## Nazwa projektu: GatherUp

**Autor projektu:** Patryk Przybylski

**Opis projektu:**

GatherUp to aplikacja umożliwiająca użytkownikom tworzenie, zarządzanie, odkrywanie i udostępnianie wydarzeń. Użytkownicy mogą zapraszać innych do wydarzeń, które sami utworzyli. Każdy z nich ma możliwość przeglądania wydarzeń innych użytkowników na listingu wydarzeń i dołączania do tych, które są dostępne publicznie, lub wysyłania prośby o udział w wydarzeniu, jeśli jego organizator wymaga akceptacji. Dzięki integracji z SignalR oraz komunikacji za pomocą WebSockets, użytkownicy otrzymują w czasie rzeczywistym powiadomienia o zmianach w obserwowanych wydarzeniach, co sprawia, że platforma jest dynamiczna i interaktywna.

**Wykorzystane technologie:**

Wykorzystane technologie i kluczowe biblioteki:

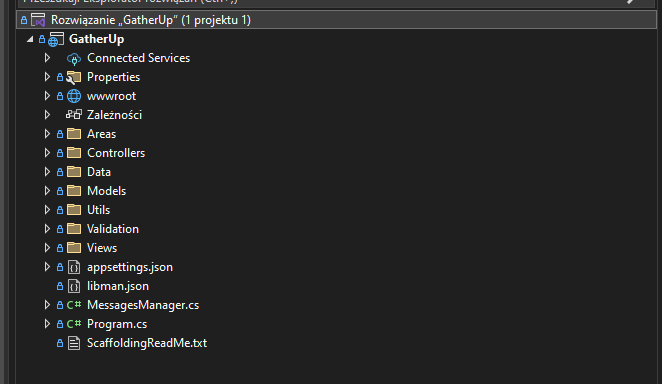
* .NET core 8
* MS SQL Server 2022 LocalDb (SQLEXPRESS) w wersji 16.0.1135.2
* Biblioteka Microsoft.AspNetCore.SignalR.Common w wersji 9.0.0

**Pierwsze uruchomienie:**

Jedyną rzeczą, którą trzeba zrobić to wykonanie polecenia Update-Database w konsoli menadżera pakietów. Po wykonaniu polecenia i wykonaniu się wszystkich migracji aplikacja jest gotowa do działania.

**Opis struktury projektu:**

Projekt zawiera typową strukturę MVC znaną z ASP .NET Core. Poniżej przedstawiam, krótki opis każdego z głównych katalogów.



**wwwroot** - znajdują się tutaj podfoldery zawierające wszelkie assety graficzne – folder „internal-images” oraz pliki .css i .js, które zawierają dodatkowe stylowania oraz skrypty.

**Areas** – katalog związany z Identity API czyli systemem autentykacji i uwierzytelniania użytkowników. Znajdują się tutaj widoki stron logowania, rejestracji itp. Które dostosowałem pod moją aplikację zmieniając ich layout oraz stylowanie.

**Controllers** – zawiera w sobie wszystkie kontrolery aplikacji dla danego modelu.

**Data** – kontekst bazy danych, jej ustawienia oraz migracje.

**Models** – wszystkie modele odzwierciedlające rzeczywistość np. Wydarzenia, zaproszenia i „follow’y”. W podkatalogu ViewModels znajdują się wszystkie viewmodele wykorzystywane w kontrolerach oraz komponentach.

**Utils** – klasy z funkcjami pomocniczymi np. Funkcja ograniczająca długość string’a, która ucina go w odpowiednim miejscu po przekroczeniu limitu.

**Validation** – własne reguły walidacji. Dodana została reguła pozwalająca na wykrycie czy użytkownik próbuje ustawić datę, która jest w przeszłości (nie chcemy tworzyć wydarzeń, które nie odbywają się w przyszłości).

**Views** – widoki aplikacji. W katalogu „Shared” znajdują się widoki typu partial oraz komponenty, które są reużywalne i pozwalają na większą modularność kodu.

Głównym plikiem naszego programu jest Program.cs, który inicjalizuje wszystkie ustawienia oraz uruchamia aplikację.

Plik MessagesManager.cs jest wykorzystywany jako główny Hub dla biblioteki SignalR. Pozwala na komunikację poprzez WebSockets z wybranymi użytkownikami.

