

RestoConnect: Restaurant REST API

IT325 Web Services Final Project

by

Mariem Ferchichi

January 2024

BA/IT Junior Student



Business Analytics Major

Information Technology Minor

Tunis Business School

Ben Arous, TUNISIA.

2023-2024

Declaration of Academic Ethics

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report.

I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: January 20th, 2024

Mariam Ferchichi

Contents

Abstract	4
1 Introduction	6
2 Explanation of the work carried out	7
2.1 Development Thought process	7
2.2 JWT Contribution	8
2.3 HTML/CSS Contribution	8
2.4 All the HTTP Methods used	8
2.4.1 GET Requests	9
2.4.2 POST Requests	11
2.4.3 PUT Requests	13
2.4.4 DELETE Requests	14
2.5 API Testing	14
2.5.1 Postman Contribution	14
2.5.2 Additinal Endpoints	14
2.6 Git Contribution	15
2.7 Database Structure	16
2.7.1 Tables	16
2.7.2 Unified Modeling Language (UML)	18
2.7.3 Database Migration	19
2.7.4 Configuration	19
3 Challenges	20
4 Evaluation and Results	21

5	Conclusion	22
A	Response examples	23
A.1	GET Requests	23
A.1.1	GET /restaurants	23
A.1.2	GET /restaurants/Menus	23
A.1.3	GET /auth	24
A.2	POST Requests	24
A.2.1	POST /register	24
A.2.2	POST /restaurants	24
A.3	PUT Requests	24
A.4	DELETE Requests	25

Abstract

Adoption of digital technologies has accelerated largely in the last decade and has reached a critical stage today. [1]Tunisia alone is a driving force of the digital economy in North Africa and across the African continent. The digital economy accounts for 11 per cent of Tunisia's gross domestic product, making it one of the strongest and fastest growing sectors in the country. [2]

A domain that is not talked about frequently is the culinary industry. Restaurants and fast food joints contribute greatly to the Tunisian economy as well as represent our country cuisine and culture.

This report delves into the design, development, and functionality of an online Restaurant API, aimed to revolutionize the culinary domain. In response to the increasing demands of a digitized dining system, the API includes multiple features aimed at enhancing operational efficiency, customer engagement, and overall dining experiences for both restaurateurs and patrons. The report comprehensively explores the technical architecture of the Restaurant API, outlining its modular and scalable design. The API's core functionalities include a list of all restaurants and their menus in the database as well as the dishes available in real time. The integration of robust security measures and adherence to industry standards are highlighted to underscore the API's commitment to data privacy and regulatory compliance. As well as a focus on user experience that is maintained through automatic documentation generation using Swagger, offering clarity and accessibility to potential integrators. This Restaurant API report serves as a comprehensive guide for restaurateurs, customers, developers, and industry stakeholders. It not only outlines the technical features of the API but also emphasizes its role in fostering innovation, efficiency, and an enhanced dining experience in an ever-evolving technological world.

Keywords : Digitisation, Restaurant, RESTAPI, python, flask-api, Postman, Swagger, redis, sqlalchemy .

Motivation

In the digital era, the restaurant industry is undergoing a transformative change in response to the changing of consumer behaviors and expectations. As more patrons seek convenience and personalization of their dining experience, the development of an online restaurant API becomes not only a strategic move but a necessity. As a student myself, I find the idea of finding restaurants within my budget near my residence extremely tempting as I won't have to waste time personally looking for them. I can look at their menu and choose my meal for the day within the comfort of my own house. I can also discover new restaurants that are not quite well-known or that serve specific meals and cater to specific diets easily.

That is why, this API can be a part of the day-to-day life of the average person, be they a student, a worker or an employer, who doesn't have enough time in their lunch break, or they can be a tourist or simply new to an area and want to see what the nearby restaurants have to offer.

