações (API) fornecida pela aplicação SideChef, especificamente uma REST API (Representational State Transfer). Esta API permite receber pedidos dos utilizadores, processá-los na base de dados e enviar as respostas de volta aos utilizadores. Ela atua como intermediária entre o utilizador e os recursos do sistema, facilitando a comunicação e a troca de dados de forma eficiente e segura.

### Casos de uso



# A aplicação

## Base de dados

### Visão geral

A base de dados da aplicação SideChef desenhada em postgreSQL desempenha um papel fundamental no armazenamento e gestão dos dados que alimentam as funcionalidades do sistema. Concebida para lidar com uma variedade de informações, desde receitas e ingredientes até perfis de utilizadores. A base de dados é consistida em 3 Tabelas, a “Users” é responsável por armazenar tudo que esta relacionado com a conta do utilizador, a tabela “Recipe” é responsável por armazenar tudo relacionado com as receitas que são disponibilizadas para o utilizador interagir, e a tabela “SavedRecipes” está relacionada com as receitas salvas pelos utilizadores para que possam ser armazenadas para o utilizador ver mais tarde.

Além disso, a base de dados suporta operações complexas, como consultas avançadas para pesquisas detalhadas de receitas, transações para garantir a consistência dos dados e backups regulares para proteção contra perda de informação. Desta forma, os utilizadores podem confiar na aplicação SideChef para fornecer informações precisas e atualizadas sempre que necessário.

A base de dados também desempenha um papel importante na segurança dos dados, implementando medidas robustas de proteção, como encriptação das passwords usando o pgcrypto e controlo de acesso, para garantir a confidencialidade e a integridade das informações dos utilizadores.

### Esquema Entidade-Relação

Uma imagem com texto, captura de ecrã, diagrama, Retângulo

Descrição gerada automaticamente

### Script

CREATE TABLE Users(

id SERIAL PRIMARY KEY,

name VARCHAR(30) NOT NULL,

email VARCHAR(40) UNIQUE NOT NULL,

username VARCHAR(20) UNIQUE NOT NULL,

password VARCHAR(300) NOT NULL

);

CREATE TABLE Recipe(

id SERIAL PRIMARY KEY,

name VARCHAR(30) NOT NULL,

preparation VARCHAR(100) NOT NULL,

prepTime TIME NOT NULL,

type VARCHAR(15) NOT NULL,

picture VARCHAR(100) NOT NULL,

ingredients VARCHAR(200) NOT NULL

);

CREATE TABLE SavedRecipe(

id SERIAL PRIMARY KEY,

name VARCHAR(30) NOT NULL,

preparation VARCHAR(100) NOT NULL,

prepTime TIME NOT NULL,

type VARCHAR(15) NOT NULL,

picture VARCHAR(100) NOT NULL,

ingredients VARCHAR(200) NOT NULL,

idUser INT NOT NULL,

idRec INT NOT NULL,

CONSTRAINT fk\_recipeSaved FOREIGN KEY (idRec) REFERENCES Recipe(id),

CONSTRAINT fk\_user FOREIGN KEY (idUser) REFERENCES Users(id)

);

## POSTMAN

No desenvolvimento da aplicação SideChef, utilizamos o Postman para testar e validar a nossa REST API. Esta ferramenta desempenha um papel crucial, permitindo-me testar todas as rotas e endpoints da API de forma eficiente.

Com o Postman, foi possível enviar uma variedade de solicitações, como GET, POST, PUT e DELETE, para interagir com os recursos disponíveis na API. Por exemplo, posso enviar solicitações GET para recuperar informações sobre receitas, POST para adicionar novas receitas à base de dados, PUT para atualizar informações existentes e DELETE para remover recursos específicos, entre outras ações.

Além disso, o Postman oferece recursos avançados, como a capacidade de criar coleções de solicitações para testes automatizados e gerar documentação da API para facilitar a compreensão e o uso por parte de outros membros da equipe.

Graças ao Postman, foi possível garantir a qualidade e a confiabilidade da nossa API, testando todos os cenários possíveis e identificando rapidamente quaisquer problemas ou falhas que possam surgir durante o desenvolvimento ou após a implementação.