**Modele aplikacji:**

**Event** – reprezentuje wydarzenie utworzone przez użytkownika. Jest trzonem aplikacji i głównym obiektem, do którego odwołują się inne modele. Pola modelu:

* **Id** – identyfikator eventu.
* **Name** – nazwa wydarzenia. Od 3 do 100 znaków.
* **Description** – opis wydarzenia. Od 1 do 500 znaków.
* **Location** – lokacja, w której odbywa się wydarzenie. Od 1 do 100 znaków.
* **Date** – data odbycia się wydarzenia. Walidowana customową metodą walidacji, sprawdza czy data na pewno jest w przyszłości.
* **Image** – Obrazek wydarzenia, kilka do wyboru.
* **UserId** – odwołanie do identyfikatora użytkownika, który jest właścicielem wydarzenia.
* **User** – relacja z użytkownikiem, który jest właścicielem wydarzenia.

**EventFollow** – sygnalizuje jaki użytkownik śledzi jakie wydarzenie. Po kliknięciu odpowiedniego przycisku tworzony jest obiekt EventFollow zawierająca informację o identyfikatorze użytkownika oraz identyfikatorze eventu. W ten sposób zbieramy wszystkie „follow’y” użytkownika. Pola:

* **Id** – identyfikator follow’a.
* **UserId** – odwołanie do identyfikatora użytkownika, który jest właścicielem follow’a.
* **User** – relacja z użytkownikiem, który jest właścicielem follow’a.
* **EventId** – odwołanie do identyfikatora wydarzenia, do którego przypisany jest follow.
* **Event** – relacja z wydarzeniem, do którego przypisany jest follow.

**EventInvitationBase** – jest bazowym modelem związanym z zaproszeniami na eventy. Wstępnym pomysłem było, aby dziedziczyły po nim dwa modele – model z zaproszeniami od organizatora wydarzenia do innych użytkowników, mógłby się nazywać np. EventInvitation oraz model z prośbą o przyjęcie do wydarzenia od użytkownika do organizatora. Ostatecznie powstał jedynie ten drugi, z nazwą **EventJoinRequest** .

* **Id** – identyfikator zaproszenia.
* **EventId** – odwołanie do identyfikatora wydarzenia, do którego przypisane jest zaproszenie.
* **Event** – relacja z wydarzeniem, do którego przypisane jest zaproszenie
* **SenderUserId** - identyfikator użytkownika wysyłającego zaproszenie/prośbę.
* **ReceiverUserId** – identyfikator użytkownika otrzymujące zaproszenie/prośbę.
* **CreatedDate** – data utworzenia.
* **ResolvedDate** – data rozwiązania zaproszenia/prośby poprzez akceptację lub odrzucenie.
* **Status** – status zaproszenia/prośby. Możliwe statusy to „Pending”, „Accepted” oraz „Rejected”.

**EventJoinRequest** – dziedziczy po **EventInvitationBase**. Reprezentuje prośbę użytkownika o przyjęcie do wydarzenia przez jego organizatora. Posiada te same pola co EventInvitationBase.

**Kontrolery aplikacji:**

**HomeController**  - domyślny kontroler w aplikacji ASP .NET Core. Metody:

* **[GET] Index()** – landing page witający użytkownika na platformie
* **[GET] Error()** – widok z błędem

**EventsController**  - kontroler sterującymi akcjami związanymi z wydarzenami. Metody:

* **[GET] Index(int pageNumber = 1, int pageSize = 10)** – widok - paginowany listing wydarzeń, na którym użytkownik widzi wszystkie istniejące wydarzenia, w tym te własne. Parametry pozwalają na ustawienie aktualnej strony oraz ilości wydarzeń per strona. Parametry przymują wartości domyślne. Widok jest renderowany na podstawie paginowanego **EventViewModel**.
* **[GET] MyEvents(int pageNumber = 1, int pageSize = 10)** – widok - paginowany listing wydarzeń utworzonych przez użytkownika (brak wydarzeń innych użytkowników). Działa na tej samej zasadzie co metoda **Index**.
* **[GET] FollowedEvents(int pageNumber = 1, int pageSize = 10)** – widok - paginowany listing wydarzeń śledzonych przez użytkownika. Działa na tej samej zasadzie co metoda **Index**.
* **[GET] Details(int id)** – widok – szczegóły związane z konkretnym wydarzeniem. W parametrze podajemy identyfikator wydarzenia w celu określenia jakie wydarzenie ma zostać wyświetlone.
* **[GET] Create()** – widok – formularz tworzenia wydarzenia
* **[GET] Edit(int id)** – widok – formularz edytowania wydarzenia. Przyjmuje jako parametr identyfikator edytowanego wydarzenia.
* **[GET] Delete(int id)** – widok – formularz usuwania wydarzenia. Przyjmuje jako parametr identyfikator usuwanego wydarzenia.
* **[POST] Create(Event event)** – metoda tworząca wydarzenie na podstawie danych przekazanych w formularzu.
* **[POST] Edit(Event event)** – metoda edytująca wydarzenie na podstawie danych przekazanych w formularzu. Po udanej edycji emituje za pomocą SignalR wiadomość dla wszystkich użytkowników, którzy śledzą to wydarzenia z informacją o tym, że zostało ono edytowane.
* **[POST] DeleteConfirmed(int id)** – metoda, która usuwania wydarzenie po wcześniejszym potwierdzeniu.