Chapter 1

Introduction

In the culinary industry, the integration of technology has become a keystone for success. As restaurants strive to meet the dynamic demands of today's digital-savvy customers, the role of Application Programming Interfaces (APIs) has become critical for connectivity, innovation, and has improved user experiences.

This report delves into the technical details of the development of a Restaurant API, an innovative solution designed to enhance customer experience with the Tunisian restaurants, make the dining experience more enjoyable, as well as advertise local food market and introduce tourists to the Tunisian cuisine.

My final Project for the IT325 Web Services course this semester consists of an online Restaurant RESTful API developed using Flask- A python Micro-web framework and other additional packages described below in Technology Stack Section. I have also used technologies such as Postman, VSCode, and Git-Version Control to maximize the project's quality.

This projects aim is to facilitate the information sharing of the Tunisian restaurant data in a secure manner to help the economy as well as tourism.

Chapter 2

Explanation of the work carried out

2.1 Development Thought process

This project was developed using **Flask** - a python micro web framework. I used a micro service architecture style as it involves developing a software system as a collection of small, independent, and loosely coupled services. Each microservice is responsible for a specific business capability and communicates with others through well-defined APIs. This promotes scalability, maintainability, and ease of development as well as the independent deployment of services.

For documentation, I used **Swagger** as it helps in maintaining up-to-date and consistent API documentation.

I used a **Test Driven Development** where tests are written before the code to catch errors easily and early in the coding process.

I also used **RedisDB** as a catching layer to improve read efficiency. Redis is an in-memory data store that can be used as a cache, message broker, and more. By storing frequently accessed data in memory, it reduces the need to retrieve data from slower storage systems like databases.

2.2 JWT Contribution

JSON Web Token is a proposed Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a public/private key. [3]

I have used JSON Web Token authentication to secure the API access. Implementation code :

```
from flask import Flask, render_template, request, jsonify, session
from flask_session import Session
import psycopg2
import jwt
from uuid import uuid4
from router import router

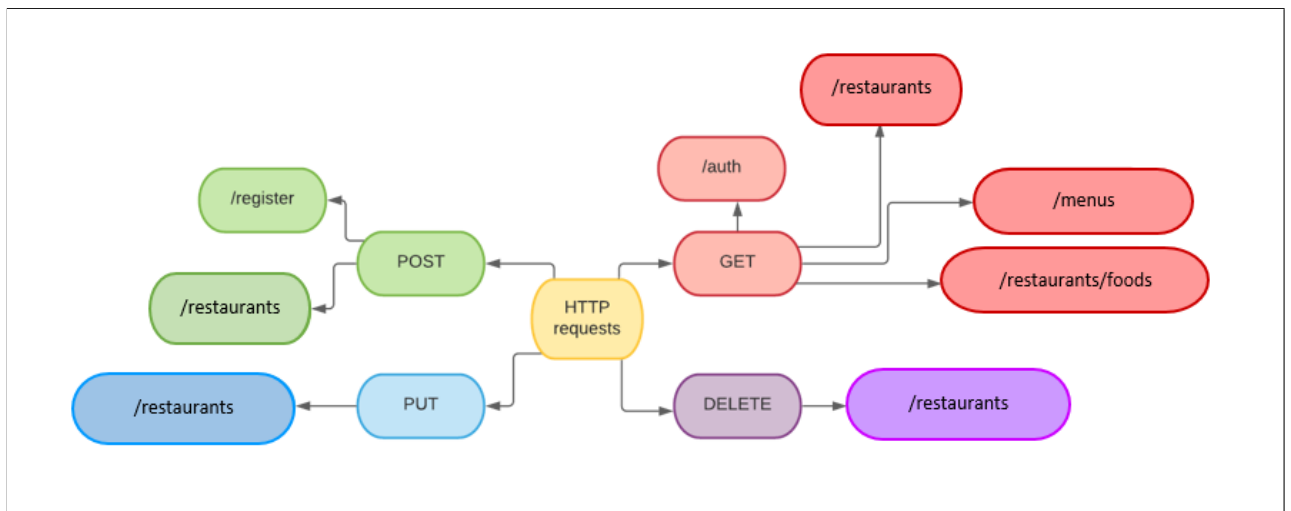
app = Flask(__name__)
app.config['SESSION_TYPE'] = 'filesystem'
app.config['SECRET_KEY'] = str(uuid4())
Session(app)
```

2.3 HTML/CSS Contribution

I used HTML and CSS to design the interactive web pages used in my app.

