universidade de aveiro
escola superior de tecnologia e
gestão de águeda

estga

Project Report
“SideChef” cooking recipe application

Mobile app development

Group 10:
David Postolea
117738
Hugo Bessa 113783

Águeda, March 15, 2024

Degree in Information Technologies 2nd
Year – 2nd Semester

Thanks

This project represents the culmination of a long journey and is the result of both of their individual efforts. We recognize that we would not have achieved this feat without the support and invaluable contribution of the people around us.

We would like to express special thanks to Professor Gonçalo Marques, for his insightful guidance and tireless dedication throughout this mobile device project. His expert knowledge and guidance were essential in shaping our approach and achieving our goals.His wisdom and competence in supervision were fundamental to the completion of this project.

Index

Index

Table index

Figure index

1. Introduction

1.1. System overview

Contemporary gastronomy is going through a digital revolution, driven by the increasing integration of technology into our daily lives. Mobile applications play a crucial role in this transformation, facilitating access to a wide range of recipes, cooking tips and practical tools that make the kitchen experience more enjoyable and inspiring. In this sense, this report proposes to analyze and explore in depth the SideChef application, a mobile platform designed to satisfy the needs and demands of culinary enthusiasts around the world.

SideChef is more than just a recipe app, it's a complete culinary ecosystem that combines a vast collection of high-quality recipes with intuitive tools and personalized features. Upon entering the app, the user is immersed in a gastronomic universe, where they can explore the most diverse recipes from around the world, find inspiration for new creations and learn culinary techniques through detailed instructions.

Among the essential features of the SideChef application are user registration and login, allowing for a personalized and continuous experience. Advanced search functionality allows users to find specific recipe names, while detailed ingredient and preparation views provide step-by-step guidance for each dish. Favorite Recipe makes it easy for users to organize and access their favorite recipes.

1.2. Client

When developing the SideChef application, it is essential to understand the users who interact with this digital culinary system. These users are represented by culinary enthusiasts, from amateur cooks to professional chefs, who turn to the app for inspiration, learning and organizing their gastronomic experiences. The application seeks to offer an intuitive and captivating interaction, responding to users' needs, from providing a wide selection of quality recipes to offering intuitive organization tools and quick access to information about dishes.

1.3. Justification of work

The development of the SideChef mobile app was undertaken to meet the increasing need for easy-to-use recipe management solutions. The project started with a collaborative effort to decide the app's functionalities and theme. We created a detailed project prototype using Figma, which guided the development process.

Our main goal was to create a smooth experience for users to search, save, and manage recipes. To achieve this, we built a strong backend using a RESTful API for secure and efficient data handling. The API was designed and implemented to support the app's needs, including user authentication, recipe retrieval, and data management.

At the same time, we focused on designing the app using Android Studio to ensure an intuitive and attractive user interface. This phase involved creating layouts and wireframes to enhance user interaction and satisfaction. Integrating the API with the Android app was crucial. This integration allowed features like user login, recipe search, and data synchronization. We wrote and adapted code to ensure the app met design specifications and provided a responsive user experience.

Throughout development, we continuously tested and improved the app to fix issues and enhance reliability and performance.

The project was documented in a detailed report, outlining each phase from start to finish. It showcases the teamwork, technical skills, and creative design involved in developing the SideChef mobile app.

This project was justified by its potential to greatly improve how users manage and discover recipes, addressing a common need in today's digital world. It is a successful example of careful planning, technical execution, and design, resulting in a functional and user-friendly mobile app.

2. Methodology

2.1. SDLC Methodology

When implementing the SideChef application, we chose to follow the SDLC methodology, namely the Waterfall model. This model is characterized by a sequential approach, where each development phase – such as analysis, design, implementation, testing and maintenance – is carried out in a linear and sequential manner, without significant overlap between the stages. This way, each phase is completed before moving on to the next, providing a clear and defined structure for the software development process. The choice of this model was based on the specific characteristics of the project and the needs identified in the initial planning phase.

2.2. Time spent

10-03 = 7h

17-03 = 9h

24-03 = 22h

31-03 = 9h

08-04 = 3h

14-04 = 3h

21-04 = 4h

28-04 = 13h

05-05 = 7h

12-05 = 18h

19-05 = 11h

08-5 = 7h

2.3. Roadmap

The first task we starting working on for the project was finding a suitable theme. Afterwards we started defining the functionalities that the should include in the app. When the functionalities were set in place we started working on the use case diagram. We gathered the necessary information for creating a database diagram. Naturally after the diagram was complete we wrote a script for creating the database. Then we did a preliminary design using Figma. After we were satisfied with the prototype of the design we started implementing in Android Studio. When the layout files were done we started working on the functionalities. During this process we connected the app to the API, added the necessary data into the databse , we added the REST API and moved on to the deployment process. When we finished we proceeded with testing the app and made sure that it runs properly. The last thing left to do was writing the project report.

2.4 Division of labour

1.Division of labor

Nº	Task	Participation(%)
1	Planning of the project and deciding the functionalities	• Hugo - 50 • David - 50
2	Choosing a theme and designing a prototype in Figma	• Hugo - 50 • David - 50
3	API integration	• Hugo - 100 • David - 0
4	Design work in Android Studio	• Hugo - 75 • David - 25
5	Writing the code for the app	• Hugo - 100 • David - 0 •
6	Final report	• Hugo - 30 • David - 70

3. Requirements Model

3.1. Requirements

3.1.1. Functional requirements

2.Requirement Table

No.	Name	Description	Priority
1	Registration and login	Allow users to register and log in to the SideChef app.	Critical
2	Advanced recipe search	Provides advanced search functionality for users to find specific recipes.	Low
3	Detailed recipe view	Present detailed information about the ingredients, preparation instructions and nutrition of the recipes.	High
4	Add to Favorites	Allow users to add favorite recipes to my recipes list.	High
5	View recipes in favorites	Allow users to view recipes in favorites	High

6	View all recipes available	Allows users to view all available recipes in the app	High

3.1.2. non-functional requirements

No.	Name	Description	Priority
1	Usability	The application must be easy to use and intuitive to ensure a good user experience.	High
2	Performance	Ensure the application responds quickly and without delays, even in high load situations.	High
3	Security	Protect user data and ensure the privacy of personal information.	High
4	Compatibility	Make sure the app is compatible with a variety of devices and operating systems.	Average
5	Reliability	Make sure the application is stable and does not experience frequent crashes or errors.	High
6	Efficiency	Effectively utilize device resources to ensure optimized performance.	Average
7	Maintainability	Facilitate application maintenance and updates to resolve issues and add new features.	Average
8	Scalability	Make sure the application is capable of handling an increase in the number of users and data.	High

3.2. Use case model

3.2.1. Overview

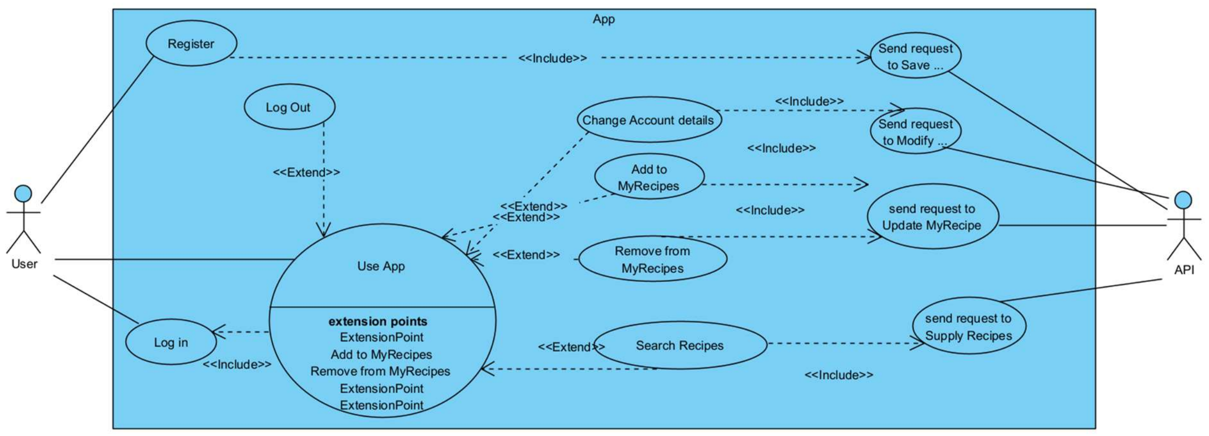
In this report, a use case diagram for SideChef will be presented. This diagram aims to offer a visual representation of the interactions between users and the system, highlighting the main actors involved and the supported usage scenarios. By examining this diagram, you will be able to clearly and concisely understand how users interact with the system and what functionality is available to them. The purpose of this document is to provide a comprehensive view of how the system works, focusing on the perspectives of users and the different roles they play during their interactions with the system.

