Tunis Business School
University of Tunis

# Web Service Project's Report

# Tunisian Culinary Chronicles

Realised by: Ranim Sayahi

Senior BA/IT

Academic Advisor : Mr . Montassar Ben Messaoud

Academic Year: 2024-2025

# Abstract

This project focuses on the design and development of a web service dedicated to Tunisian recipes, with the aim to preserve cultural heritage and provide an engaging user experience.
The system contains features about recipe management, filtering by celebrations, seasons or ethnicities but also the sharing of tips and feedbacks to make the user experience more interactive.
Using a Flask-based backend and an embedded database engine, the service is able to support functionalities such as management of user and admin accounts, recipe searches, creation and updates, management of feedbacks and tips related to recipes and recipe details' updating and fetching such as ingredients, celebrations etc..
The project highlights the significance of traditional Tunisian cuisine while leveraging modern technology to promote accessibility and user interaction in order to provide a great user experience.
This work contributes to both cultural preservation and the sharing of our country's culinary specialties with the rest of the world.

# Contents

General Introduction . . . . . 1

**1 Context and Taxonomy** **2**

1.1 Context . . . . . 2

1.2 Taxonomy . . . . . 2

1.2.1 Restful APIs . . . . . 2

1.2.2 Importance of HTTP request methods in Data Management Systems . . 2

1.3 Similar APIs Research . . . . . 3

**2 Design and Methodology** **4**

2.1 Design . . . . . 4

2.1.1 Class Diagram . . . . . 5

2.1.2 Database Design . . . . . 6

2.1.3 Explanation of class diagram relationships . . . . . 9

2.2 Description of Endpoints . . . . . 10

2.2.1 What is an endpoint . . . . . 10

2.2.2 Recipe's endpoints . . . . . 10

2.2.3 User endpoints . . . . . 11

2.2.4 RecipeStep endpoints . . . . . 11

2.2.5 Ingredient endpoints . . . . . 12

2.2.6 RecipeIngredient endpoints . . . . . 12

2.2.7 Celebration endpoints . . . . . 13

2.2.8 RecipeCelebration endpoints . . . . . 13

2.2.9 Ethnicity endpoints . . . . . 14

2.2.10 RecipeEthnicity endpoints . . . . . 14

2.2.11 Season endpoints . . . . . 15

2.2.12 SeasonalRecipe endpoints . . . . . 15

2.2.13 Tip endpoints . . . . . 15

2.2.14 Feedback endpoints . . . . . 16

2.3 External API integration . . . . . 16

2.4 Data Connection . . . . . 16

2.5 Security . . . . . 16

**3 Implementation** **17**

3.1 System Architecture and Code implementation . . . . . 17

3.1.1 **Back-end layer** . . . . . 17

3.1.2 **Testing layer** . . . . . 20

3.1.3 **Front-end layer** . . . . . 20

General Conclusion . . . . . 22

# List of Figures

2.1 Class Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.1 Flow of the project . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

# List of Tables

1.1 Description of HTTP Methods and Their respective Roles . . . . . . . . . . . . 3

2.1 Recipe's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.2 User's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.3 RecipeStep's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.4 Ingredient's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.5 RecipeIngredient's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . 12
2.6 Celebration's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
2.7 RecipeCelebration's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . 13
2.8 Ethnicity's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.9 RecipeEthnicity's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.10 Seasons's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.11 SeasonalRecipes's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . 15
2.12 Tip's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.13 Feedback's Endpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

3.1 RBAC security mechanism . . . . . . . . . . . . . . . . . . . . . . . . . . 18
3.2 Description of error handling scenarios implemented . . . . . . . . . . . . 19
3.3 Description of Successful HTTP requests scenarios implemented . . . . . . . . 19

# General Introduction

Food plays a crucial role in shaping cultural identity, acting as a bridge that connects individuals to their heritage and traditions. Our Tunisian cuisine represents a unique culinary experience with its rich and diverse history, offering a vibrant array of recipes that reflect the essence of its culture, influenced by Mediterranean, Berber, Arab, and European cuisines.
Whoever tries the Tunisian delicacies gets immediately attached to its flavorfulness and Succulence. It is more than just a meal. It's a sensory journey and a heartfelt expression of culture, where you could discover unique recipes from different ethnicities to serve at different occasions and seasons.
However, Despite the rich and diverse history of Tunisian cuisine, many people around the world, particularly non-Arab individuals, are not fully aware of the unique traditional dishes that Tunisia has to offer. This lack of awareness is because there isn't enough resources dedicated specifically towards this subject or references that showcase the depth and variety of our centuries-old gastronomy. Global culinary trends often spotlight more widely known cuisines, and ours is relatively underrepresented.
This project aims to fill this gap by providing an accessible, user- friendly platform that highlights the flavors, ingredients, and cooking techniques that define Tunisian cuisine. It also aims to make people who are not familiar with it, experience our unique roots and enjoy the diversity of its flavors.
The website will serve a bunch of functionalities for users to explore traditional Tunisian recipes fully, with an emphasis on ease of use, accessibility, and engagement. The platform will include recipes going from appetizers, main course to deserts and the ability to connect users with the culinary treasures of Tunisia while preserving its rich gastronomic heritage for future generations.
All of this will be managed by an admin to make sure the content shared on the website is community friendly and serves for the purpose of our project.
Throughout our endeavor, there will be a development of both the front-end and back-end components of the website, utilizing modern web service technologies to create an interactive and user-friendly experience.

