Web Programming Project Report

BonApp! A cooking recipes website

Abderraouf MEDJADJ
Bachelor in Computer Science and Mathematics (Magistère)
Université Paris-Saclay

Danilo VULETIC
Bachelor in Computer Science
Université Paris-Saclay

May 1, 2025

Abstract

This project aims to design and develop a bilingual web platform for managing and sharing cooking recipes, using PHP, jQuery (a JavaScript framework), HTML, and CSS. The application supports multiple user roles including cooks, chefs, translators, and administrators, each with specific permissions and responsibilities. Recipes are stored in a semi-complete JSON file and enriched collaboratively through user input. Key features include AJAX-powered dynamic content updates, multilingual support with live translation interfaces, user role management, recipe publishing workflows, and advanced filtering/searching functionalities. The interface is designed to be modern, responsive, and user-friendly, with minimal page reloads and a focus on usability. This report details the technical implementation and architecture of the system, with a particular focus on AJAX interactions and data processing.

Contents

1 Introduction

1.1 Context and Motivation

Web technologies have evolved to support highly interactive and user-friendly applications. This project explores how PHP and jQuery can be combined to build a dynamic website with features such as AJAX-based interactions and structured backend logic.

1.2 Objectives

The primary goal is to design and implement a feature-rich web application while respecting best practices of code modularity, responsiveness, and user experience. Key functionalities include form processing, dynamic content loading, and database (JSON files)/API interaction through AJAX.

1.3 Report Overview

This report is organized as follows: Section 2 describes the tools and technologies used; Section 3 presents the system architecture; Section 4 and 5 detail the implementation and AJAX interactions. The report concludes with testing analysis and future improvement suggestions.

2 Tools and Technologies

This project made use of a variety of tools and technologies to support the full web development lifecycle, including design, implementation, version control, and collaboration.

Programming Languages and Technologies

The core development stack consists of:

- **PHP** for server-side logic and data processing.
- **JavaScript**, enhanced with **jQuery** for client-side scripting and **AJAX** for asynchronous communication.
- **HTML** and **CSS** for structuring and styling the user interface.

Development Environment

- The project was developed primarily using **Visual Studio Code (VSCode)** as the integrated development environment (IDE).

Version Control

- **Git** was used for version control, enabling efficient code tracking, collaboration, and rollback capabilities.

Data Management

- A **JSON** file was used to store and manage recipes and user data, following a structured yet flexible format suitable for incremental enrichment.

Modeling Tools

- **Lucidchart** was used for UML modeling, including system architecture and user flow diagrams.
- **Figma** was employed for UI/UX prototyping and interface design planning.

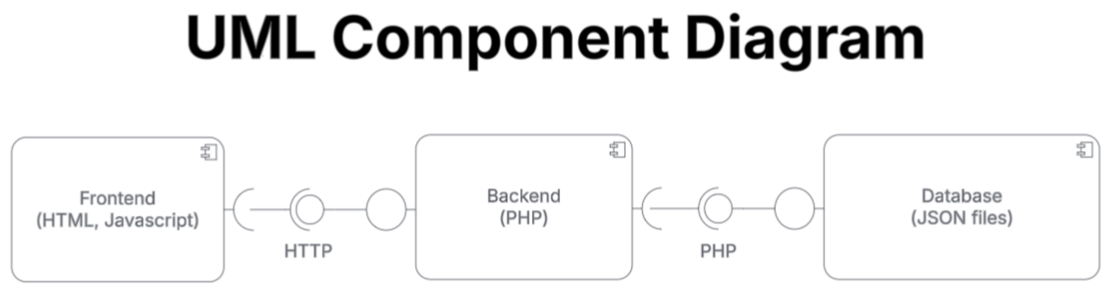
Collaboration and Communication

- Communication and task coordination were facilitated through platforms such as **Discord**, **Google Docs**, and **Google Sheets**, enabling real-time collaboration and shared document editing.

3 System Architecture

The web application follows a modular architecture separating the backend logic, user interface components, static resources, and data storage. The stack is composed of:

- **PHP** for the backend logic and server-side routing.
- **HTML**, **CSS**, and **JavaScript** (with **jQuery** and **AJAX**) for the frontend interface and interactivity.
- **JSON files** used as a lightweight NoSQL-style database for storing recipes and user data.