## API REST

<https://sidechef-api.vercel.app/>  
https://sidechef-api.vercel.app/doc/

A REST API (Representational State Transfer Application Programming Interface) desempenha um papel fundamental na arquitetura e funcionamento da aplicação SideChef. Esta interface de programação permite uma comunicação eficiente e segura entre o cliente, a aplicação SideChef e a base de dados subjacente. Hospedada no Vercel, uma plataforma de hospedagem moderna e escalável, a nossa REST API garante uma disponibilidade constante e um desempenho otimizado.

Através da REST API, os utilizadores da aplicação SideChef podem enviar pedidos para realizar diversas ações a partir do link https://sidechef-api.vercel.app/, como pesquisar receitas, salvar receitas favoritas, criar contas, obter utilizadores, entre outras. Estes pedidos são então processados pela API, que os encaminha para a base de dados correspondente, onde são executadas as operações necessárias.

Após o processamento na base de dados, a REST API retorna as respostas pertinentes aos utilizadores, fornecendo informações atualizadas, confirmações de ações realizadas e outros dados relevantes. Esta troca de dados ocorre de forma eficiente, graças ao protocolo HTTP subjacente, que permite a transferência de informações de forma rápida e segura entre o cliente e o servidor. A nossa REST API é constituída por 2 ficheiros principais (index.py e bd.py).

A documentação em relação aos vários endpoints da api podem-se encontrar aqui:  
https://sidechef-api.vercel.app/doc/

* ***Index.py***

import os

from datetime import datetime, timedelta

from functools import wraps

import jwt

import psycopg2

from flask import Flask, jsonify, request

import db

app = Flask(\_\_name\_\_)

app.debug = True

app.config['SECRET\_KEY'] = os.getenv('SECRET\_KEY', 'mysecretkey')

NOT\_FOUND\_CODE = 401

OK\_CODE = 200

SUCCESS\_CODE = 201

NO\_CONTENT\_CODE = 204

BAD\_REQUEST\_CODE = 400

UNAUTHORIZED\_CODE = 401

FORBIDDEN\_CODE = 403

NOT\_FOUND = 404

SERVER\_ERROR = 500

@app.route('/', methods = ["GET"])

def home():

return "Welcome to API!"

@app.route('/login', methods=['POST'])

def login():

data = request.get\_json()

if "username" not in data or "password" not in data:

return jsonify({"error": "invalid parameters"}), BAD\_REQUEST\_CODE

user = db.login(data['username'], data["password"])

if user is None:

return jsonify({"error": "Check credentials"}), NOT\_FOUND\_CODE

token = jwt.encode(

{'user\_id': user['id'], 'exp': datetime.utcnow() + timedelta(minutes=40)}, app.config['SECRET\_KEY'], 'HS256')

user["token"] = token.decode('UTF-8')

#user["token"] = token

return jsonify(user), OK\_CODE

@app.route("/register", methods=['POST'])

def register():

data = request.get\_json()

if "name" not in data or "email" not in data or "username" not in data or "password" not in data:

return jsonify({"error": "invalid parameters"}), BAD\_REQUEST\_CODE

if (db.user\_exists(data)):

return jsonify({"error": "user already exists"}), BAD\_REQUEST\_CODE

user = db.add\_user(data)

return jsonify(user), SUCCESS\_CODE

def auth\_required(f):

@wraps(f)

def decorated(\*args, \*\*kwargs):

if "Authorization" not in request.headers:

return jsonify({"error": "Token not provided"}), FORBIDDEN\_CODE

token = request.headers['Authorization']

# Remove Bearer from token

token = token.split(' ')[1]

try:

data = jwt.decode(token, app.config['SECRET\_KEY'], algorithms=['HS256'])

except jwt.ExpiredSignatureError:

return jsonify({"error": "Token expirado", "expired": True}), UNAUTHORIZED\_CODE

except jwt.InvalidTokenError:

return jsonify({"error": "Token inválido"}), FORBIDDEN\_CODE

request.user = db.get\_user(data['user\_id'])

return f(\*args, \*\*kwargs)

return decorated

@app.route("/changeUser/<int:id\_user>", methods=['PUT'])