# Chapter 1

# Context and Taxonomy

## 1.1 Context

Tunisian cuisine is a vibrant blend of Mediterranean, Arab, and African influences, characterized by its spicy flavors, various spices, and diverse culinary centuries-old traditions.
However, despite the richness of this culture there aren't a lot of resources, besides a few, such as [10] "Our Tunisian Table" or [2]"Nessma Cuisine" that focus solely on Tunisian Cuisine and provide a unique user experience.
That's why this project is going to be a digital platform that aims to create a centralized database of recipes and cultural insights categorized by ethnicities, seasons, and celebrations, making it accessible to users around the world.
This project's primary goal is to not only provide users with access to authentic Tunisian recipes but also to act as a cultural repository that highlights the history and traditions behind the food, ensuring that the essence of Tunisian cuisine is passed down to future generations.

## 1.2 Taxonomy

### 1.2.1 Restful APIs

A[4] **RESTful API** is an application programming interface (API) that follows the design principles of the REST architectural style. REST is short for representational state transfer, and is a set of rules and guidelines about how you should build a web API.
Modern culinary websites around the world, use RESTful APIs in order to provide dynamic, user-friendly and efficient systems that allow users to fetch recipes easily and rapidly.

### 1.2.2 Importance of HTTP request methods in Data Management Systems

[6]HTTP defines a set of request methods to indicate the purpose of the request and what is expected if the request is successful.These methods are used to create a uniform and predictable way for clients and servers to communicate and exchange data over the web. They are characterised by providing standardization, statelesness, resource management etc..

We can list different types of methods:

| HTTP Method | Role |
|---|---|
| GET | Used to retrieve a resource from the server |
| POST | Used to submit data to the server for processing |
| DELETE | Used to delete a resource from the server |
| UPDATE | Used to update a resource on the server |
| PATCH | Used to partly update a resource on the server |

Table 1.1: Description of HTTP Methods and Their respective Roles

In the context of Culinary websites , HTTP methods are very important in terms of managing recipes ensuring easy flow of data throughout the system.

## 1.3 Similar APIs Research

List of similar [7] APIs that work on food and culinary websites:

- **Spoonacular API**: This API works on displaying recipes using specific ingredients or finding calorie details for a recipe.
- **TheMealDB API** :This API works on Providing a random recipe feature for users to discover new dishes.
- **Yummly API** : This API works on offering personalized recipe suggestions

# Chapter 2

# Design and Methodology

## 2.1 Design

In the design phase of the Tunisian recipe management project, the focus was on developing a RESTful API specifically tailored to manage recipe-related data. The API handles various aspects of the system, including recipe management, highlighting the history and traditions behind the food and the sharing of tips and feedbacks. It provides real-time access to recipe information for users and admins, enabling them to
efficiently interact with them. The API's endpoints are designed to allow users and admin to seamlessly interact with the database, each of them having their own role-based permissions. By providing easy access to up-to-date, detailed information, this design ensures that users can quickly discover recipes, share their cultural knowledge, and engage with the platform in meaningful ways. This system supports a more dynamic and efficient platform for celebrating Tunisian culinary heritage and promoting community engagement. Also with the integration of the translation external API, the project becomes all the more inclusive and reaches a global target.

### 2.1.1 Class Diagram

This UML class diagram below represents the classes and the relationships in the model.

