Uniwersytet Zielonogórski
Wydział Informatyki, Elektrotechniki i Automatyki

Platforma .NET – Projekt
Prowadzący: dr inż. Marek Sawerwain

Tytuł raportu/sprawozdania

Wykonał: Damian Radecki, Grupa dziekańska: 33-INF-SSI-SP
Projekt realizowano razem z:
Damian Kurkiewicz

Data oddanie projektu: DD mmmm YYYY

Ocena: .....

Spis treści

Spis listingów

Motto:
Pisanie raportu przywilejem każdego studenta.

1 Wprowadzenie

1.1 Aplikacje czasu rzeczywistego

Aplikacje działające na żywo oferują wiele korzyści, które są sporym ułatwieniem dla użytkowników podczas używania takiej aplikacji. Czynności wykonywane bez odświeżania strony nie tylko skracają czas wykonywania czynności czy obsługiwania samej witryny to jeszcze znacznie ułatwiają komunikację, unikają blokowania strony i tworzą bardziej intuicyjny interface. Takie aplikacje internetowe staja się normą w dzisiejszych czasach. Każde przeładowanie strony jest nie komfortowe i stwarza pewnego rodzaju niebezpieczeństwo wykradnięcia danych. Serwisy internetowe obsługujące komunikację real-time z klientem są lepiej zabezpieczone i działają wydajnościowo lepiej. Powstawało wiele technologii do wsparcia komunikacji na żywo, które działają zarówno po stronie witryny i serwera. Są to między innymi WebSocket, SignalR, RabbitMQ czy Apache Kafka. Wszystkie z nich są dziś globalnie używane do wsparcia przekazu informacji.

1.2 Opis działania

Projekt chatu na żywo jest aplikacją, która wspiera komunikację między użytkownikami, aby ich konwersacje nie działały w stylu w jakim działa klasyczny serwer e-mail. Założenie projektu są takie, aby użytkownicy bez przeładowania strony mogli wymieniać miedzy sobą wiadomości. Dodatkowo wszelkie powiadomienia przychodzą również bez zbędnego odświeżania witryny. Sprawia to, że witryna jest bardziej intuicyjna, łatwiejsza i szybsza w obsłudze. Taka architektura aplikacji jest przyjazna użytkownikowi, od którego będzie wymagana minimalny wysiłek w trakcie używania strony. Celem takiej aplikacji jest też maksymalne bezpieczeństwo wspierane przez bearen token i autoryzację użytkowników z zachowaniem szyfrowania danych poufnych. Architektura zapewnia, że nie będziemy otrzymywać wiadomości od niezaakceptowanych użytkowników lecz daje możliwość uczestnictwa w czatach posiadających osoby nieznajome poprzez mechanizm grup. W grupach każdy użytkownik może zaprosić swoich znajomych co może spowodować komunikację między nieznajomymi w danym czacie.

1.3 Grupa docelowa

Grupą docelową są wszyscy użytkownicy którzy cenią sobie bezpieczeństwo i wygodę. Chcą szybko skomunikować się ze swoimi przyjaciółmi bądź grupą docelową bez żadnych opóźnień czy niepotrzebnych przeładować strony. Są pewni tego, że ich dane są przechowywane w bezpiecznym miejscu i nikt nie wkradnie się na ich konto. Mogą to być zarówno firmy, które chcą komunikować się między sobą i ewentualnie z klientami poprzez utworzenie chatu dla grupy użytkowników jak i dla szkół, uniwersytetów, grup pracowników, przyjaciół czy kolegów.