@auth\_required

def update\_user(id\_user):

data = request.get\_json()

if "name" not in data or "email" not in data or "username" not in data or "password" not in data:

return jsonify({"error": "invalid parameters"}), BAD\_REQUEST\_CODE

if (db.user\_exists(data)):

return jsonify({"error": "user already exists"}), BAD\_REQUEST\_CODE

user = db.change\_user(id\_user, data)

return jsonify(user), SUCCESS\_CODE

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

@app.route("/getUser/<string:username>", methods=['GET'])

@auth\_required

def get\_user(username):

user = db.get\_user(username)

if user is None:

return jsonify({"Error": "No user found"}), NOT\_FOUND\_CODE

return jsonify(user), OK\_CODE

@app.route("/getRecipe/<int:id\_recipe>", methods=['GET'])

@auth\_required

def get\_recipe(id\_recipe):

recipe = db.getRecipe(id\_recipe)

if recipe is None:

return jsonify({"Error": "No Recipe found"}), NOT\_FOUND\_CODE

return jsonify(recipe), OK\_CODE

@app.route("/getSavedRecipes\_user/<int:id\_user>", methods=['GET'])

@auth\_required

def get\_recipes(id\_user):

recipes = db.getSaved\_recipes(id\_user)

if recipes is None:

return jsonify({"Error": "No saved recipes found in this user"}), NOT\_FOUND\_CODE

return jsonify(recipes), OK\_CODE

@app.route("/addRecipe", methods=['POST'])

@auth\_required

def add\_saved\_recipe():

data = request.get\_json()

if "name" not in data or "preparation" not in data or "prepTime" not in data or "type" not in data or "picture" not in data or "ingredients" not in data or "idUser" not in data or "idRec" not in data:

return jsonify({"Error": "Invalid parameters"}), BAD\_REQUEST\_CODE

recipe = db.add\_recipe(data)

return jsonify(recipe), SUCCESS\_CODE

@app.route('/deleteRecipe/<int:recipe\_id>', methods=['DELETE'])

@auth\_required

def delete\_saved\_recipe(recipe\_id):

if db.remove\_recipe(recipe\_id):

return jsonify({"message": "Recipe removed with success"}), OK\_CODE

else:

return jsonify({"error": "Recipe not found"}), FORBIDDEN\_CODE

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

* ***Bd.py***

import os

from re import I

import psycopg2

def getConnection():

return psycopg2.connect(host=os.environ.get("DB\_HOST"), database = os.environ.get("DB\_NAME"), user = os.environ.get("DB\_USER"), password = os.environ.get("DB\_PASS"))

def login(username, password):

try:

with getConnection() as conn:

with conn.cursor() as cur:

query = "SELECT \* FROM Users WHERE username = %s AND password = crypt(%s, password)"

cur.execute(query, [username, password])

userRow = cur.fetchone()

user = None

if userRow is None:

return None

user = {

"id": userRow[0],

"username": userRow[5],

}

except (Exception, psycopg2.Error) as error:

print("Error while connecting to PostgreSQL", error)

finally:

if conn:

cur.close()

conn.close()

return user

def user\_exists(user):

try:

with getConnection() as conn:

with conn.cursor() as cur:

query = "SELECT \* FROM Users WHERE username = %s"

cur.execute(query, [user["username"]])

existing\_user = cur.fetchone()

except (Exception, psycopg2.Error) as error:

print("Error while connecting to PostgreSQL", error)

finally:

if conn:

cur.close()

conn.close()

return existing\_user is not None

def get\_user(username):

try:

with getConnection() as conn:

with conn.cursor() as cur:

query = "SELECT \* FROM Users WHERE username = %s"

cur.execute(query, [username])

userRow = cur.fetchone()

user = {

"id": userRow[0],

"name": userRow[1],

"email": userRow[2],

"username": userRow[3],

"password": userRow[4],

}

except (Exception, psycopg2.Error) as error:

print("Error while connecting to PostgreSQL", error)

finally:

if conn:

cur.close()

conn.close()

return user

def add\_user(user):

try:

with getConnection() as conn:

with conn.cursor() as cur:

query = "INSERT INTO Users (name, email, username, password) VALUES (%s, %s, %s, crypt(%s, gen\_salt('bf'))) RETURNING \*"

cur.execute(query, [user["name"], user["email"], user["username"], user["password"]])