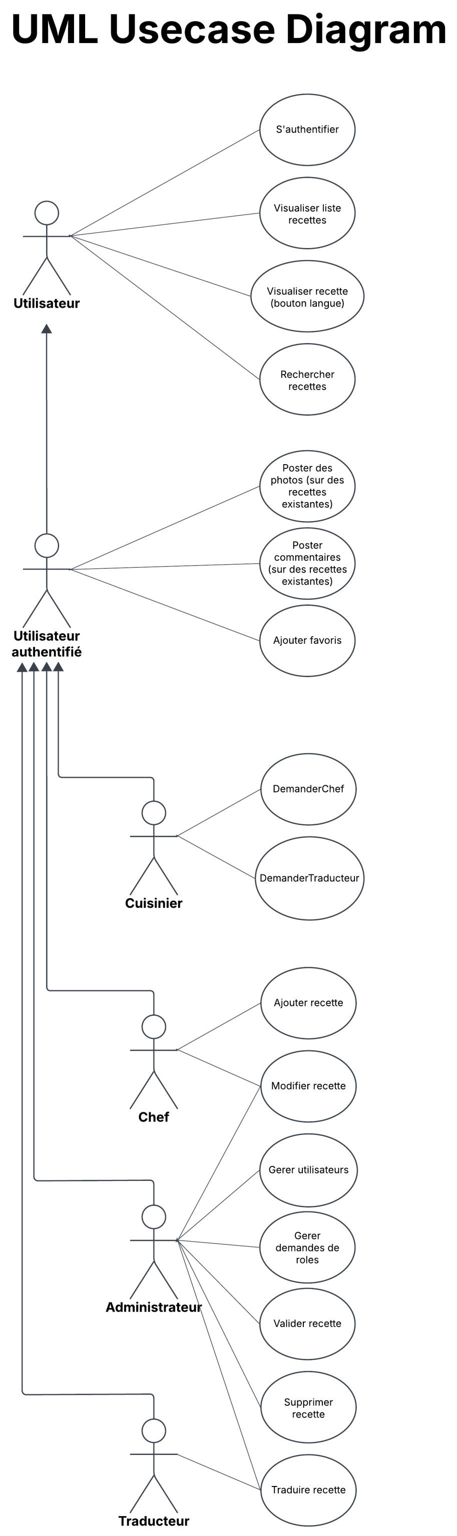
Project Folder Structure

The application is organized into clearly defined folders, as illustrated below:

- **api/** — Contains controller scripts that handle AJAX requests and data processing.

- **assets/images/** — Organized subfolders for storing user-uploaded images, recipe images, and website assets.
- **components/** — Reusable frontend PHP components such as navigation bars, footers, messages, and session handling.
- **css/** — Stylesheets used to design the layout and appearance of the web application.
- **db/** — Contains the ‘recipes.json‘ and ‘users.json‘ files that serve as the main data storage.
- **js/** — JavaScript scripts that enhance frontend behavior.
- **public/** — Contains dynamic view files written in PHP, such as the main homepage and other user-facing interfaces.
- **index.php** — The main entry point for the application.

4 Features and Functionalities



5 Implementation Details

5.1 Backend Logic with PHP

The PHP backend is responsible for processing all business logic, handling form submissions, validating user input, and interacting with the data layer (JSON files). Each type of functionality is handled by a corresponding script in the `api/` folder.

The general processing flow for backend operations follows these steps:

1. **Input Validation:** All user inputs are sanitized and validated on the server side to prevent malicious injection or incorrect data formatting.
2. **JSON File Access:** PHP reads the relevant JSON file (`users.json` and/or `recipes.json`) using built-in functions like `file_get_contents()` and decodes the content with `json_decode()`.
3. **Data Processing:** Based on the action (e.g., login, add recipe, add comment), the PHP script updates the appropriate data structure in memory.
4. **File Overwrite:** If changes are made, the updated data is encoded back into JSON using `json_encode()` and written back to the file using `file_put_contents()`.

5.2 AJAX and jQuery Interactions

To ensure a fluid and responsive user experience, all client-server interactions are handled asynchronously using AJAX requests made via jQuery. These calls interact with the PHP API without reloading the page.