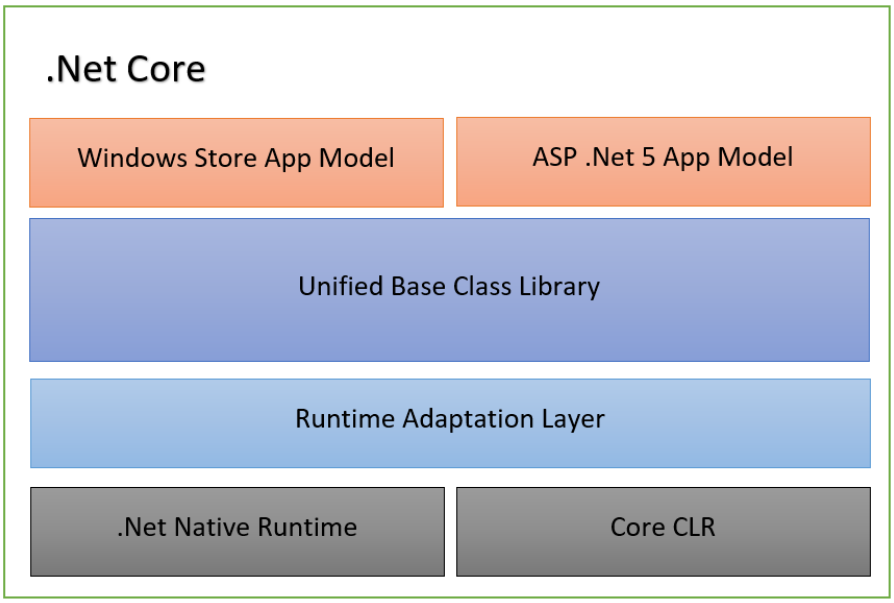
2 Użyte technologie

2.1 .Net Core

Popularny, nowoczesny i wydajny framework oparty o otwartoźródłowa implementację, który został wydany w 2016 roku do ogólnego przeznaczenia. Stanowi zestaw bibliotek pozwalający tworzyć wieloplatformowe aplikacje o wysokim stopniu bezpieczeństwa. Framework ten pozwala na pisanie aplikacji między innymi przeznaczonych do obliczeń chmurowych, IoT oraz jak w naszym przypadku do pisania web serwisu.

Rysunek 1: Logo .Net Core

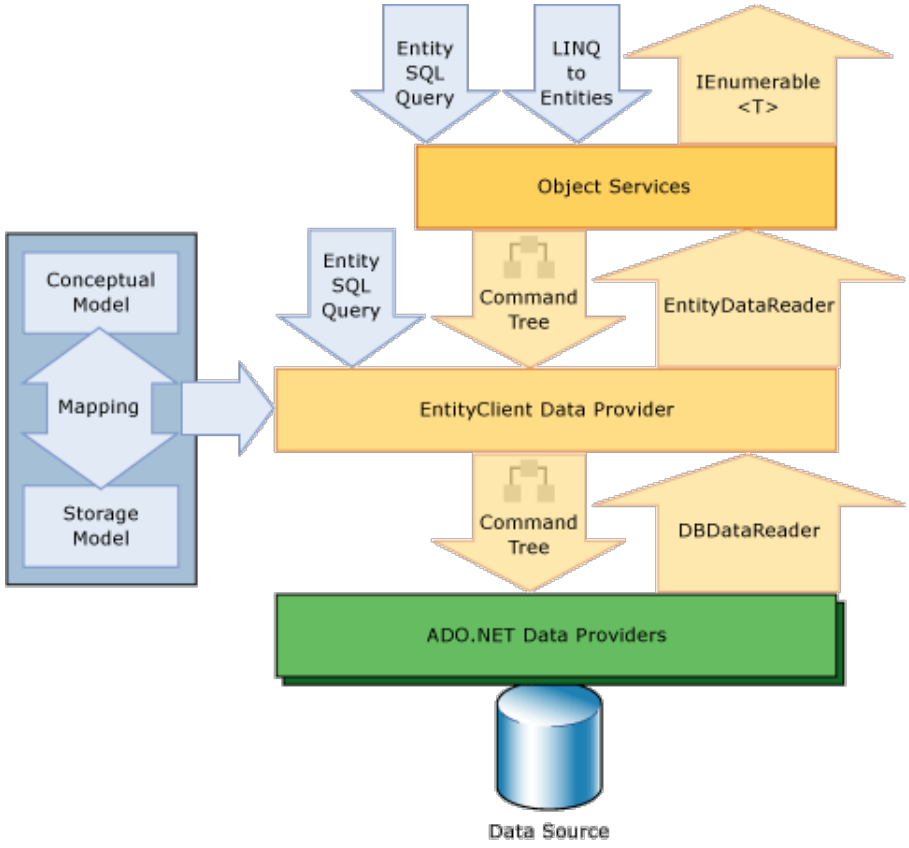
Framework .Net Core został przez nas wybrany, ponieważ jest to nowa oraz dobrze prosperująca technologia wprowadzająca dużą dawkę świeżości podczas tworzenia nowego oprogramowania. Posiada wsparcie dla tworzenia web serwisów opartych o metodykę REST, poprzez dodanie nowych i gotowych do działania bibliotek. Platforma .Net Core jest znacznie wydajniejsza od .Net Framework. Wprowadza znaczące usprawnienia przekładające się na szybkość działania pisanych programów. Architektura przedstawia się w sposób następujący.



