



Politechnika  
Śląska

**POLITECHNIKA ŚLĄSKA**  
**WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI**  
**KIERUNEK: INFORMATYKA**

Praca dyplomowa inżynierska

System zarządzania siecią siłowni oparty o bazę dokumentową

autor: Seweryn Gładysz

kierujący pracą: dr inż. Ewa Płuciennik

Gliwice, styczeń 2022



# Spis treści

<b>Streszczenie</b>	<b>1</b>
<b>1 Wstęp</b>	<b>3</b>
<b>2 Analiza tematu</b>	<b>5</b>
<b>3 Wymagania i narzędzia</b>	<b>9</b>
3.1 Wymagania funkcjonalne . . . . .	9
3.2 Wymagania niefunkcjonalne . . . . .	12
3.3 Diagram przypadków użycia (UML) . . . . .	13
3.4 Narzędzia . . . . .	13
3.5 Wykorzystane zewnętrzne rozwiązania . . . . .	15
3.6 Metodyka pracy nad projektowaniem i implementacją . . . . .	16
<b>4 Specyfikacja zewnętrzna</b>	<b>19</b>
4.1 Wymagania sprzętowe . . . . .	19
4.2 Instalacja . . . . .	20
4.2.1 Instalacja bazy danych MongoDB . . . . .	20
4.2.2 Instalacja środowiska .NET 5 . . . . .	25
4.2.3 Instalacja środowiska Node.js . . . . .	27
4.2.4 Instalacja <i>http-server</i> . . . . .	31
4.3 Sposób aktywacji . . . . .	31
4.3.1 Uruchomienie warstwy serwerowej . . . . .	31
4.3.2 Uruchomienie warstwy prezentacji . . . . .	33
4.4 Kategorie użytkowników . . . . .	34

4.5	Kwestie bezpieczeństwa . . . . .	34
4.6	Przykład działania . . . . .	35
4.6.1	Rejestracja i logowanie . . . . .	35
4.6.2	Administracja systemem . . . . .	37
4.6.3	Wydarzenia . . . . .	44
4.6.4	Treningi indywidualne . . . . .	47
4.6.5	Karnety . . . . .	50
4.6.6	Tryb bramek . . . . .	51
<b>5</b>	<b>Specyfikacja wewnętrzna</b>	<b>53</b>
5.1	Architektura systemu . . . . .	53
5.2	Organizacja bazy danych . . . . .	56
5.3	Przegląd najważniejszych klas . . . . .	57
5.4	Przegląd wzorców projektowych . . . . .	61
<b>6</b>	<b>Weryfikacja i walidacja</b>	<b>65</b>
6.1	Sposoby testowania . . . . .	65
6.2	Organizacja testów . . . . .	67
6.3	Przykłady wykrytych i usuniętych błędów . . . . .	69
<b>7</b>	<b>Podsumowanie i wnioski</b>	<b>71</b>
	<b>Bibliografia</b>	<b>74</b>
	<b>Spis skrótów i symboli</b>	<b>77</b>
	<b>Źródła</b>	<b>79</b>
	<b>Spis załączników elektronicznych</b>	<b>81</b>

# Streszczenie

Czas pandemii koronawirusa stawia przed przedsiębiorcami nowe wyzwania, a cyfryzacja pozwala na automatyzację wielu czynności. Właściciele sieci siłowni są zmuszeni do ciągłego dostosowywania zasad działania siłowni do wymogów związanych z obostrzeniami sanitarnymi. Dzięki rozwojowi aplikacji internetowych istnieje możliwość zarządzania biznesem zarówno z poziomu komputera osobistego jak i smartfona.

W pracy przedstawiono projekt aplikacji internetowej, która pozwala na zarządzanie siecią obiektów siłowni. Szczególny nacisk położono na możliwość definiowania zasad działania poszczególnych obiektów w kontekście obowiązujących restrykcji sanitarnych. System korzysta z dokumentowej bazy danych w celu skorzystania z zalet technologii *NoSQL*. Warstwa prezentacji wykorzystuje możliwości współczesnych przeglądarek internetowych.

W ramach pracy omówiono ideę stojącą za stworzeniem systemu oraz sposób jego projektowania. Opisane zostały metodyki pracy nad projektem, wykorzystane technologie oraz wybrane wzorce projektowe, które zostały użyte przy pracy nad aplikacją.