All AJAX calls follow this general pattern:

```
// Template for AJAX Call with jQuery
$.ajax({
    url: 'api/egController.php',
    type: 'POST',
    data: {
        action: 'doSomething',  // Operation to be handled by egController
        param1: value1,
        param2: value2,
        ...
    },
    success: function(response) {
        // Handle response: update DOM, show message, etc.
    },
    error: function(xhr, status, error) {
        console.error('AJAX Error:', error);
    }
});
```

Each AJAX request sends one or more parameters to a PHP controller, which then processes the request and returns a response, often in JSON format or as HTML fragments. This mechanism is used for:

- Authenticating users (signin/signup/signout)
- Fetching, filtering or searching recipes
- Managing a user's favorite recipes
- Managing users in general
- Saving, editing or deleting recipes
- Submitting comments and photos

5.2.1 User Login

Purpose: Authenticate existing users
Inputs: Username and password strings
Output: Redirects to homepage on success, shows error on failure

```
$.ajax({
    url: "http://localhost:3000/api/userController.php",
    method: "POST",
    data: { action: "signin", username: username, password: password }
})
.done(function() { window.location.href = "/"; })
.fail(function(xhr) {
    let err = JSON.parse(xhr.responseText);
    $("#error-message").text(err?.error || "Error occurred");
});
```

5.2.2 User Registration

Purpose: Create new user account
Inputs: Username, email, password (with both client and server-side validation)
Output: Redirects to homepage on success, shows error on failure

```
$.ajax({
    url: "http://localhost:3000/api/userController.php",
    method: "POST",
    data: {
        action: "signup",
        username: username,
        email: email,
        password: password,
        role: "COOK"
    }
})
.done(function() { window.location.href = "/"; })
.fail(function(xhr) {
    let err = JSON.parse(xhr.responseText);
    $("#error-message").text(err?.error || "Error occurred");
});
```

5.2.3 User Logout

Purpose: Terminate user session
Inputs: None (uses server-side session)
Output: Clears local storage and redirects to homepage

```
$.ajax({
    url: "http://localhost:3000/api/userController.php",
    method: "POST",
    data: { action: "signout" }
})
.done(function() {
    localStorage.removeItem('likedRecipes');
    window.location.href = "/";
})
.fail(function(err) {
    console.log(err);
});
```

5.2.4 Load All Recipes

Purpose: Fetch all validated recipes
Inputs: None
Output: Displays recipe cards

```
$.ajax({
    url: "http://localhost:3000/api/recipeController.php",
    method: "GET",
    data: { action: "getall" }
});
```

5.2.5 Search Recipes

Purpose: Find recipes by search query
Inputs: Search query string
Output: Displays matching recipes

```
$.ajax({
    url: "http://localhost:3000/api/recipeController.php",
    method: "GET",
    data: { action: "getbyquery", query: query }
});
```

5.2.6 Filter by Category

Purpose: Get recipes by category
Inputs: Category name string
Output: Displays filtered recipes

```
$.ajax({
    url: "http://localhost:3000/api/recipeController.php",
    method: "GET",
    data: { action: "getbycategory", category: category }
});
```

5.2.7 Get Single Recipe

Purpose: Fetch details for one recipe
Inputs: Recipe ID
Output: Returns recipe JSON data

```
$.ajax({
    url: "http://localhost:3000/api/recipeController.php",
    method: "GET",
    data: { action: "get", id: id }
});
```

5.2.8 Favorite Recipes

Purpose: Manage user favorites
Operations:

- Get favorites: `action: "get"`
- Add favorite: `action: "add"`
- Remove favorite: `action: "delete"`

```
// Example: Add favorite
$.ajax({
    url: "http://localhost:3000/api/favoriteController.php",
    method: "POST",
    data: { action: "add", id_usr: userId, id_rec: recipeId }
});
```

5.2.9 Role Requests

Purpose: Request special user roles
Inputs: User ID and requested role
Output: Success/error message

```
$.ajax({
    url: "http://localhost:3000/api/userController.php",
    method: "POST",
    data: {
        action: "request-role",
        id: userId,
        requested_role: "CHEF"
    }
});
```