Rysunek 2: Architektura .NET Core

2.2 Entity Framework

Entity Framework to zestaw technologii ADO.NET, obsługujące opracowywanie aplikacji zorientowanych na dane. Framework ten pozwala modelować konkretne jednostki, relację czy też logikę działania bazy danych. Obsługuję wiele systemów magazynowania danych takich jak na przykład MySQL, który to został przez nas wybrany do realizacji projektu. Całość jest konfigurowalna z poziomu kodu co jest bardzo wygodnym rozwiązaniem. Pozwala deweloperom na współpracę z danymi w postaci obiektów. Model fizyczny jest rafinowany przez administratorów bazy danych w celu zwiększenia wydajności, ale programiści piszący kod aplikacji przede wszystkim zawężają się do pracy z modelem logicznym, pisząc zapytania SQL i wywołując procedury składowane. Modele domen są zwykle używane jako narzędzie do przechwytywania i komunikowania się z wymaganiami aplikacji, często tak jak w przypadku diagramów obojętnych, które są wyświetlane i omówione w wczesnych etapach projektu, a następnie porzucone. Wiele zespołów programistycznych pomija Tworzenie modelu koncepcyjnego i rozpoczyna się od określenia tabel, kolumn i kluczy w relacyjnej bazie danych.

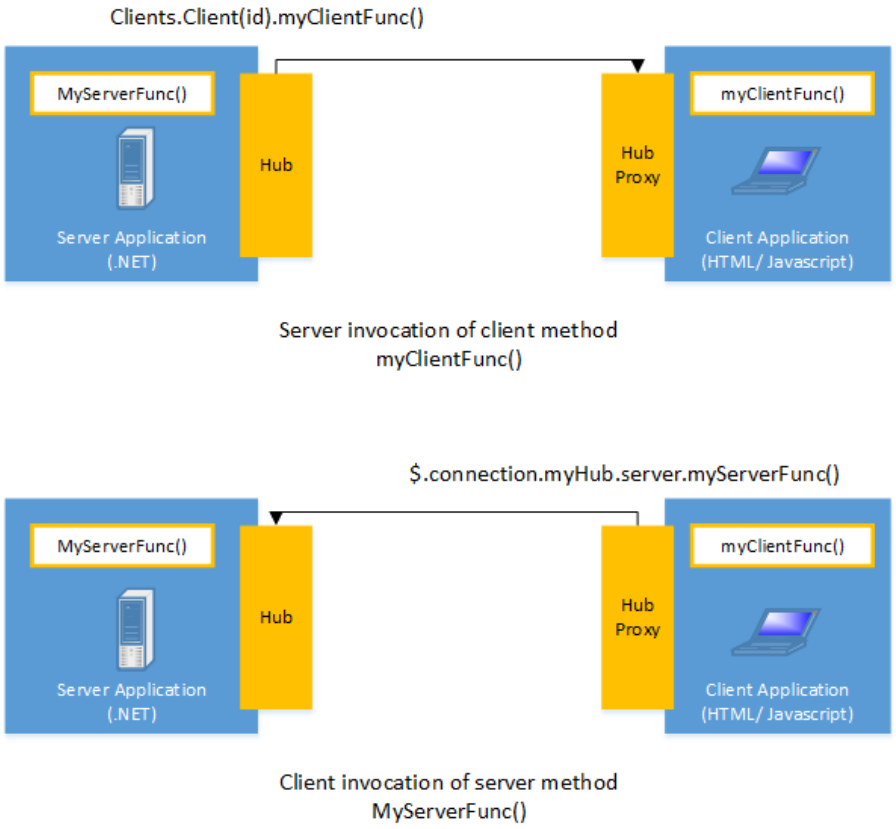


Rysunek 3: Architektura Entity Framework do uzyskania dostępu do danych

2.3 SignalR

Biblioteka dla dewelpoarów ASP.NET, która przyśpiesza proces dodawania funkcji sieci web w czasie rzeczywistym do aplikacji. Często wykorzystuje się tą bibliotekę w czatach internetowych, lecz może ona o wiele więcej. SignalR może być używany też w takich aplika-

cjach jak pulpity nawigacyjne, aplikacje do monitorowania czy formularze działające w czasie rzeczywistym. Biblioteka umożliwia również zupełnie nowe typy aplikacji internetowych, które wymagają aktualizacji wysokiej częstotliwości z serwera, na przykład gier w czasie rzeczywistym. Komunikacja między stroną a serwerem odbywa się poprzez tak zwane Hub'y. Bardzo dobrze ukazuje to rysunek poniżej.



