Raport tygodniowy 7.04 – 20.04

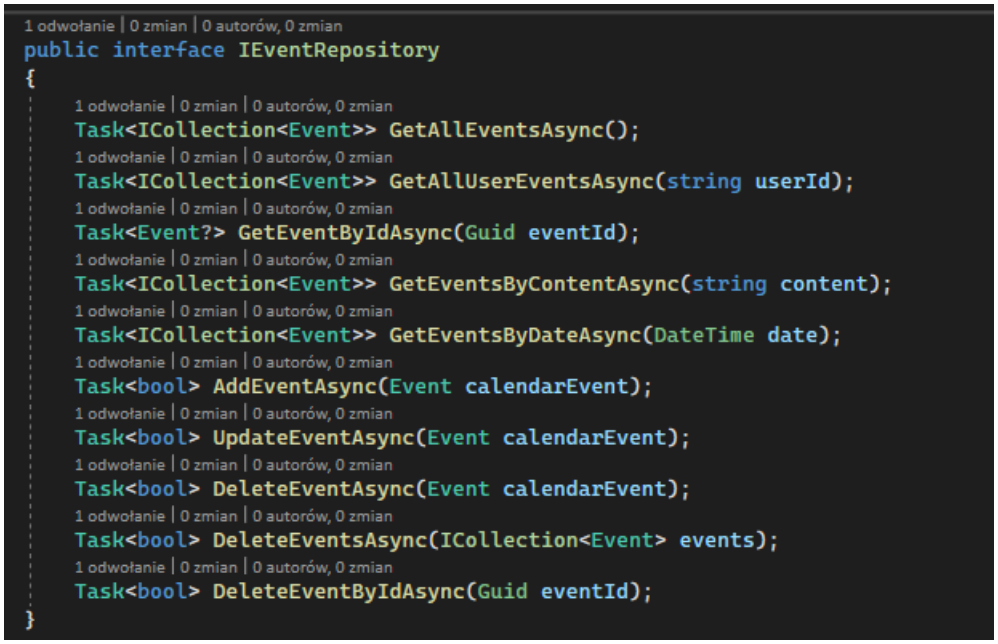
W tym czasie wraz z całą sekcją omówiliśmy jakie będą niezbędne klasy w projekcie – jaki będzie ogólny zarys bazy danych.

Następnie czytałam i słuchałam na temat wzorca projektowego repozytorium (poniżej linki):

https://www.programmingwithwolfgang.com/repository-pattern-net-core/

https://www.youtube.com/watch?v=lI9oOGuXaQY

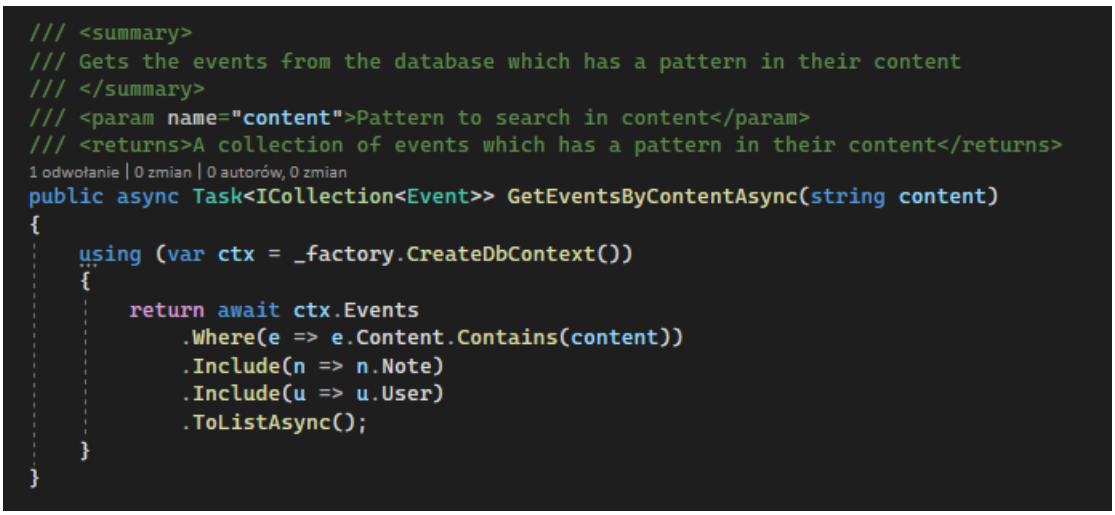
Kolejnym etapem mojej pracy było napisanie repozytorium do encji wydarzenia. Pracę zaczęłam od utworzenia interfejsu IEventRepository zawierającego deklaracje metod do dodawania, usuwania, modyfikacji i odczytywania wydarzenia z bazy danych.



```
1 odwołanie | 0 zmian | 0 autorów, 0 zmian
public interface IEventRepository
{
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<ICollection<Event>> GetAllEventsAsync();
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<ICollection<Event>> GetAllUserEventsAsync(string userId);
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<Event?> GetEventByIdAsync(Guid eventId);
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<ICollection<Event>> GetEventsByContentAsync(string content);
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<ICollection<Event>> GetEventsByDateAsync(DateTime date);
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<bool> AddEventAsync(Event calendarEvent);
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<bool> UpdateEventAsync(Event calendarEvent);
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<bool> DeleteEventAsync(Event calendarEvent);
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<bool> DeleteEventsAsync(ICollection<Event> events);
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    Task<bool> DeleteEventByIdAsync(Guid eventId);
}
```

Rysunek 1 Deklaracje metod w interfejsie

Następnie utworzyłam klasę EventRepository oraz zaimplementowałam metody z utworzonego interfejsu.



```
/// <summary>
/// Gets the events from the database which has a pattern in their content
/// </summary>
/// <param name="content">Pattern to search in content</param>
/// <returns>A collection of events which has a pattern in their content</returns>
1 odwołanie | 0 zmian | 0 autorów, 0 zmian
public async Task<ICollection<Event>> GetEventsByContentAsync(string content)
{
    using (var ctx = _factory.CreateDbContext())
    {
        return await ctx.Events
            .Where(e => e.Content.Contains(content))
            .Include(n => n.Note)
            .Include(u => u.User)
            .ToListAsync();
    }
}
```

Rysunek 2 Implementacja metody GetEventByContentAsync

Kod interfejsu IEventRepository oraz klasy EventRepository dodałam do dokumentacji na stronę o nazwie repozytorium.

Ostatnią rzeczą jaką zrobiłam było utworzenie pliku dla testów zaimplementowanego przeze mnie repozytorium oraz napisanie pierwszych testów. Przykładowy test przedstawiony został poniżej:

```
[Fact(DisplayName = "User is able to add a event to the database")]
0 | Odwołania: 0 | 0 zmian | 0 autorów, 0 zmian
public async Task AddEvent()
{
    // Arrange
    var ne = new EventRepository(new EventRepositoryShould());
    DateTime date = DateTime.Now;
    var calendarEvent = new Event(null, "example content", date, date, new ApplicationUser());

    // Act
    var result = await ne.AddEventAsync(calendarEvent);

    // Assert
    var events = await _context.Events.ToListAsync();
    Assert.True(result);
    Assert.Single(events);
    await ne.DeleteEventAsync(calendarEvent);
}
```

Rysunek 3 Test metody AddEventAsync