5.2.10 Fetch Recipe

Purpose: Get recipe data by ID
Inputs: Recipe ID (string/number)
Output: Populates recipe page or shows error

```
$.ajax({
    url: "/api/recipeController.php",
    type: 'GET',
    data: {action: "get", id: recId},
    dataType: "json",
    success: function(response) {
        recipeData = response;
        createRecipeContent(recipeData);
    },
    error: function() {
        recipeError(recId);
    }
});
```

5.2.11 Get All Users

Purpose: Fetch all users (admin only)
Inputs: None
Output: Array of user objects

```
$.ajax({
    url: "/api/userController.php",
    method: "GET",
    data: { action: "getall" }
});
```

5.2.12 Edit User

Purpose: Update user details
Inputs: User ID, new username, new role
Output: Updated user object

```
$.ajax({
    url: "/api/userController.php",
    method: "POST",
    data: {
        action: "put",
        id: userId,
        username: newName,
        role: newRole
    }
});
```

5.2.13 Delete User

Purpose: Remove user account
Inputs: User ID
Output: Success confirmation

```
$.ajax({
    url: "/api/userController.php",
    method: "POST",
    data: {
        action: "delete",
        id: userId
    }
});
```

5.2.14 Validate Role Request

Purpose: Approve role upgrade request
Inputs: User ID
Output: Updated user object

```
$.ajax({
    url: "/api/userController.php",
    method: "POST",
    data: {
        action: "validate-role",
        id: userId
    }
});
```

5.2.15 Get Recipe by Author

Purpose: Fetch chef's recipes
Inputs: Author user ID
Output: Array of recipe objects

```
$.ajax({
    url: "/api/recipeController.php",
    method: "GET",
    data: {
        action: "getbyauthor",
        author: userId
    }
});
```

5.2.16 Add Recipe

Purpose: Create new recipe
Inputs: FormData with all recipe fields
Output: New recipe ID

```
$.ajax({
    url: "/api/recipeController.php",
    method: "POST",
    contentType: false,
    processData: false,
    data: formData  // Contains action:"add" + all fields
});
```

5.2.17 Validate Recipe

Purpose: Approve pending recipe
Inputs: Recipe ID
Output: Updated recipe object

```
$.ajax({
    url: "/api/recipeController.php",
    method: "POST",
    data: {
        action: "put",
        id: recipeId,
        validated: true
    }
});
```

5.2.18 Get Current Session

Purpose: Check logged-in status
Inputs: None
Output: Session data with userID and role

```
$.ajax({
    url: "/api/userController.php",
    method: "GET",
    data: { action: "getsession" }
});
```

5.2.19 Submit Comment

Purpose: Add a comment to a recipe
Inputs: Recipe ID, user ID, comment text (optional image)
Output: Success/error message, refreshes recipe to show new comment

```
$.ajax({
    url: "/api/recipeController.php",
    type: 'POST',
    data: {
        action: "addComment",
        user_id: userId,
        user_name: userName,
        id: recipeId,
        text: commentText,
        image: commentImageFile  // optional
    },
    contentType: false,
    processData: false,
    success: function(response) {
        // Handle response
        fetchRecipe(recipeId); // Refresh comments
    }
});
```

5.2.20 Get User Session

Purpose: Check if user is logged in
Inputs: None
Output: Returns user ID or null

```
$.ajax({
    url: "http://localhost:3000/api/userController.php",
    method: "POST",
    data: { action: "getsession" }
});
```