Rysunek 4: SignalR przykład działania

2.4 MySql

Baza danych rozwijana przez firmę Oracle. MySql jest to otwarto źródłowy system zarządzania relacyjnymi bazami danych. System ten obsługuję język zapytań SQL, który służy do pisania zapytań do tej bazy. MySql jest relacyjną bazą danych. Model relacyjny w prosty i intuicyjny sposób przedstawia dane w tabelach. Każdy wiersz w tabeli jest rekordem z unikatowym identyfikatorem zwanym kluczem. Kolumny tabeli zawierają atrybuty danych, a każdy rekord zawiera zwykle wartość dla każdego atrybutu, co ułatwia ustalenie relacji między poszczególnymi elementami danymi.

Rysunek 5: Logo MySQL

2.5 Angular

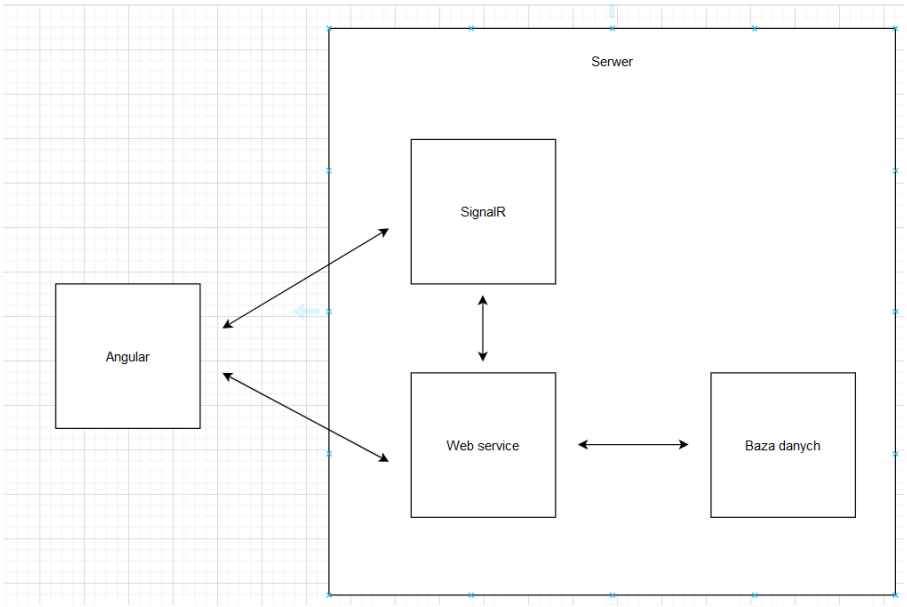
Otwarto źródłowy framework JavaScript, zaprojektowany i napisany przez inżynierów z Google. Ich celem było zrewolucjonizowanie projektowania części wizualnej stron internetowych. Szybko zyskał popularność wśród programistów JavaScript, którzy zaczęli odstawiać go na rzecz jQuery. Jego największą zaletą oraz najbardziej rozpoznawalną cechą jest integracja z atrybutami HTML. Framework ten umożliwia proste wdrożenie wzorca MVC(Model-View-Controller), dzięki czemu testowanie jak i rozwój aplikacji nie sprawia wielu problemów.