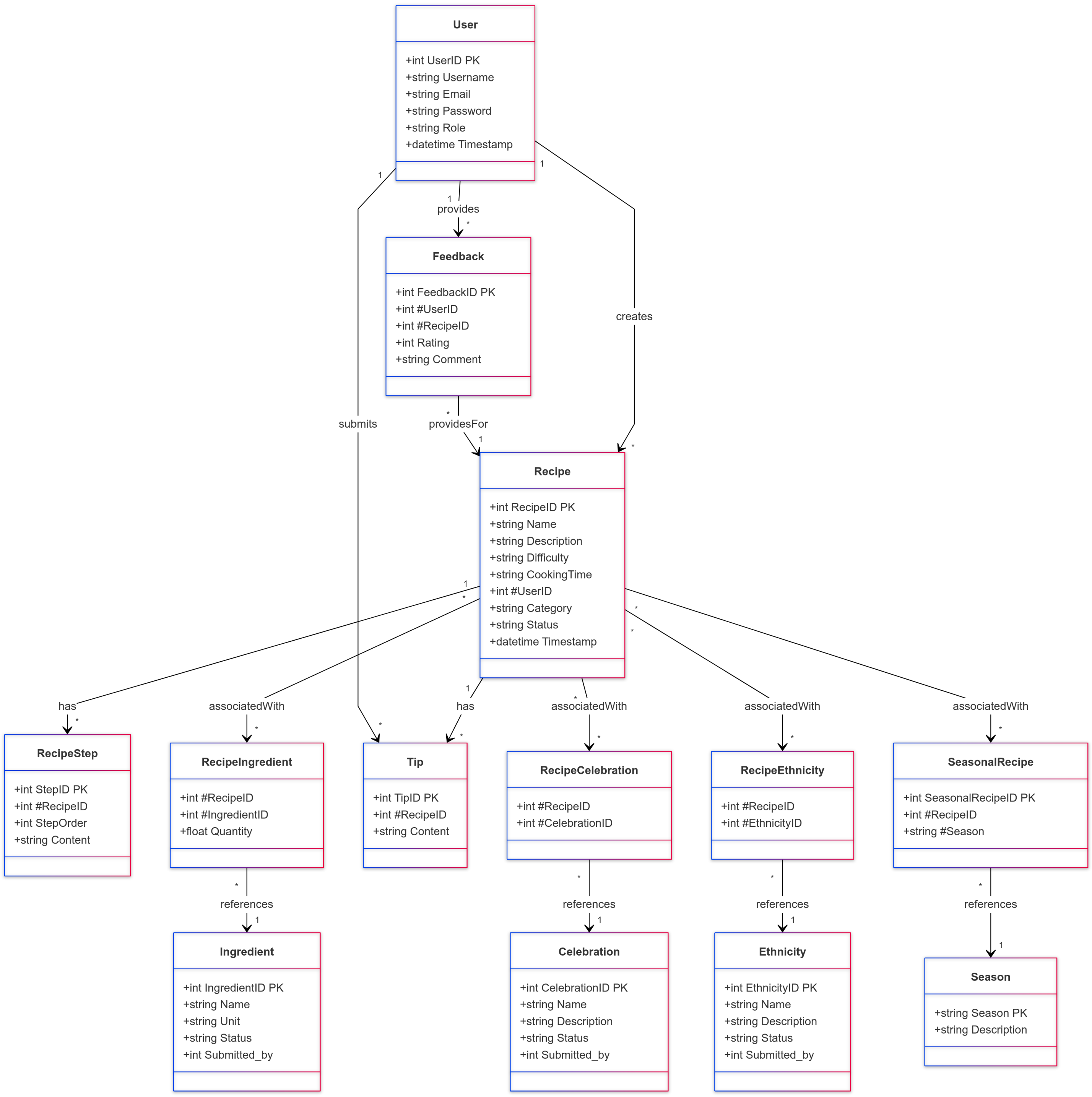

Figure 2.1: Class Diagram

### 2.1.2 Database Design

Like shown in figure 2.1 the database design is the following :

**Recipe Model** :
This model represents recipes, it is a major component and is composed of :

- **RecipeID**:The primary key of the Recipe table.
- **Name**: The name of the recipe.
- **Description**: Description of the recipe.
- **Difficulty**: To represent the difficulty level of the recipe ( must be one of the following: Easy, Medium, Difficult).
- **CookingTime**: The duration the recipe.
- **Category**: Main dish, Dessert or Starter meals.
- **Status**: A string field with a default value of "pending." It indicates the status of the recipe.
- **Timestamp**: The datetime of the creation of the recipe.
- **UserID**: The ID of the user who shared the recipe.(FK)

**RecipeStep Model** :
This model breaks down a recipe into sequential steps, detailing the preparation process and it is composed of :

- **StepID**: The primary key of the RecipeStep table.
- **RecipeID** : The ID of the featured recipe (FK).
- **StepOrder** : The order of the step created.
- **Content** : A text about the content of the step to be shared.

**Ingredient Model** :
This model represents ingredients is composed of :

- **IngredientID**: The primary key of the Ingredient table.
- **Name** : The name of the ingredient.
- **Unit** : The unit of the ingredient.
- **Status** : A string representing the status of the ingredient, it is set to "pending" by default when creating it.
- **Submitted_by** : User who created the ingredient.