2.4 All the HTTP Methods used

The figure below shows all HTTP requests used in this API.



2.4.1 GET Requests

GET /restaurants

This method returns all the restaurants available in the database in JSON format.

```
from flask import request
from flask_restful import Resource
from model import db, redis_cache, RestaurantModel, RestaurantSchema
from Constants import RESTAURANT_LIST
import json
restaurants_schema = RestaurantSchema(many=True)
restaurant_schema = RestaurantSchema()
from ast import literal_eval

TAG = "Restaurant"

class RestaurantResource(Resource):
    def __init__(self):
        self.tag = "RestaurantResource"

    def get(self):
        if redis_cache.exists(RESTAURANT_LIST):
            print("[%s.%s]Getting Restaurant Data from redis Cache"%(TAG,self.tag))
            restaurants = redis_cache.__getitem__(RESTAURANT_LIST)
            restaurants = literal_eval(restaurants.decode('utf8'))
        else:
            print("[%s.%s]Getting Restaurant Data from sqlite db"%(TAG,self.tag))
            restaurants = RestaurantModel.query.all()
            restaurants = restaurants_schema.dump(restaurants).data
            redis_cache.__setitem__(RESTAURANT_LIST,restaurants)

        return {'status': 'success', 'data': restaurants}, 200
```

GET /menus

This method returns all the menu available in the database in JSON format.

```
from flask import jsonify, request
from flask_restful import Resource
from model import db, redis_cache, MenuModel, RestaurantModel, MenuSchema
from Constants import MENU_LIST
menus_schema = MenuSchema(many=True)
menu_schema = MenuSchema()
from ast import literal_eval

TAG = "Menu"

class MenuResource(Resource):
    def __init__(self):
        self.tag = "MenuResource"

    def get(self):
        if redis_cache.exists(MENU_LIST):
            print("[%s.%s]Getting Menu Data from redis Cache"%(TAG,self.tag))
            menus = redis_cache.__getitem__(MENU_LIST)
            menus = literal_eval(menus.decode('utf8'))
        else:
            print("[%s.%s]Getting Menu Data from sqlite db"%(TAG,self.tag))
            menus = MenuModel.query.all()
            menus = menus_schema.dump(menus).data
            redis_cache.__setitem__(MENU_LIST,menus)

        return {"status":"success", "data":menus}, 200
```

GET /foods

This method returns all the foods available in the database in json format.

```
from flask import jsonify, request
from flask_restful import Resource
from model import db, redis_cache, FoodModel, RestaurantModel, FoodSchema
from Constants import FOOD_LIST
from ast import literal_eval
TAG = "Food"
foods_schema = FoodSchema(many=True)
food_schema = FoodSchema()

class FoodResource(Resource):
    def __init__(self):
        self.tag = "FoodResource"

    def get(self):
        if redis_cache.exists(FOOD_LIST):
            print("[%s.%s]Getting Food Data from redis Cache"%(TAG,self.tag))
            foods = redis_cache.__getitem__(FOOD_LIST)
            foods = literal_eval(foods.decode('utf8'))
        else:
            print("[%s.%s]Getting Food Data from sqlite db"%(TAG,self.tag))
            foods = FoodModel.query.all()
            foods = foods_schema.dump(foods).data
            redis_cache.__setitem__(FOOD_LIST,foods)

        return {"status":"success", "data":foods}, 200
```