3.2.2. Actors

The "Client" actor represents the end user who interacts directly with the SideChef application. This user can access the application through a user interface in a mobile application. The "Customer" is responsible for using the features offered by the SideChef application to achieve their culinary goals, such as finding recipes and following step-by-step cooking instructions.

The "API" actor represents the application programming interface (API) provided by the SideChef application, specifically a REST API (Representational State Transfer). This API allows you to receive requests from users, process them in the database and send responses back to users. It acts as an intermediary between the user and system resources, facilitating communication and data exchange in an efficient and secure manner.

3.2.3. Use cases



1.Use cases

4. The app

4.1. Data base

4.1.1. Overview

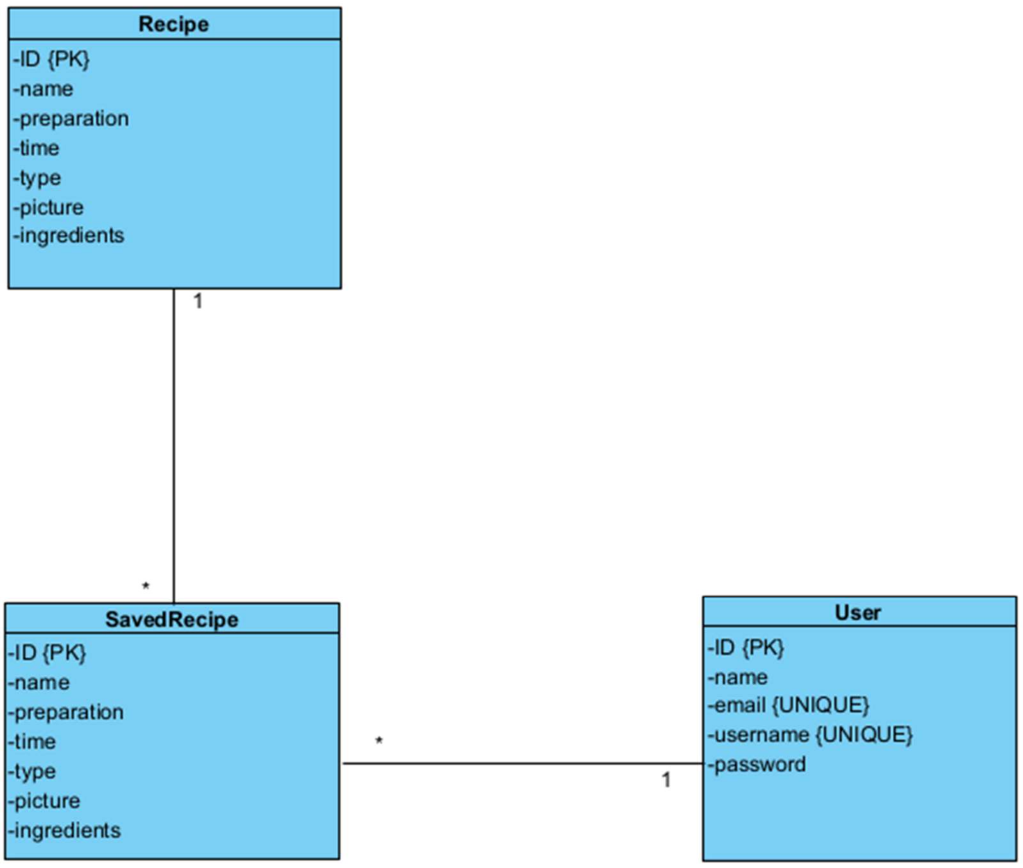
The SideChef application database designed in postgreSQL plays a fundamental role in storing and managing the data that feeds the system's functionalities. Designed to handle a variety of information, from recipes and ingredients to user profiles. The database consists of 3 Tables, the "Users" is responsible for storing everything related to the user account, the "Recipe" table is responsible for storing everything related to the recipes that are made available for the user to interact with, and the "SavedRecipes" table is related to recipes saved by users so that they can be stored for the user to view later.

Additionally, the database supports complex operations such as advanced queries for detailed recipe searches, transactions to ensure data consistency, and regular backups to protect against information loss. This way, users can trust the SideChef app to provide accurate and up-to-date information whenever necessary.

The database also plays an important role in data security, implementing robust protection measures, such as password encryption using pgcrypto and access control, to guarantee the confidentiality and integrity of user information.

4.1.2. Entity-Relationship Schema

2.Entity-Relationship Schema



4.1.3. Script

```
CREATE TABLE Users(

  id SERIAL PRIMARY KEY,

  name VARCHAR(40) 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(600) NOT NULL,
  prepTime Int NOT NULL,
  type VARCHAR(15) NOT NULL,
  picture TEXT NOT NULL,
  ingredients VARCHAR(300) NOT NULL
);

CREATE TABLE SavedRecipe(
  id SERIAL PRIMARY KEY,
  name VARCHAR(30) NOT NULL,
  preparation VARCHAR(600) NOT NULL,
  prepTime Int NOT NULL,
  type VARCHAR(15) NOT NULL,
  picture TEXT NOT NULL,
  ingredients VARCHAR(300) 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)
```

);

4.2. POSTMAN

When developing the SideChef application, we used Postman to test and validate our REST API. This tool plays a crucial role in allowing us to test all API routes and endpoints efficiently.

With Postman, it was possible to send a variety of requests, such as GET, POST, PUT and DELETE, to interact with the resources available in the API. For example, I can send GET requests to retrieve information about recipes, POST to add new recipes to the database, PUT to update existing information and DELETE to remove specific resources, among other actions.

Additionally, Postman offers advanced features such as the ability to create collections of requests for automated testing and generate API documentation to make it easier for other team members to understand and use.

Thanks to Postman, we were able to ensure the quality and reliability of our API by testing all possible scenarios and quickly identifying any issues or failures that may arise during development or after implementation.

4.3. REST API

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

REST plays a fundamental role in the architecture and functioning of the SideChef application. This programming interface allows efficient and secure communication between the client, the SideChef application and the underlying database. Hosted on Vercel, a modern and scalable hosting platform, our REST API guarantees constant availability and optimized performance.

Through the REST API, users of the SideChef application can send requests to perform various actions using the link https://sidechef-api.vercel.app/, such as searching for recipes, saving favorite recipes, creating accounts, obtaining users, among others . These requests are then processed by the API, which forwards them to the corresponding database, where the necessary operations are performed.

After processing in the database, the REST API returns relevant responses to users, providing updated information, confirmations of actions performed and other relevant data. This data exchange occurs efficiently, thanks to the underlying HTTP protocol, which allows information to be transferred quickly and securely between the client and the server. Our REST API consists of 2 main files (index.py and bd.py).

