1. Wstęp i cel projektu

Nazwa projektu: OnlineCookbook
Cel: Aplikacja służy do zarządzania przepisami kulinarnymi w formie interaktywnego „notesu kucharskiego” dostępnego online. Użytkownicy mogą:

- przeglądać listę przepisów,
- dodawać własne przepisy,
- edytować i usuwać przepisy,
- zarządzać kategoriami.

Technologie: Projekt wykorzystuje .NET (w wersji 6 lub 7) wraz z Entity Framework Core do komunikacji z bazą danych oraz mechanizmy autoryzacji oparte na ASP.NET Identity (lub dedykowanych tabelach użytkowników). Dodatkowo zaimplementowano WebAPI do obsługi operacji CRUD na głównej encji – przepisie (Recipe).

2. Wymagania i Środowisko Uruchomieniowe

2.1. Wymagania Sprzętowe i Systemowe

- **System operacyjny:** Windows 10/11, macOS lub Linux (z zainstalowanym .NET SDK)
- **RAM:** Minimum 2 GB (zalecane 4 GB lub więcej)
- **Baza danych:** MS SQL Server (lub inny system, po odpowiedniej konfiguracji)

2.2. Wymagania Programowe

- **.NET SDK:** Wersja zgodna z projektem (np. .NET 6 lub .NET 7)
- **Środowisko IDE:** Visual Studio 2022, Visual Studio Code, Rider lub inne kompatybilne z .NET
- **Przeglądarka:** Chrome, Firefox, Edge, Safari – do testowania interfejsu webowego
- **Docker (opcjonalnie):** Jeśli projekt ma wspierać uruchamianie w kontenerze (plik Dockerfile jest dołączony)

3. Instalacja i uruchomienie

3.1. Pobranie i Rozpakowanie

- Rozpakuj archiwum **OnlineCookbook.zip** w wybranym folderze.
- Otwórz projekt w swoim środowisku deweloperskim.

3.2. Konfiguracja Bazy Danych

- W pliku konfiguracyjnym (**appsettings.json** lub **appsettings.Development.json**) znajduje się sekcja ConnectionStrings.

  Przykładowa konfiguracja:

  "ConnectionStrings": {

  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=OnlineCookbookDB;Trusted_Connection=True;"

  }

- W razie potrzeby dostosuj łańcuch połączenia (np. zmień nazwę serwera, bazy danych lub poświadczenia) do używanego systemu bazodanowego.

3.3. Przygotowanie Bazy Danych

- W terminalu, w katalogu zawierającym plik projektu (*.csproj*), wykonaj migracje:

  dotnet ef database update

- Polecenie utworzy lub zaktualizuje strukturę bazy danych zgodnie z modelem projektu.

3.4. Uruchomienie Aplikacji

- Aplikację możesz uruchomić poprzez wpisanie w terminalu:

  dotnet run

- Alternatywnie, uruchom projekt bezpośrednio z poziomu IDE.
- Domyślnie aplikacja dostępna jest pod adresem: https://localhost:7197

4. Konfiguracja Aplikacji i Testowe Dane

4.1. Łańcuch połączenia z bazą danych

Jak wspomniano wyżej, łańcuch znajduje się w pliku konfiguracyjnym appsettings.json. W zależności od środowiska i używanego silnika bazy danych, łańcuch może wymagać modyfikacji (np. nazwy serwera, bazy, poświadczeń).

4.2. Testowi użytkownicy i hasła

- **Administrator**
    - Login: Admin
    - Hasło: Admin123!
- **Przykładowa tabela użytkowników testowych**

Login	Hasło
michal123	Michal1!
mati34	Fgh345!
iwona15	pOjkgh5!
ania47	Tyhgr6!
brat67	gjkFii90!
Fifi99	hHuyfg88!
hubert55	BhJk44@
jan55	Kofg85#
iza333	dhhjT3#
tom98	yrgvD44#

Dzięki temu można od razu zalogować się i testować funkcjonalność w zależności od uprawnień.

5. Opis działania aplikacji z punktu widzenia użytkownika

5.1. Interfejs Użytkownika

- **Strona Główna:** Wyświetla listę dostępnych przepisów oraz menu nawigacyjne, w którym znajdują się linki do:
    - Strony startowej („OnlineCookbook”)
    - Listy kategorii („Categories”)
    - Listy przepisów („Recipes”)
    - Opcji logowania/wylogowania („Zaloguj” / „Wyloguj”)

5.2. Rejestracja / Logowanie

- **Rejestracja**: Nowy użytkownik może się zarejestrować, podając email, hasło i imię/pseudonim.
- **Logowanie**: Po zalogowaniu użytkownik zyskuje dostęp do funkcji związanych z tworzeniem i edycją przepisów. W zależności od roli (User / Administrator) zakres uprawnień może być inny. Administrator ma dodatkowy dostęp do panelu zarządzania

5.3. Formularze

Przykładowe formularze:

1. **Formularz dodawania/edycji przepisu**
    - Encja główna: Recipe (Przepis)
    - Powiązana encja: Ingredient (Składnik) lub Category (Kategoria)
    - Użytkownik może wprowadzić tytuł, opis, ocenę 1-5, kategorię.
2. **Formularz rejestracji**
    - Encja główna: User (Użytkownik)
    - Powiązana encja: UserRole (lub odwołanie do ról w Identity)
    - Użytkownik wprowadza dane logowania
3. **Formularz zarządzania kontem** (dla administratora)
    - Encja: User
    - Powiązana encja: UserRole
    - Administrator może zmieniać role użytkownikom

