Campus Cafeteria Platform Report

TFG Coding Challenge

Prince Moyo

1. Abstract

This report outlines the development of an online platform for a campus cafeteria, aimed at improving planning and determining dish popularity. The project involved creating a REST API for cafeteria workers to manage dishes and a simple interface for students to view and rate them.

2. Introduction

In today's digital era, online platforms have become essential for streamlining operations and enhancing user experiences. The campus cafeteria sought to develop an online cafeteria platform that would allow cafeteria workers to manage dishes effectively while providing students with the ability to view and rate their meals. This report discusses the requirements, design, and implementation of the system, which was developed using C# and ASP.NET Core for the REST API, with Entity Framework Core for database interactions and SQLite as the database.

3. Requirements Captured

Functional Requirements

- **Cafeteria Workers:**
    - Create, update, and delete dishes.
    - Each dish must have a name, description, and price.
- **Students:**
    - View available dishes.
    - Rate dishes on a scale of 1 to 5.

Non-Functional Requirements

- **Performance:** The system should handle multiple simultaneous requests without significant delays.
- **Scalability:** The platform should be able to scale as the number of dishes grows.

4. Explanation of the Most Important Classes and Methods

4.1 Dish.cs

The Dish class is a model representing a dish in the cafeteria. It contains the following properties and methods:

Properties:

- **Id (int):** A unique identifier for each dish.
- **Name (string):** The name of the dish.
- **Description (string):** A detailed description of the dish.
- **Price (decimal):** The price of the dish.
- **Rating (double):** The average rating of the dish.
- **RatingCount (int):** The number of ratings the dish has received.
- **ImageUrl (string):** An optional URL for the dish's image.

Constructor:

- **Dish():** A parameterless constructor for Entity Framework Core.
- **Dish(string name, string description, decimal price, double rating, string? imageUrl = null):** A constructor that initializes a new instance of the Dish class with the specified parameters, setting default values for RatingCount to 0.

4.2 DishesController.cs

The DishesController class is responsible for handling API requests related to dishes. It provides endpoints for creating, updating, deleting, and retrieving dishes.

Key Methods:

- **GetDishes:**
    - **HTTP Method:** GET
    - **Description:** Retrieves a list of all dishes from the database. It returns the dish details to the client.
- **CreateDish:**
    - **HTTP Method:** POST
    - **Description:** Accepts dish details from the client to create a new dish. It saves the dish in the database and returns the created dish.
- **UpdateDish:**
    - **HTTP Method:** PUT

    - **Description:** Accepts updated details for a specific dish and modifies the corresponding record in the database.

- **DeleteDish:**
    - **HTTP Method:** DELETE
    - **Description:** Deletes a specified dish from the database based on the dish ID provided in the request.
- **RateDish:**
    - **HTTP Method:** POST
    - **Description:** Accepts a rating from students, updates the dish's rating and rating count in the database, and recalculates the average rating.

4.3 script.js

The script.js file contains the client-side JavaScript code that interacts with the REST API and updates the user interface dynamically.

Key Functions:

- **fetchDishes(userRole, sortOption = ''):**
    - Fetches dishes from the API and displays them based on the user's role (worker or student). It also supports sorting by rating or price.
- **searchDishes():**
    - Allows students to filter the displayed dishes based on a search input. It hides dishes that do not match the search criteria.
- **showRatingForm(dishId):**
    - Displays a modal form for students to submit their ratings for a specific dish.
- **rateDish(dishId):**
    - Submits the rating for a dish to the API and refreshes the dish list to reflect the new rating.
- **showCreateForm():**
    - Displays a form for cafeteria workers to create a new dish.
- **createDish():**
    - Collects data from the form and sends it to the API to create a new dish.
- **showUpdateForm(id, name, description, price):**
    - Populates the update form with existing dish details for modification.