Rysunek 6: Logo Angular

3 Projekt

3.1 Struktura projektu

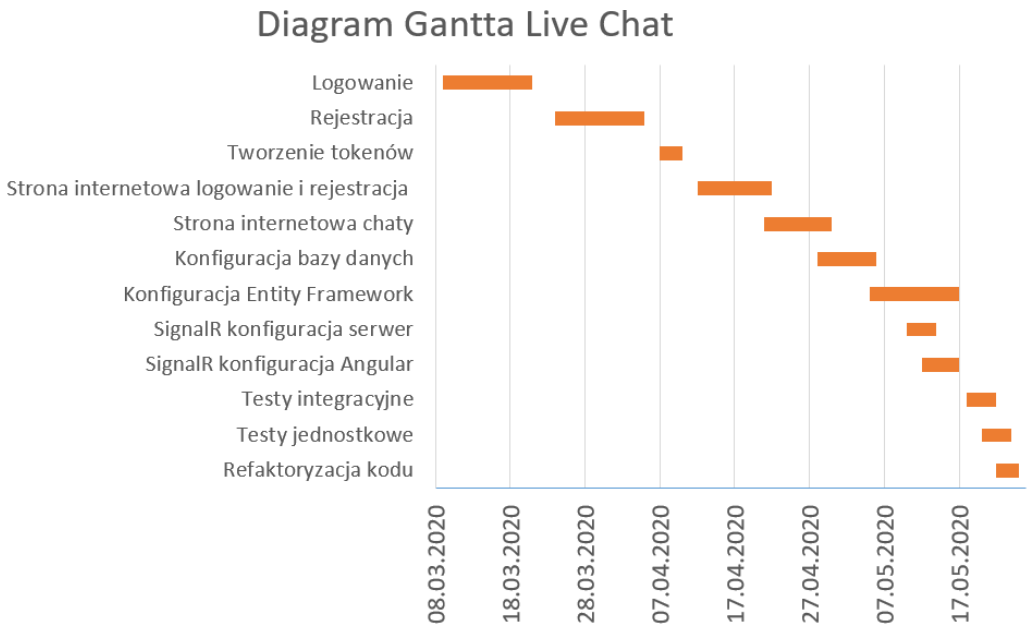
Diagram pokązujący architekturę całego projektu opisuje główne komponenty i występujące między nimi połączenia. Strona internetowa napisana w Angularze jest podłączona bezpośrednio z signalR i web serwisem. Miedzy nim występują połączenia dwukierunkowe. Z tym pierwszym strona internetowa utrzymuje stałe połączenie, aby zapewnić natychmiastową wymianę danych. Drugi wymieniony komponent odpowiada na zadane żądania. SignalR został wbudowany w web serwis jako integralna część, ale jest traktowany osobno, ponieważ utrzymuje z klientem innego typu połączenie. Między nimi także następuje wymiana danych. Ostatnim komponentem projektu jest baza danych, która jest podłącza tylko do web serwisu. To on stanowi most do pobierania wszelkich zawartych tam informacji.



Rysunek 7:

3.2 Diagram Gantta

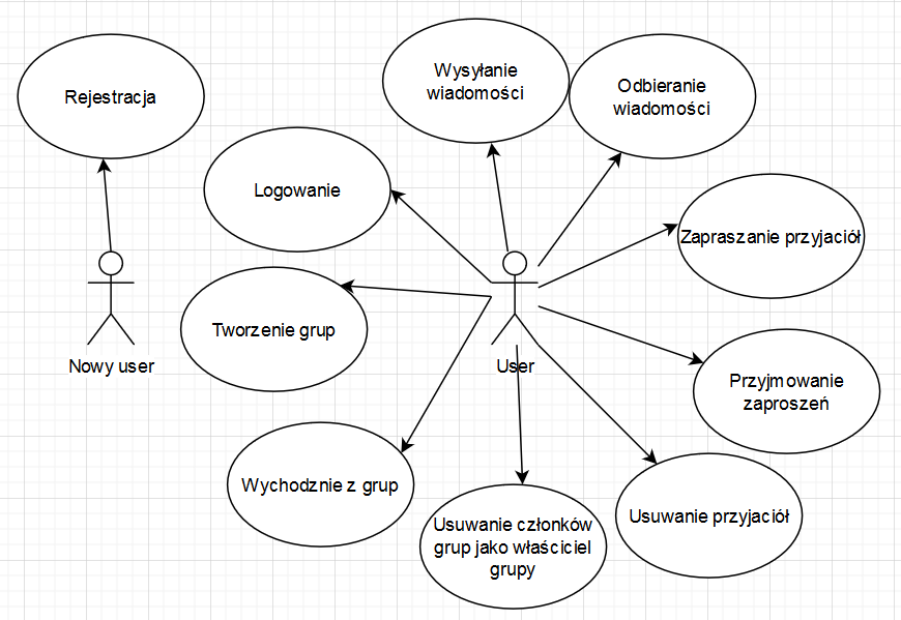
Diagram Gantta dla naszego projektu przedstawia się następująco:



Rysunek 8: Diagram Gantta Live Chat

3.3 Use Cases

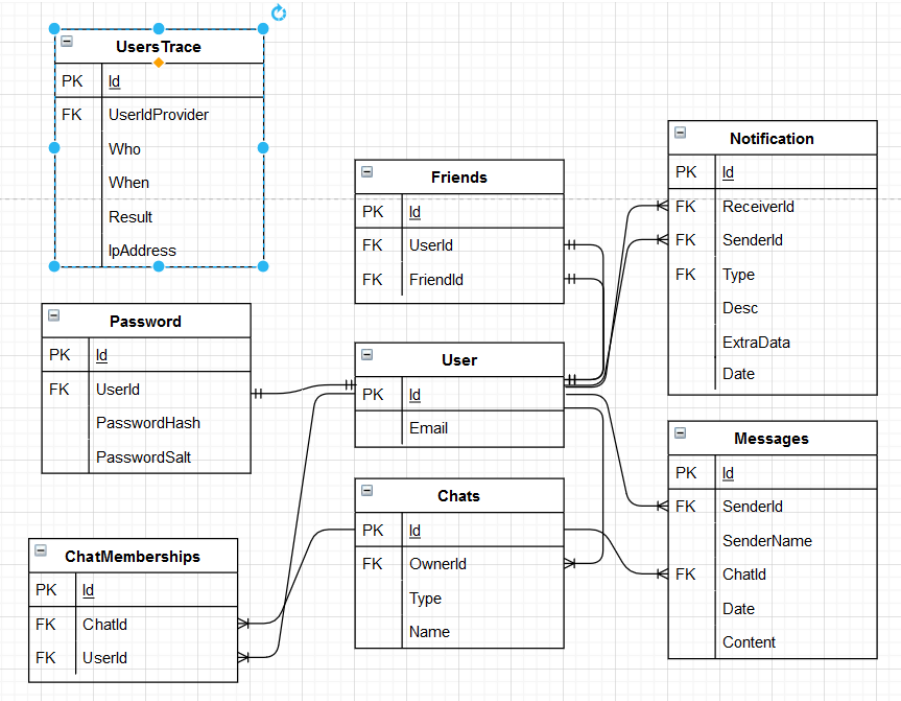
Use case pokazują fukcjonalności projektu z podziałem na rodzaje użytkownik danej aplikacji. W live chat bedziemy wyrózniać dwa rodzaje użytkowników. Będzie to nowy user, który będzie miał możliwość jedynie rejestracji i drugi to będzie już utworzony user, który będzie mógł korzystać ze wszystkich korzyści programu.



Rysunek 9:

3.4 Struktura bazy danych

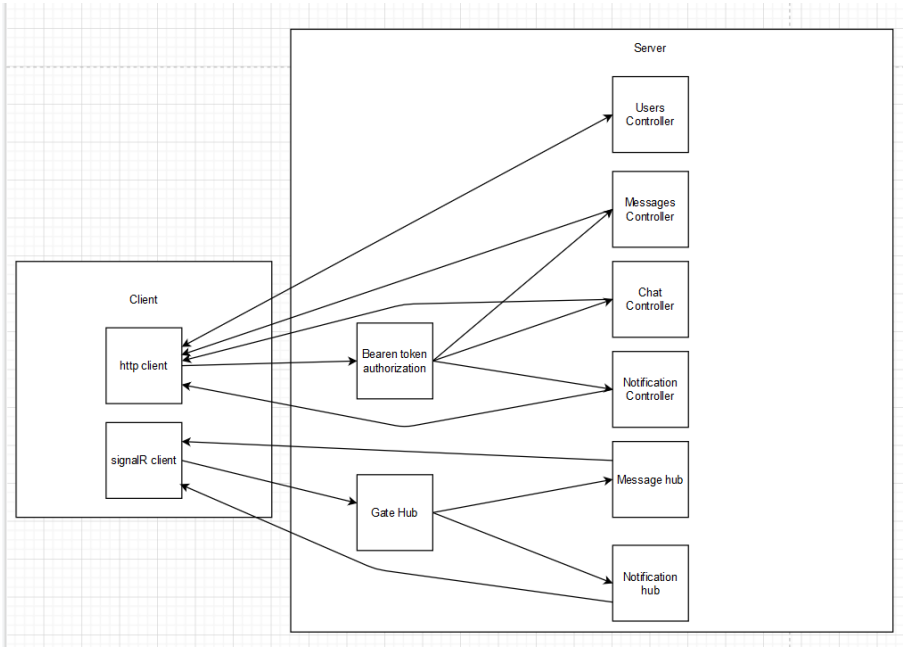
Struktura bazy danych jest prosta i czytelna. Dla takiego projektu baza danych jest fundamentem. Musimy mieć miejsce do przechowywania informacji o użytkownikach, relacjach, wiadmościach czy czatach. Tabele opieraja się na relacjach między sobą. Sama struktura w mysql została wygenerowana przez Entity Framework Core, który zrobił w najbardziej optymalny i wydajny sposób.