Documentation regarding the various API endpoints can be found here: 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
from flask_swagger_ui import get_swaggerui_blueprint

import db

app = Flask(__name__)
app.debug = True
app.config['SECRET_KEY'] = os.getenv('SECRET_KEY', 'mysecretkey')

NOT_FOUND_CODE = 404
OK_CODE = 200
SUCCESS_CODE = 201
NO_CONTENT_CODE = 204
BAD_REQUEST_CODE = 400
UNAUTHORIZED_CODE = 401
FORBIDDEN_CODE = 403
SERVER_ERROR = 500

@app.route('/static/<path:path>')
def send_static(path):
    return send_from_directory('static', path)


SWAGGER_URL = '/doc'
API_URL = '/static/swagger.json'
swaggerui_blueprint = get_swaggerui_blueprint(
    SWAGGER_URL,
    API_URL,
    config={
        'app_name': "SideChef-REST-API"
    }
)
app.register_blueprint(swaggerui_blueprint, url_prefix=SWAGGER_URL)



@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

    if db.email_exists(data):
        return jsonify({"error": "Email 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", methods=['PUT'])
@auth_required
```

```
def update_user():
    data = request.get_json()

    if "name" not in data or "email" not in data or "username" not in data:
        return jsonify({"error": "invalid parameters"}), BAD_REQUEST_CODE

    if db.user_exists(data) and db.get_user_by_username(data["username"])["id"] != request.user["id"]:
        return jsonify({"error": "Username already exists"}), BAD_REQUEST_CODE

    if db.email_exists(data) and db.get_user_by_email(data["email"])["id"] != request.user["id"]:
        return jsonify({"error": "Email already exists"}), BAD_REQUEST_CODE

    user = db.change_user(request.user["id"], data)

    return jsonify(user), SUCCESS_CODE



@app.route("/changePassword", methods=['PUT'])
@auth_required
def update_password():
    data = request.get_json()

    if "password" not in data:
        return jsonify({"error": "invalid parameters"}), BAD_REQUEST_CODE

    user = db.change_password(request.user["id"], data)
    return jsonify(user), SUCCESS_CODE



@app.route("/getUser", methods=['GET'])
@auth_required
def get_user():
    user = db.get_user(request.user["id"])

    if user is None:
        return jsonify({"Error": "No user found"}), NOT_FOUND_CODE

    return jsonify(user), OK_CODE



@app.route("/getRecipes/<string:name_recipe>", methods=['GET'])
@auth_required
def get_recipes(name_recipe):
    recipes = db.getRecipes(name_recipe)

    if recipes is None:
        return jsonify({"Error": "No Recipes found"}), NOT_FOUND_CODE

    return jsonify(recipes), OK_CODE



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

```
def get_all_recipes():
    recipes = db.getAllRecipes()
    if recipes is None:
        return jsonify({"Error": "Couldnt get recipes"}), NOT_FOUND_CODE

    return jsonify(recipes), OK_CODE



@app.route("/getSavedRecipes_user", methods=['GET'])
@auth_required
def get_SavedRecipes():
    recipes = db.getSaved_recipes(request.user["id"])

    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()
    recipe = None
    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 "idRec" not in data:
        return jsonify({"Error": "Invalid parameters"}), BAD_REQUEST_CODE
```

```
 data["idUser"] = request.user["id"]

 if not db.SavedRecipe_exists(data["idRec"], data["idUser"]):

   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, request.user["id"]):

   return jsonify({"message": "Recipe removed with success"}),
OK_CODE

 else:

   return jsonify({"error": "Recipe not found"}), FORBIDDEN_CODE

if __name__ == "__main__":

 app.run()
```

<u>Bd.py</u>

```
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[3],
                    }
    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):
  count = 0
  try:
    with getConnection() as conn:
      with conn.cursor() as cur:
        cur.execute("SELECT * FROM users WHERE username = %s",
[user["username"]])
        count = cur.rowcount
  except (Exception, psycopg2.Error) as error:
    print("Error while connecting to PostgreSQL", error)
  finally:
    if conn:
      cur.close()
      conn.close()
  return count > 0

def email_exists(user):
  count = 0
  try:
    with getConnection() as conn:
      with conn.cursor() as cur:
        cur.execute("SELECT * FROM users WHERE email = %s",
[user["email"]])
        count = cur.rowcount
  except (Exception, psycopg2.Error) as error:
    print("Error while connecting to PostgreSQL", error)
  finally:
    if conn:
      cur.close()
      conn.close()
  return count > 0



def get_user(id):
        try:
                with getConnection() as conn:
                        with conn.cursor() as cur:
                                query = "SELECT * FROM Users WHERE id
= %s"
                                cur.execute(query, [id])
                                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 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_data = {
                    "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)
        if conn:
            conn.rollback()
    finally:
        if conn:
            cur.close()
            conn.close()
    return user_data


def get_user_by_username(username):
    try:
        with getConnection() as conn:
            with conn.cursor() as cur:
                query = "SELECT * FROM Users WHERE username = %s"
                cur.execute(query, [username])
                user_row = cur.fetchone()
                if user_row:
                    return {
                        "id": user_row[0],
                        "name": user_row[1],
                        "email": user_row[2],
                        "username": user_row[3],
                    }
                else:
                    return None
    except (Exception, psycopg2.Error) as error:
        print("Error while connecting to PostgreSQL", error)
    finally:
        if conn:
            cur.close()
            conn.close()

def get_user_by_email(email):
    try:
        with getConnection() as conn:
            with conn.cursor() as cur:
                query = "SELECT * FROM Users WHERE email = %s"
                cur.execute(query, [email])
                user_row = cur.fetchone()
                if user_row:
                    return {
                        "id": user_row[0],
                        "name": user_row[1],
                        "email": user_row[2],
                        "username": user_row[3],
                    }
                else:
                    return None
    except (Exception, psycopg2.Error) as error:
        print("Error while connecting to PostgreSQL", error)
    finally:
        if conn:
            cur.close()
            conn.close()


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 WHERE id = %s RETURNING *"
                cur.execute(query,      [user["name"],      user["username"],
user["email"], 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)
        if conn:
            conn.rollback()
    finally:
        if conn:
            cur.close()
            conn.close()
        return user


def change_password(id, user):
    try:
        with getConnection() as conn:
            with conn.cursor() as cur:
                query = "UPDATE Users SET password = crypt(%s, 
gen_salt('bf')) WHERE id = %s RETURNING *"
                cur.execute(query, [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)
        if conn:
            conn.rollback()
    finally:
        if conn:
            cur.close()
            conn.close()
        return user

def SavedRecipe_exists(id_recipe, id_user):
    count = 0
    try:
```

```
    with getConnection() as conn:
      with conn.cursor() as cur:
        query = "SELECT * FROM SavedRecipe WHERE idrec = %s
AND iduser = %s"
        cur.execute(query, [id_recipe, id_user])
        count = cur.rowcount
  except (Exception, psycopg2.Error) as error:
    print("Error while connecting to PostgreSQL", error)
  finally:
    if conn:
      cur.close()
      conn.close()
  return count > 0


def getRecipes(name_recipe):
  recipes = []
  try:
    with getConnection() as conn:
      with conn.cursor() as cur:
        query = "SELECT * FROM Recipe WHERE name ILIKE %s"
        cur.execute(query, ['%' + name_recipe + '%'])
        for recipe in cur.fetchall():
          recipe = {
            "id": recipe[0],
            "name": recipe[1],
            "preparation": recipe[2],
            "prepTime": recipe[3],
            "type": recipe[4],
            "picture": recipe[5],
            "ingredients": recipe[6],
          }
          recipes.append(recipe)
  except (Exception, psycopg2.Error) as error:
    print("Error while connecting to PostgreSQL", error)
  finally:
    if conn:
      cur.close()
      conn.close()
  return recipes
```

```
def getAllRecipes():
    try:
        with getConnection() as conn:
            with conn.cursor() as cur:
                query="SELECT * FROM Recipe"
                cur.execute(query)
                recipes = []
                for recipe in cur.fetchall():
                    recipe = {
                        "id": recipe[0],
                        "name": recipe[1],
                        "preparation": recipe[2],
                        "prepTime": recipe[3],
                        "type": recipe[4],
                        "picture": recipe[5],
                        "ingredients": recipe[6],
                    }
                    recipes.append(recipe)
    except (Exception, psycopg2.Error) as error:
        print("Error while connecting to PostgreSQL", error)
    finally:
        if conn:
            cur.close()
            conn.close()
    return recipes


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 = []

                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)
    except (Exception, psycopg2.Error) as error:
      print("Error while connecting to PostgreSQL", error)
    finally:
      if conn:
        cur.close()
        conn.close()
      return recipes


def add_recipe(recipe):
  try:
    with getConnection() as conn:
      with conn.cursor() as cur:
        query = "INSERT INTO SavedRecipe (name, preparation,
preptime, type, picture, ingredients, iduser, idrec) VALUES (%s, %s,
%s, %s, %s, %s, %s, %s) RETURNING *"
        cur.execute(query, [recipe["name"], recipe["preparation"],
recipe["prepTime"], recipe["type"], recipe["picture"],
recipe["ingredients"] ,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)
        if conn:
            conn.rollback()
    finally:
        if conn:
            cur.close()
            conn.close()



def remove_recipe(id_recipe, idUser):
    try:
        with getConnection() as conn:
            with conn.cursor() as cur:
                query = "DELETE FROM SavedRecipe WHERE id = %s AND
iduser = %s "
                cur.execute(query, [id_recipe, idUser])
                conn.commit()

                value = None
                if cur.rowcount > 0:
                    value = cur.rowcount

    except (Exception, psycopg2.Error) as error:
        print("Error while connecting to PostgreSQL", error)
        if conn:
            conn.rollback()
    finally:
        if conn:
            cur.close()
            conn.close()
        return value
```