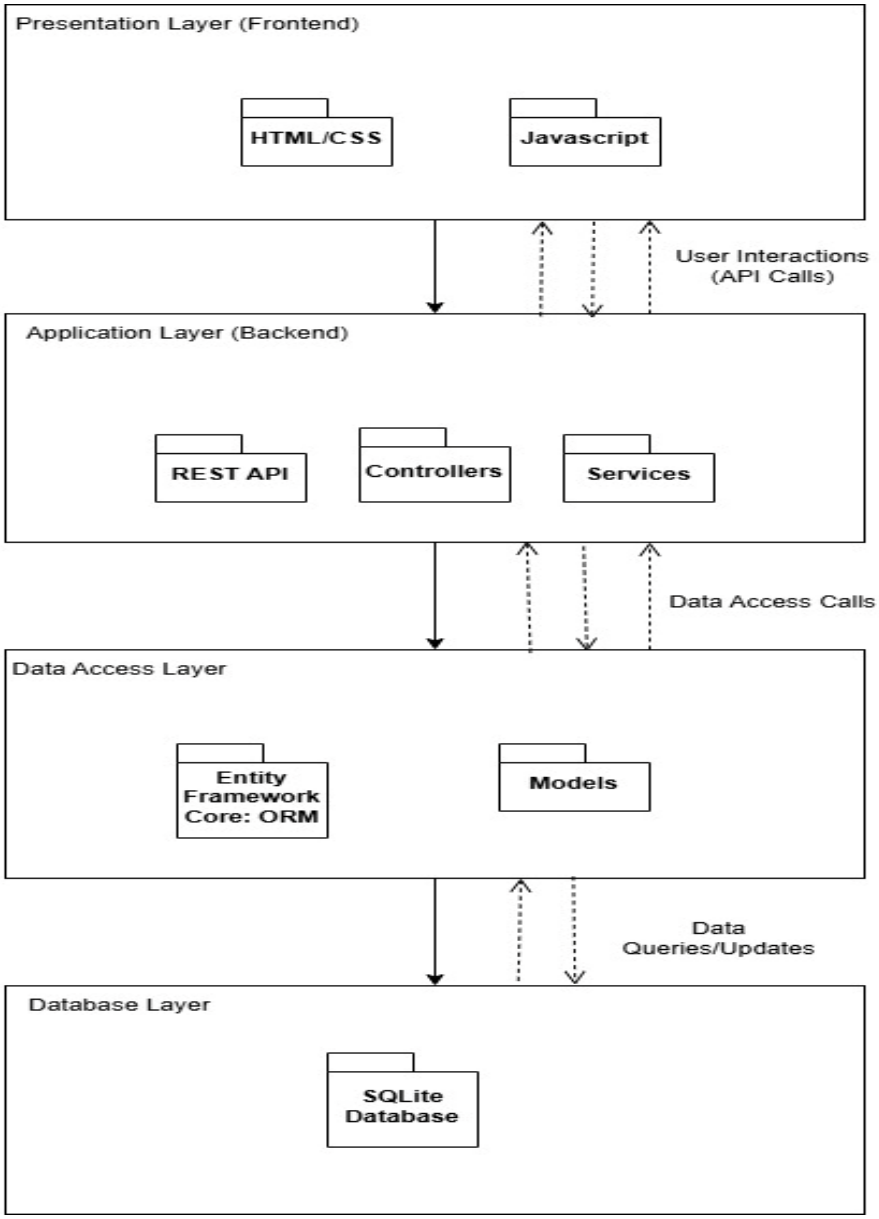
- **updateDish():**
    - Sends updated dish details to the API to save changes.
- **deleteDish(dishId):**
    - Sends a request to delete a specific dish and refreshes the list afterward.
- **sortDishes():**
    - Triggers a fetch for dishes sorted by the selected criteria from a dropdown.
- **logout():**
    - Clears the user role from session storage and redirects to the login page.

5. Technologies Used

- **Programming Languages:** C#.
- **Framework:** ASP.NET Core for building the REST API.
- **Database:** SQLite for data storage.
- **ORM:** Entity Framework Core for database interactions.
- **Frontend:** HTML/CSS and JavaScript for the user interface.

6. Architecture Diagram

A Layered Architecture Diagram illustrates the flow of data between the frontend, backend, and database. It shows how user interactions are handled at each layer of the system.


Presentation Layer (Frontend)
HTML/CSS
Javascript
User Interactions
(API Calls)
Application Layer (Backend)
REST API
Controllers
Services
Data Access Calls
Data Access Layer
Entity
Framework
Core: ORM
Models
Data
Queries/Updates
Database Layer
SQLite
Database