GET /auth

Generate a JWT authentication token.

```
# LOGIN (get a token)
@app.route('/auth', methods=['GET'])
def authenticate():
    username = request.args.get('username')
    password = request.args.get('password')

    try:
        cursor = pool.cursor()
        cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
        verify = cursor.fetchone()

        if verify is None:
            return jsonify(f"There is no user with the username {username}")

        stored_password = verify[1]

        if password == stored_password:
            user = {
                'username': username,
                'pw': password
            }
            token = jwt.encode({'user': user}, 'secretkey', algorithm='HS256').decode('utf-8')
            session['token'] = token # Store token in session
            return jsonify({'token': token})
        else:
            return jsonify(f"Password for user {username} is incorrect")

    except Exception as e:
        print(str(e))
        return jsonify("Error occurred during authentication")
```

2.4.2 POST Requests

POST /register

Register in the API after specifying a valid username and password in the body of the request.

```
# REGISTER
@app.route('/register', methods=['POST'])
def register():
    username = request.json.get('username')
    password = request.json.get('password')

    try:
        cursor = pool.cursor()
        cursor.execute("INSERT INTO users(username, pw) VALUES (%s, %s)", (username, password))
        pool.commit()
        return jsonify(f"User {username} added successfully.")
    except Exception as e:
        print(str(e))
        return jsonify("Error occurred during registration")

    finally:
        cursor.close()
```

POST /restaurants

This method posts a new restaurant and accept application/JSON format for the operation with "name" as the only and the required parameter for the JSON.

```

def post(self):
    json_data = request.get_json(force=True)
    if not json_data:
        | | return {'message': 'No input data provided'}, 400
    # Validate and deserialize input
    data, errors = restaurant_schema.load(json_data)
    if errors:
        | return errors, 422
    restaurant = RestaurantModel.query.filter_by(name=data['name']).first()
    if restaurant:
        | return {'message': 'Restaurant already exists'}, 400
    restaurant = RestaurantModel(
        | name=json_data['name']
        | )

    db.session.add(restaurant)
    db.session.commit()

    result = restaurant_schema.dump(restaurant).data

    return { "status": 'success', 'data': result }, 201

```

POST /menus

This method posts a new menu and accept application/JSON format for the operation with "name" and "restaurantId" as the required parameter for the JSON.

```

def post(self):
    json_data = request.get_json(force=True)
    if not json_data:
        | | return {'message': 'No input data provided'}, 400
    # Validate and deserialize input
    data, errors = menu_schema.load(json_data)
    if errors:
        | return {"status": "error", "data": errors}, 422
    restaurant_id = RestaurantModel.query.filter_by(id=data['restaurant_id']).first()
    if not restaurant_id:
        | return {'status': 'error', 'message': 'Menu Restaurant not found'}, 400
    menu = MenuModel(
        | restaurant_id=json_data['restaurant_id'],
        | name=json_data['name']
        | )
    db.session.add(menu)
    db.session.commit()

    result = menu_schema.dump(menu).data

    return {'status': "success", 'data': result}, 201

class MenuItemResource(Resource):
    def get(self, id):
        menu = MenuModel.query.filter_by(id=id)
        menu = menu_schema.dump(menu).data
        return {'status': 'success', 'data': menu}, 200

```

POST /foods

This method posts a new food and accept application/JSON format for the operation with "name" and "restaurantId" as the required parameter for the JSON.

```
def post(self):
    json_data = request.get_json(force=True)
    if not json_data:
        | return {'message': 'No input data provided'}, 400
    # Validate and deserialize input
    data, errors = food_schema.load(json_data)
    if errors:
        | return {"status": "error", "data": errors}, 422
    restaurant_id = RestaurantModel.query.filter_by(id=data['restaurant_id']).first()
    if not restaurant_id:
        | return {'status': 'error', 'message': 'food Restaurant not found'}, 400
    food = FoodModel(
        | menu_id=json_data['menu_id'],
        | restaurant_id=json_data['restaurant_id'],
        | name=json_data['name'],
        | description = json_data['description']
    )
    db.session.add(food)
    db.session.commit()

    result = food_schema.dump(food).data

    return {'status': "success", 'data': result}, 201
```