- ***Requirements.txt***

```
flask

psycopg2-binary

djangorestframework-jwt

flask-swagger-ui
```

- ***Vercel.json***

```
• {"version":two,
•  "builds":
  [{"src":"./index.py","use":"@vercel/python"}],
•  "routes": [{"src":"/(.*)","dest":"/"}]
• }
```

4.4. Application

4.4.1. Deployment Process

To deploy our mobile app using Vercel, follow these steps: Vercel is a cloud platform optimized for hosting modern web applications and static websites. First, ensure your project is production-ready by updating dependencies, running tests, and building the project if necessary. Create a Vercel account at vercel.com using GitHub, GitLab, or Bitbucket. Install the Vercel CLI with npm install -g vercel and navigate to your project directory in the terminal. Initialize your project with Vercel by running vercel and follow the prompts to link your project to your Vercel account. To deploy, use the command vercel --prod, which builds and deploys your project, providing a unique URL for your app. Also for new deployments you only need to git push to the GitHub repositor linked to the Vercel project. For automatic deployments, go to the Vercel dashboard, select your project, navigate to the "Git" tab, and connect your repository, configuring it to deploy from the desired branch. Manage environment variables via the "Settings" tab in the dashboard under "Environment Variables." Vercel offers a dashboard to monitor deployments, view logs, and manage settings. Additional useful commands include vercel for preview deployments, vercel ls to list all deployments, and vercel rollback [deployment-id] to revert to a previous deployment.

4.4.2. Design

The design of the XML layouts tries to fit in with the chosen theme of the project. The choice of color palette plays a fundamental role in conveying the visual identity and creating an appropriate atmosphere. The predominant shades of orange and hints of gray were strategically selected to create a modern and harmonious aesthetic.

Orange tones are associated with energy, enthusiasm, evoking a feeling of warmth and appetite. This choice is particularly effective in stimulating user interest and creating an inviting atmosphere when exploring new recipes. On the other hand, gray, being a neutral color, adds sophistication and elegance to the design. It balances the vibrant orange, providing a subtle contrast that highlights important elements in the interface.

In addition to conveying a cohesive visual identity, the colors are visually pleasing and easy to read, ensuring a positive user experience. The arrangement of recipes in the application is designed to offer a visually attractive and intuitive experience for users. Each recipe is presented on cards (CardViews) with rounded corners and a slight elevation to create a sense of depth.

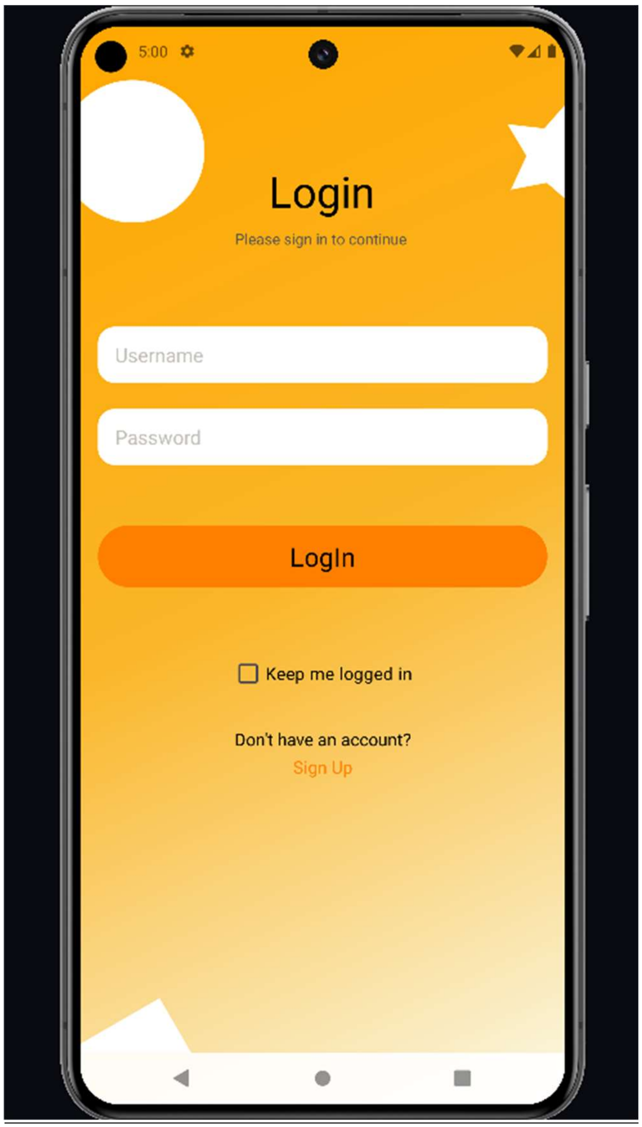
Inside each card, the elements are organized in a clear and concise manner. A representative image of the recipe takes up a significant portion of the space, providing users with a quick and captivating view of the dish. The image is centered and sized to fit onto the card. Below the image, the recipe title is displayed in bold and centered, ensuring immediate identification of the dish. The text is easy to read and contrasts well with the background, making it easy to read. Next to the title, important information about the recipe is displayed, such as the type of dish (e.g. "Meat") and the estimated preparation time (e.g. "90m"). This information is displayed in a clear and concise way, allowing users to quickly identify the type of recipe and assess whether they have time available to prepare it. Additionally, a heart icon next to the preparation time indicates whether the recipe is marked as a favorite by the user. This feature allows users to save their favorite recipes for future reference.

Overall, the arrangement of recipes in this layout is designed to maximize usability and aesthetics, providing users with a pleasant and efficient experience when exploring and interacting with recipes in the app.

4.4.3. Activities (pages)

Main Activity

3.Login Screen



The MainActivity is the main activity that handles the user login interface. It checks whether the login data is stored securely if the user has previously chosen to remain logged in and, if so, redirects the user to MainPageNav. Otherwise, it displays the login screen.

- UI Components

➢ **TextView loginText:**Displays the title "Login".

➢ **TextView loginDescription:**Displays the description "Please sign in to continue".

➢ **EditText usernameInput:**Input field for username.

➢ **EditText passwordInput:**Input field for password.

➢ **Button buttonLogIn:**Button to start the session.

➢ **CheckBox keeploginBox:**Option to keep the user authenticated.

- **TextView signUpLink:**Link to the registration page.
- **TextView labelValidation:**Validation and error messages.
- **FrameLayout frameProgress:**Layout for displaying a progress indicator during authentication.
- **ProgressBar progressBar:**Progress indicator

  - Functionalities

- **Login Data Verification:**Uses EncryptedSharedPreferences to check for saved login data from a previous session.