conn.commit()

userRow = cur.fetchone()

user = {

"id": userRow[0],

"name": userRow[1],

"email": userRow[2],

"username": userRow[3],

}

except (Exception, psycopg2.Error) as error:

print("Error while connecting to PostgreSQL", error)

finally:

if conn:

cur.close()

conn.close()

return user

def change\_user(id, user):

try:

with getConnection() as conn:

with conn.cursor() as cur:

query = "UPDATE Users SET name = %s, username = %s, email = %s, password = crypt(%s, gen\_salt('bf')) WHERE id = %s RETURNING \*"

cur.execute(query, [user["name"], user["username"], user["email"], user["password"], id])

conn.commit()

userRow = cur.fetchone()

user = {

"id": userRow[0],

"name": userRow[1],

"email": userRow[2],

"username": userRow[3],

}

except (Exception, psycopg2.Error) as error:

print("Error while connecting to PostgreSQL", error)

finally:

if conn:

cur.close()

conn.close()

return user

def getRecipe(id\_recipe):

try:

with getConnection() as conn:

with conn.cursor() as cur:

query="SELECT \* FROM Recipe WHERE id = %s"

cur.execute(query, [id\_recipe])

recipe = cur.fetchone()

recipe = {

"id": recipe[0],

"name": recipe[1],

"preparation": recipe[2],

"prepTime": recipe[3],

"type": recipe[4],

"picture": recipe[5],

"ingredients": recipe[6],

}

except (Exception, psycopg2.Error) as error:

print("Error while connecting to PostgreSQL", error)

finally:

if conn:

cur.close()

conn.close()

return recipe

def getSaved\_recipes(id\_user):

try:

with getConnection() as conn:

with conn.cursor() as cur:

query = "SELECT \* FROM SavedRecipe WHERE idUser = %s"

cur.execute(query, [id\_user])

rows = cur.fetchall()

recipes = None

for recipe in rows:

recipe = {

"id": recipe[0],

"name": recipe[1],

"preparation": recipe[2],

"prepTime": recipe[3],

"type": recipe[4],

"picture": recipe[5],

"ingredients": recipe[6],

"id\_recipe": recipe[7],

}

recipes.append(recipe)

return recipes

except (Exception, psycopg2.Error) as error:

print("Error while connecting to PostgreSQL", error)

finally:

if conn:

cur.close()

conn.close()

def add\_recipe(recipe):

try:

with getConnection() as conn:

with conn.cursor() as cur:

query = "INSERT INTO SavedRecipe VALUES (%s, %s, %s, %s, %s, %s, %s) RETURNING \*"

cur.execute(query, [recipe["name"], recipe["preparation"], recipe["prepTime"], recipe["type"], recipe["picture"], recipe["idUser"], recipe["idRec"]])

recipe = None

recipe = cur.fetchone()

recipe = {

"id": recipe[0],

"name": recipe[1],

"preparation": recipe[2],

"prepTime": recipe[3],

"type": recipe[4],

"picture": recipe[5],

"ingredients": recipe[6],

"id\_user": recipe[7],

"id\_recipe": recipe[8],

}

conn.commit()

return recipe

except (Exception, psycopg2.Error) as error:

print("Error while connecting to PostgreSQL", error)

finally:

if conn:

cur.close()

conn.close()

def remove\_recipe(id\_recipe):

try:

with getConnection() as conn:

with conn.cursor() as cur:

query = "DELETE FROM SavedRecipe WHERE id = %s"

cur.execute(query, [id\_recipe])

conn.commit()

value = None

if cur.rowcount > 0:

value = cur.rowcount

except (Exception, psycopg2.Error) as error:

print("Error while connecting to PostgreSQL", error)

finally:

if conn:

cur.close()

conn.close()

return value

* ***Requirements.txt***

flask

psycopg2-binary

djangorestframework-jwt

* ***Vercel.json***
* {"version":2,
* "builds": [{ "src": "./index.py", "use": "@vercel/python" }],
* "routes": [{ "src": "/(.\*)", "dest": "/" }]
* }

## Aplicativo

### Processo de Deployment

### Projeto

### Atividades (páginas de inscrição)

# Conclusão

# Bibliografia