Zadanie domowe ASP.NET: System Rezerwacji Salek Konferencyjnych

Proszę zaimplementować aplikację webową do rezerwacji salek konferencyjnych w biurze. System pozwala zalogowanym użytkownikom rezerwować dostępne salki w określonych przedziałach czasowych.

Wymagania techniczne:

- Dane o salkach i rezerwacjach proszę przechowywać w pamięci w klasie-repozytorium, która musi być przygotowana na obsługę wielu jednoczesnych zapytań (problem wyścigu przy rezerwacji tego samego terminu).
- **Model Salki:** Powinien zawierać unikalną **nazwę** (np. "Kreatywna") oraz **pojemność** (liczba miejsc).
- **Model Rezerwacji:** Powinien łączyć **użytkownika** z **salką** i przechowywać **czas rozpoczęcia** oraz **czas zakończenia** rezerwacji.
- Istnieją dwie role: **Administrator** (login: `admin`), który zarządza salkami, oraz **Użytkownik**, który dokonuje rezerwacji.

Wskazówki rozwiązania obsługi **problemu wyścigu** (race condition), który może wystąpić, gdy wielu użytkowników próbuje zarezerwować ten sam termin jednocześnie:

- Do przechowywania danych w pamięci (salek, rezerwacji) użyj kolekcji bezpiecznych wątkowo, na przykład `ConcurrentDictionary<TKey, TValue>`. Zapewniają one podstawową ochronę przy prostych operacjach dodawania czy usuwania.
- Samo użycie `ConcurrentDictionary` nie wystarczy do operacji rezerwacji. Cała logika, która najpierw sprawdza dostępność terminu, a następnie dodaje nową rezerwację, musi zostać wykonana jako jedna, **atomowa** operacja. Osiągniesz to, zamykając kluczowy fragment kodu w bloku `lock (...)`.
- Pamiętaj, aby zarejestrować swoje repozytorium w kontenerze Dependency Injection jako **Singleton**. Dzięki temu cała aplikacja będzie korzystać z jednej, współdzielonej instancji danych w pamięci.

Akcje do zaimplementowania i punktacja

Akcje dostępne dla administratora:

- **Room/Manage (10 pkt.)**
  - Jeden widok do zarządzania salkami. Powinien wyświetlać listę istniejących salek i zawierać formularz do dodawania nowej. Przy każdej salce na liście powinien znajdować się przycisk do jej usunięcia.
- **Init (5 pkt.)**
  - Akcja, która tworzy kilku przykładowych użytkowników oraz 3-4 salki konferencyjne o różnych nazwach i pojemnościach.

Akcje dostępne dla wszystkich użytkowników:

- **Account/Login/{login} (5 pkt.)**
  - Loguje użytkownika na podstawie loginu z adresu URL. Po zalogowaniu przekierowuje na główny widok kalendarza (`/Booking/Calendar`).
- **Account/Logout (5 pkt.)**
  - Wylogowuje bieżącego użytkownika.

Główne funkcje systemu (wymagają zalogowania):pl

- **Booking/Calendar (25 pkt.)**
  - Najważniejszy widok aplikacji, przedstawiający tygodniowy lub dzienny harmonogram rezerwacji dla wszystkich salek.
  - Interfejs powinien być czytelny, np. w formie tabeli, gdzie kolumny to salki, a wiersze to godziny.

    - Użytkownik, klikając na wolny termin, powinien móc dokonać rezerwacji (np. poprzez okno modalne lub prosty formularz).
- **API: Booking/GetForDay (10 pkt.)**
    - Akcja zwracająca w formacie JSON wszystkie rezerwacje na dany dzień. Będzie wykorzystywana przez JavaScript do dynamicznego rysowania harmonogramu w widoku `Calendar`.
- **API: Booking/Create (15 pkt.)**
    - Akcja przyjmująca dane nowej rezerwacji (ID salki, start, koniec).
    - **Kluczowa logika:** Musi zweryfikować, czy termin jest wolny, czy czas rozpoczęcia jest przed czasem zakończenia i czy rezerwacja nie koliduje z inną.
    - W przypadku sukcesu rezerwuje salkę i zwraca JSON ze statusem powodzenia. W przypadku porażki (np. termin zajęty) zwraca JSON z odpowiednim komunikatem błędu.
- **Booking/MyBookings (10 pkt.)**
    - Prosty widok listy, na której zalogowany użytkownik widzi swoje nadchodzące rezerwacje. Przy każdej rezerwacji powinien być przycisk pozwalający ją anulować (co powinno wywoływać akcję `Booking/Cancel`).
- **Booking/ExportMyBookings (10 pkt.)**
    - Akcja, która generuje i pozwala pobrać plik w formacie **iCalendar (`.ics`)** zawierający wszystkie nadchodzące rezerwacje zalogowanego użytkownika. Umożliwi to importowanie ich do zewnętrznych kalendarzy (Google, Outlook).
- **Walidacja po stronie serwera (5 pkt.)**
    - Oprócz sprawdzania dostępności terminu, proszę dodać walidację na modelu rezerwacji, np. że minimalny czas rezerwacji to 15 minut, a maksymalny to 3 godziny. Komunikaty o błędach walidacji powinny być wyświetlane użytkownikowi.