

Dokumentacja projektu

pt.: „Kalendarz Świąt nietypowych”

Data wykonania:

17.06.2022

Grupa: P4

Konrad Szpytma

Dominik Zając

Michał Maciej

Wojciech Łuszczak

Albert Kupiec

Spis treści

Cel projektu	3
Wykorzystane technologie	4
Backlog	4
Wstępny projekt UI/UX	7
Tryb jasny	7
Tryb ciemny	9
Przykładowe historyjki użytkownika	12
Użytkownik może zobaczyć wygląd kalendarza po wejściu w aplikację	12
Użytkownik może zalogować się do aplikacji	12
Użytkownik może zarejestrować się w serwisie (stworzenie konta, podanie emaila, hasła)	13
Użytkownik może otworzyć link wysłany na email, w celu potwierdzenia konta	14
Baza danych	15
Projekt systemu	16
Realny diagram przypadków użycia	16
Teoretyczny diagram przypadków użycia	17
Diagram ERD	18
Diagram DFD	19
Diagram FHD	20
Diagramy sekwencji poszczególnych funkcjonalności	21
Rejestracja użytkownika:	21
Modyfikacja użytkownika:	22
Dodawanie wydarzenia:	22
Modyfikacja wydarzenia:	23
Dodawanie święta:	23
Edycja święta:	24
Dokumentacja kodu źródłowego	24
Dokumentacja użytkownika	26

1. Cel projektu

Serwis "Kalendarz świąt nietypowych" ma na celu przesyłanie powiadomień na urządzenie, które będą informowały użytkownika jakie danego dnia jest nietypowe święto. Kalendarz ten będzie pełnił także funkcję zwykłego kalendarza, gdzie będzie można zapisywać swoje wydarzenia i otrzymywać o nich powiadomienia. Do każdego wydarzenia, które użytkownik umieści w kalendarzu będzie można przypisać kolor wydarzenia (np. czerwony, niebieski, zielony, itd.), pozwoli to użytkownikom na uporządkowanie wydarzeń według kolorów. Serwis będzie posiadał także funkcję otrzymywania powiadomień, nie tylko związanych z świętami nietypowymi, ale także z wydarzeniami, które użytkownik sam umieścił w swoim kalendarzu. Powiadomienia będą wysyłane na urządzenie zgodnie z preferencjami użytkownika, będzie to możliwe dzięki opcji „dodaj powiadomienie”, gdzie użytkownik będzie w stanie wybrać kiedy otrzyma daną wiadomość (np. tydzień przed, dzień przed, 2h przed, itd.). Do głównego tytułu wydarzenia w funkcjach dodatkowych użytkownik będzie mógł dodać lokalizację wydarzenia, opis, załącznik oraz będzie miał możliwość przypisania osoby do wydarzenia. Dodatkowo użytkownik będzie w stanie wybrać czas trwania wydarzenia (cały dzień, od danej daty do danej daty, dokładne godziny), a także czy czynność powtarza się, czy nie. Kalendarz będzie można udostępniać innym użytkownikom posiadającym aplikację, by użytkownicy mogli posiadać jeden wspólny kalendarz. Aplikacja będzie posiadała funkcję wyświetlania kalendarza w układzie tygodniowym, miesięcznym, dziennym lub rocznym, dzięki temu użytkownik będzie mógł dostosować użyteczność aplikacji pod swoje preferencje. Aplikacja będzie posiadała możliwość wyboru trybu jasnego, z białym tłem i czarnymi napisami oraz trybu czarnego – z czarnym tłem i białymi napisami. Użytkownik będzie posiadał także możliwość dodania tła do swojego kalendarza.

2. Wykorzystane technologie

Frontend:

- React.js – biblioteka języka programowania JavaScript, użyta została do utworzenia graficznego interfejsu użytkownika.
- Typescript – język oprogramowania będący nadzbiorem języka Javascript
- Redux - biblioteka javascript do zarządzania i centralizacji stanu aplikacji

Backend:

- ASP.NET Core – platforma do tworzenia połączonych z internetem aplikacji z obsługą chmury. Wykorzystuje język programowania C#.

Hosting:

- Azure

Obsługa projektu:

- Github classroom
- Trello

Aplikacja “Kalendarz Świąt Nietypowych” jest aplikacją Webową. Użytkownik do jej uruchomienia korzysta z przeglądarki internetowej.

3. Backlog

Jako...	Funkcjonalność	Sprint	Status
Użytkownik	Użytkownik może zobaczyć wygląd kalendarza po wejściu w aplikację	2	Done
Użytkownik	Użytkownik może zapisać swoje wydarzenie w wybrany dzień w kalendarzu	3	Done
Użytkownik	Użytkownik może zalogować się do aplikacji	1	Done
Użytkownik	Użytkownik może wybrać układ kalendarza (tygodniowy, dzienny, miesięczny, roczny)	2	Not exactly
Użytkownik	Użytkownik może zarejestrować się w serwisie (utworzenie konta, podanie emaila, hasła)	1	Done
Użytkownik	Użytkownik może utworzyć wydarzenie	3	Done

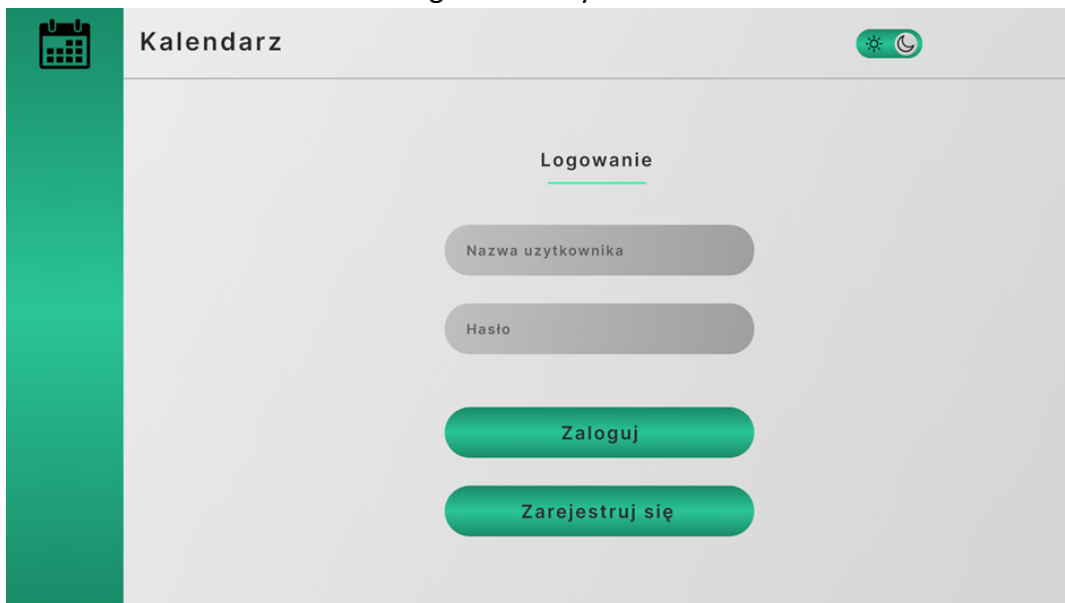
Jako...	Funkcjonalność	Sprint	Status
Użytkownik	Użytkownik może zobaczyć wygląd kalendarza po wejściu w aplikację	2	Done
Użytkownik	Użytkownik może usunąć swoje wydarzenie	4	Done
Użytkownik	Użytkownik może edytować opis swojego wydarzenia	4	Done
Użytkownik	Użytkownik może edytować lokalizację swojego wydarzenia	4	Done
Użytkownik	Użytkownik może zresetować hasło, w przypadku gdy o nim zapomni	4	Not exactly
Użytkownik	Użytkownik może edytować swoje wydarzenie	4	Done
Użytkownik	Użytkownik może zobaczyć w kalendarzu zakładkę z wypisanymi świętami nietypowymi na każdy dzień	3	Not exactly
Użytkownik	Użytkownik może usunąć opis swojego wydarzenia	4	Done
Użytkownik	Użytkownik może usunąć lokalizację swojego wydarzenia	4	Done
Użytkownik	Użytkownik może wybrać wygląd aplikacji (jasny lub ciemny)	3	Done
Użytkownik	Użytkownik może wybrać częstotliwość powiadomienia do wydarzenia (np. godzinę przed, tydzień przed)	4	Not exactly
Użytkownik	Użytkownik może wybrać godzinę, w której będzie otrzymywał codzienne powiadomienie o dzisiejszym święcie nietypowym		Not done
Użytkownik	Użytkownik może dołączyć załącznik do swojego wydarzenia		Not done

Jako...	Funkcjonalność	Sprint	Status
Użytkownik	Użytkownik może zobaczyć wygląd kalendarza po wejściu w aplikację	2	Done
Użytkownik	Użytkownik może usunąć załącznik dołączony do swojego wydarzenia		Not done
Użytkownik	Użytkownik może przypisać kolor osobie dodanej swojego wydarzenia		Not done
Użytkownik	Użytkownik może udostępnić swój kalendarz poprzez wysłanie linku		Not done
Użytkownik	Użytkownik może zmienić wygląd aplikacji	2	Done
Użytkownik	Użytkownik może dodać tło do swojego kalendarza		Not done
Użytkownik	Użytkownik może usunąć tło ze swojego kalendarza		Not done
Użytkownik	Użytkownik może zmienić tło swojego kalendarza		Not done

4. Wstępny projekt UI/UX

1. Tryb jasny

Logowanie użytkownika:



Kalendarz

Logowanie

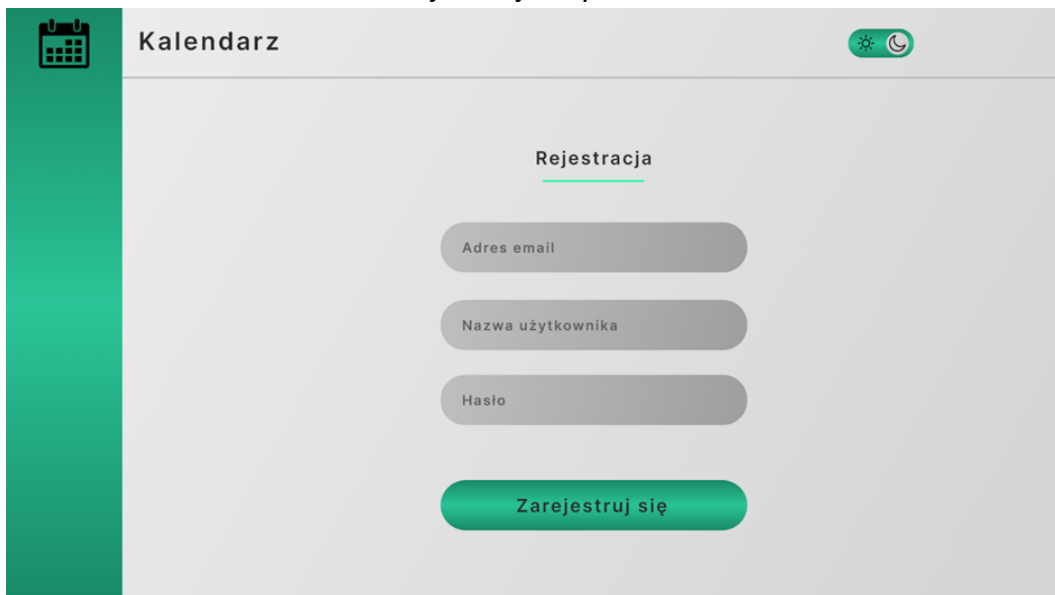
Nazwa użytkownika

Hasło

Zaloguj

Zarejestruj się

Rejestracja użytkownika:



Kalendarz

Rejestracja

Adres email

Nazwa użytkownika

Hasło

Zarejestruj się

Przeglądanie strony głównej kalendarza:

Kalendarz

Ważne nadchodzące wydarzenia:

- Kolokwium zaliczeniowe 10.06.2022 12:00
- Mecz piłkarski 10.06.2022 15:00

Miesiąc Tydzień Dzień

PN 9	WT 10	ŚR 11	CZ 12	PT 13	SO 14	N 15
12:00						
12:30		Nowy system testów 12:30-14:00			Zaliczenie 12:00-14:00	
13:00			Zaliczenie 13:30-15:00		Politechnika Rzeszowska	
13:30			Politechnika Rzeszowska			

Maj | 9-15 | 2022

Wydarzenia:

Piątek 10.06.2022

- Kolokwium zaliczeniowe 12:00 - 14:00 Politechnika Rzeszowska

Sobota 11.06.2022

- Spotkanie biznesowe 12:00 - 14:00 Lokalizacja nieznana

Poniedziałek 13.06.2022

- Kolokwium zaliczeniowe 10:00 - 11:00 Politechnika Rzeszowska

Wtorek 14.06.2022

- Kolokwium zaliczeniowe 13:00 - 14:00 Politechnika Rzeszowska

Dodawanie wydarzenia przez użytkownika:

Kalendarz

Nowe wydarzenie

Kolokwium zaliczeniowe

Dodaj lokalizację

Dodaj powiadomienie

Dodaj opis

Dodaj osobę

Poniedziałek 13.06.2022 10:00
Poniedziałek 13.06.2022 11:00

Dodaj załącznik

Ustaw powtarzalność

Domyślny kolor

Wydarzenia:

Piątek 10.06.2022

- Kolokwium zaliczeniowe 12:00 - 14:00 Politechnika Rzeszowska

Sobota 11.06.2022

- Spotkanie biznesowe 12:00 - 14:00 Lokalizacja nieznana

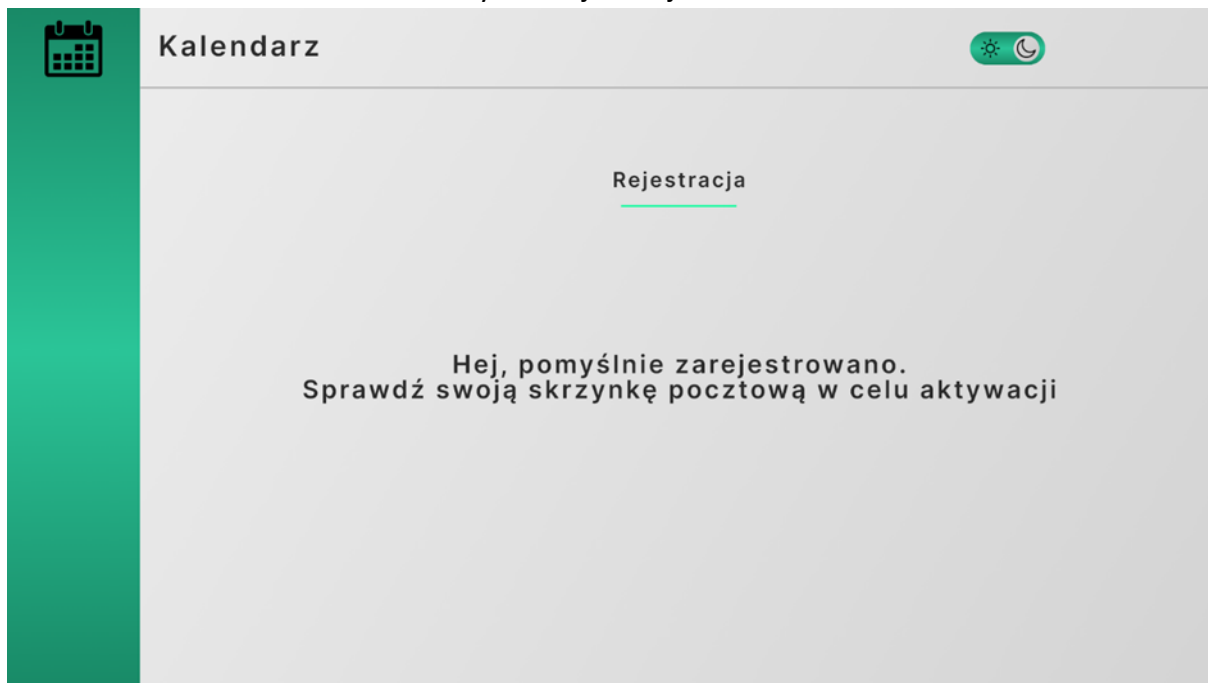
Poniedziałek 13.06.2022

- Kolokwium zaliczeniowe 10:00 - 11:00 Politechnika Rzeszowska

Wtorek 14.06.2022

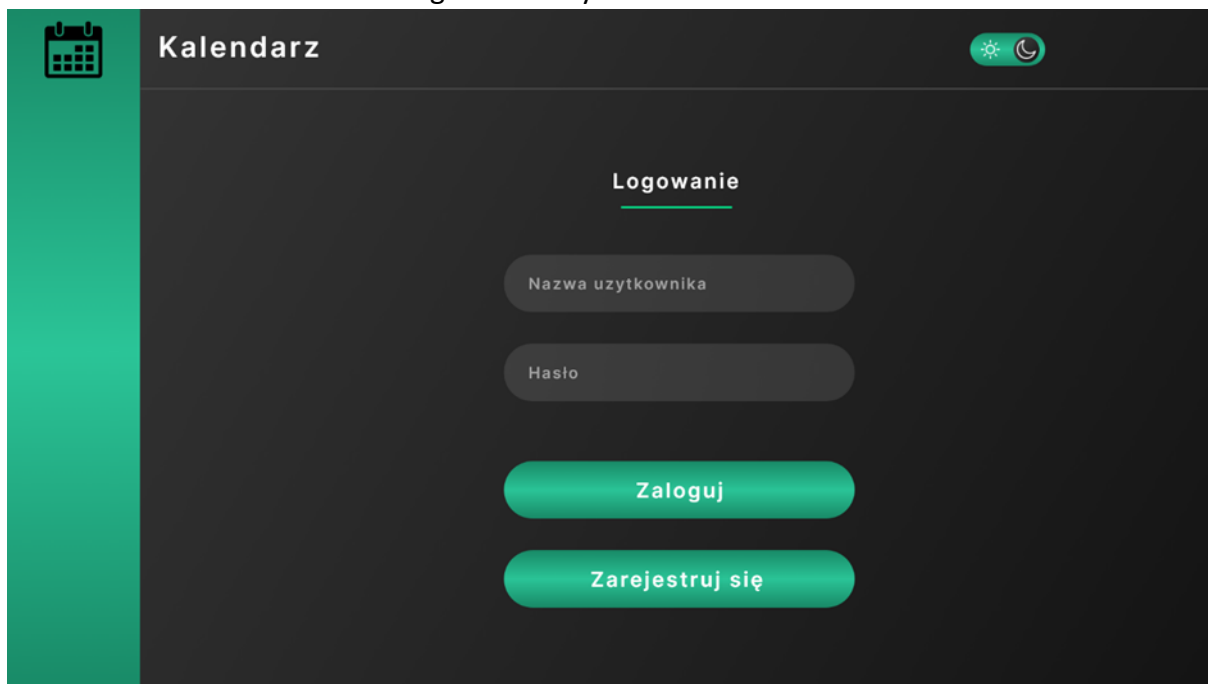
- Kolokwium zaliczeniowe 13:00 - 14:00 Politechnika Rzeszowska

Pomyślna rejestracja:



2. Tryb ciemny

Logowanie użytkownika:



Rejestracja użytkownika:

Kalendarz

Rejestracja

Adres email

Nazwa użytkownika

Hasło

Zarejestruj się

Przeglądanie strony głównej kalendarza:

Kalendarz

Ważne nadchodzące wydarzenia:

- Kolokwium zaliczeniowe
10.06.2022 12:00
- Mecz piłkarski
10.06.2022 15:00

Miesiąc
Tydzień
Dzień

PN	WT	ŚR	CZ	PT	SO	N
9	10	11	12	13	14	15
12:00						
12:30		Nowy odcinek serialu 12:30-13:00			Zaliczenie 12:30-13:00	
13:00			Zaliczenie 13:30-13:00		Politechnika Rzeszowska	
13:30			Politechnika Rzeszowska			

Maj | 9-15 | 2022

Wydarzenia:

Piątek 10.06.2022

- Kolokwium zaliczeniowe
12:00 - 14:00
Politechnika Rzeszowska

Sobota 11.06.2022

- Spotkanie biznesowe
12:00 - 14:00
Lokalizacja nieznana

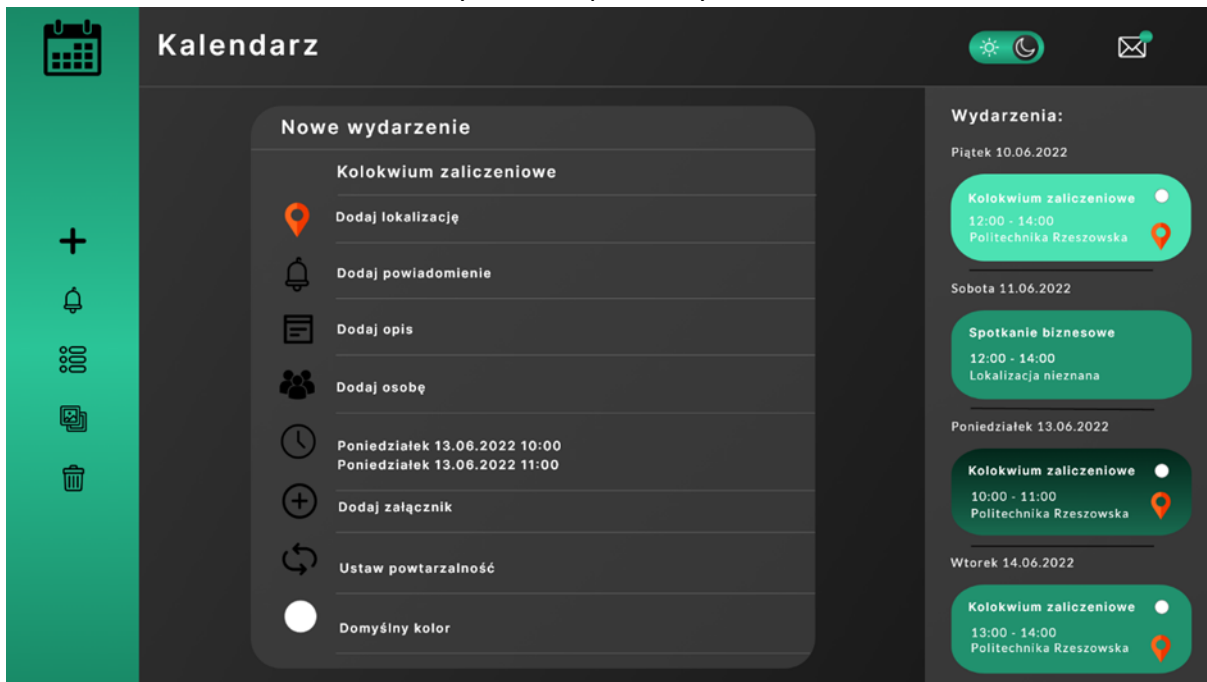
Poniedziałek 13.06.2022

- Kolokwium zaliczeniowe
10:00 - 11:00
Politechnika Rzeszowska

Wtorek 14.06.2022

- Kolokwium zaliczeniowe
13:00 - 14:00
Politechnika Rzeszowska


Dodawanie wydarzenia przez użytkownika:





Kalendarz


Nowe wydarzenie


Kolokwium zaliczeniowe


 Dodaj lokalizację


 Dodaj powiadomienie


 Dodaj opis

 Dodaj osobę

 Poniedziałek 13.06.2022 10:00
Poniedziałek 13.06.2022 11:00

 Dodaj załącznik

 Ustaw powtarzalność

 Domyślny kolor

Wydarzenia:

Piątek 10.06.2022

Kolokwium zaliczeniowe
12:00 - 14:00
Politechnika Rzeszowska

Sobota 11.06.2022

Spotkanie biznesowe
12:00 - 14:00
Lokalizacja nieznana

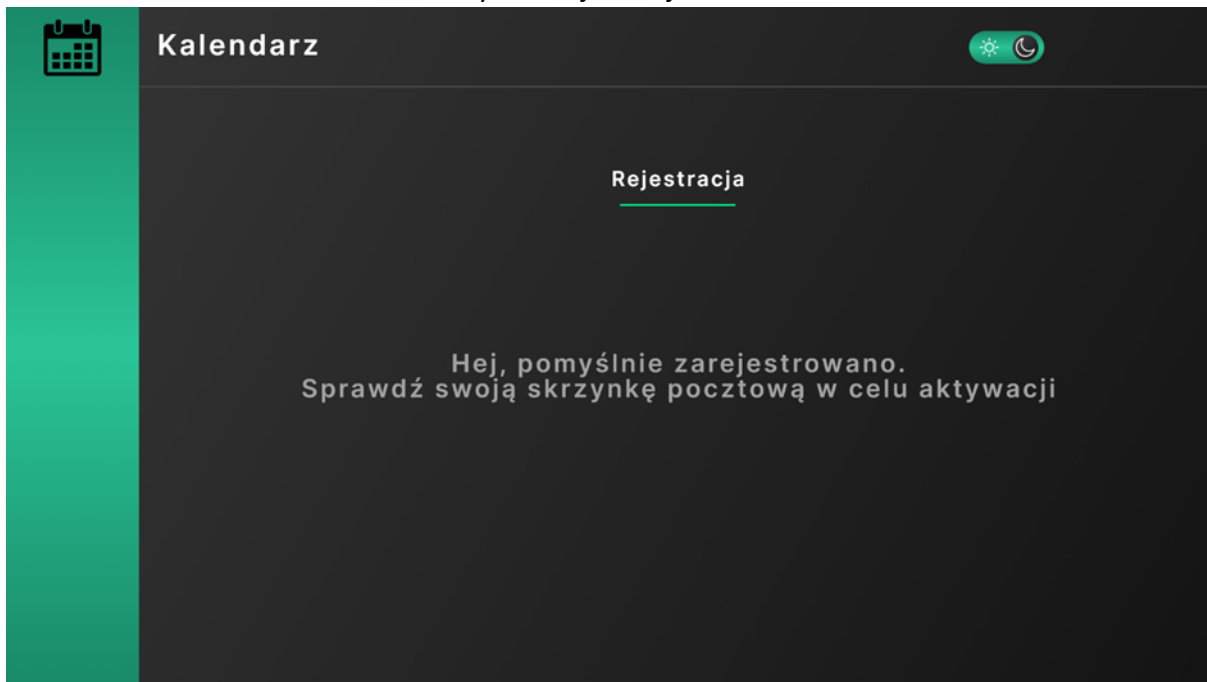
Poniedziałek 13.06.2022

Kolokwium zaliczeniowe
10:00 - 11:00
Politechnika Rzeszowska

Wtorek 14.06.2022

Kolokwium zaliczeniowe
13:00 - 14:00
Politechnika Rzeszowska

Pomyślna rejestracja:



Kalendarz

Rejestracja

Hej, pomyślnie zarejestrowano.
Sprawdź swoją skrzynkę pocztową w celu aktywacji

5. Przykładowe historyjki użytkownika

Użytkownik może zobaczyć wygląd kalendarza po wejściu w aplikację

Scenariusze testowe:

Scenariusz 1 --> Poprawne przygotowanie wyglądu kalendarza

Przyjmując, że użytkownik wszedł w aplikację kiedy przycisnął znacznik, który ma przekierować go do kalendarza wtedy aplikacja wyświetla wygląd kalendarza

Scenariusz 2 --> Niepoprawne przygotowanie wyglądu kalendarza

Przyjmując, że użytkownik wszedł w aplikację kiedy przycisnął znacznik, który ma przekierować go do kalendarza wtedy aplikacja wyświetla komunikat błędu i nie wyświetla wyglądu kalendarza

Kryteria akceptacji User Stories:

- Czy po wejściu w aplikację pokazuje się wygląd kalendarza?
- Czy stworzona grafika kalendarza jest zgodna z wymaganiami klienta?
- Czy po przyciśnięciu przycisku przenosi użytkownika do odpowiedniej strony?

Użytkownik może zalogować się do aplikacji

Scenariusze testowe:

Scenariusz 1 --> Użytkownik jest w stanie zalogować się do aplikacji

Przyjmując, że użytkownik poprawnie wszedł w aplikację i wpisał swoje dane, kiedy wybierze opcję "zaloguj" wtedy użytkownik zostaje zalogowany do aplikacji

Scenariusz 2 --> Użytkownik nie jest w stanie zalogować się do aplikacji

Przyjmując, że użytkownik poprawnie wszedł w aplikację i wpisał swoje dane, kiedy wybierze opcję "zaloguj" wtedy użytkownik zostaje poinformowany, że nie może być zalogowany do aplikacji

Kryteria akceptacji User Stories:

- Czy użytkownik po wciśnięciu ikony "zaloguj się" zostaje zalogowany w aplikacji?
- Czy użytkownik po zalogowaniu jest w stanie korzystać z funkcjonalności aplikacji?
- Czy użytkownik może wybrać różne opcje zalogowania się? (zaloguj się z Google, zaloguj się z Facebook, itd.)?

Użytkownik może zarejestrować się w serwisie (stworzenie konta, podanie emaila, hasła)

Scenariusze testowe:

Scenariusz 1 --> Użytkownik może zarejestrować się w aplikacji

Przyjmując, że użytkownik poprawnie wpisał swoje dane, w wyznaczone do tego miejsca, kiedy wciśnie przycisk "zarejestruj" wtedy zostanie zarejestrowany w aplikacji i będzie mógł się zalogować

Scenariusz 2 --> Użytkownik nie może zarejestrować się w aplikacji

Przyjmując, że użytkownik poprawnie wpisał swoje dane, w wyznaczone do tego miejsca, kiedy wciśnie przycisk "zarejestruj" wtedy zostanie powiadomiony o błędzie i nie zostanie zarejestrowany w aplikacji

Kryteria akceptacji User Stories:

- Czy użytkownik posiada opcje "podanie email", "stwórz konto", "podaj hasło"?
- Czy użytkownik może wpisać swój email w wyznaczone do tego pole?
- Czy użytkownik może wcisnąć przycisk "zarejestruj się"?
- Czy użytkownik jest w stanie wpisać hasło w wyznaczone do tego pole?

- Czy użytkownik po wciśnięciu "zarejestruj się" może być zalogowanym w aplikacji?

Użytkownik może otworzyć link wysłany na email, w celu potwierdzenia konta

Scenariusze testowe:

Scenariusz 1 --> Użytkownik może otworzyć link wysłany na email, w celu potwierdzenia konta

Przyjmując, że użytkownik wpisał poprawne dane w odpowiednie pola, kiedy wciśnie przycisk "Zarejestruj się", wtedy na podany przy rejestracji email zostanie wysłany link do aktywacji konta

Scenariusz 2 --> Użytkownik nie może otworzyć link wysłany na email, w celu potwierdzenia konta

Przyjmując, że użytkownik wpisał poprawne dane w odpowiednie pola, kiedy wciśnie przycisk "Zarejestruj się", wtedy na podany przy rejestracji email nie zostanie wysłany link do aktywacji konta, przez co użytkownik nie będzie w stanie potwierdzić prawdziwość swojego konta

Kryteria akceptacji User Stories:

- Czy użytkownik po wciśnięciu "Zarejestruj się" otrzymuje link na podany przy rejestracji email?
- Czy użytkownik może wejść w link wysłany na podany przy rejestracji email?
- Czy po wejściu w link konto zostaje aktywowane?

6. Baza danych

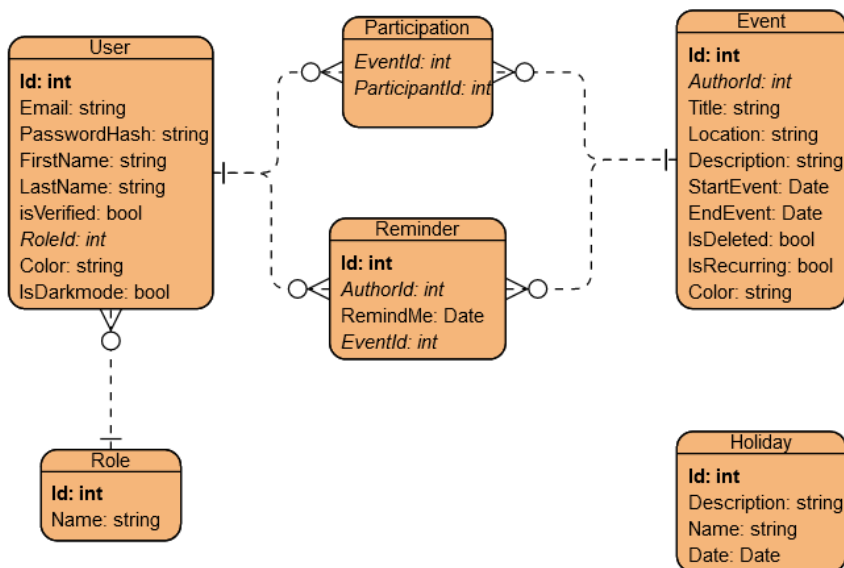
Baza danych składa się z sześciu tabel: User, Role, Participation, Reminder, Event oraz Holiday. Przy tworzeniu projektu korzystaliśmy z metody *Code First*. Korzystanie z C# (.NET) na backendzie usprawniło pracę dzięki EntityFramework Core. Dzięki temu utworzony został kod w programie, a następnie na podstawie niego utworzona została baza danych z określonymi zasadami. Pierwszym krokiem było utworzenie odpowiednich encji, które będą wykorzystywane, na podstawie diagramu ERD zostały utworzone odpowiednie pliki. Wykonane encje zostały wykorzystane w DbCalendarContext, który jest ogólnym programem komunikującym się z bazą danych za pośrednictwem różnych funkcji. W pliku również znajdują się również zasady ustawiania bazy danych, przypisanie odpowiednich kluczy głównych oraz odpowiednia konfiguracja kluczy obcych.

Dzięki EntityFramework, przy każdym utworzeniu bazy danych, dodawane są nietypowe święta z pliku tekstowego *holidaysJSON.txt*, który zostaje odpowiednio zdeserializowany w projekcie .NET, tak, aby właściwe dane zostały dodane do bazy.

```
using (StreamReader r = new StreamReader("holidaysJSON.txt"))
{
    string json = r.ReadToEnd();
    List<HolidayDTO> items =
    JsonConvert.DeserializeObject<List<HolidayDTO>>(json);
    int id_num = 1;
    foreach (var item in items)
    {
        modelBuilder.Entity<Holiday>().HasData(new Holiday { Id =
id_num, Name = item.Name, Date = item.Date, Description =
item.Description });
        id_num++;
    }
}
```

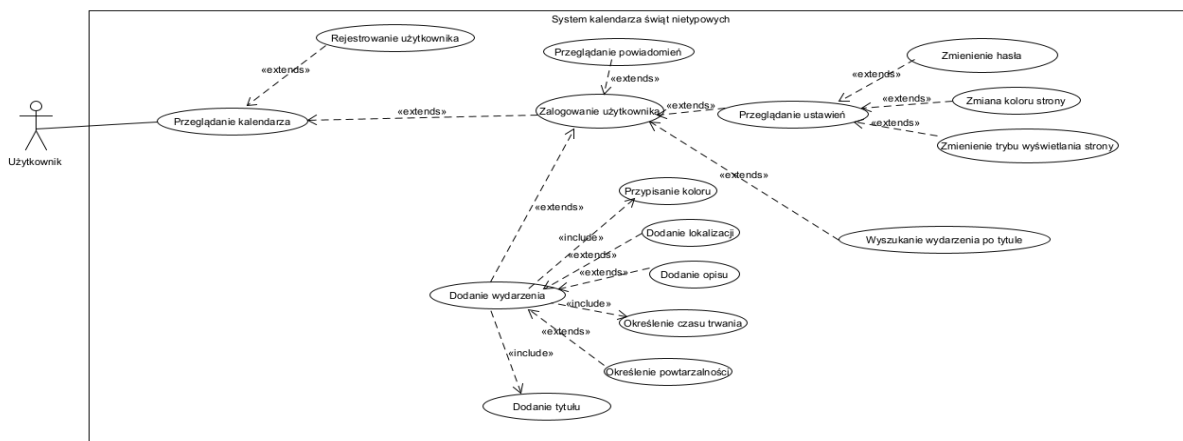
Na samym końcu pliku wyrażona jest metoda OnConfiguring, która wykonywana jest przy różnych zmianach, które zostały utworzone w serwisach.

Po utworzeniu takiego pliku można przejść do utworzenia migracji. Migracja jest obiektem, który posiada informacje na temat zmian utworzonych w encjach, modelach oraz kontekście bazy danych, które zostały napisane na backendzie. Zawarte są dodane zmiany, tak, aby w przypadku chęci cofnięcia ich, był możliwy łatwy sposób. Pomaga to w naprawie błędów.