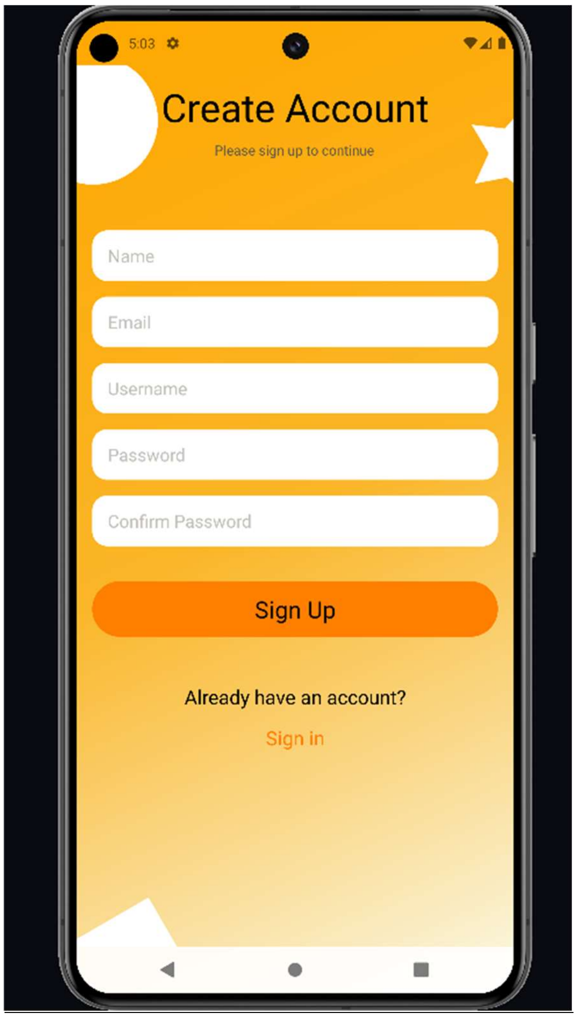
- **Credential Validation:**Validate the username and password before proceeding with the login attempt. The username must be between 5 and 15 characters long, and the password must contain at least one capital letter, one digit and one special character.

- **Login:**Sends a request to the API to authenticate the user. If successful, it obtains user profile information and redirects to MainPageNav.

- **Keep Login:**If the "Keep me logged in" option is checked, the login data is stored securely, to keep the login the next time until the logout is done.

CreateAccount

4.Create Account Screen



CreateAccount is the activity responsible for registering new users in the application. It provides an account creation form and validates the data entered, as well as communicating with the API to create the user account.

- UI Components

➢ **TextView signInLink: Link to redirect the user to the login page.**
➢ **EditText inputName: Input field for the user name.**
➢ **EditText inputEmail: Input field for the user's email.**
➢ **EditText inputUsername: Input field for the username.**
➢ **EditText inputPassword: Input field for the password.**
➢ **EditText inputConfirmPassword: Input field to confirm the password.**
➢ **Button signUpButton: Button to submit the registration form.**
➢ **TextView labelValidation: Validation and error messages.**
➢ **ProgressBar progressBar: Progress indicator during account creation.**
➢ **FrameLayout frameProgress: Layout to display the progress indicator.**

- Functionalities

- **View Configuration:**Configures the user interface and initializes UI components. Adds behavior to buttons and input fields, including redirection to the login page.

- **Validation of Credentials: (validateCredentials)**Validates input fields based on regex patterns.
  Ensure that:
  The name contains only letters and is between 2 and 30 characters long.
  The email follows the standard format.
  The username is between 5 and 15 characters long.
  The password contains at least 8 characters, including an uppercase letter, a digit, and a special character.
  Password confirmation matches the password.

- **Account Creation:**createAccount – Sends a POST request to the API to create a new account. Data is sent in JSON format. Uses the Volley library to manage HTTP requests. Ensures safe and correct execution of the request by disabling input fields and displaying a progress indicator during the operation.

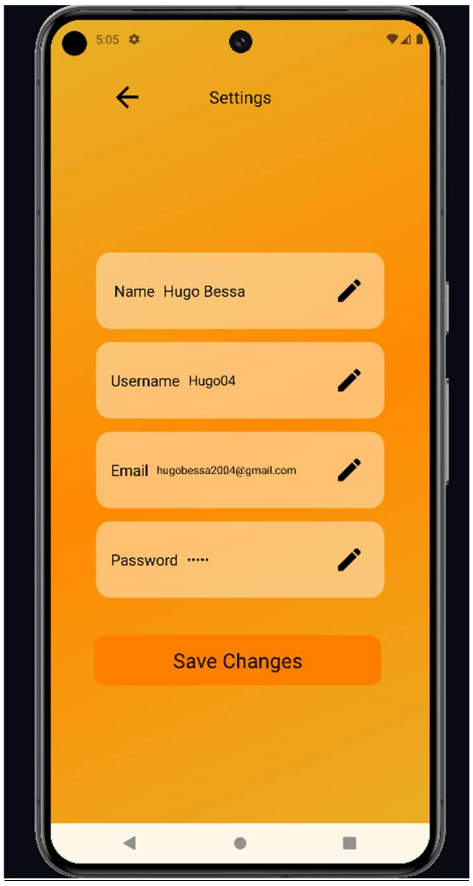
- **API Response Management:**Successful Response: Redirects the user to the login page (changeToSignIn). Response Error: Shows appropriate error messages based on the type of error (eg, AuthFailureError, NetworkError, TimeoutError, ServerError).

- **Show Error:**showError: Displays error messages in the UI, configuring the labelValidation color and text.

- **Redirection to Login:**changeToSignIn: Starts the MainActivity and ends the CreateAccount activity.

Settings

5.Settings Screen



The "Settings" page is responsible for configuring user information, such as name, username, password and email. Allows the user to make changes as needed.

- UI Components

➢ **EditText editTextCurrentName:**Edit field for the user's current name.
➢ **EditText editTextCurrentUserName:**Edit field for current username.
➢ **EditText editTextCurrentPassword:**Edit field for current password.
➢ **EditText editTextCurrentEmail:**Edit field for the user's current email.
➢ **ImageButton imageViewEditName, imageViewEditUserName, imageViewEditPassword, imageViewEditEmail:**Edit icons associated with each field.
➢ **Button saveChangesButton:**Button to save the changes made.
➢ **TextView labelValidation:**Text field to display validation or error messages.
➢ **ProgressBar progressBar:**Progress indicator during the execution of operations.
➢ **FrameLayout frameProgress:**Layout that contains the progress bar.

- Functionalities

- **Initial Page Setup:**Loads the user data received from the intent and displays it in the edit fields. Configures button and icon listeners to allow editing of text fields.

- **Save Changes:**Pressing the "Save Changes" button checks what information has been changed and takes appropriate action. If only the password is changed, the changePassword function is called to update the password in the database. If information other than the password is changed, the changeCredentials function is called to update the name, username and email in the database, if the username, name or email are changed, the application clears all activities from the stack and restarts the MainPageNav (addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP or Intent.FLAG_ACTIVITY_CLEAR_TASK or Intent.FLAG_ACTIVITY_NEW_TASK)

- **Data Validation:**Before saving changes, validate the data entered in the editing fields. Validation includes checking for valid email formats, appropriate length usernames, and strong passwords.

- **Communication with the API:**Uses the Volley library to send HTTP PUT requests to the API to change user information. Handles different types of API response errors such as authentication errors, network errors, and server errors.

- **Navigation:**Configures the backButton to call the changeToMainActivity method, which ends the current activity and returns to the previous one.

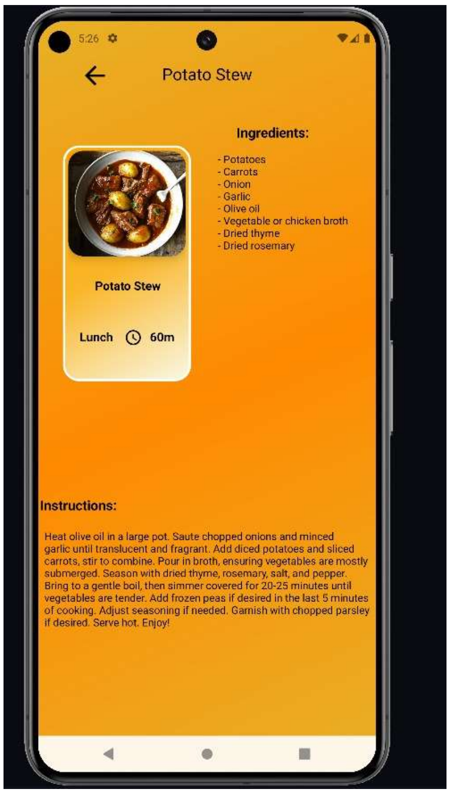
- **Progress Management:**Displays a progress bar while waiting for the API response. Disables edit fields and buttons when performing asynchronous operations.

- **Safe Storage:**loadPassword and loadToken methods: Load the securely stored password and authentication token using EncryptedSharedPreferences. SaveToken Method: Stores the authentication token securely.

- **Token Revalidation:**loginBeforeChangeCredentials/ChangePassword method: Revalidates the authentication token if the request fails due to authentication problems. handleLoginResponse method: Processes the token revalidation response and stores the new token.