6. Integracja z Entity Framework Core

- **Model Danych:** Zdefiniowano encje, takie jak:
    - **Recipe** – główna encja przepisów
    - **Ingredient** – składniki przepisu
    - **Category** – kategorie przepisów
    - **User** oraz **Role** – zarządzanie użytkownikami i ich uprawnieniami

- **Kontekst Bazy Danych:**
  Klasa AppDbContext dziedziczy po DbContext i zawiera DbSety dla poszczególnych encji, np.:

```
public class AppDbContext : DbContext {
  public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }
  public DbSet<Recipe> Recipes { get; set; }
  public DbSet<Category> Categories { get; set; }
  public DbSet<Ingredient> Ingredients { get; set; }
  // Inne DbSety...}
```

- **Migracje:**
  Folder **Migrations** zawiera wszystkie pliki migracyjne, które tworzą oraz modyfikują strukturę bazy danych.

7. Autoryzacja Użytkowników

- **Mechanizm:**
  Autoryzacja została zaimplementowana z użyciem ASP.NET Identity lub dedykowanych tabel użytkowników, co umożliwia podział na role.
- **Role:**
  - **Administrator:** Pełne uprawnienia – możliwość zarządzania wszystkimi przepisami oraz użytkownikami.
  - **User:** Uprawnienia ograniczone do zarządzania własnymi przepisami.
- **Konfiguracja:**
  Przykładowa konfiguracja w pliku Program.cs/Startup.cs:

```
builder.Services.AddIdentity<AppUser, IdentityRole>()
  .AddEntityFrameworkStores<AppDbContext>()
  .AddDefaultTokenProviders();

builder.Services.AddAuthorization(options => {
  options.AddPolicy("RequireAdminRole", policy =>
policy.RequireRole("Administrator"));
});
```

- **Ograniczenie Dostępu:**
  W kontrolerach lub widokach stosuje się atrybuty, np.:

```
[Authorize(Roles = "Administrator")]
public IActionResult AdminPanel() {
   // Logika panelu administracyjnego
}
```

8. Interfejs WebAPI

- **Cel:**
  Umożliwienie wykonywania operacji CRUD na głównej encji (Recipe) poprzez dedykowane endpointy.
- **Przykładowe Endpointy:**
  - **GET /api/recipes:** Pobiera listę wszystkich przepisów.
  - **GET /api/recipes/{id}:** Zwraca szczegółowe dane przepisu o wskazanym identyfikatorze.
  - **POST /api/recipes:** Dodaje nowy przepis (dane przesyłane w formacie JSON).
  - **PUT /api/recipes/{id}:** Aktualizuje istniejący przepis.
  - **DELETE /api/recipes/{id}:** Usuwa przepis o podanym identyfikatorze.
- **Bezpieczeństwo:**
  Niektóre operacje mogą być ograniczone do autoryzowanych użytkowników lub wyłącznie administratorów (przy użyciu atrybutów [Authorize]).

9. Scenariusze Testowe

9.1. Testy Logowania i Uprawnień

- **Jako Użytkownik:**
    - Wejdź na stronę logowania (/Account/Login) i użyj danych testowego użytkownika.
    - Sprawdź, czy widoczna jest opcja dodania nowego przepisu oraz możliwość edycji tylko własnych wpisów.
- **Jako Administrator:**
    - Zaloguj się przy użyciu danych administratora.
    - Zweryfikuj dostęp do panelu administracyjnego i możliwość zarządzania wszystkimi przepisami oraz użytkownikami.

9.2. Testy Operacji CRUD

- **Dodawanie Przepisu:**
  Wypełnij formularz dodawania przepisu (tytuł, opis, kategoria, składniki) i zatwierdź. Upewnij się, że przepis pojawił się na liście.
- **Edycja Przepisu:**
  Zmodyfikuj istniejący przepis (np. zmień tytuł lub opis) i zapisz zmiany. Sprawdź, czy aktualizacja została poprawnie zapisana.
- **Usuwanie Przepisu:**
  Usuń wybrany przepis i potwierdź, że nie pojawia się już na liście.
- **Wyświetlanie Szczegółów:**
  Kliknij na wybrany przepis i zweryfikuj, czy widoczne są pełne dane przepisu.

10. Dodatkowe uwagi i możliwe rozszerzenia

- **Walidacja Danych:**
    - W formularzach stosowane są atrybuty walidacyjne (np. [Required], [StringLength], [EmailAddress]), co zapewnia poprawność wprowadzanych danych.
- **Bezpieczeństwo**:
    - Hasła przechowywane w postaci zahashowanej (ASP.NET Identity robi to domyślnie).
    - Zabezpieczenie API (JWT lub cookie authentication).

    - Ochrona przed atakami CSRF (w przypadku formularzy w MVC – domyślna ochrona AntiForgeryToken).
- **Interfejs użytkownika**:
    - Możliwość dodania responsywnego layoutu (Bootstrap, Tailwind, itp.).
    - Obsługa wielu języków interfejsu.
- **Konteneryzacja**:
    - Plik Dockerfile umożliwiający zbudowanie obrazu i uruchomienie aplikacji w kontenerze Docker.
- **Międzynarodowość:**
    - Potencjalne rozszerzenie o obsługę wielu języków interfejsu.