2.4.3 PUT Requests

PUT /restaurants

Same as POST with additional feature of updating the restaurant object too.

```
def put(self):
    json_data = request.get_json(force=True)
    if not json_data:
        | return {'message': 'No input data provided'}, 400
    # Validate and deserialize input
    data, errors = restaurant_schema.load(json_data)
    if errors:
        | return errors, 422
    restaurant = RestaurantModel.query.filter_by(id=data['id']).first()
    if not restaurant:
        | return {'message': 'Restaurant do not exist'}, 400
    restaurant.name = data['name']
    db.session.commit()

    result = restaurant_schema.dump(restaurant).data

    return { "status": 'success', 'data': result }, 204
```

2.4.4 DELETE Requests

DELETE /restaurants

This method deletes the given restaurant if the restaurantId exists.

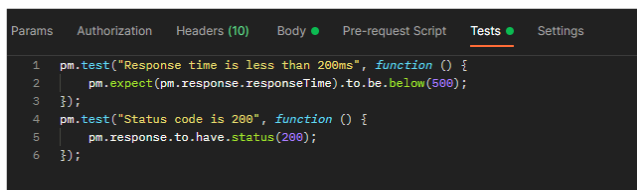
```
def delete(self):
    json_data = request.get_json(force=True)
    if not json_data:
        return {'message': 'No input data provided'}, 400
    # Validate and deserialize input
    data, errors = restaurant_schema.load(json_data)
    if errors:
        return errors, 422
    restaurant = RestaurantModel.query.filter_by(id=data['id']).delete()
    db.session.commit()
    result = restaurant_schema.dump(restaurant).data

    return {"status": 'success', 'data': result}, 204
```

2.5 API Testing

2.5.1 Postman Contribution

Postman is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated. [4] I have used Postman for the automatic testing of my API requests, as well as some snippets such as "Response Time less than 200ms", "Status Code is 200"..etc.



```
1 pm.test("Response time is less than 200ms", function () {
2     pm.expect(pm.response.responseTime).to.be.below(500);
3 });
4 pm.test("Status code is 200", function () {
5     pm.response.to.have.status(200);
6 });
```

2.5.2 Additinal Endpoints

<http://127.0.0.1:5000/api/restaurants/id>

Returns the particular restaurant with id = id if it exists

http://127.0.0.1:5000/api/restaurants/id/foods

Returns all the foods available in the particular restaurant with id = id, if the restaurant it exists

http://127.0.0.1:5000/api/restaurants/id/foods/foodId

Returns the particular food with id = foodId in the particular restaurant with id = id if it exists.

http://127.0.0.1:5000/api/restaurants/id/menus

Returns all the menus available in the particular restaurant with id = id, if the restaurant it exists

http://127.0.0.1:5000/api/restaurants/id/menus/menuId

Returns the particular menu with id = menuId in the particular restaurant with id = id if it exists.

```
api_bp = Blueprint('api', __name__)
api = Api(api_bp)

# Route For Restaurants endpoints
api.add_resource(RestaurantResource, '/restaurants')
api.add_resource(RestaurantItemResource, '/restaurants/<int:id>')
api.add_resource(RestaurantFoodResource, '/restaurants/<int:id>/foods')
api.add_resource(RestaurantFoodItemResource, '/restaurants/<int:id>/foods/<int:foodId>')

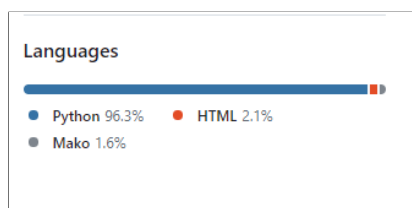
# Route for Menu endpoint
api.add_resource(MenuResource, '/menus')
api.add_resource(MenuItemResource, '/menus/<int:id>')
api.add_resource(MenuFoodResource, '/menus/<int:id>/foods')
api.add_resource(MenuFoodItemResource, '/menus/<int:id>/foods/<int:foodId>')

# Route For Food endpoint
api.add_resource(FoodResource, '/foods')
```

2.6 Git Contribution

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. [5]

I have used Git-Version Control to save changes and coordinate between the different repositories that I had (local and remote).



