Programowanie zaawansowane: Dokumentacja techniczna projektu

Łukasz Michalak (138325) Piotr Namysł (138328)

Wirtualny Ogrodnik.

Opis:

Aplikacja umożliwia użytkownikom śledzenie pielęgnacji roślin w domu, pozwalając na dodawanie roślin do "wirtualnej hodowli", ustawianie przypomnień o podlewaniu, nawożeniu i przesadzaniu, a także prowadzenie dziennika wzrostu rośliny. Funkcje obejmują: dodawanie roślin ręcznie, wyświetlanie powiadomień o pielęgnacji, dokumentowanie wzrostu rośliny oraz śledzenie zadań związanych z podlewaniem i wzrostem roślin.

Specyfikacja wykorzystanych technologii:

- Platforma: .NET 8

- Backend: ASP.NET Core Web API

- Frontend: Blazor

- Baza danych: PostgreSQL

- Konteneryzacja: Docker

Instrukcje pierwszego uruchomienia projektu:

1. Upewnij się, że masz zainstalowanego dockera
2. Przejdź do katalogu głównego projektu
3. Uruchom skrypt "START_SCRIPT.sh", skrypt ten uruchamia zarówno bazę danych, migrację bazy, backend oraz frontend
4. Aplikacja jest gotowa do działania

Warstwy aplikacji:

1. Frontend (Blazor Server):

   Widoki i interfejs użytkownika, np. formularze dodawania roślin, listy roślin i harmonogramy pielęgnacji.

2. Backend (ASP.NET Core API):

Logika aplikacji i połączenie z bazą danych.

3. Baza danych (PostgreSQL):

   Przechowuje dane użytkowników, roślin, harmonogramów i logów wzrostu.

Główne foldery projektu:

VirtualGardener.Server: Logika backendu, kontrolery API, konfiguracja bazy danych.

I. Controllers: Kontrolery API dla obsługi żądań.
II. Models: Modele danych i encje.
III. Services: Logika biznesowa i interfejsy usług.
IV. Database: Konfiguracja migracji EF Core dla PostgreSQL.
V. Migrations Migracje bazy danych
VI. Utilities Klasy i metody pomocnicze

VirtualGardener.Shared: Modele współdzielone między frontendem a backendem.

I. Models: klasy CareTask, Plant, UserBase.

VirtualGardener.Client: Frontend Blazor.

I. Components: Komponenty interfejsu użytkownika.
II. Services: Obsługa komunikacji z API backendu.
III. Models: Modele danych
IV. wwwroot Pliki statyczne

Modele współdzielone

CareTaskType

Definicja typów zadań pielęgnacyjnych:

- Watering - Podlewanie.
- Measuring - Mierzenie.
- Fertilizing - Nawożenie.
- Pruning - Przycinanie.
- PestControl - Kontrola szkodników.

- Repotting - Przesadzanie.

Frequency

Enumeracja definiująca częstotliwości:

- Daily - Codziennie.
- EveryOtherDay - Co drugi dzień.
- OnceAWeek - Raz w tygodniu.
- TwiceAMonth - Dwa razy w miesiącu.
- Monthly - Raz w miesiącu.

PlantType

Typy roślin:

- Flower - Kwiat.
- Tree - Drzewo.
- Shrub - Krzew.
- Grass - Trawa.
- Herb - Zioło.
- Vegetable - Warzywo.
- Fruit - Owoc.

Role

Role użytkowników:

- User - Zwykły użytkownik.

LogInRequest

Model danych do logowania:

- Email *(string)* - Adres e-mail użytkownika.
- Password *(string)* - Hasło użytkownika.

CareTask

Model zadania pielęgnacyjnego:

- Id *(Guid)* - Identyfikator zadania.
- PlantId *(Guid)* - Identyfikator rośliny.

- ActionType *(CareTaskType)* - Typ zadania.
- TaskDate *(DateTime)* - Data zadania.
- Notes *(string?)* - Notatki. (w przypadku zadania "Measuring" wpisać sam wzrost w cm)

Plant

Model rośliny:

- Id *(Guid)* - Identyfikator rośliny.
- Name *(string?)* - Nazwa rośliny.
- Type *(PlantType)* - Typ rośliny.
- PlantingDate *(DateTime)* - Data zasadzenia.
- WateringFrequency *(Frequency)* - Częstotliwość podlewania.
- FertilizingFrequency *(Frequency)* - Automatycznie obliczana częstotliwość nawożenia.
- Location *(string?)* - Lokalizacja.
- Notes *(string?)* - Notatki.
- IsIndoor *(bool)* - Czy roślina jest wewnętrzna.
- CareTasks *(List?)* - Lista zadań pielęgnacyjnych.

UserBase

Model użytkownika:

- Id *(Guid)* - Identyfikator użytkownika.
- Name *(string?)* - Imię użytkownika.
- Email *(string?)* - E-mail użytkownika.
- Role *(Role)* - Rola użytkownika.

Backend (Serwer)

PlantEntity

Encja rośliny:

- Dziedziczy po Plant z VirtualGardener.Shared

- User *(UserEntity)* - Właściciel rośliny.
- **Metoda:** ToPlant() - Konwersja encji na model Plant.

UserEntity

Encja użytkownika:

- Id *(Guid)* - Identyfikator użytkownika.
- Name *(string)* - Imię.
- Email *(string)* - E-mail.
- Password *(string)* - Hasło (wymaga szyfrowania).
- Role *(Role)* - Rola użytkownika.
- Plants *(ICollection)* - Kolekcja roślin użytkownika.

ServerSettings

Konfiguracja serwera:

- ConnectionString *(string)* - Ciąg połączenia do bazy danych.

AddPlantRequest

Model żądania dodania rośliny:

- Id *(Guid)* - Identyfikator.
- Name *(string)* - Nazwa.
- Type *(PlantType)* - Typ.
- PlantingDate *(DateTime)* - Data zasadzenia.
- WateringFrequency *(Frequency)* - Częstotliwość podlewania.
- Location *(string)* - Lokalizacja.
- Notes *(string?)* - Notatki.
- IsIndoor *(bool)* - Czy roślina jest wewnętrzna.

User

Model danych użytkownika:

- Name *(string)* - Imię.
- Email *(string)* - E-mail.
- Password *(string)* - Hasło.

UserDto

Rozszerzenie UserBase do transferu danych.

PlantController

Opis kontrolera: Obsługuje zarządzanie roślinami.

Metody:

- **GET /getPlants/{userId}** (GetPlantsAsync)
    - Zwraca listę roślin użytkownika.
    - Parametry: userId *(Guid)* - Identyfikator użytkownika.
    - Zwraca: List<Plant>.
- **GET /getPlantDetails/{plantId}/{userId}** (GetPlantDetailsAsync)
    - Zwraca szczegóły rośliny.
    - Parametry:
        - userId *(Guid)* - Identyfikator użytkownika.
        - plantId *(string)* - Identyfikator rośliny.
    - Zwraca: Plant.
- **POST /add/{userId}** (AddPlantAsync)
    - Dodaje nową roślinę.
    - Parametry:
        - userId *(Guid)* - Identyfikator użytkownika.
        - Body: AddPlantRequest - Dane rośliny.
    - Zwraca: status operacji.
- **POST /addCareTask/{plantId}/{userId}** (AddCareTaskAsync)
    - Dodaje zadanie pielęgnacyjne.
    - Parametry:
        - userId *(Guid)* - Identyfikator użytkownika.
        - plantId *(string)* - Identyfikator rośliny.
        - Body: CareTask - Dane zadania.
    - Zwraca: status operacji.

- **DELETE /deletePlant/{plantId}/{userId}** (DeletePlantAsync)
    - Usuwa roślinę.
    - Parametry:
        - userId *(Guid)* - Identyfikator użytkownika.
        - plantId *(string)* - Identyfikator rośliny.
    - Zwraca: status operacji.

AuthController

Opis kontrolera: Obsługuje autoryzację i rejestrację użytkowników.

Metody:

- **POST /register** (RegisterAsync)
    - Rejestruje nowego użytkownika.
    - Parametry:
        - Body: User - Dane użytkownika.
    - Zwraca: status operacji.
- **POST /logIn** (SignInAsync)
    - Loguje użytkownika.
    - Parametry:
        - Body: LogInRequest - Dane logowania (email i hasło).
    - Zwraca: UserDto - Dane zalogowanego użytkownika.

Helpers

Opis: Klasa narzędziowa dla różnych funkcjonalności serwera (aktualnie pusta).

Result

Opis: Model wyniku operacji, używany w serwerze.

Typy wyników:

- **ResultStatus**:
    - Success - Operacja zakończona sukcesem.
    - Warning - Ostrzeżenie.
    - Error - Błąd.

- **ResultStatusCode**:
    - Ok - Operacja udana.
    - NoDataFound - Brak danych.
    - DataAlreadyExist - Dane już istnieją.
    - AccessForbidden - Brak dostępu.
    - BadRequest - Nieprawidłowe żądanie.
    - DatabaseError - Błąd bazy danych.
    - UserCreationFailed - Błąd podczas tworzenia użytkownika.
    - Unknown - Nieznany błąd.
    - InternalServerError - Wewnętrzny błąd serwera.

Interfejsy i klasy:

- **IResult** - Bazowy interfejs wyniku.
- **IResult<T>** - Wynik z danymi generycznymi.
- **Result** - Implementacja wyniku bez danych.
- **Result<T>** - Implementacja wyniku z danymi typu generycznymi.

Frontend:

ServerSettings

Konfiguracja frontendu:

- BaseUrl *(string)* - Adres bazowy API.

User

Rozszerzenie UserBase:

- Password *(string?)* - Hasło (opcjonalne, np. podczas logowania).

UserAuthState

Stan uwierzytelnienia użytkownika:

- Rozszerza UserBase.

Najciekawsze funkcjonalności:

Własny system logowania oparty na localstorage

Automatyczne migracje bazy danych po uruchomieniu projektu

Automatyzacja uruchomienia projektu