Recipe Details

6.Recipe Details Screen



Ingredients:

- Potatoes
- Carrots
- Onion
- Garlic
- Olive oil
- Vegetable or chicken broth
- Dried thyme
- Dried rosemary

Potato Stew

Lunch 60m

Instructions:

Heat olive oil in a large pot. Saute chopped onions and minced garlic until translucent and fragrant. Add diced potatoes and sliced carrots, stir to combine. Pour in broth, ensuring vegetables are mostly submerged. Season with dried thyme, rosemary, salt, and pepper. Bring to a gentle boil, then simmer covered for 20-25 minutes until vegetables are tender. Add frozen peas if desired in the last 5 minutes of cooking. Adjust seasoning if needed. Garnish with chopped parsley if desired. Serve hot. Enjoy!

The "click_recipe_details" page is responsible for showing detailed information about a recipe, including name, type, preparation time, ingredients and instructions. Furthermore, it allows the user to return to the main page.

- UI Components

ImageButton backButton: Button to return to the previous page.

TextView recipeNameLabel: Text field to display the name of the recipe.

TextView recipeSmallNameLabel: Text field to display the short name of the recipe.

TextView recipeTypeLabel: Text field to display the type of recipe (eg, dessert).

TextView recipeTimeLabel: Text field to display recipe preparation time.

TextView recipeIngredientsLabel: Text field to display the list of ingredients.

TextView recipeInstructionsLabel: Text field to display preparation instructions.

ImageView recipeImageView: Recipe image.

- Functionalities

➢ **Data Loading:**Gets the revenue data from the Intents passed to the activity. Variables such as recipeId, recipeName, recipePreparation, recipePrepTime, recipeType, recipeImage and recipeIngredients are initialized with the received values.

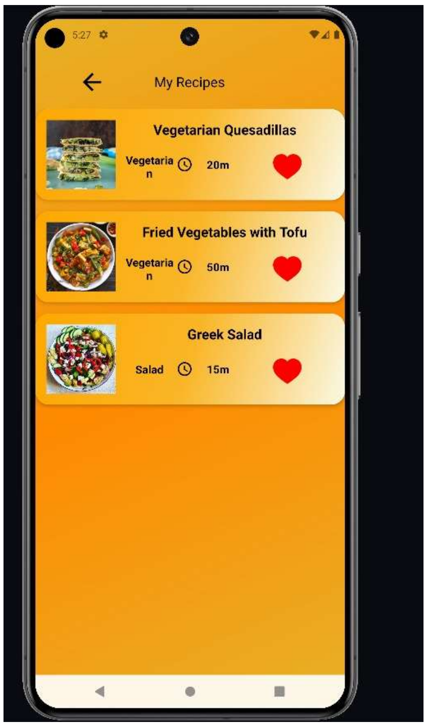
➢ **Data Display:**Formats the list of ingredients into a readable and presentable list. Updates UI components with recipe data.

➢ **Navigation:**Configures the backButton to call the changeToMainActivity method, which ends the current activity and returns to the previous one.

➢ **Ingredient Formatting:**Converts the received string of ingredients into a list of strings. Formats each ingredient to begin with a capital letter and adds a bullet ("-") before each one.

<u>**MyRecipes**</u>

7.My Recipes Screen



The "My Recipes" page is responsible for displaying the user's favorite recipes. It allows you to load recipes from the server, display them in a list and remove them from the favorites list.

- UI Components

ImageButton backButton: Button to return to the previous page.

ProgressBar progressBar: Progress bar to indicate ongoing operations.

FrameLayout frameProgress: Layout to encapsulate the progress bar.

RecyclerView recyclerView: Component to display the list of saved recipes.

- Functionalities

- **setupView method:**Initializes UI components. Configures the RecyclerView with a linear layout manager. Gets user data from the Intents passed to the activity. Displays the progress bar and disables the back button while data is loaded. Calls the getSavedRecipes method to get the user's saved recipes.
- **Response Handling:**handleGetSavedRecipesResponse method: Processes the request response, converts JSON data into recipe objects and configures the RecyclerView with an adapter. Configures click events for each item in the recipe list, allowing detailed viewing or removal of the recipe.

- **changeToProfile method:**End the current activity and return to the profile page.

- **RecyclerView Adapter:**MyRecipes_RecyclerViewAdaptor Class: Adapts saved recipe data to be displayed in the RecyclerView. Configures the view of each item in the list. Implements click events for detailed viewing and removal of recipes.

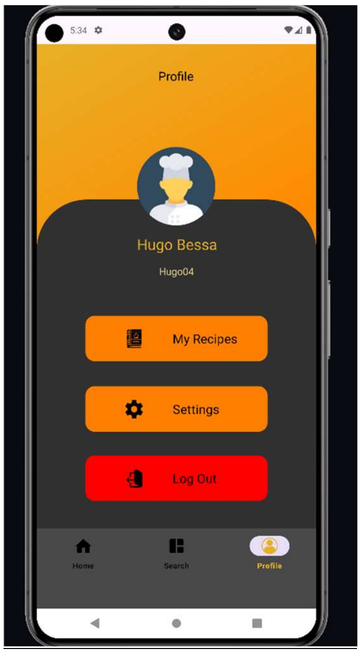
- **Safe Storage:**loadPassword and loadToken methods: Load the securely stored password and authentication token using EncryptedSharedPreferences. SaveToken Method: Stores the authentication token securely.

- **Token Revalidation:**loginBeforeGetRecipes method: Revalidates the authentication token if the request fails due to authentication problems. handleLoginResponse method: Processes the token revalidation response and stores the new token.

- **Recipe Removal:**removeFavoriteRecipe method: Sends a DELETE request to remove a recipe from the favorites list. loginBeforeRemoveRecipe method: Revalidates the authentication token before removing the recipe, if necessary. handleLoginRemoveRecipeResponse method: Processes the token revalidation response for revenue removal.

Profile

8.Profile Screen



The "Profile" snippet displays user information, such as name and username, and offers buttons to navigate to other application features.

- UI Components

➢ **Button myrecipesButton:**Button to navigate to the saved recipes page.

➢ **Button settingsButton:**Button to navigate to the settings page.

➢ **Button logoutButton:**Button to log out.

➢ **TextView name_label:**Label to display the user's name.

➢ **TextView username_label:**Label to display the username.

- Functionalities

➢ **setupView method:**Initializes UI components. Gets user data from the Intents passed to the activity. Sets text labels with user information. Configures click events for buttons.

➢ **changeToMyRecipes method:**Navigate to the "MyRecipes" page, passing user information through Intent.

- **changeToSettings method:**Navigate to the "Settings" page, passing user information through Intent.

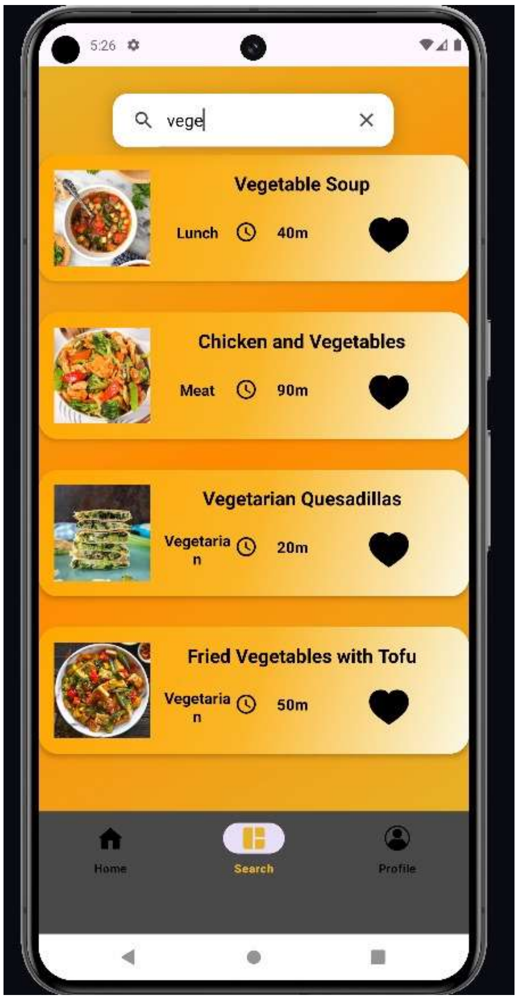
- **changeToLogIn method:**Navigate to the login page and end the current activity.