**RecipeIngredient Model** :
This model acts as a bridge between recipes and ingredients, defining which ingredients and how much of each are required for a recipe and it is composed of :

- **RecipeID**: Foreign key referencing recipe table.
- **IngredientID** : Foreign key referencing Ingredient table.
- **Quantity** : The quantity of the ingredient to add to recipe.

**User Model** :
This model Represents the users of the platform and it is composed of :

- **UserID**: The primary key of the user table.
- **Username** : A string representing The username of the user.
- **Email** : A string representing The email of the user.
- **Password** : A string representing The password of the user.
- **Role** : Can be either Admin or user.
- **Timestamp** : DateTime of user account creation.

**Tip Model** :
This model provides additional cooking advice or suggestions related to specific recipes, submitted by users and is composed of :

- **TipID**: The primary key of the Tip table.
- **RecipeID** : Foreign key referencing recipe table.
- **Content** : The content of the tip to be shared.

**Celebration Model** :
This model represents cultural or festive occasions and their associated details (There's a predefined list of celebrations that a user can append to it under admin approval.) It is composed of :

- **CelebrationID**: The primary key of the Celebration table.
- **Name** : The name of the celebration.
- **Description** : The description of the celebration to be shared.
- **Status** : A string representing the status of the celebration, it is set to "pending" by default when creating it.
- **Submitted_by** : User who created the celebration.

**RecipeCelebration Model** :
This model links recipes to celebrations, showing which recipes are suited for specific occasions and is composed of :

- **RecipeID**: Foreign key referencing recipe table.
- **CelebrationID** : Foreign key referencing Celebration table.

**Ethnicity Model** :
This model represents the cultural or ethnic origin of recipes, providing background information about their roots and is composed of :

- **EthnicityID**: The primary key of the Ethnicity table.
- **Name** : The name of the Ethnicity.
- **Description** : The description of the Ethnicity to be shared.
- **Status** : A string representing the status of the Ethnicity, it is set to "pending" by default when creating it.
- **Submitted_by** : User who created the Ethnicity.

=> There's a predefined list of ethnicities that a user can append to it under admin approval.
**RecipeEthnicity Model** :
This model links recipes to ethnicities, associating them with specific cultural backgrounds and is composed of :

- **RecipeID**: Foreign key referencing recipe table.
- **EthnicityID** : Foreign key referencing Ethnicity table.

**Season Model** :
This model represents different seasons and is composed of :

- **Season**: The primary key of the season table.
- **Description** : The description of the season to be shared.

=>There's a predefined list of seasons but you can't append to it.

**SeasonalRecipe Model** :
Links recipes to specific seasons, indicating when they are most appropriate to prepare.

- **SeasonalRecipeID**: Primary key of the SeasonalRecipe table.
- **RecipeID**: Foreign key referencing recipe table.
- **Season** : Foreign key referencing season table.

**Feedback Model** :
This model captures user reviews of recipes, including ratings and comments, to provide insights and recommendations and is composed of :

- **FeedbackID**: The primary key of the Feedback table.
- **UserID** : Foreign key referencing user table.
- **RecipeID** : Foreign key referencing recipe table.
- **Rating** : The rating to give to a recipe.
- **Comment** : A comment or Feedback you want to give about the recipe.

=> A user can't make more than one feedback per recipe

Important Note

**Explanation of why we did a seperate class for ingredient and RecipeIngredient (same spirit for celebration, ethnicity and season)** :
A single ingredient (e.g., "Onion") can appear in multiple recipes with different quantities. So, without a separate class, you couldn't track how much of each ingredient is needed for a specific recipe. Also, it avoids duplicating ingredient data for every recipe, making the system more modular and efficient.
=> Handling Many-to-Many Relationships and Avoiding Data Redundancy

**Explanation of why we did not create a seperate class for RecipeStep:**
Steps are unique to recipes unlike ingredients or categories and don't necessarily require reusability. So RecipeStep already captures all necessary attributes (e.g., order, content, and recipe association) that's why separating them would add complexity without adding significant value.

### 2.1.3 Explanation of class diagram relationships

**User "1" –> "*" Recipe**: A user can create multiple recipes, but each recipe is submitted by a single user.

**Recipe "" –> "" RecipeIngredient**: A recipe can have multiple ingredients, and an ingredient can be used in multiple recipes, enabling the many-to-many relationship.

**RecipeIngredient "*" –> "1" Ingredient**: Each recipe ingredient entry references a single ingredient to specify its details and quantity.

**Recipe "1" –> "*" Tip**: A recipe can have multiple tips to guide users, providing additional insights for its preparation.

**Recipe "" –> "" RecipeCelebration**: A recipe can be associated with multiple celebrations, and a celebration can feature multiple recipes, establishing a many-to-many relationship.