2.7 Database Structure

The database consists of 4 tables: User, Restaurant, Menu, Food. The relations between them is as follows:

- Each user has multiple restaurants.
- Each restaurant has one menu and multiple food items.
- Each menu has one restaurant and multiple food items.

2.7.1 Tables

Table "Restaurant"

containing 2 columns :

1. id : Restaurant ID
2. name : Restaurant name

Table "Menu"

containing 3 columns :

1. id : Menu ID
2. name : Menu name
3. RestaurantId : Restaurant ID

Table "Food"

containing 6 columns :

1. id : Food item ID
2. name : Food item name
3. description : Food item description
4. creationDate : when was it cooked (default is current time)
5. RestaurantId : Restaurant ID
6. MenuID: Menu ID

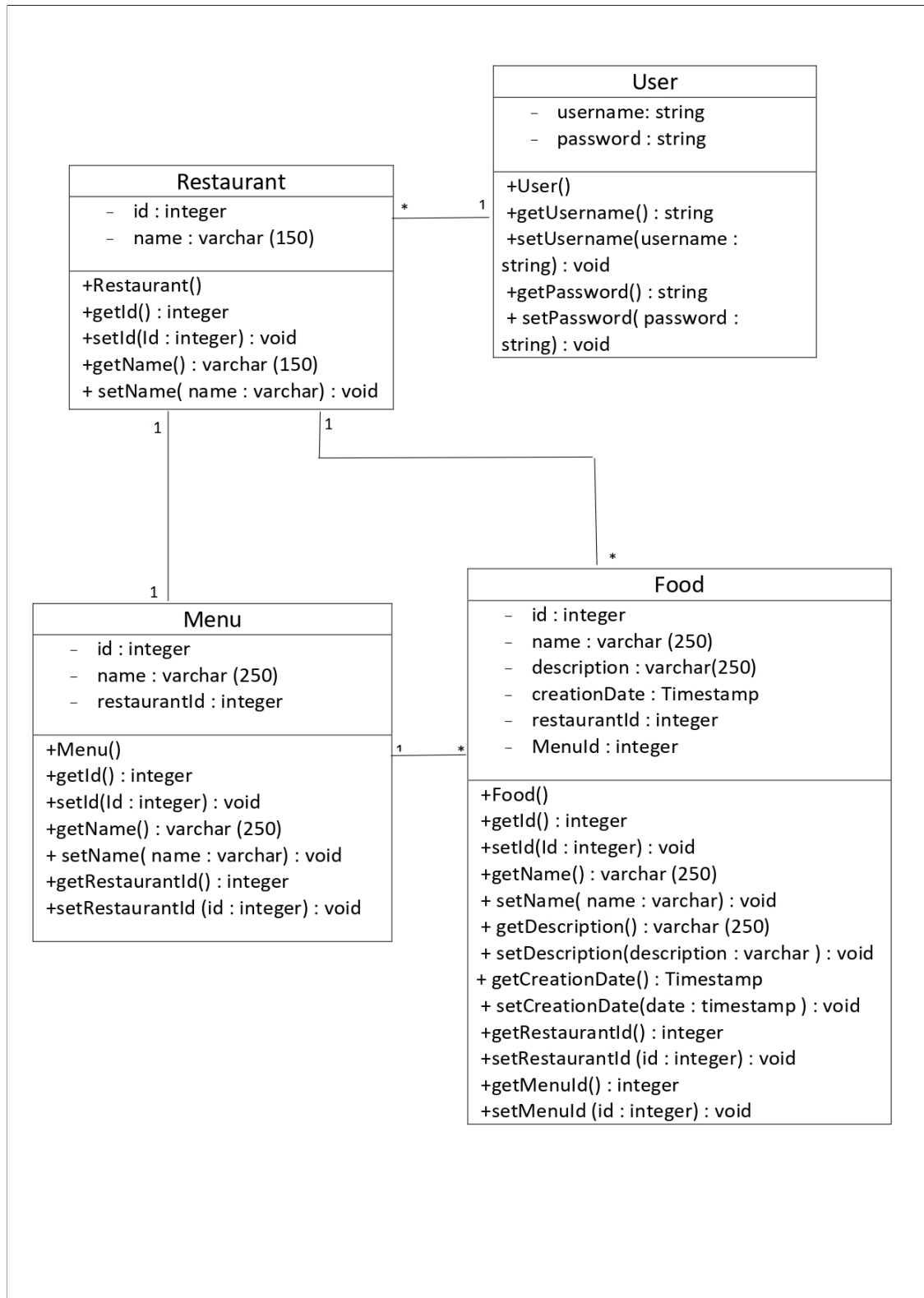
Table "user"