- **deleteDataLogIn method:**Erases the login data stored on the device.

- **loadToken method:**Loads the securely stored authentication token using EncryptedSharedPreferences.

Search

9.Search Screen



The "Search" snippet makes it easy to search for recipes and display the results. It also manages interaction with the API and handling of user sessions.

- UI Components

- **SearchView searchBar:**Search bar to enter recipe name.

- **ProgressBar progressBar:**Progress bar to indicate loading.
- **FrameLayout frameProgress:**Layout to contain the progress bar.
- **RecyclerView recyclerView:**List to display search results.

  - Functionalities

- **setupView method:**Initializes UI components. Sets the color of text in the search bar. Gets user data from the Intents passed to the activity. Configures the listener for the search bar to handle search submissions.

- **getRecipes method:**Makes an API request to get revenue based on the keyword. Generate the UI state during the request (showing/hiding the progress bar). Defines the headers and content type of the request.

- **handleGetRecipesResponse method:**Handles the API response. Configures the RecyclerView to display search results. Defines click events on list items to view details or add to favorites.

- **Session and Login Management:**

loginBeforeGetRecipes method:Performs user login before obtaining recipes, if necessary. Makes a request to the API to authenticate the user and renew the token.

handleLoginResponse method:Handles the login response and saves the renewed token. Make the request to obtain revenue with the new token.

loginBeforeAddRecipe method:Log the user in before adding a recipe to favorites, if necessary.

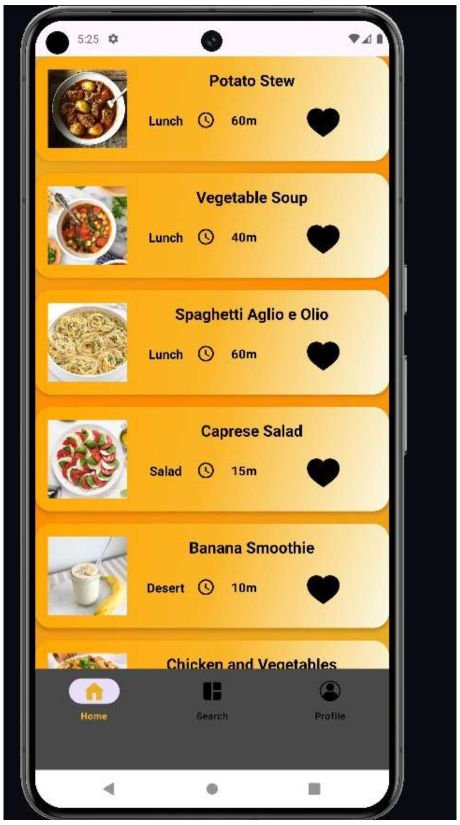
handleLoginAddRecipeResponse method:Handles the login response and saves the renewed token. Add the recipe to your favorites with the new token.

- **Favorites Management:**favoriteRecipe method: Makes a request to the API to add a recipe to favorites. Defines the headers and content type of the request.

- **Safe Storage:**loadToken, loadPassword, SaveToken methods: Use EncryptedSharedPreferences to securely store and retrieve the user's token and password.

Home

10.Home Screen



The "Home" fragment allows the visualization of recipes obtained from the API and the management of user favorites and sessions.

- UI Components

ProgressBar progressBar: Progress bar to indicate loading.
FrameLayout frameProgress: Layout to contain the progress bar.
RecyclerView recyclerView: List to display the recipes obtained.

- Functionalities

➢ **setupView method:**Initializes UI components. Gets user data from the Intents passed to the activity. Displays the progress bar while recipes are loaded. Calls the getRecipes method to get recipes from the API.

➢ **getRecipes method:**Makes a request to the API to obtain recipes. Generate the UI state during the request (showing/hiding the progress bar). Defines the headers and content type of the request. Implements a retry policy for the request.

- **handleGetRecipesResponse method:**Handles the API response. Configures the RecyclerView to display recipes. Defines click events on list items to view details or add to favorites.

- **Session and Login Management:**

loginBeforeGetRecipes method:Performs user login before obtaining recipes, if necessary. Makes a request to the API to authenticate the user and renew the token.

handleLoginResponse method:Handles the login response and saves the renewed token. Make the request to obtain revenue with the new token.

loginBeforeAddRecipe method:Log the user in before adding a recipe to favorites, if necessary.

handleLoginAddRecipeResponse method:Handles the login response and saves the renewed token. Add the recipe to your favorites with the new token.

- **Favorites Management:**favoriteRecipe method: Makes a request to the API to add a recipe to favorites. Defines the headers and content type of the request.

- **Safe Storage:**loadToken, loadPassword, SaveToken methods: Use EncryptedSharedPreferences to securely store and retrieve the user's token and password.

4.5. Security

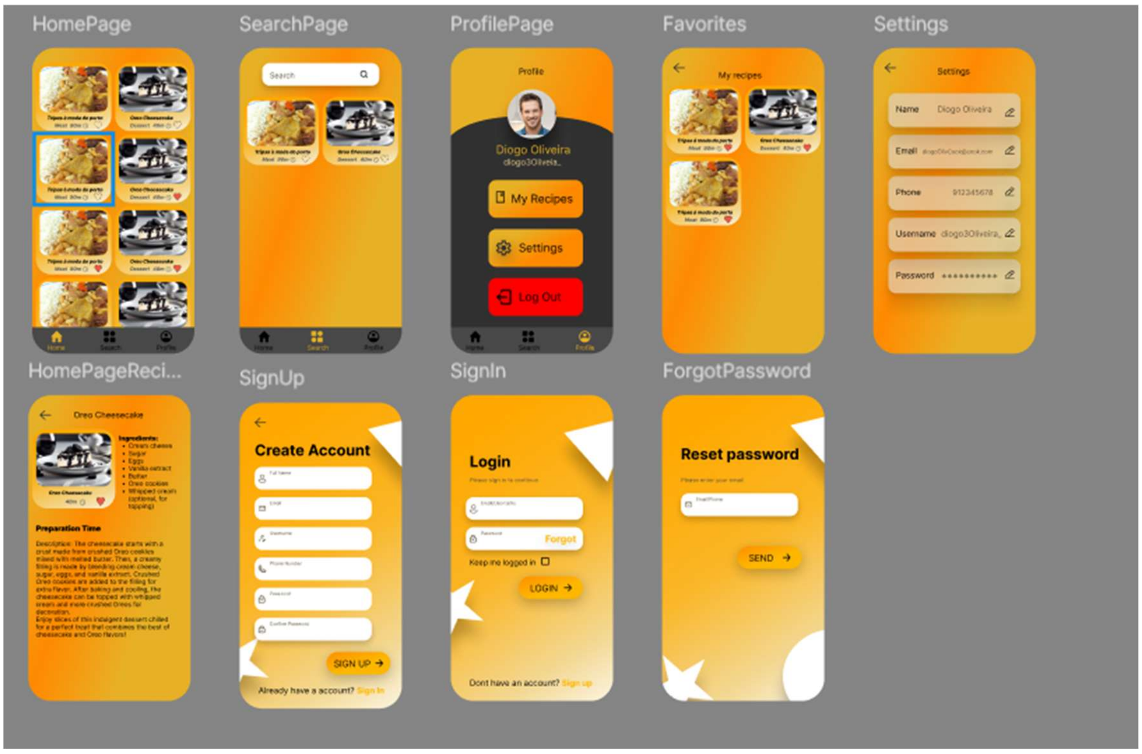
Sensitive data such as tokens are securely stored using EncryptedSharedPreferences and MasterKey, ensuring that any data stored in shared preferences is encrypted and protected from unauthorized access. The fragment also retrieves and uses sensitive user data securely from intents, including user_id, username, token, email, and name. Additionally, it clears stored user data on logout to prevent unauthorized access to user information.