**RecipeCelebration "*" –> "1" Celebration**: Each entry links a single recipe to a single celebration.

**Recipe "" –> "" RecipeEthnicity**: A recipe can have multiple ethnic origins, and an ethnicity can encompass multiple recipes, establishing a many-to-many relationship.

**RecipeEthnicity "*" –> "1" Ethnicity**: Each entry connects a recipe to a single ethnic origin.

**Recipe "" –> "" SeasonalRecipe**: A recipe can be associated with multiple seasons, and a season can include multiple recipes, forming a many-to-many relationship.

**SeasonalRecipe "*" –> "1" Season**: Each entry links a recipe to a single season.

**User "1" –> "*" Feedback**: A user can provide feedback on multiple recipes, but each feedback entry is tied to a single user.

**Feedback "*" –> "1" Recipe**: Multiple feedback entries can be linked to a single recipe, enabling users to review and rate it.

**Recipe "1" –> "*" RecipeStep**: A recipe consists of multiple steps, which define its preparation process in a specific sequence.

**User "1" –> "*" Tip**: Users can submit multiple tips for various recipes, contributing to the platform's knowledge base.

## 2.2 Description of Endpoints

### 2.2.1 What is an endpoint

Endpoints are important aspects of interacting with server-side web APIs, as they specify where resources lie that can be accessed by third party software. Usually the access is via a URI to which HTTP requests are posted, and from which the response is thus expected.

### 2.2.2 Recipe's endpoints

Table 2.1 below represents the recipe's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /recipes | Get a list of all recipes |
| GET | /recipes/<recipeid> | Get details of a specific recipe by ID |
| GET | /recipe/search | Search for a specific recipe based on certain attributes |
| GET | /users/<userid>/recipes | Get all recipes created by a specific user |
| GET | /recipe/<recipeid>/details | Get details of a specific recipe including its ingredients, tips, steps etc.. |
| POST | /recipes | Create a new recipe |
| POST | /recipe/<recipeid>/translate | Translate recipe details |
| PUT | /recipes/<recipeid> | Update an existing recipe |
| PUT | /recipe/<recipeid>/approve | Admin approves creation of a recipe |
| PUT | /recipe/<recipeid>/reject | Admin rejects creation of a recipe |
| DELETE | /recipes/<recipeid> | Delete a recipe |

Table 2.1: Recipe's Endpoints

### 2.2.3 User endpoints

Table 2.4 below represents the User's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /users | Get all users |
| GET | /profile | Get the logged in user's details |
| POST | /register | User registration |
| POST | /login | User login |
| POST | /logout | User logout |
| UPDATE | /profile | Update a user's credentials |
| DELETE | /profile | Delete a user account |

Table 2.2: User's Endpoints

- **Important note**: It is crucial to note that if a user decides to delete his account all of the content he posted he shared will be automatically deleted too.

### 2.2.4 RecipeStep endpoints

Table 2.3 below represents the RecipeStep's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /recipe/<recipeid>/steps | Get all steps of a specific recipe |
| GET | /recipe/<recipeid>/steps/<stepid> | Get a specific step of a specific recipe |
| POST | /recipe/<recipeid>/steps | Create a new step for a specific recipe |
| PUT | /recipe/<recipeid>/steps/<stepid> | Update a step for a specific recipe |
| DELETE | /recipe/<recipeid>/steps/<stepid> | Delete a step for a specific recipe |

Table 2.3: RecipeStep's Endpoints

### 2.2.5 Ingredient endpoints

Table 2.2 below represents the Ingredient's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /ingredients | Get all ingredients |
| GET | /ingredients/<ingredientid> | Get an ingredient by id |
| GET | /ingredients/search | search for an ingredient by certain attributes |
| POST | /ingredients | Create a new ingredient |
| PATCH | /ingredients/<ingredientid> /approve | approve or reject creation of a new ingredient |
| PUT | /ingredients/<ingredientid> | Update an ingredient |
| DELETE | /ingredients/<ingredientid> | Delete an ingredient |

Table 2.4: Ingredient's Endpoints

### 2.2.6 RecipeIngredient endpoints

Table 2.2 below represents the RecipeIngredient's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /recipe-ingredients | Get all recipes with their associated ingredients |
| GET | /recipe-ingredients/<recipeid> | Get ingredients of a specific recipe |
| GET | /recipe-ingredients/search | Search for recipe ingredients of a specific recipe |
| POST | /recipe-ingredients | add a new ingredient for a specific recipe |
| PUT | /recipe-ingredients/<recipeid>/ <ingredientid> | Update the quantity of an ingredient for a specific recipe |
| DELETE | /recipe-ingredients/<recipeid>/ <ingredientid> | Delete an ingredient from a specific recipe |