containing 2 columns :

1. username : name of the user.
2. password : password of the user.

2.7.2 Unified Modeling Language (UML)

The figure below shows the UML of the project.



2.7.3 Database Migration

Python code:

```
from flask_script import Manager
from flask_migrate import Migrate, MigrateCommand
from model import db
from run import create_app
from config import PresentConfig

app = create_app(PresentConfig)

migrate = Migrate(app, db)
manager = Manager(app)
manager.add_command('db', MigrateCommand)

if __name__ == '__main__':
    manager.run()
```

2.7.4 Configuration

Python code:

```
class BaseConfig:
    SQLALCHEMY_ECHO = False
    SQLALCHEMY_TRACK_MODIFICATIONS = True
    SQLALCHEMY_DATABASE_URI = "sqlite:///app.db"
    REDIS_HOST = "localhost"
    REDIS_PASSWORD = "61a62a841f54d08ec165faf2ad20544e9f6fb8d37d10bf429b6e9f6e2807e0c4"
    REDIS_PORT = 6379

class DevelopmentConfig(BaseConfig):
    SQLALCHEMY_DATABASE_URI = "sqlite:///dev.db"

class TestingConfig(BaseConfig):
    SQLALCHEMY_DATABASE_URI = "sqlite:///test.db"

class ProductionConfig(BaseConfig):
    SQLALCHEMY_DATABASE_URI = "sqlite:///base.db"

PresentConfig = BaseConfig
```

Chapter 3

Challenges

While developing this API, I faced several problems and I tried to overcome them. Some of these challenges are as follows:

- One of the challenges was passing headers, parameters, and body objects in the browser. I overcame with an extension called ModHeader Chrome Extension.
- Another aspect of the code where I had difficulties is the implementation of Authentication and Authorization mechanisms to secure the API and control access to resources. Choosing the right authentication method (e.g., API keys, OAuth) and managing user permissions is critical. I used industry-standard authentication protocols and implemented token-based authentication. I used a code from the internet to secure the authentication and authorization methods.
- Handling errors effectively and providing meaningful error messages was also quite tricky. Consistent error responses must be defined and communicated. Thus I had to be careful as I implemented standardized error responses and provide detailed error messages.

Chapter 4

Evaluation and Results

A potential development gap in my project lies in the limited functions used as this code can encompass many more features. An additional feature could be ordering and reservations thus allowing customers to place orders or make reservations with utmost ease, irrespective of their physical location and offering a hassle-free experience, eliminating the need for phone calls or in-person visits.

Another enhancement could be the option to add reviews and share your opinion and experience with a particular restaurant.

Further development could be added such as the location and the path to reach the restaurants as well as the estimated time of arrival.

There is also the feature of tailored recommendations where by leveraging past customer data, the API can provide personalized recommendations based on past orders, preferences, and dietary restrictions. This level of personalization adds value to the customer experience, making them feel understood and appreciated.