5.2.21 Get User Name

Purpose: Fetch username by ID
Inputs: User ID
Output: Returns username string

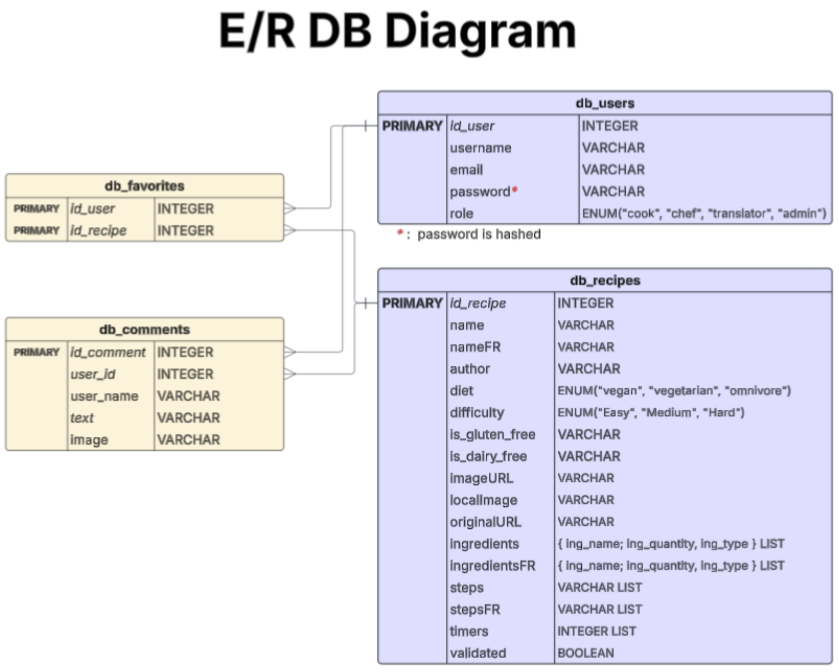
```
$.ajax({
    url: "http://localhost:3000/api/userController.php",
    method: "GET",
    data: { action: "get", id: id }
});
```

5.3 Data Storage and Access

Data is persistently stored in two structured JSON files:

- `users.json` – Contains user credentials, roles, and profile information.
- `recipes.json` – Stores all submitted recipes with fields such as title, ingredients, instructions, language, tags (e.g., gluten-free, vegan), author, photos, and status.

These files are accessed directly by PHP scripts that load the content into associative arrays for manipulation and then write back the changes when necessary. Since the application does not rely on a traditional SQL database, the JSON structure is designed to be both flexible and easily extendable.



6 Testing

Throughout the development of the project, testing was primarily conducted manually. Each feature was tested iteratively as it was implemented, focusing on user interactions, data validation, and dynamic behavior of the interface.

Manual testing included:

- Verifying form submissions and field validations.
- Testing AJAX interactions to ensure correct data retrieval and updates without page reloads.
- Ensuring correct role-based access and functionality for all user types (cooks, chefs, translators, administrators).
- Checking multilingual consistency between the French and English versions.
- Verifying that JSON files were correctly read from and written to.

Automated testing was not implemented due to the scope and timeframe of the project. However, manual testing proved sufficient for validating core functionalities and ensuring a stable user experience.

7 Conclusion

Summary

The project successfully resulted in a fully functional and responsive web application that meets the specifications defined in the initial project brief. All core features were implemented, including multilingual support (French and English), user role management (cooks, chefs, translators, and administrators), dynamic recipe interactions using AJAX, and structured data storage in JSON format. The interface is intuitive, consistent across roles, and minimizes page reloads to enhance the user experience.

Reflection

The development process went smoothly overall, with clear role separation, modular design, and effective use of asynchronous JavaScript to ensure a dynamic and fluid interface. One potential area for improvement is the use of browser-side caching and cookies to persist more client-side state, which could further improve performance and reduce the frequency of JSON reads on the server.

Future Work

Several possible extensions and improvements could be considered for future development:

- Adding support for more languages to expand accessibility beyond French and English.

- Migrating from JSON files to a relational SQL database for better scalability and data integrity.
- Implementing automated testing (unit and integration tests) to ensure robustness.
- Improving the admin dashboard for monitoring and analytics.
- Enhancing accessibility and mobile responsiveness further.

8 References

- Figma — UI/UX Prototyping:
  `https://www.figma.com/design/NyB4LKoGR54vbYt6SXSGQa/BonApp`
- Lucidchart — UML Diagrams (Use Case and Component):
  `https://lucid.app/lucidchart/ffddfb44-5500-45f1-8a3a-26debbbebb6e/edit?`
  `view_items=WS3UijJ6vvip&invitationId=inv_2710e264-e324-4cca-890a-5d5ee3655fc8`
- Google Sheets — API Endpoint Reference Sheet:
  `https://docs.google.com/spreadsheets/d/1467LAE8aKKDLa12HgJG6x6j2FLv8Qg0kXiARXtdkU4s`
  `edit?usp=sharing`
- jQuery AJAX Documentation:
  `https://api.jquery.com/jquery.ajax`
- PHP Manual:
  `https://www.php.net/manual/en/`