Table 2.5: RecipeIngredient's Endpoints

### 2.2.7 Celebration endpoints

Table 2.2 below represents the Celebration's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /celebrations | Get all celebrations |
| GET | /celebrations/<celebrationid> | Get a celebration by id |
| POST | /celebrations | Create a new celebration |
| PUT | /celebrations/<celebrationid> | Update an existing celebration |
| PUT | /celebrations/<celebrationid> /approve | approve creation of a celebration |
| PUT | /celebrations/<celebrationid> /reject | reject creation of a celebration |
| DELETE | /celebrations/<celebrationid> | Delete a celebration |

Table 2.6: Celebration's Endpoints

### 2.2.8 RecipeCelebration endpoints

Table 2.2 below represents the RecipeCelebration's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /recipe-celebrations | Get all recipes with their associated celebrations |
| GET | /recipe-celebrations/<recipeid> | Get celebrations associated with a specific recipe |
| GET | /recipe-celebrations/<celebrationsname> /recipes | Get all recipes associated with a specific celebration |
| POST | /recipe-celebrations | Associate new celebration with specific recipe |
| PUT | /recipe-celebrations/<celebrationsid> /recipes/<recipeid> | Update the association between a celebration and recipe |
| DELETE | /recipe-celebrations/<celebrationsid> /recipes/<recipeid> | Delete a celebration from a specific recipe |

Table 2.7: RecipeCelebration's Endpoints

### 2.2.9 Ethnicity endpoints

Table 2.2 below represents the Ethnicity's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /Ethnicities | Get all Ethnicities |
| GET | /Ethnicities/<Ethnicitiesid> | Get an Ethnicity by id |
| POST | /Ethnicities | Create a new Ethnicity |
| PUT | /Ethnicities/<Ethnicitiesid> | Update an existing Ethnicity |
| PATCH | /Ethnicities/<Ethnicitiesid> /approve | approve creation of an Ethnicity |
| PATCH | /Ethnicities/<Ethnicitiesid> /reject | reject creation of an Ethnicity |
| DELETE | /Ethnicities/<Ethnicitiesid> | Delete a Ethnicity |

Table 2.8: Ethnicity's Endpoints

### 2.2.10 RecipeEthnicity endpoints

Table 2.2 below represents the RecipeEthnicity's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /recipe-ethnicities | Get all recipes with their associated ethnicities |
| GET | /recipe-ethnicities/<recipeid> | Get ethnicities associated with a specific recipe |
| GET | /recipe-ethnicities/<ethnicitiesname> /recipes | Get all recipes associated with a specific ethnicity |
| POST | /recipe-ethnicities | Associate new ethnicity with specific recipe |
| PUT | /recipe-ethnicities/<ethnicitiesid> /recipes/<recipeid> | Update the association between an ethnicity and recipe |
| DELETE | /recipe-ethnicities/<recipeid> /<ethnicitiesid> | Delete an ethnicity from a specific recipe |

Table 2.9: RecipeEthnicity's Endpoints

### 2.2.11 Season endpoints

Table 2.2 below represents the Season's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /seasons | Get all predefined seasons |
| GET | /seasons/<seasonid> | Get a predefined season by id |

Table 2.10: Seasons's Endpoints

### 2.2.12 SeasonalRecipe endpoints

Table 2.2 below represents the SeasonalRecipe's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /recipes/search | search for a recipe by associated season or name |
| GET | /seasons/<seasonid>/recipes | Get all recipes for a specific season |
| POST | /seasons/<seasonid>/recipes | Create a new season, recipe association |
| DELETE | /seasons/<seasonid>/recipes /<recipeid> | Remove a recipe from a season |

Table 2.11: SeasonalRecipes's Endpoints

### 2.2.13 Tip endpoints

Table 2.2 below represents the Tip's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /tips/<tipid> | Get a specific tip by id |
| GET | /recipes/<recipeid>/tips | Get all tips of a specific recipe |
| POST | /tips | Create a new tip for a recipe |
| PUT | /tips/<tipid> | Update an existing tip |
| PUT | /tips/<tipid>/approve | Admin approves creation of a tip |
| PUT | /tips/<tipid>/reject | Admin rejects creation of a tip |
| DELETE | /tips/<tipid> | Delete a tip |

Table 2.12: Tip's Endpoints

### 2.2.14 Feedback endpoints

Table 2.2 below represents the Feedback's endpoints:

| HTTP method | Endpoint Path | Description |
|---|---|---|
| GET | /feedback/recipe/<recipeid> | Get all feedbacks for specific recipe |
| POST | /feedback | Create a new feedback |
| PUT | /feedback/<feedbackid> | Update an existing feedback |
| DELETE | /feedback/<feedbackid> | Delete a feedback |

Table 2.13: Feedback's Endpoints

## 2.3 External API integration

To enhance the accessibility of the application, the [1]**MyMemory Translation API** was integrated to provide recipe translations. This allows users to translate recipe details, such as names and descriptions, into multiple languages. By leveraging this external API, the system dynamically retrieves accurate translations, ensuring a better user experience for non-native speakers and promoting the cultural diversity of Tunisian recipes to a global audience.