Rysunek 10:

3.5 Komunikacja z web serwisem

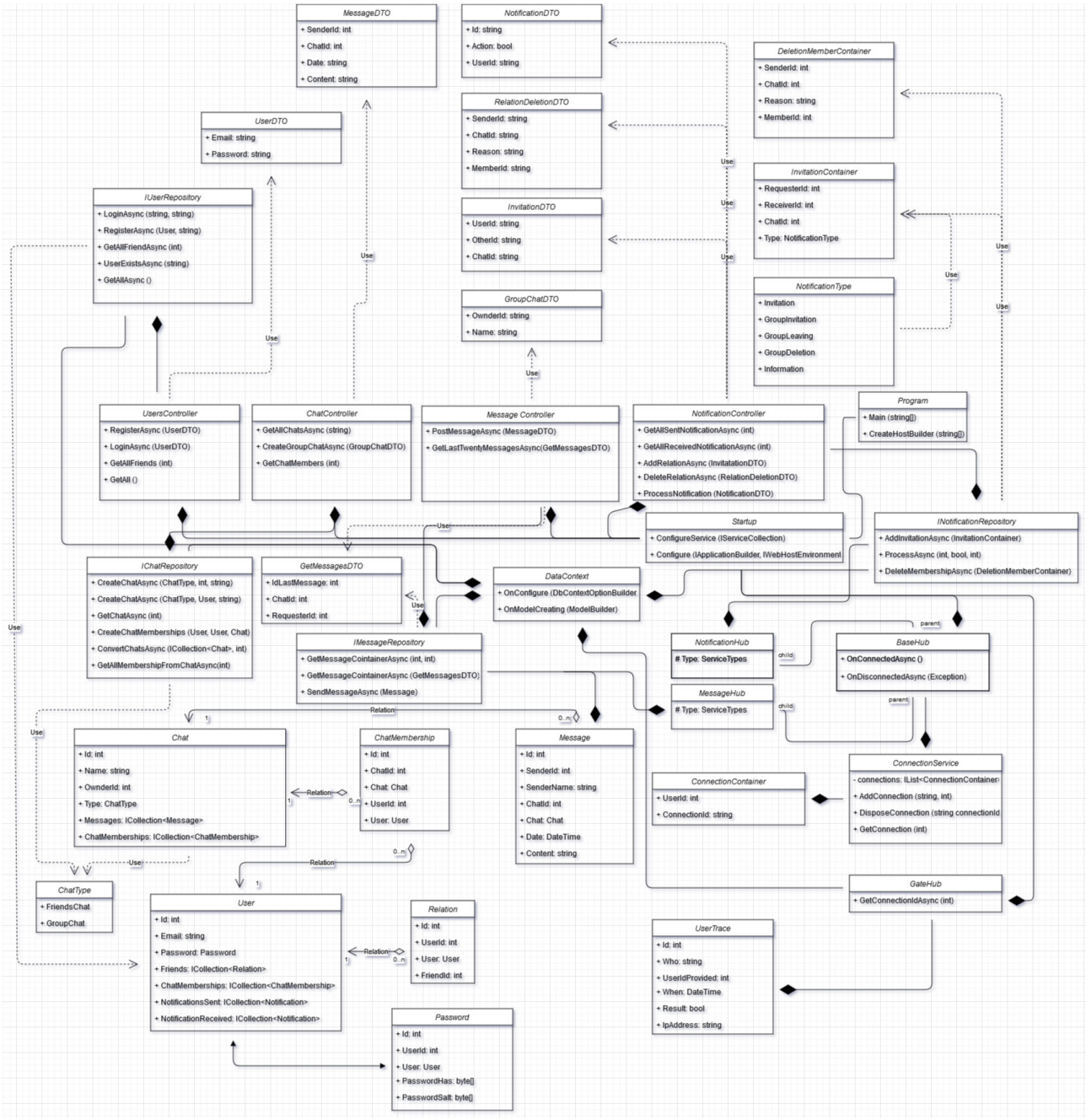
Komunikacja z web serwisem została podzielona na część dla każdego usera i część tylko dla zalogowanych. Po zalogowaniu wytwarzany jest token w User Controller i przesyłany z powrotem do klienta. Takowy token jest wykorzystywany do autoryzacji podczas dostępu do pozostałych kontrolerów. W przypadku SignalR, tylko zalogowani użytkownicy mogą się podłączyć. Wywoływana jest metoda RPC do walidacji użytkownika. Jest to druga część autoryzacji co czyni ją dwuetapową. Ta część polega na sprawdzeniu czy email zawarty w tokenie zgadza się z id usera. Wywoływana jest wówczas baza danych do pobrania emailu zawartego w rekodzie o danym id i przyrównanie tej wartości z wartością email zawartą w tokenie. Prosty diagram komunikacji z web serwisem jest następujący.



Rysunek 11:

3.6 Diagram klas

Diagram klas przedstawia od strony czysto technicznej połączenia między poszczególnymi klasami, interfejsami czy typami prostymi. Pozwala zauważyć pewne założenia, schematy, które ułatwią implementowanie całego mechanizmu, aby uniknąć powtarzania się kodu czy zminimalizować ilość użytych linijek.