Overall, the project provides a solid foundation for an online restaurant API but leaves room for further development of a comprehensive and delightful experience.

Chapter 5

Conclusion

In summary, this technical report serves as a guide for customers, restaurant owners, developers, and industry stakeholders seeking to leverage the potential of an online Restaurant API. The purpose of this project was to build an API with not only a practical purpose, but also a social and economic one. It also serves a touristic purpose as it would be a helpful tool for both locals and tourists and easy to use for all users looking to discover various local restaurants and try new and traditional dishes. The paper explores different technologies and approaches in the development an easy-to-use API that helps in the exploration of Tunisian restaurants and its delightful cuisine.

Overall, Tunisia is a beautiful country and has a lot to offer and I sincerely hope that through this project I helped bettering our country's future and facilitating our lives through efficient technologies and I hope that that was conveyed throughout my work.

Mariam Ferchichi



Appendix A

Response examples

A.1 GET Requests

A.1.1 GET /restaurants

```
[
  {
    "id ": 25893,
    "name ": "DAR EL JELD ",
  },
  {
    "id ": 247933,
    "name ": "EL WALIMA",
  },
  {
    "id ": 24752,
    "name ": "CAPITOLE",
  },
  ...
]
```

A.1.2 GET /restaurants/Menus


```
{
  "id": 25893,
  "name": "DAR EL JELD",
  "menu_id": 156666,
}
```

A.1.3 GET /auth

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7InVzZXJuYW1lIjoic2FycmEiLCJwdyI6InNhenJhMTIzIn0sImhhdCI6MTY0Mjk1NDgxNiwiZXhwIjozNjQyOTU2MDE2fQ.IRYit-pRnuXxxQtDIv8W0U17QuuXaPOa8pwZxXx-ul8"
}
```

A.2 POST Requests

A.2.1 POST /register

"User mariem added successfully."

A.2.2 POST /restaurants

```
{
  "id": 247933,
  "name": "EL WALIMA",
  "menu_id": "156216"
}
```

A.3 PUT Requests

PUT /restaurants

"the restaurant was successfully updated!"

A.4 DELETE Requests

DELETE /restaurants

" the restaurant was successfully deleted!"

Technology Stack

- **Flask** - Web Microframework for Python.
- **Flask-restful** - Extension for flask for quickly building REST APIs.
- **Swagger** - Automatic Documentation for the REST endpoints.
- **Flask-migrate** - An extension that handles SQLAlchemy database migrations for Flask applications using Alembic.
- **Marshmallow** - A serializer and deserializer framework for converting complex data types, such as objects to and from native Python data types.
- **Flask-sqlalchemy** - This is an extension of flask that add supports for SQLAlchemy
- **Flask-marshmallow** - An integration layer for flask and marshmallow
- **Marshmallow-sqlalchemy** - This adds additional features to marshmallow.
- **Sqlite3** - Database for the project. It comes built in with python.
- **RedisDB** - Key-Value based No-SQL DB to optimize relational database by improving Read by caching data efficiently.
- **Flask-Redis** - An flask extension of RedisPy to use Redis with Python and Flask easily.

Bibliography

- [1] A. B. Youssef, “Digital transformation in tunisia: Under which conditions could the digital economy benefit everyone?,” in *ERF Working Papers Series*, pp. 1–42, Nov 2021.
- [2] N. Schraepel, “Digital innovations create jobs and support transparent and effective public administration,” *GIZ deutsche gesellschaft für internationale zusammenarbeit*, p. 1, 2020.
- [3] “Jwt.” https://en.wikipedia.org/wiki/JSON_Web_Token. [Online; accessed 23-Jan-2022].
- [4] G. Romero, “What is postman api test,” *encora*, p. 1, June 2021.
- [5] “Git.” <https://git-scm.com/>. [Online; accessed 23-Jan-2022].