## 2.4 Data Connection

In the data connection phase of the project, I used SQLite as the database management system, integrated within the VSCode environment and [8]SQLAlchemy as the ORM to interact with the SQLite database.
SQLite is a lightweight, serverless, and self-contained relational database engine, ideal for small to medium-scale applications, which aligns well with the project's requirements.

## 2.5 Security

In the development of the project, security was a primary consideration in order to protect user data and ensure the integrity and authenticity of the system. The explanation of the security measures developed are in section 3.1.1

# Chapter 3

# Implementation

## 3.1 System Architecture and Code implementation

The system architecture of the Tunisian Culinary Chronicles project refers to the conceptual model that defines the structure, components, and interactions of the system. It outlines how the different parts work together to achieve a specific goal which is the delivery of a user-friendly platform. The primary components of the project are:

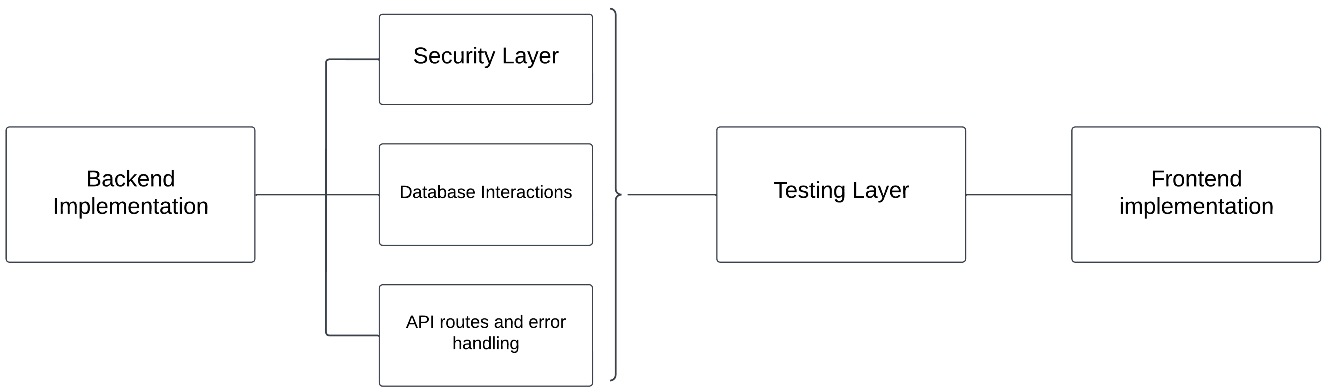

Figure 3.1: Flow of the project

### 3.1.1 Back-end layer

The backend of the project was developed using mainly Python and [3] Flask framework. It is responsible for managing all user-required functions, including database management,ensuring a secure navigation and interaction with the website, and error handling. A virtual environment was also implemented where dependencies for the project were installed. It ensures that the project has its own libraries and versions, independent of system-wide installations or other projects. My project's structure is as follows: -**Models** is a folder that contains the Model structure of all of my classes.
-**Resources** is a folder that contains the endpoints related to each of my classes separately.
-**App.py** serves as the main entry point of the application, handling the initialization of the Flask app, configuration settings, route definitions, and the connection between the backend, database, and other components.
-**Schemas.py** is a file that defines how data is structured and validated.

-**Requirements.txt** lists all the dependencies and libraries required for the project
-**.flaskenv** configures environment variables for a Flask application
-**seed.py** contains input about predefined data to put in my database
-**translator.py** File that contains data about my external API
-**Dockerfile** configures the implementation of my docker container
-**instances/data.db** Contains my sqlite database

- **Database layer**

[9]In this project, I used SQLite as an internal database due to its lightweight, serverless nature, making it ideal for local development and small-scale applications. SQLAlchemy was employed as the ORM (Object Relational Mapper) to interact with the database, enabling seamless integration with Python. This setup eliminated the need for an external database server, simplifying the configuration process and allowing all data to be stored and managed locally within the project directory. Through SQLAlchemy, I efficiently performed CRUD operations while maintaining the structure and relationships of the database models.