In the CreateAccount activity, input validation is performed using regular expressions to ensure that user inputs such as name, email, username, password, and confirmPassword meet specific criteria. This helps in preventing injection attacks and ensures data integrity. The activity also uses the Volley library to handle network requests, ensuring secure transmission of user data by setting the appropriate content type and employing a retry policy. Detailed error handling is provided for different types of network errors, including authentication, network, timeout, and server errors, which enhances the robustness of the network communication. Moreover, the activity provides user feedback in case of validation or network errors without exposing sensitive information, thus maintaining a secure user experience. Additionally, the application employs pgcrypto in pgAdmin for encrypting data at the database level, adding an extra layer of security for stored sensitive information. Regarding the API security our project incorporates measures to ensure the integrity and confidentiality of user data and API interactions. The backend API uses JSON Web Tokens (JWT) for secure authentication, as implemented in the index.py file. JWTs are generated during user login and validated for protected routes, ensuring that only authenticated users can access certain endpoints. The tokens include an expiration time to limit their validity period, enhancing security. Additionally, parameterized queries are used extensively in db.py for all CRUD operations. These queries, implemented in functions such as login, add_user, and change_password, prevent SQL injection attacks by safely handling user inputs when interacting with the PostgreSQL database. The use of pgcrypto for password hashing and verification is another critical security feature. Within the db.py file, pgcrypto functions ensure that passwords are stored securely in the database, adding a layer of encryption to user credentials. These combined practices, including JWT-based authentication, parameterized queries, and secure password hashing with pgcrypto, contribute to a secure app.

4.6. Prototype

One of the first staged of the app development process was the visual prototype. We used Figma to try and figure out what our app should look like. We can see that at first glance most things are similar but there have been design changes throughout the development process.

11.Figma Prototype



4.7. Adapters

4.7.1 Home_RecyclerViewAdapter.kt

The Home_RecyclerViewAdapter class in the SideChef Android application is designed to display a list of recipes in a RecyclerView efficiently, managing recipe data and providing an interactive user interface. It uses a ViewHolder class to hold references to key views like recipeImage, recipeTitle, recipeType, recipeTime, and favoritedIcon. The onCreateViewHolder method inflates the layout for each recipe item, while onBindViewHolder binds the recipe data to these views, decodes Base64 encoded images for display, and sets click listeners for the item and favorite icon. Clicking an item invokes onItemClick, and clicking the favorite icon invokes onImageClick and changes its color. The getItemCount method returns the total number of recipes. Error handling ensures stability during image decoding. This adapter enhances user experience by providing a practical way to browse and manage recipes.

4.7.2 MyRecipes_RecyclerViewAdaptor.kt

The MyRecipes_RecyclerViewAdapter class within the SideChef Android application serves to efficiently display a list of saved recipes in a RecyclerView. It manages recipe data and facilitates user interaction. The adapter extends RecyclerView.Adapter and utilizes a ViewHolder class to manage references to essential views such as recipeImage, recipeTitle, recipeType, recipeTime, and favoritedIcon. The onCreateViewHolder method inflates the layout for each recipe item, while onBindViewHolder binds the recipe data to these views, including decoding Base64 encoded images for display. Click listeners are set for both the item and the favorite icon, with the former invoking onItemClick and the latter invoking onImageClick. Clicking the favorite icon also removes the corresponding recipe from the list and updates the RecyclerView accordingly. The getItemCount method determines the total number of saved recipes. This adapter enhances user experience by providing a practical means to view and manage saved recipes.

4.7.3 Search_RecyclerViewAdapter.kt

The Search_RecyclerViewAdapter class in the SideChef Android app facilitates the presentation of recipes in a RecyclerView, enhancing the user's browsing experience. It extends RecyclerView.Adapter and utilizes a ViewHolder to manage views such as recipeImage, recipeTitle, recipeType, recipeTime, and favoritedIcon. During onCreateViewHolder, the adapter inflates the layout for each recipe item, and onBindViewHolder binds the recipe data to these views, including decoding Base64 encoded images for display. Click listeners are set for both the item and the favorite icon, with the former invoking onItemClick and the latter invoking onImageClick. Clicking the favorite icon also changes its color to indicate selection. The getItemCount method determines the total number of recipes in the list. This adapter contributes to a smooth and engaging user experience by enabling users to browse and interact with recipes effortlessly.

4.8 API files

4.8.1 ChangeUserData.kt

The ChangeUserData class in the SideChef Android app is a data model used for representing user data changes. It is annotated with @SerializedName to specify the JSON keys corresponding to each property. This data class includes properties for the user's ID, name, username, and email. By utilizing this class, the app can easily serialize and deserialize user data when communicating with the backend API, ensuring smooth data exchange and consistency between the app and server.

4.8.2 GetRecipesResponseData.kt

The GetRecipesResponseData class in the SideChef Android app serves as a data model for representing recipe information obtained from the backend API. Annotated with @SerializedName to map JSON keys to corresponding properties, this data class encapsulates various details of a recipe, including its ID, name, preparation instructions, preparation time, type, picture (encoded as a string), and ingredients. By using this class, the app can seamlessly deserialize JSON responses from the server into Java objects, facilitating the display and utilization of recipe data within the application.

4.8.3 GetSavedRecipesResponseData.kt

The GetSavedRecipesResponseData class in the SideChef Android app functions as a data model designed to represent saved recipe information retrieved from the backend API. Annotated with @SerializedName to map JSON keys to corresponding properties, this data class encapsulates various details of a saved recipe, including its ID, name, preparation instructions, preparation time, type, picture (encoded as a string), ingredients, and the ID of the original recipe it corresponds to. By utilizing this class, the application can effectively deserialize JSON responses from the server into Java objects, enabling the seamless handling and utilization of saved recipe data within the app.

4.8.4 GetUserProfileData.kt

The GetUserProfileData class within the SideChef Android app serves as a data model tailored to represent user profile information fetched from the backend API. Annotated with @SerializedName to facilitate the mapping of JSON keys to corresponding properties, this data class encapsulates essential user details such as their ID, name, email, and username. By leveraging this class, the application can efficiently deserialize JSON responses received from the server into Java objects, enabling seamless integration and utilization of user profile data throughout the app's functionality.

4.8.5 LoginResponseData.kt

The GetUserProfileData class within the SideChef Android app serves as a data model tailored to represent user profile information fetched from the backend API. Annotated with @SerializedName to facilitate the mapping of JSON keys to corresponding properties, this data class encapsulates essential user details such as their ID, name, email, and username. By leveraging this class, the application can efficiently deserialize JSON responses received from the server into Java objects, enabling seamless integration and utilization of user profile data throughout the app's functionality.

4.9 Data Classes

4.9.1 UserData.kt

The UserData data class within the SideChef app's DataClasses package serves as a representation of user-related data. With nullable properties for the user's ID, name, email, and username, this data class is designed to encapsulate user information retrieved from various sources within the application. By employing such a data structure, the app can efficiently manage and manipulate user data throughout its functionality, facilitating tasks such as user profile management and interaction with backend services.

5. Conclusion

By examining SideChef in detail, we highlight its features that enrich users' gastronomic experience. This application is not limited to presenting recipes, but offers additional features that make the journey in the kitchen easier. Advanced search makes it easy to find specific recipes, while a detailed display of ingredients and preparation steps provides clear, precise guidance for each dish. The ability to organize favorite recipes is another valuable feature, allowing users to easily save and access their favorites. The intuitive interface makes navigating the application simple and accessible, even for less experienced users.

While SideChef doesn't offer features like how-to videos or a built-in cooking community, its emphasis on organization, search, and easy access to recipes makes for a more effective and enjoyable user experience.

Developing the SideChef project was a stimulating and rewarding experience for our team. During the development process, we delved into the world of digital gastronomy, exploring creative ways to make the application not only functional, but also engaging and inspiring for users. The challenge of creating a platform that meets the needs of culinary enthusiasts around the world was something that motivated us from the beginning. Investigating user preferences, understanding culinary trends and designing innovative solutions to simplify the kitchen experience was incredibly interesting and enriching.

Furthermore, contributing to an application that can be part of people's daily lives, making their culinary experience easier and more enjoyable, brought a sense of purpose to the project. Knowing that our work can help users discover new recipes, improve their cooking skills and explore different gastronomic cultures is extremely rewarding. Therefore, the development of SideChef was not just a technical exercise, but also an opportunity to apply our interest in mobile application development.

6. Bibliography

https://vercel.com/

[illegible]