MessageDTO
+ SenderId: int
+ ChatId: int
+ Date: string
+ Content: string
NotificationDTO
+ Id: string
+ Action: bool
+ UserId: string
DeletionMemberContainer
+ SenderId: int
+ ChatId: int
+ Reason: string
+ MemberId: int
RelationDeletionDTO
+ SenderId: string
+ ChatId: string
+ Reason: string
+ MemberId: string
UserDTO
+ Email: string
+ Password: string
InvitationContainer
+ RequesterId: int
+ ReceiverId: int
+ ChatId: int
+ Type: NotificationType
IUserRepository
+ LoginAsync (string, string)
+ RegisterAsync (User, string)
+ GetAllFriendAsync (int)
+ UserExistsAsync (string)
+ GetAllAsync ()
InvitationDTO
+ UserId: string
+ OtherId: string
+ ChatId: string
NotificationType
+ Invitation
+ GroupInvitation
+ GroupLeaving
+ GroupDeletion
+ Information
GroupChatDTO
+ OwnderId: string
+ Name: string
UsersController
+ RegisterAsync (UserDTO)
+ LoginAsync (UserDTO)
+ GetAllFriends (int)
+ GetAll ()
ChatController
+ GetAllChatsAsync (string)
+ CreateGroupChatAsync (GroupChatDTO)
+ GetChatMembers (int)
Message Controller
+ PostMessageAsync (MessageDTO)
+ GetLastTwentyMessagesAsync(GetMessagesDTO)
NotificationController
+ GetAllSentNotificationAsync (int)
+ GetAllReceivedNotificationAsync (int)
+ AddRelationAsync (InvitatationDTO)
+ DeleteRelationAsync (RelationDeletionDTO)
+ ProcessNotification (NotificationDTO)
Program
+ Main (string[])
+ CreateHostBuilder (string[])
Startup
+ ConfigureService (IServiceCollection)
+ Configure (IApplicationBuilder, IWebHostEnvironment
INotificationRepository
+ AddInvitationAsync (InvitationContainer)
+ ProcessAsync (int, bool, int)
+ DeleteMembershipAsync (DeletionMemberContainer)
IChatRepository
+ CreateChatAsync (ChatType, int, string)
+ CreateChatAsync (ChatType, User, string)
+ GetChatAsync (int)
+ CreateChatMemberships (User, User, Chat)
+ ConvertChatsAsync (ICollection<Chat>, int)
+ GetAllMembershipFromChatAsync(int)
GetMessagesDTO
+ IdLastMessage: int
+ ChatId: int
+ RequesterId: int
DataContext
+ OnConfigure (DbContextOptionBuilder
+ OnModelCreating (ModelBuilder)
IMessageRepository
+ GetMessageCointainerAsync (int, int)
+ GetMessageCointainerAsync (GetMessagesDTO)
+ SendMessageAsync (Message)
NotificationHub
Type: ServiceTypes
BaseHub
+ OnConnectedAsync ()
+ OnDisconnectedAsync (Exception)
MessageHub
Type: ServiceTypes
Chat
+ Id: int
+ Name: string
+ OwnderId: int
+ Type: ChatType
+ Messages: ICollection<Message>
+ ChatMemberships: ICollection<ChatMembership>
ChatMembership
+ Id: int
+ ChatId: int
+ Chat: Chat
+ UserId: int
+ User: User
Message
+ Id: int
+ SenderId: int
+ SenderName: string
+ ChatId: int
+ Chat: Chat
+ Date: DateTime
+ Content: string
ConnectionContainer
+ UserId: int
+ ConnectionId: string
ConnectionService
- connections: IList<ConnectionContainer>
+ AddConnection (string, int)
+ DisposeConnection (string connectionId
+ GetConnection (int)
ChatType
+ FriendsChat
+ GroupChat
User
+ Id: int
+ Email: string
+ Password: Password
+ Friends: ICollection<Relation>
+ ChatMemberships: ICollection<ChatMembership>
+ NotificationsSent: ICollection<Notification>
+ NotificationReceived: ICollection<Notification>
Relation
+ Id: int
+ UserId: int
+ User: User
+ FriendId: int
GateHub
+ GetConnectionIdAsync (int)
UserTrace
+ Id: int
+ Who: string
+ UserIdProvided: int
+ When: DateTime
+ Result: bool
+ IpAddress: string
Password
+ Id: int
+ UserId: int
+ User: User
+ PasswordHas: byte[]
+ PasswordSalt: byte[]
Use
Relation
child
parent
1
0..n

Rysunek 12: