Zadanie domowe: System rezerwacji „Project Defense”

Proszę zaimplementować system do zarządzania zapisami studentów na oddawanie projektów w wyznaczonych salach laboratoryjnych. W ramach zadania można uzyskać 110%.

Struktura rozwiązania (Solution)

Solution powinno składać się z przynajmniej 3 projektów, wzorując się na strukturze z zadań przykładowych:

1. **Aplikacja internetowa ASP.NET Core (Razor Pages)** – Główny interfejs użytkownika dla studentów i prowadzących, realizujący logikę biznesową oraz udostępniający API.
2. **Biblioteka klas (.NET 8)** – Współdzielony model danych (encje), który będzie używany zarówno przez aplikację webową, jak i konsolową.
3. **Aplikacja konsolowa (.NET 8)** – Prosty klient pozwalający na interakcję z systemem (np. przeglądanie wolnych terminów) poprzez wywołania API.

Model Danych (Podstawowe Encje)

Sercem systemu powinien być model danych obejmujący m.in.:

- **Użytkownik** (dziedziczący z `IdentityUser`) – Z podziałem na role "Student" i "Prowadzący".
- **Sala** – Encja przechowująca dostępne sale (np. `Nazwa`, `Numer Sali`).
- **DostępnośćProwadzącego** – Encja, w której prowadzący definiuje swoje "okna" dostępności (powiązanie z `Prowadzącym`, `Salą`, `DataPoczątkowa`, `DataKońcowa`, `GodzinaRozpoczecia`, `GodzinaZakonczenia` oraz `CzasTrwaniaSlotuWMIN`).
- **Rezerwacja** – Encja reprezentująca pojedynczy slot czasowy. Powinna być powiązana z `DostępnośćProwadzącego`, mieć `CzasRozpoczecia`, `CzasZakonczenia` oraz `StudentId` (FK do Użytkownika, *nullable* jeśli slot jest wolny).
- **BlokadaStudenta** (Opcjonalnie) – Encja do przechowywania "zbanowanych" studentów.

PUNKTACJA I WYMAGANIA

(20 pkt.) Moduł Prowadzącego

Prowadzący, po zalogowaniu, musi mieć możliwość pełnego zarządzania systemem:

- Pełne operacje **CRUD** na encji **Sala** (dodawanie/edycja/usuwanie sal).
- Definiowanie swojej dostępności: Prowadzący wybiera salę, zakres dat (np. od 10.11 do 20.11), zakres godzin (np. 8:00-12:00) oraz czas trwania pojedynczego slotu (np. 15 minut).
- **Logika Biznesowa:** Na podstawie tych danych system musi **automatycznie wygenerować** listę dostępnych, pustych `Rezerwacji` (slotów).
- Prowadzący widzi listę wszystkich rezerwacji (kto, gdzie, kiedy).

- Prowadzący może **anulować** dowolną rezerwację (nawet zajętą przez studenta) lub **przepisać** studenta na inny wolny termin.
- Prowadzący może **zablokować** wybrany okres (np. z powodu choroby), co powinno anulować rezerwacje i uniemożliwić nowe w tym czasie.
- Prowadzący może **zbanować** studenta, uniemożliwiając mu logowanie lub rezerwację.

(20 pkt.) Moduł Studenta

Student, po zalogowaniu, ma ograniczone możliwości interakcji:

- Widzi listę **tylko wolnych** terminów (tych, które nie są zajęte, nie są zablokowane i nie odbyły się w przeszłości).
- Student może wybrać **tylko jeden** dostępny termin. System nie może pozwolić na zarezerwowanie więcej niż jednego slotu jednocześnie.
- Student może **anulować** swoją rezerwację (co zwalnia termin dla innych).
- Student może **zmienić** swój termin (co jest równoznaczne z anulowaniem starego i zarezerwowaniem nowego).

(20 pkt.) Walidacja i Logika Systemowa

System musi poprawnie walidować wszystkie formularze i operacje:

- Walidacja wprowadzanych danych (np. czas trwania slotu musi być > 0; zakresy dat i godzin nie mogą być sprzeczne).
- Zapobieganie konfliktom: Prowadzący nie może zdefiniować dwóch nakładających się "Dostępności" dla tej samej sali w tym samym czasie.
- Logika wygasania: Rezerwacje, które już się odbyły (data i godzina z przeszłości) powinny być **automatycznie blokowane** i niemożliwe do zmiany (ani przez studenta, ani przez prowadzącego).
- Formularze powinny być intuicyjne, np. lista dostępnych slotów powinna być czytelna (np. "Sala 101, Pn. 10.11.2025, 10:15-10:30").

(10 pkt.) Uwierzytelnianie i Autoryzacja

- Należy użyć opcji **"Pojedyncze konta"** (`IdentityContext`).
- Wymagana jest rejestracja z **potwierdzeniem przez e-mail** (należy zaimplementować `IEmailSender` używając np. SendGrid, MailGun) .
- Należy zaimplementować **Role** ("Student" i "Prowadzący") i odpowiednio zabezpieczyć funkcjonalności (np. `[Authorize(Roles = "Prowadzący")]`).
- Użytkownik niezalogowany nie ma dostępu do żadnych funkcjonalności (widzi tylko stronę logowania/rejestracji).

(10 pkt.) Internacjonalizacja

- System musi wspierać przynajmniej dwie kultury: **pl-PL** oraz **en-US**.
- Szczególny nacisk należy położyć na poprawną walidację i wyświetlanie **dat i godzin** zgodnie z wybraną kulturą.

(10 pkt.) Minimal API i Swagger

- Należy udostępnić **Minimal API**:
  - `MapGet` pozwalający na pobranie listy wolnych terminów (`/api/slots/available`).
  - `MapGet` pozwalający na pobranie listy sal (`/api/rooms`).
  - `MapPost` pozwalający studentowi na rezerwację terminu (np. `/api/slots/{id}/book`).
- Należy zarejestrować **Swagger** (Middleware i UI).
- Do komunikacji przez API należy wydzielić osobne obiekty **DTO**.

(10 pkt.) Klient Konsolowy

- Należy zaimplementować prostego klienta konsolowego, który wykorzysta API.
- Klient powinien umożliwiać (np. po podaniu ID studenta lub tokenu):
  - Wyświetlenie stanu dostępnych terminów (wywołanie GET).
  - Dokonanie rezerwacji (wywołanie POST/PUT).

(10 pkt.) Eksport Danych

- W panelu prowadzącego należy dodać funkcjonalność eksportu "wykresu rezerwacji" (listy rezerwacji) dla wybranej sali i zakresu dat.
- Aplikacja powinna pozwalać na wybór formatu eksportu:
  - Prosty plik **.txt**.
  - Arkusz kalkulacyjny **.xlsx** (np. przy użyciu biblioteki ClosedXML).
  - Plik **.pdf** (np. przy użyciu biblioteki QuestPDF).