7. Projekt systemu

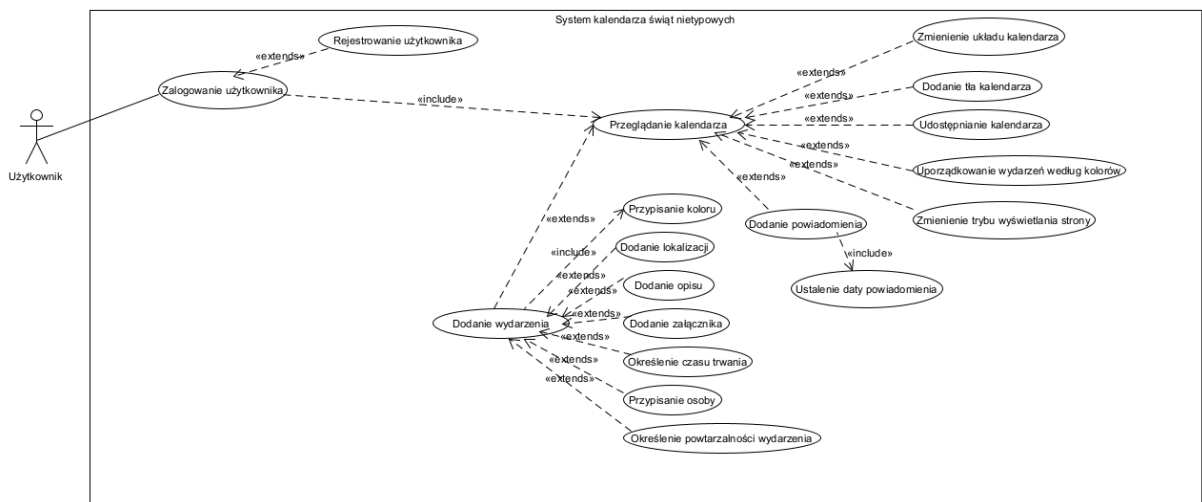
1. Rzeczywisty diagram przypadków użycia



Użytkownik przeglądając kalendarz ma możliwość zalogowania się lub zarejestrowania, po poprawnym zalogowaniu użytkownik może przeglądać swoje ustawienia gdzie może zmienić niektóre elementy serwisu (swoje hasło, kolor strony, tryb wyświetlania strony) a także dodać wydarzenie, któremu może opcjonalnie nadać mu lokalizację, opis, powtarzalność. Przy dodawaniu wydarzenia użytkownik musi nadać mu tytuł, określić czas trwania (początkowo jest przyjmowana bazowa wartość) oraz przypisać kolor (początkowo jest przyjmowana

bazowa wartość). Użytkownik może także przeglądać swoje powiadomienia i wyszukiwać wydarzenie po tytule.

2. Planowany diagram przypadków użycia



Użytkownik ma opcję zalogowania lub rejestracji, po zalogowaniu użytkownik przegląda kalendarz i ma możliwość dodania tła do kalendarza, zmiany układu kalendarza, udostępnienia kalendarza, uporządkowania wydarzeń według kolorów, zmiany trybu wyświetlania strony a także może dodać powiadomienie (musi przy tym ustalić jego datę) lub dodać wydarzenie z możliwością określenia: koloru, lokalizacji, opisu, załącznika, czasu trwania, powtarzalności wydarzenia, osoby przypisanej do wydarzenia.

3. Diagram ERD

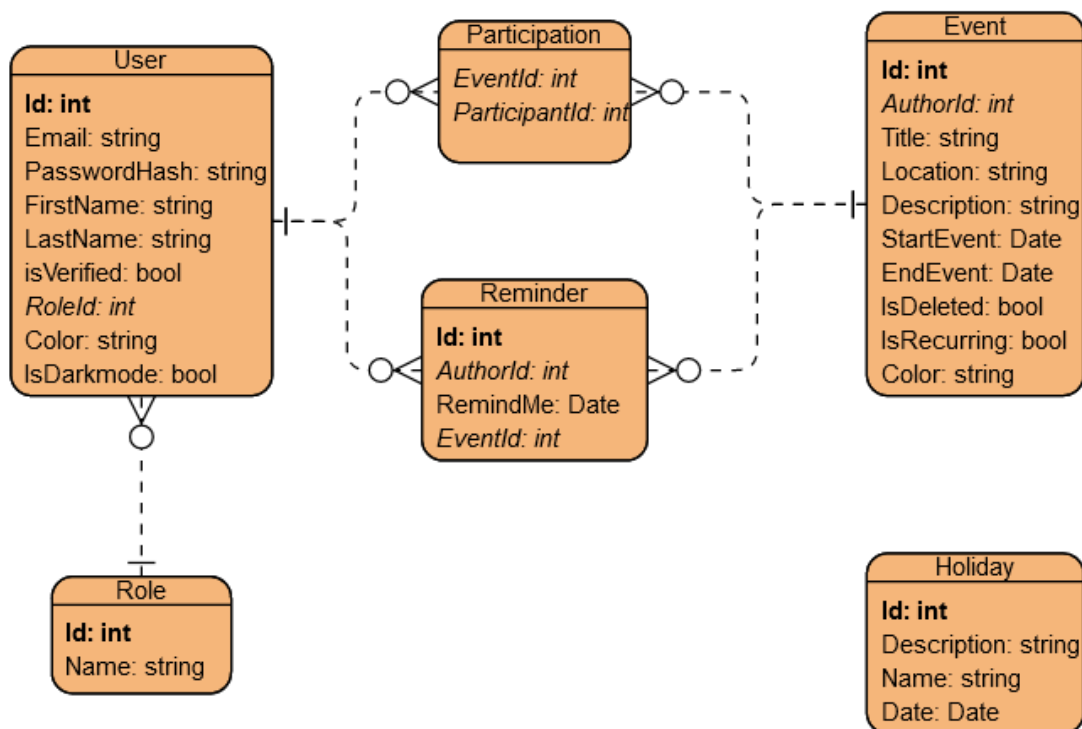
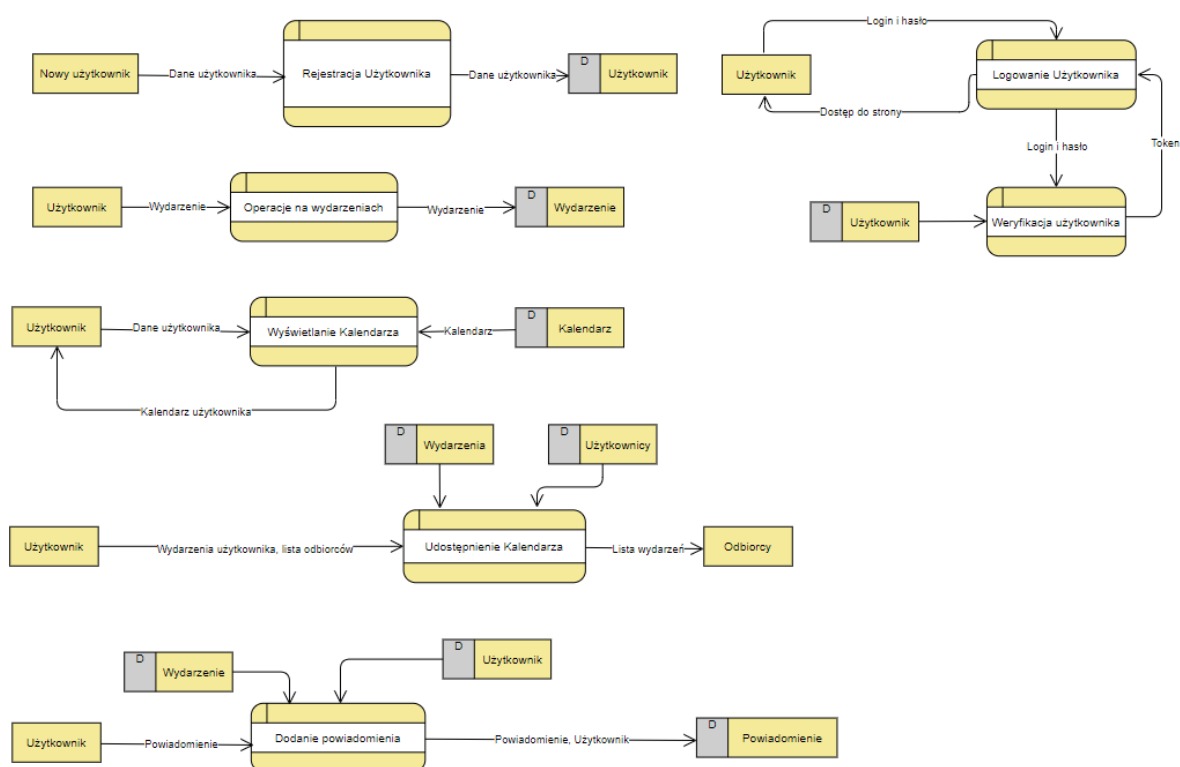


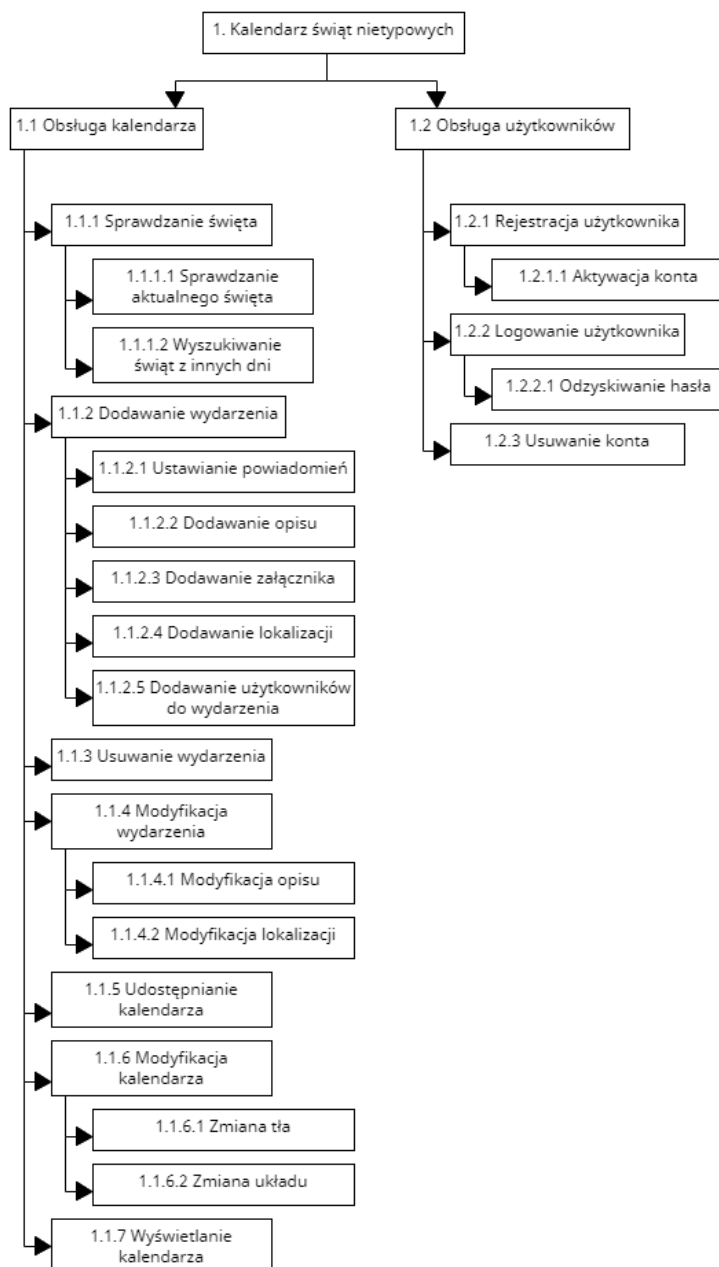
Diagram składa się z 6 encji. Użytkownik może mieć wiele przypomnień, przypomnienie może być przypisane do jednego użytkownika, użytkownik może mieć wiele uczestnictw oraz uczestnictwo może być przypisane do jednego użytkownika. Wydarzenie może mieć wiele przypomnień dotyczących go i wiele uczestnictw, przypomnienie i uczestnictwo dotyczy jednego eventu. Użytkownik może mieć jedną rolę a rola może być przypisana do wielu użytkowników. Istnieje także encja święta odpowiadająca za święto nietypowe, które nie jest zależne od użytkownika. Encje posiadają atrybuty encji potrzebne do poprawnego i przejrzystego działania serwisu.

4. Diagram DFD



W procesie rejestracji agent - użytkownik przesyła swoje dane do funkcji rejestracja użytkownika a ona przesyła te dane do magazynu danych dla użytkownika. Przy operacjach na wydarzeniach użytkownik przesyła wydarzenie do odpowiedniej funkcji a ona przesyła wydarzenie do magazynu danych dla wydarzenia. Podczas wyświetlania kalendarza użytkownik wysyła dane użytkownika do funkcji, która pobiera dane o kalendarzu użytkownika z magazynu danych dla kalendarza i zwraca użytkownikowi jego kalendarz. W procesie dodawania powiadomienia użytkownik przesyła powiadomienie do funkcji która pobiera dane z magazynów danych dla wydarzenia i użytkownika oraz je kontroluje i przesyła powiadomienie użytkownika do magazynu danych dla powiadomienia. Podczas procesu logowania użytkownik wysyła swój login i hasło do funkcji odpowiedzialnej za logowanie, następnie ta funkcja przesyła login i hasło użytkownika do funkcji weryfikującej te dane. Funkcja weryfikująca pobiera dane z magazynu danych dla użytkownika i w przypadku poprawności zwraca do funkcji logowania token. Funkcja logowania w takim przypadku udziela użytkownikowi dostępu do strony.

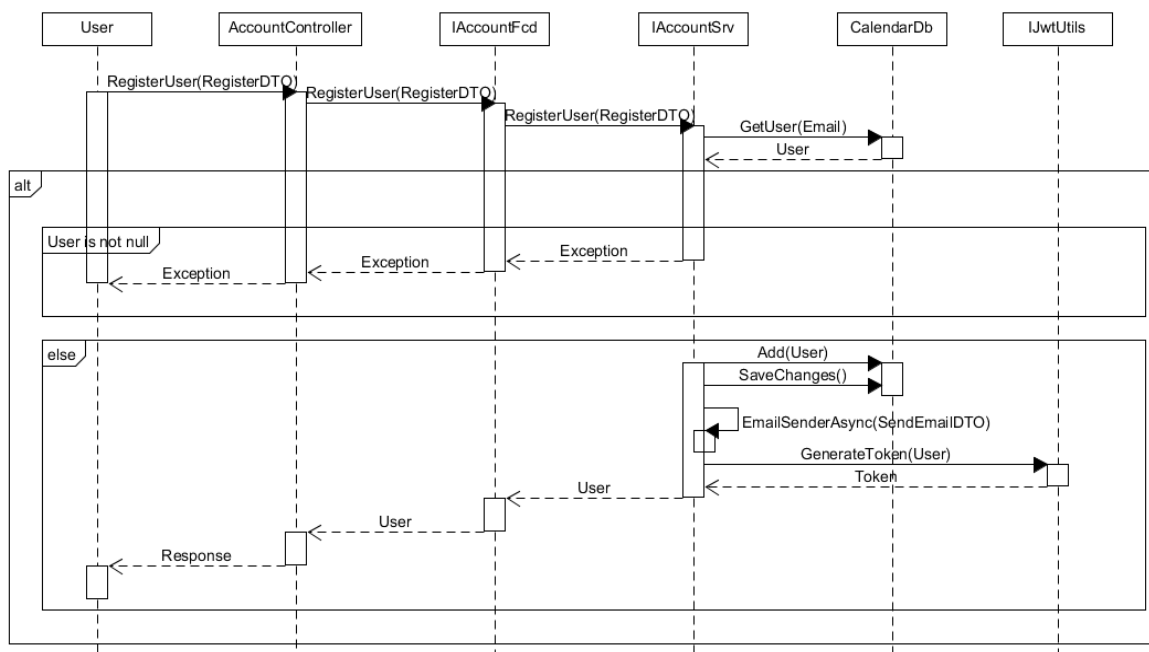
5. Diagram FHD



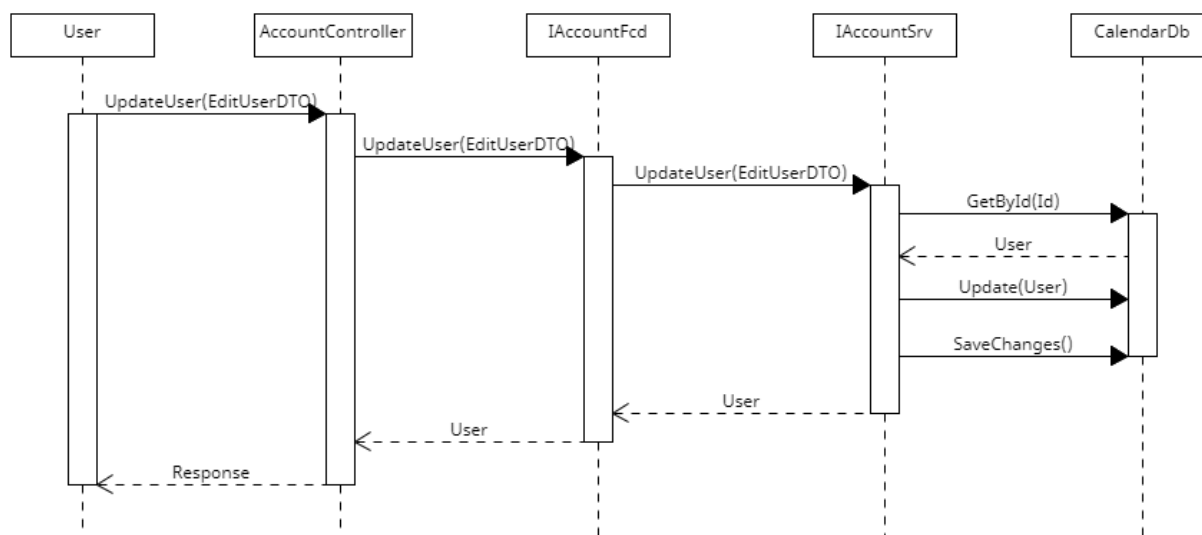
Kalendarz świąt nietypowych dzieli się na 2 główne podfunkcje odpowiadające za obsługę użytkownika oraz obsługę kalendarza. Obsługa użytkownika składa się z rejestracji użytkownika, logowania użytkownika i usuwania konta. W skład obsługi kalendarza wchodzi sprawdzanie świąt, dodawanie wydarzenia, usuwanie wydarzenia, modyfikacja wydarzenia, udostępnienie kalendarza, modyfikacja kalendarza, wyświetlanie kalendarza.

8. Diagramy sekwencji poszczególnych funkcjonalności i przepływu danych

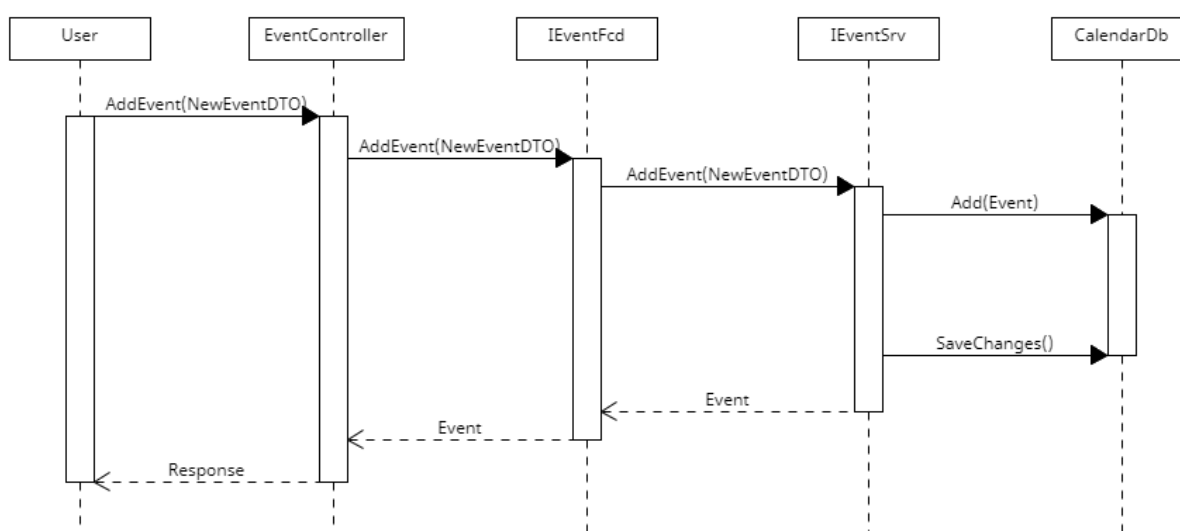
Rejestracja użytkownika:



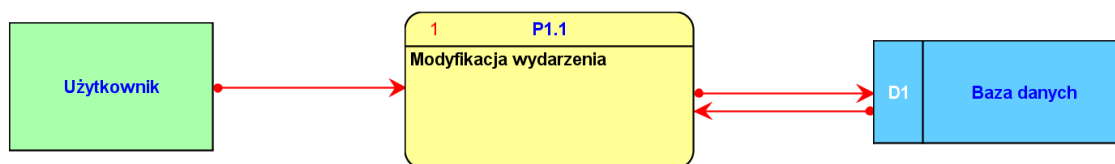
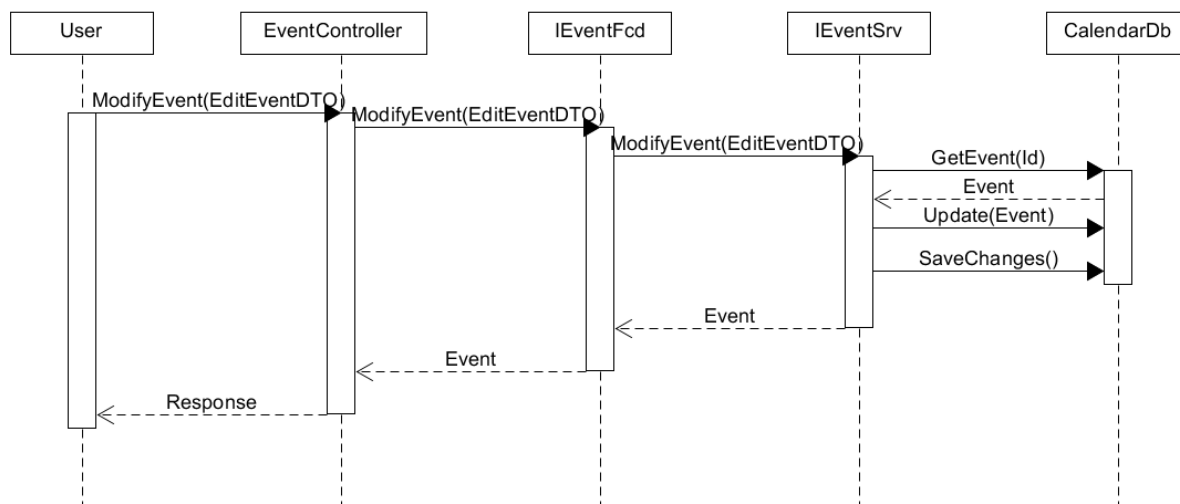
Modyfikacja użytkownika:



Dodawanie wydarzenia:

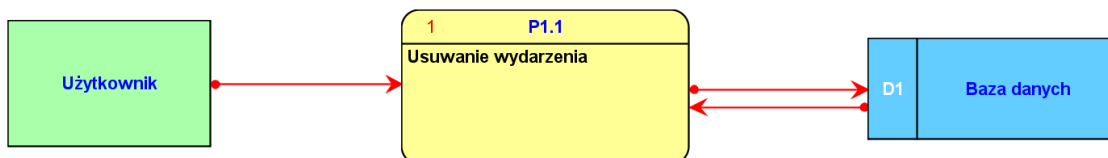
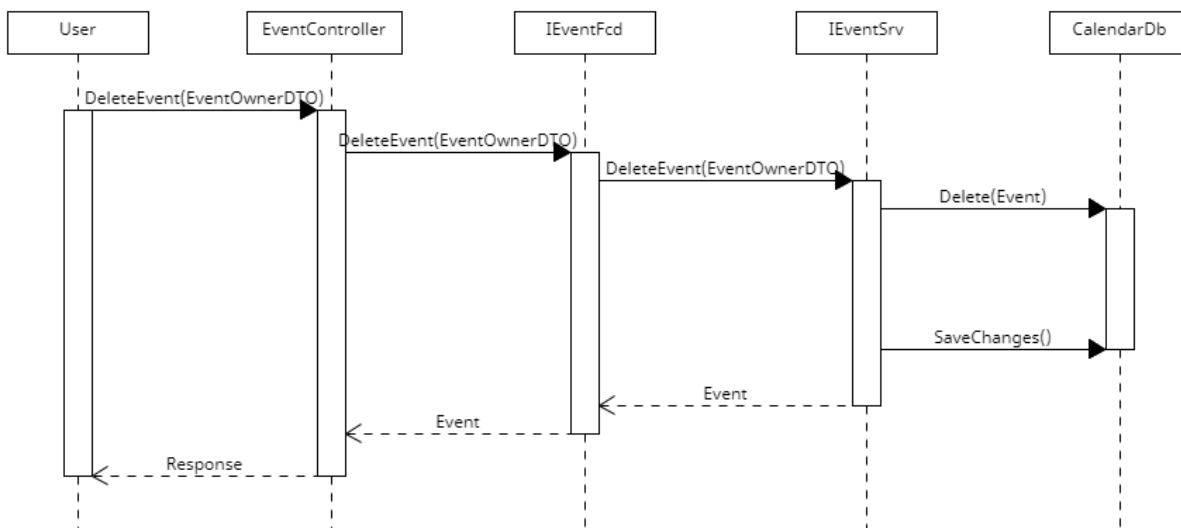


Modyfikacja wydarzenia:



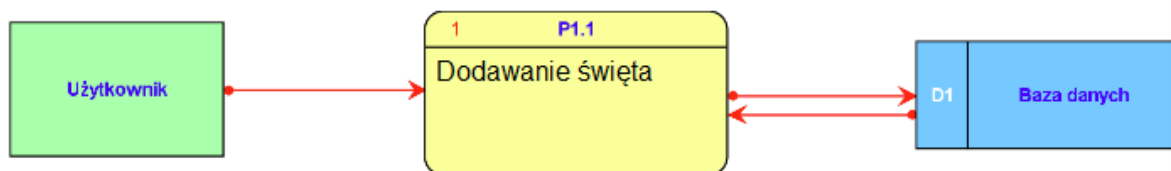
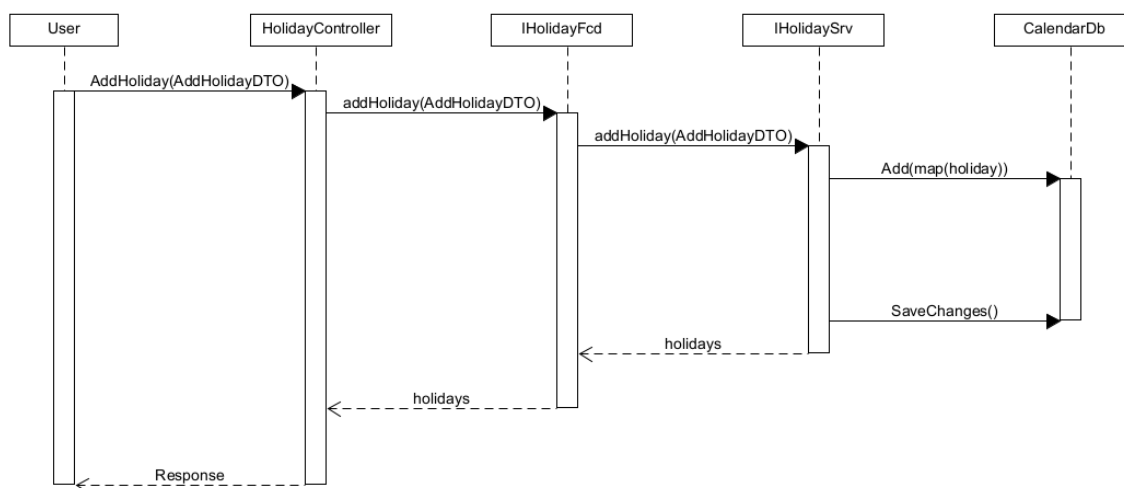
Ukazane powyżej diagramy (kolejno sekwencji oraz przepływu danych) ukazują działanie funkcji modyfikacji utworzonego wydarzenia. Użytkownik wchodzi w interakcję z systemem przy użyciu funkcji `ModifyEvent` na obiekcie `EditEventDTO`. Następnie przechodząc przez kolejne warstwy systemu na końcu następuje komunikacja z bazą danych gdzie zapisywane są wprowadzone modyfikacje.

Usuwanie wydarzenia:



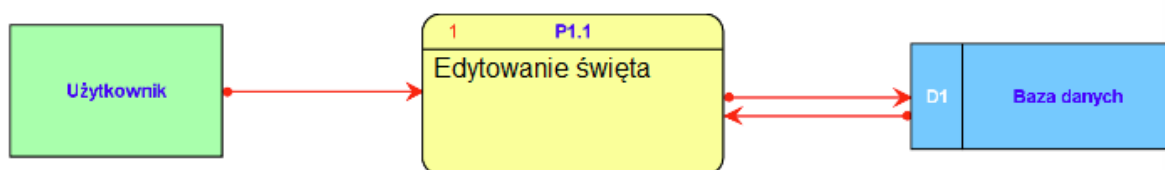
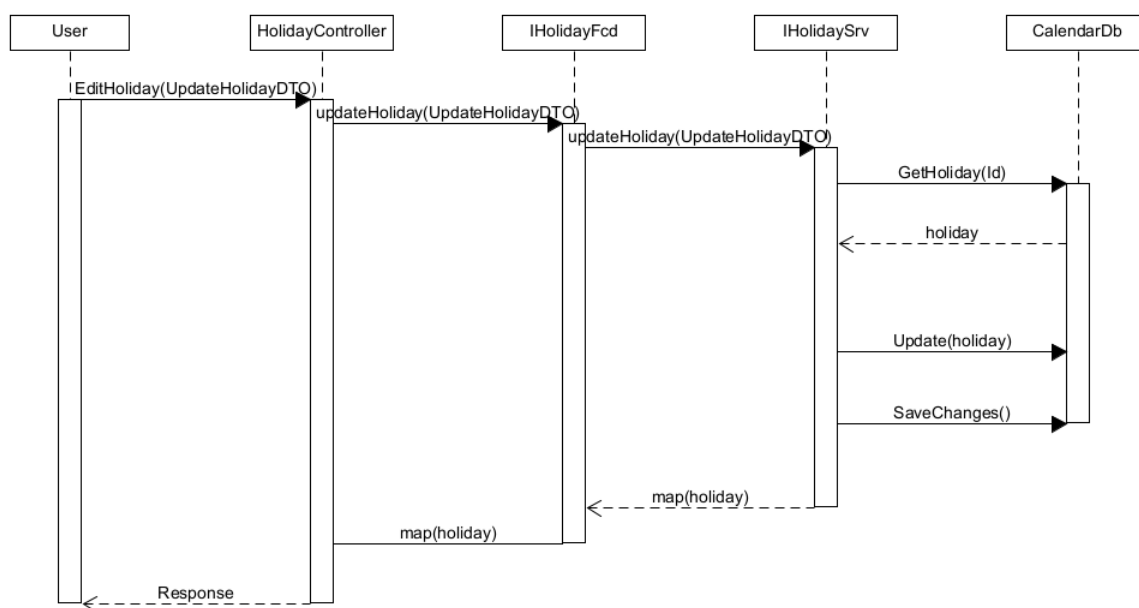
Ukazane powyżej diagramy (kolejno sekwencji oraz przepływu danych) ukazują działanie funkcji usuwania utworzonego wydarzenia. Użytkownik wchodzi w interakcję z systemem przy użyciu funkcji DeleteEvent na obiekcie EventOwnerDTO. Następnie przechodząc przez kolejne warstwy systemu na końcu następuje komunikacja z bazą danych skąd usuwane jest dane wydarzenie oraz zapisywane są wprowadzone zmiany.

Dodawanie Świąt:



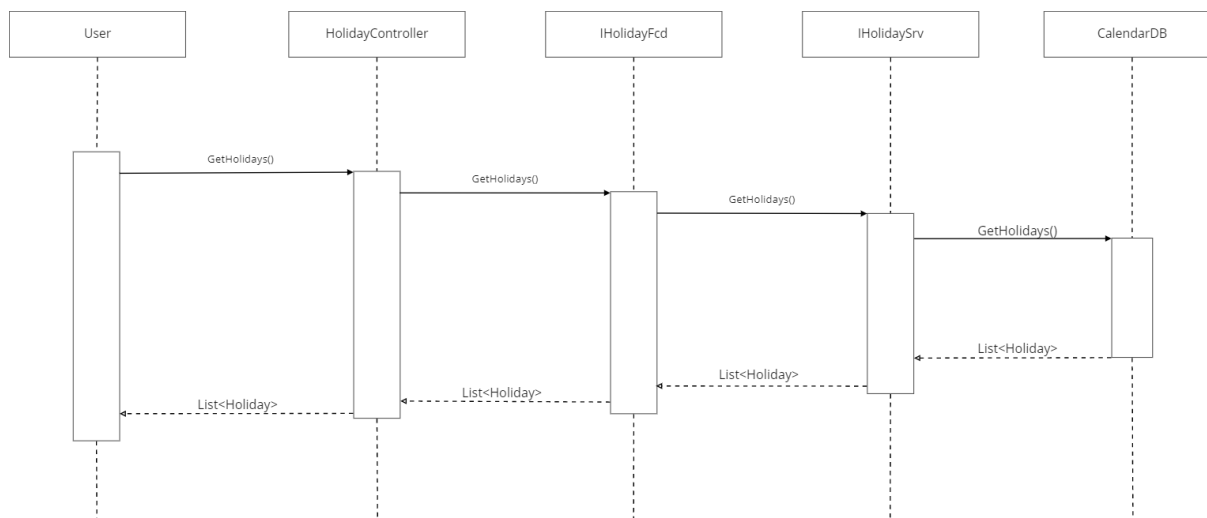
Ukazane powyżej diagramy (kolejno sekwencji oraz przepływu danych) ukazują działanie funkcji dodawania świąt. Użytkownik wchodzi w interakcję z systemem przy użyciu funkcji AddHoliday na obiekcie AddHolidayDTO. Następnie przechodząc przez kolejne warstwy systemu na końcu następuje komunikacja z bazą danych gdzie zapisywane są wprowadzone zmiany.

Edycja Święta:



Ukazane powyżej diagramy (kolejno sekwencji oraz przepływu danych) ukazują działanie funkcji edytowania święta. Użytkownik wchodzi w interakcję z systemem przy użyciu funkcji EditHoliday na obiekcie UpdateHolidayDTO. Następnie przechodząc przez kolejne warstwy systemu na końcu następuje komunikacja z bazą danych gdzie zapisywane są wprowadzone zmiany.

Pobieranie Świąt:



Ukazany powyżej diagram sekwencji ukazują działanie funkcji pobierania świąt. Użytkownik wchodzi w interakcję z systemem przy użyciu funkcji GetHolidays. Następnie przechodząc przez kolejne warstwy systemu na końcu następuje komunikacja z bazą danych gdzie przeprowadzana jest operacja pobrania danych.

9. Dokumentacja kodu źródłowego

Utworzone modele transferu danych (DTO):

Account:

- ChangePasswordDTO
- DeleteUserDTO
- EditUserDTO
- EmailDTO
- LoginDTO
- RegisterDTO
- ResetPasswordDTO

Event:

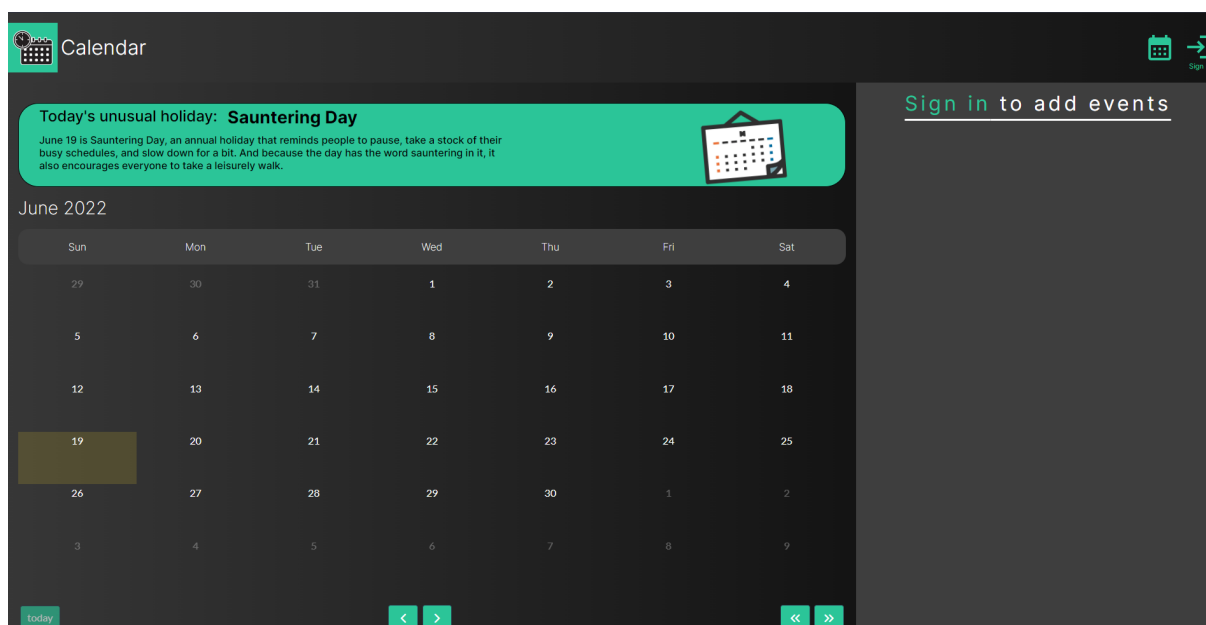
- NewEventDTO
- EditEventDTO
- EventOwnerDTO
- GetByUserDTO

Holiday:

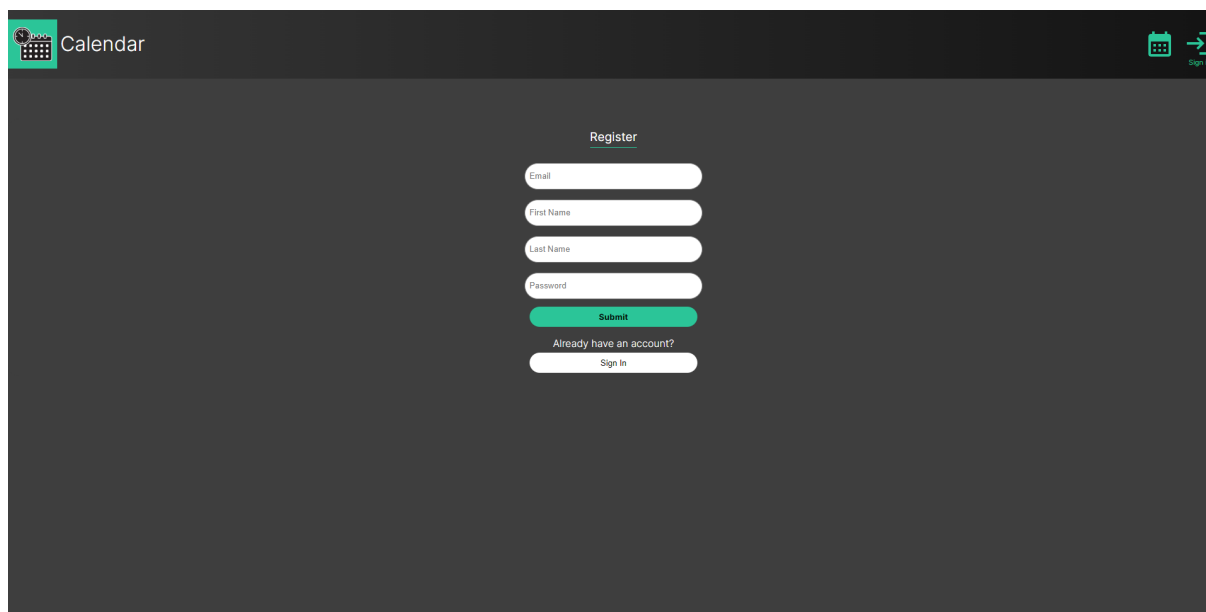
- AddHolidayDTO
- DeleteHolidayDTO
- UpdateHolidayDTO
- HolidayDTO

10. Dokumentacja użytkownika

Ekran główny aplikacji przed zalogowaniem:

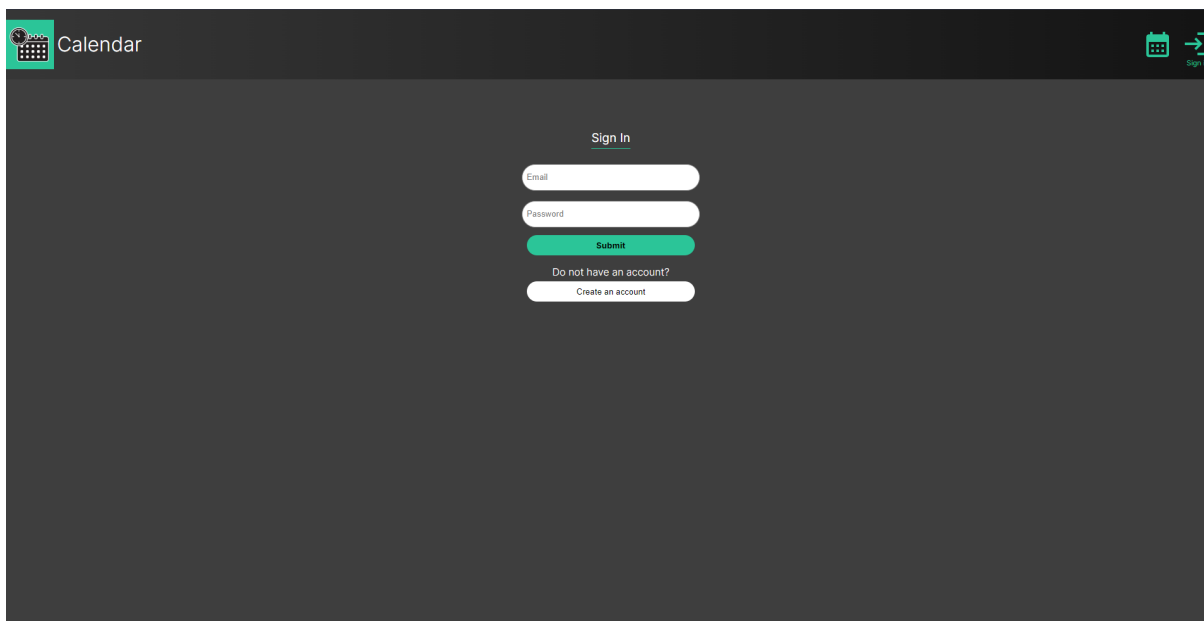


Rejestracja nowego użytkownika:

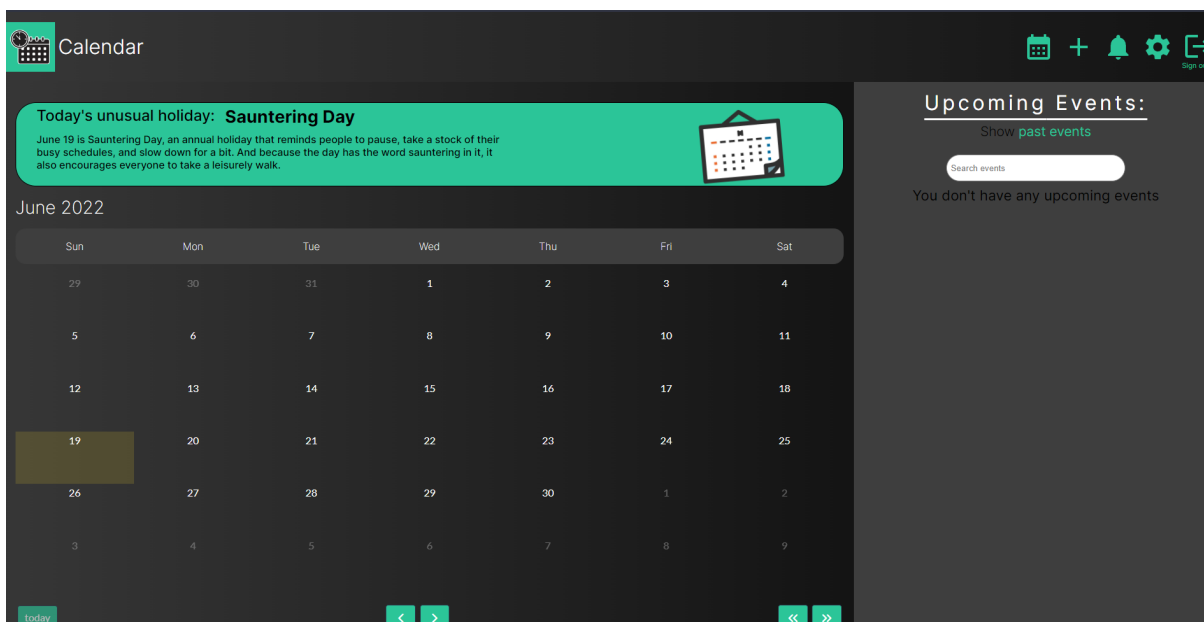


Dokończenie procesu rejestracji wymaga kliknięcia w link potwierdzający wysyłany na podany adres E-mail

Logowanie użytkownika:



Ekran główny aplikacji po zalogowaniu:



Today's unusual holiday: **Sauntering Day**
June 19 is Sauntering Day, an annual holiday that reminds people to pause, take a stock of their busy schedules, and slow down for a bit. And because the day has the word sauntering in it, it also encourages everyone to take a leisurely walk.

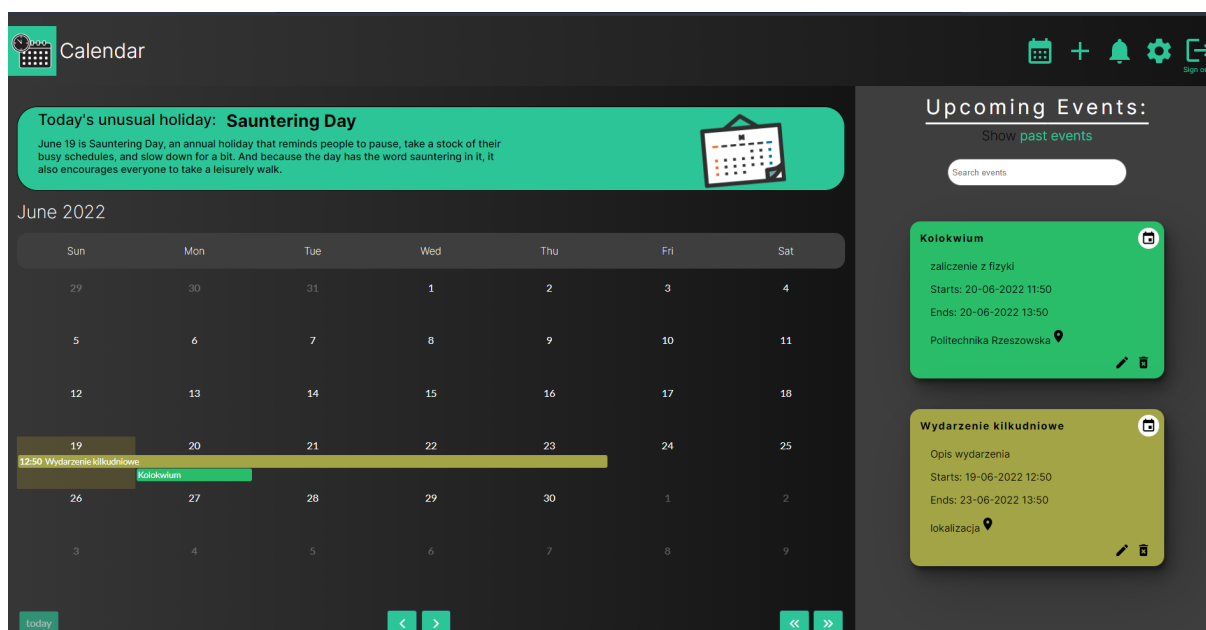
June 2022

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9


today < > << >>






Upcoming Events:
Show past events
Search events
You don't have any upcoming events

Główny ekran po dodaniu przykładowych wydarzeń:



Dodawanie nowego wydarzenia:

 Calendar


    


Add Event

Title

Description

Location


19.06.2022 09:50 






19.06.2022 09:50 

☒ Choose event color

☐ Recurring event

Zmiana ustawień użytkownika:

 Calendar

Change your Settings

Change your personal data:

name

surname

user@user.user

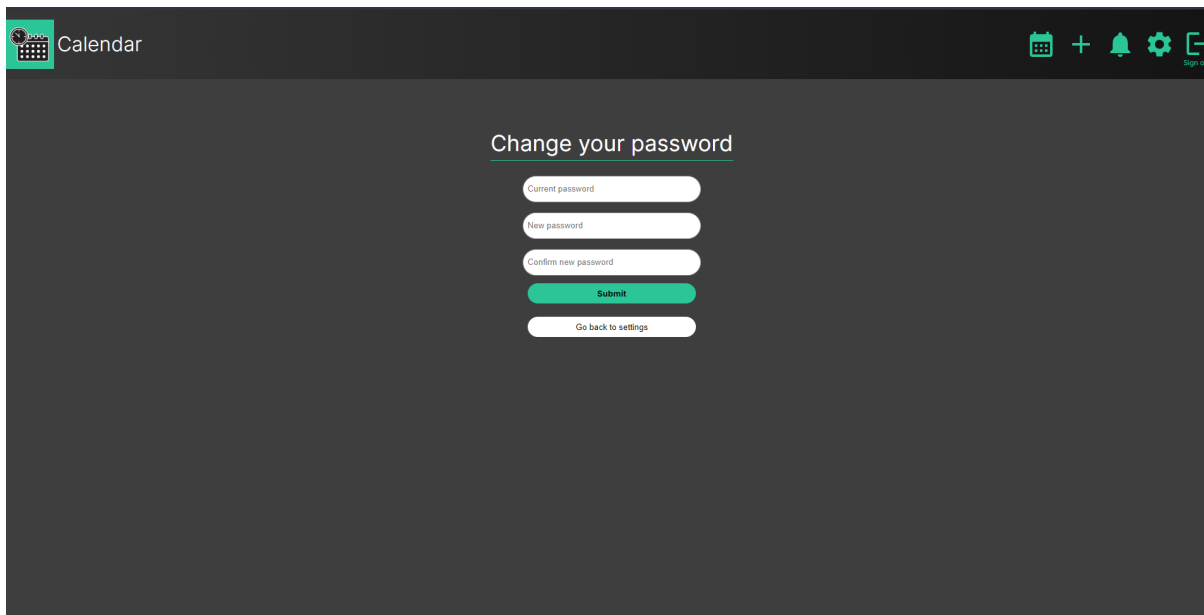
Choose a color of your Calendar:

☒ Cherry ☐ Orange ☐ Dark Lemon ☐ Monstera

☐ Default ☐ Sky ☐ Raspberry ☐ Lavender

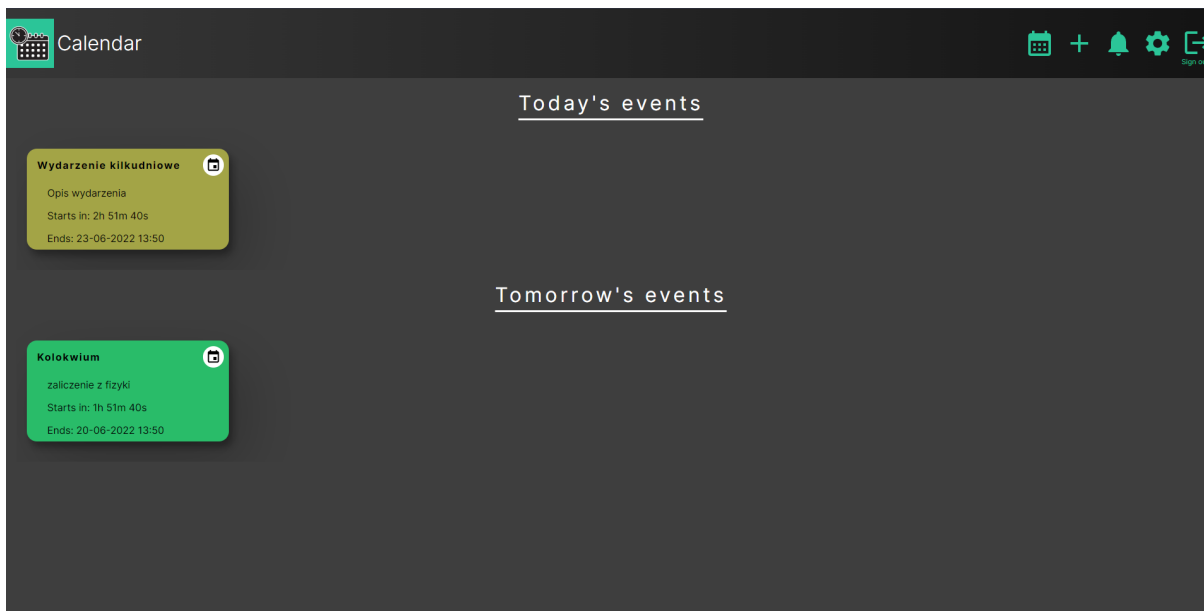
☒ Darkmode

Zmiana hasła:



The screenshot shows a web interface for a calendar application. At the top left is a 'Calendar' header with a calendar icon. At the top right are icons for calendar, add, notifications, settings, and a 'Sign out' button. The main content area is titled 'Change your password' and contains four input fields: 'Current password', 'New password', and 'Confirm new password'. Below these fields are two buttons: a green 'Submit' button and a white 'Go back to settings' button.

Powiadomienia użytkownika:



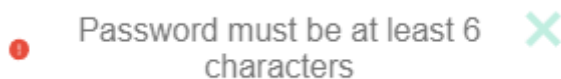
The screenshot shows the 'Today's events' section of the calendar application. It features two event cards. The first card, titled 'Wydarzenie kilkudniowe', is yellow and contains the text 'Opis wydarzenia', 'Starts in: 2h 51m 40s', and 'Ends: 23-06-2022 13:50'. The second card, titled 'Kolokwium', is green and contains the text 'zaliczenie z fizyki', 'Starts in: 1h 51m 40s', and 'Ends: 20-06-2022 13:50'. Both cards have a small camera icon in the top right corner. The interface also includes the 'Calendar' header and the top navigation icons.

Działanie akcji rejestracji użytkownika po stronie frontendu:

Po naciśnięciu przycisku "Sign in" uruchamiana jest funkcja handleSubmit:

```
const handleSubmit = (
  event: React.MouseEvent<HTMLButtonElement, MouseEvent>
) => {
  event.preventDefault();
  for (let [key, value] of Object.entries(credits)) {
    if (!registerValidator(key, value)) {
      return;
    }
  }
  dispatch(registerAction(credits));
  setCredits(registerInitialState);
};
```

Funkcja zapobiega domyślnemu działaniu akcji wysłania formularza (np. odświeżenie strony), następnie w pętli sprawdzane są wszystkie wpisane przez użytkownika wartości pod kątem poprawności (np. hasło musi być dłuższe niż 5 znaków), znajdują się one w zmiennej *credits*. W razie wykrycia błędnych danych wypisywany jest odpowiedni komunikat na ekranie.



Po poprawnym przejściu przez walidator danych wysyłana jest akcja *registerAction* z danymi i dane z formularza są czyszczone.

```
export const registerAction = createAsyncThunk(
  ADD_USER,
  async (credential: register) => {
    try {
      return await authSrv.register(credential);
    } catch (e: any) {
      return e.json();
    }
  }
);
```

Akcja *registerAction* uruchamia w serwisie autoryzacji funkcję *register* przesyłając dane dalej.

```
async register(credential: register) {  
  try {  
    return await api  
      .post(controllerPath + "register", credential)  
      .then((r) => r.data);  
  } catch (e) {  
    console.error(e);  
  }  
},
```

Funkcja register serwisu authSrv wysyła zapytanie do api i oczekuje na odpowiedź. Jeśli operacja się powiodła, użytkownik dostaje informacje o tym, że musi wejść na swoją pocztę i kliknąć link. Po tym zostanie on zweryfikowany i będzie mógł zalogować się na swoje konto.

Działanie akcji rejestracji użytkownika po stronie backendu:

Z frontendu napływa odpowiednie zapytanie, które jest przez program odpowiednio interpretowane i w zależności od ścieżki uruchamiany jest odpowiedni kod. W tym przypadku zapytanie jest wysyłane do AccountController, który przy wykorzystaniu interfejsu fasady, przesyła zapytanie dalej. Kontroler na podstawie ścieżki oraz otrzymanego czasownika http odróżnia, którą funkcję włączyć. Tak wygląda akcje dla rejestracji użytkownika w kontrolerze:

```
[HttpPost("register")]  
0 references | Wojciech Łuszczak, 1 day ago | 1 author, 2 changes  
public ActionResult RegisterUser([FromBody] RegisterDTO registerDTO)  
{  
  ...  
  return Ok(_accountFcd.RegisterUser(registerDTO));  
}
```

Następnie przechodzi do fasady, jest to wzorec projektowy, który odpowiada za wykonywanie funkcji nie związanych z połączeniem bazy danych. W tej funkcjonalności, jedyną akcją było zmniejszenie wszystkich liter e-mail, w przypadku, gdyby użytkownik wpisał dane z wciśniętym klawiszem Caps lock. Po zmniejszeniu liter, wykonywany jest kod z serwisu odpowiedzialnego za obsługę konta.

```
2 references | Wojciech Łuszczak, 9 days ago | 1 author, 4 changes
public UserDTO RegisterUser(RegisterDTO registerDTO)
{
    registerDTO.Email = registerDTO.Email.ToLower();
    return _accountSrv.RegisterUser(registerDTO);
}
```

W serwisie wykonywany jest cały kod odpowiedzialny za rejestrację. Na samym początku funkcji sprawdzane są otrzymane dane, jeśli użyty e-mail znajduje się już w bazie danych, program przerywa kod i wysyłany jest wyjątek z odpowiednią wiadomością. W przypadku, gdy e-mail jest wolny możliwe jest utworzenie użytkownika i dodanie go do bazy danych. Później utworzony zostaje model DTO użytkownika, który zostaje przesłany do funkcji wysyłającej mail z linkiem do potwierdzenia konta. Później hasło zostało

zaszyfrowane, tak, aby nie było jawnych informacji w bazie danych.

```
2 references | Wojciech Łuszczak, 9 hours ago | 1 author, 10 changes
public UserDTO RegisterUser(RegisterDTO registerDTO)
{
    var emailInUse = _dbContext.Users.Any(u => u.Email == registerDTO.Email);
    if (emailInUse) throw new EmailTakenException();
    var newUser = new User
    {
        Email = registerDTO.Email,
        FirstName = registerDTO.FirstName,
        LastName = registerDTO.LastName,
        PasswordHash = "",
        isVerified = false,
        IsDarkmode = true,
        Color = "#2BC598"
    };

    _dbContext.Users.Add(newUser);
    _dbContext.SaveChanges();

    var newUserDTO = new UserDTO
    {
        Id = _dbContext.Users.FirstOrDefault(u => u.Email == registerDTO.Email).Id,
        FirstName = newUser.FirstName,
        LastName = newUser.LastName,
        Email = newUser.Email
    };

    var emailToSend = new SendEmailDTO() { user = newUserDTO, Email = registerDTO.Email };
    EmailSenderAsync(emailToSend);

    var hashedPassword = _passwordHasher.HashPassword(newUser, registerDTO.Password);
    newUser.PasswordHash = hashedPassword;

    _dbContext.SaveChanges();

    var userAuth = new UserAuthorizeDTO
    {
        Id = newUser.Id,
        Email = newUser.Email,
        FirstName = newUser.FirstName,
        LastName = newUser.LastName,
        PasswordHash = newUser.PasswordHash
    };

    return new UserDTO()
    {
        Id = newUser.Id,
        Email = newUser.Email,
        FirstName = newUser.FirstName,
        LastName = newUser.LastName,
        Token = null,
        Color = "#2BC598",
        IsDarkmode = true
    };
}
```

Na frontend zostaje wysłany odpowiednie dane i na podstawie ich oraz architektury stanów, w zależności od stanu zostaje wywołane odpowiednie akcje.

Działanie akcji dodania wydarzenia po stronie frontendu:

Po naciśnięciu przycisku “Submit” uruchamiana jest funkcja handleSubmit:

```
const handleSubmit = (e: React.MouseEvent<HTMLButtonElement, MouseEvent>) => {
  e.preventDefault();
  credits.authorId = currentAuthorId;
  credits.isRecurring = checked;
  credits.isDeleted = false;
  for (let [key, value] of Object.entries(credits)) {
    if (!addEventValidator(key, value)) {
      return;
    }
  }
  if (!dateValidator(credits.startEvent, credits.endEvent)) return;
  dispatch(
    addEventAction({
      authorId: credits.authorId,
      title: credits.title,
      description: credits.description,
      location: credits.location,
      participantsEmails: credits.participantsEmails,
      startEvent: credits.startEvent,
      endEvent: credits.endEvent,
      color: credits.color,
      isRecurring: credits.isRecurring,
      isDeleted: credits.isDeleted,
    })
  );
  setCredits(initialState);
  setChecked(false);
};
```

Funkcja ta jest bardzo podobna do poprzedniej, różni się walidatorem danych oraz tym, że ręcznie są przypisane: id użytkownika, powtarzalność wydarzenia, oraz flaga isDeleted ustawiana jest na false. Następnie wysyłana jest akcja i czyszczony jest formularz.

```
export const addEventAction = createAsyncThunk(
  ADD_EVENT,
  async (credential: addEvent) => {
    try {
      return await eventSrv.addEvent(credential);
    } catch (e: any) {
      return e.json();
    }
  }
);
```

Akcja `addEventAction` uruchamia w serwisie wydarzeń funkcję `addEvent` przysyłając dane dalej.

```
async addEvent(credential: addEvent) {  
  try {  
    return await api  
      .post(controllerPath + "create", credential)  
      .then((r) => r.data);  
  } catch (e) {  
    console.error(e);  
  }  
},
```

Funkcja `register` serwisu `eventSrv` wysyła zapytanie do api i oczekuje na odpowiedź.

```
.addCase(addEventAction.fulfilled, (state, action) => {  
  if (action.payload !== undefined) {  
    const newEvent: event = {  
      id: action.payload.id,  
      authorId: action.payload.authorId,  
      title: action.payload.title,  
      description: action.payload.description,  
      location: action.payload.location,  
      participantsEmails: action.payload.participantsEmails,  
      startEvent: action.payload.startEvent,  
      endEvent: action.payload.endEvent,  
      color: action.payload.number,  
      isDeleted: action.payload.isDeleted,  
      isRecurring: action.payload.isRecurring,  
      isEditable: true,  
    };  
    const newEvents = state.events;  
    newEvents.push(newEvent);  
    state.events = newEvents;  
  }  
})
```

Po poprawnym wykonaniu zapytania oraz uzyskaniu odpowiedzi tworzone jest nowe wydarzenie i dodawane jest do istniejących wydarzeń w `state`. `State` jest to pojemnik na dane przechowywane po stronie frontu, z którego poszczególne komponenty mogą wyciągać informacje i zmieniać wygląd aplikacji oraz wyświetlane dane.

Przykład:

```
const calendarColors = useAppSelector((state) => SelectColors(state));
```

ta linijka kodu wywołuje selector `SelectColors`, który przypisuje do zmiennej `calendarColors` informacje o wyglądzie kalendarza.

```
export const SelectColors = (state: RootState) => {  
  return {  
    color: state.currentUser.color,  
    isDarkmode: state.currentUser.isDarkmode  
  }  
}
```

Selector pobiera z *state* bieżący kolor kalendarza oraz informacje o trybie wyświetlania.