**EventFollowsController**  - kontroler sterującymi akcjami związanymi z followami wydarzeń. Metody:

* **[POST] Create(EventFollow eventFollow)** – metoda tworząca follow’a dla aktualnie zalogowanego użytkownika, dla danego wydarzenia. Jest wywoływana ze js’owego skryptu za pomocą fetch API.
* **[POST] DeleteConfirmed(DeleteFollowRequest request)** – metoda, która usuwa dla obecnego użytkownika follow dla wybranego wydarzenia. Potrzebuje do tego id follow’a. Ten przekazywany jest jako pole w body requestu. Wywoływana ze js’owego skryptu za pomocą fetch API.

**EventJoinRequestsController**  - kontroler sterującymi akcjami związanymi z prośbami o przyjęcie do wydarzeń. Metody:

* **[GET] OwnRequests(int pageNumber = 1, int pageSize = 20)** – widok - paginowany listing próśb o przyjęcie, które użytkownik wysłał do innych użytkowników. Parametry pozwalają na ustawienie aktualnej strony oraz ilości próśb per strona. Parametry przymują wartości domyślne. Widok jest renderowany na podstawie paginowanego **EventJoinRequestViewModel**.
* **[GET] ForeignRequests(int pageNumber = 1, int pageSize = 20)** – widok - paginowany listing próśb o przyjęcie, które zostały wysłane do użytkownika i użytkownik może potwierdzić/odrzucić. Parametry pozwalają na ustawienie aktualnej strony oraz ilości próśb per strona. Parametry przymują wartości domyślne. Widok jest renderowany na podstawie paginowanego **EventJoinRequestViewModel**.
* **[POST] Create(EventFollow eventJoinRequest)** – metoda tworząca prośbę o przyjęcie dla aktualnie zalogowanego użytkownika, dla danego wydarzenia. Jest wywoływana ze js’owego skryptu za pomocą fetch API. Status prośby automatycznie jest ustawiany jako Pending.
* **[POST] DeleteConfirmed(DeleteJoinRequestRequest request)** – metoda, która usuwa dla obecnego użytkownika prośbę o przyjęcie dla wybranego wydarzenia. Potrzebuje do tego id prośby o przyjęcie. Ta przekazywany jest jako pole w body requestu. Wywoływana ze js’owego skryptu za pomocą fetch API.
* **[POST] ChangeStatus (AcceptOrRejectJoinRequestRequest request)** – metoda, która zmienia status dla wybranego wydarzenia. Potrzebuje do tego id prośby o przyjęcie oraz docelowego statusu. Te przekazywane są jako pola w body requestu. Wywoływana ze js’owego skryptu za pomocą fetch API.

**System użytkowników:**

System użytkowników jest prosty i dzieli się jedynie na zalogowanych i niezalogowanych, bez podziału na role. Użytkownicy niezalogowani mogą jedynie przeglądać istniejące wydarzenia innych użytkowników. W celu utworzenia własnego wydarzenia, dołączenia lub śledzenia czyjegoś wydarzenia – muszą się zarejestrować oraz zalogować.

Po zalogowaniu pojawiają się w menu nawigacyjnym dodatkowe akcje – Moje wydarzenia, Obserwowane wydarzenia, Zarządzanie zaproszeniami oraz zarządzanie kontem.

**Najciekawsze funkcjonalności:**

* Aplikacja wykorzystuje skrypty javascriptowe do tworzenia obiektów EventFollow oraz EventJoinRequest poprzez fetch API. Oznacza to dla użytkownika większą dynamikę interfejsu i lepszy user experience; strona nie musi być w ten sposób przeładowywana przy każdym requeście.
* Użycie biblioteki SignalR pozwala na dynamiczne komunikowanie się między sesjami użytkowników. Wykorzystuje to w aplikacji w celu dynamicznego wyświetlania powiadomień o zmianach w obserwowanych wydarzeniach. Jeśli użytkownik A obserwuje wydarzenie użytkownika B, a użytkowik B dokona w nim edycji, to użytkownik A zobaczy w prawym górnym rogu powiadomienie o tym, że dane wydarzenie zostało edytowane i jakie pola się zmieniły. Można to sprawdzić otwierając dwie przeglądarki i logując się na dwóch różnych kontach.