- **Security layer**

In this project, the security layer was established to make sure of the protection of data and user interactions. That's why a Role-Based Access Control (RBAC) security mechanism was implemented to restrict access to resources.

| | Admin | User |
|---|---|---|
| See approved content | YES | YES |
| Modify content You shared | YES | YES |
| Delete content You shared | YES | YES |
| See pending and rejected content | YES | NO |
| Modify other users' content | YES | NO |
| Delete other users' content | YES | NO |
| Approve or reject content | YES | NO |
| See all user profiles | YES | NO |
| Modify or delete your profile | YES | YES |

Table 3.1: RBAC security mechanism

The table 3.1 sums up the different permissions each user has. The admin basically has access to all resources and can perform any functionality. However, the user have limited access to certain actions and data

[5]- **JSON Web Tokens (JWT)** were also implemented in this project for authentication and authorization to add another layer of security. The tokens are generated upon User Login. If their credentials (email and password) are valid, the system generates a JWT containing the user's identity and roles (e.g., "user" or "admin"). This token gives the user access to protected Routes once you send JWT in the request's Authorization header.
As mentioned before, there are Role-Based Permissions: Based on the user's role within the token, the backend grants or denies access to specific resources or operations.

- A **Password Hashing** security layer was also implemented to ensure the confidential data (ie. the passwords) of each user are securily stored. **Bycript** was used to ensure the protection against brute force attacks by generating a hash and a salt combination to increase the complexity of decrypting it.

- **Error Handling layer**

In the project, several error handling scenarios were implemented to ensure the program runs smoothly and be more user-friendly offering clear guidance during failure and improving overall stability.

| Error | Description |
|---|---|
| 403 | The user is not authorized to view the resource |
| 500 | Internal server error |
| 404 | Resource not found |
| 409 | A resource with the same name or credentials already exists |
| 401 | Invalid credentials |
| 422 | Shorter than minimum length |

Table 3.2: Description of error handling scenarios implemented

| Successful HTTP requests | Description |
|---|---|
| 200 | Request was successful. |
| 201 | Resource has been successfully created |

Table 3.3: Description of Successful HTTP requests scenarios implemented

### 3.1.2 Testing layer

Testing was conducted using **Insomnia**, a tool designed for API development and debugging. Each API endpoint was rigorously tested to ensure it met the required specifications, including returning correct data formats, handling valid and invalid inputs gracefully, and maintaining consistent performance.
It was also used to test the security implementation by inserting in the Authorization Header the authentication tokens that were passed to the server, verifying the user's identity and granting access to protected routes. Overall, the testing phase ensured that the API was reliable and ready for potential integrations with front-end applications or external services.

### 3.1.3 Front-end layer

A really simple user-friendly interface was implemented by providing a well-designed and interactive frontend, the application allows users to explore Tunisian recipes belonging to diffrent celebrations, ethnicities and seasons.

# Threats to validity

This is Release 1 of the project cause despite the efforts to ensure the quality and accuracy of the outcome, certain threats to validity must be acknowledged.

- **Internal Limitations**
  - The use of a local SQLite database limits scalability for larger, real-world datasets so in future releases the implementation of data migration to an external database must be thought of.
  -Ensuring that feedback data is representative and using appropriate statistical methods for analysis can help ensure robust results.
  -We can in the future implement additional features such as Email Notifications to Notify users when their recipe is approved or rejected etc..
  -Include different other external APIs to make the website all the more resourceful.

- **External Limitations**
  - Due to the short amount of time on hand, the implementation of the frontend was not ideal. Even though, it is able to serve the basic functionalities, it still lacks major improvements in layout to make it more attractive and more captivating.
  -A seperated interface should be implemented for the admin that is not the same as the one of the user

All these issues are going to be addressed in future releases to ensure the improvement of the project and an ideal end result.

# General Conclusion

Tunisian Culinary Chronicles marks a significant step towards preserving and celebrating Tunisia's rich culinary heritage in a digital era. It's a project dedicated to share not only the Tunisian recipes but also demonstrates the power of technology in bridging cultural and linguistic gaps.
By combining a well-structured backend, a simple frontend, this platform makes it easy for users to engage and share. It leverages a robust RESTful API to handle data management and ensures seamless interaction between users and the database. With the additional layer of security, such as RBAC and token-based authentication,the project offers a rich, culturally inclusive and secure experience.
Despite challenges, including the various limitations and the time constraint, the project sets a foundation for future enhancements and releases, both for the backend and frontend.
This project demonstrates the potential of combining technology with cultural and heritage preservation, aiming to connect Tunisian culinary traditions with a global audience while fostering a sense of pride and belonging within the local community.

# Bibliography

[1] MyMemory Translation API. Mymemory - machine translation api, 2025.

[2] Nessma Cuisine. Nessma cuisine - tunisian recipes.

[3] Flask. Flask official documentation.

[4] Red Hat. What is a restful api?

[5] JWT.io. Introduction to json web tokens.

[6] Mozilla Developer Network (MDN). Http methods.

[7] RapidAPI. Food apis collection.

[8] SQLAlchemy. Sqlalchemy documentation.

[9] SQLite. Sqlite documentation.

[10] Our Tunisian Table. Tunisian bread: Rougag, 2018.