**Słowa kluczowe:** aplikacja internetowa, dokumentowa baza danych, .NET



# Rozdział 1

## Wstęp

Głównym celem tego projektu jest stworzenie systemu informatycznego pozwalającego na prowadzenie sieci siłowni z wykorzystaniem dokumentowej bazy danych. Podstawowym założeniem projektu jest przygotowanie systemu w taki sposób, aby właściciele mogli dostosowywać zasady działania obiektów sportowych do aktualnie obowiązujących restrykcji sanitarnych. Prowadzenie obiektu siłowni jest utrudnione w czasie pandemii ze względu na częste zmiany w przepisach związanych z ograniczeniami pandemicznymi. Wymienione wcześniej problemy były głównym powodem powstania tego projektu.

Wykorzystanie dokumentowej bazy danych jest związane z zaletami baz *NoSQL*. Niektóre cechy wcześniej wymienionych baz nie są dostępne dla relacyjnych baz danych. Jedną z takich cech jest duża podatność na zmiany w strukturze względem relacyjnych baz oraz przystosowanie do skalowania wszcz[13].

Pracę podzielono na rozdziały, gdzie pierwszy został poświęcony analizie tematu. Analiza zawiera informacje o idei powstania systemu oraz wyzwaniach z którymi projekt będzie musiał się mierzyć. Kolejny rozdział został poświęcony wymaganiom i narzędziom. Wymagania w rozdziale 3 zostały podzielone na funkcjonalne i нефункционалне, а в kolejnych podrozdziałach opisano narzędzia, które wykorzystano do ich realizacji. Rozdział 4 opisuje wymagania sprzętowe, kategorie użytkowników oraz przykłady działania aplikacji. Poza tym zawiera wyczerpującą instrukcję instalacji i uruchamiania systemu. W rozdziale 5 skupiono się na opisie architektury i przyjętej organizacji bazy danych. W kolejnych podrozdziałach

przedstawiono przegląd najważniejszych klas i wzorców projektowych. Rozdział 6 poświęcono testom jakim został poddany system. W ramach rozdziału opisano sposoby i organizacje testów z podziałem na testy jednostkowe i manualne. Ostatni rozdział 7 przedstawia wnioski i obserwacje do jakich doszło w trakcie prac. Rozdział kończy się przedstawieniem propozycji usprawnienia systemu w przypadku kontynuowania pracy nad nim.



## Rozdział 2

### Analiza tematu

Na rynku siłowni widać coraz większą konkurencję, a właściciele szukają sposobów jak zachęcić nowych klientów do uczęszczania na ich siłownię. Wiele z sieci pozwala na całodobowe korzystanie z ich obiektów, aby sprostać wymaganiom jak największej grupy klientów. Stawia to nowe wyzwania w organizacji pracy i sposobie działania obiektów. Dużym usprawnieniem byłaby możliwość zrezygnowania z recepcji na rzecz bramek wejściowych i wyjściowych, które kontrolowałyby ważność karnetu oraz pilnowałyby, aby w tej samej chwili na siłowni nie znajdowała się zbyt duża liczba osób.

Innym problemem jest coraz większa liczba sieci siłowni, która w swojej ofercie zawiera możliwość udziału w wydarzeniach grupowych, które pozwalają na uczestnictwo w grupowej sesji, gdzie pracownik siłowni nadzoruje czy uczestnicy wykonują ćwiczenia w sposób bezpieczny dla ich zdrowia. Pozwolenie na zgłoszenie chęci udziału w takim wydarzeniu poprzez aplikację internetową może zwiększyć atrakcyjność oferty danej sieci siłowni w obliczu rosnącej konkurencji na rynku sieci siłowni.

Sieć siłowni zderza się z problemami, które nie są znane pojedynczym obiektom. Zarządzanie wyposażeniem może być utrudnione ze względu na liczbę posiadanych urządzeń i liczbę obiektów siłowni należących do sieci. System powinien gromadzić informacje na temat posiadanego sprzętu, aby skrócić czas, jaki jest potrzebny do przeprowadzenia inwentaryzacji posiadanych akcesoriów i sprzętów do ćwiczeń.

Innym wyzwaniem stawianym przed właścicielami siłowni jest stworzenie oferty

treningów personalnych dla klientów, którzy oczekują indywidualnego podejścia do treningu. Rozwiązaniem jest stworzenie modułu, który pozwalałby na rezerwowanie przez klientów terminów treningów. Takie rozwiązanie może zwiększyć atrakcyjność danej sieci oraz pozwolić na sprawniejsze organizowanie czasu trenerów personalnych należących do sieci siłowni.

Wyzwaniem stawianym przez pandemię koronawirusa jest reagowanie na częste zmiany w przepisach dotyczących wstępu na obiekty sportowe. Taka sytuacja zmusza właścicieli do ciągłego monitorowania sytuacji i reagowania na zmiany dotyczące przepisów sanitarnych. Każda taka zmiana zmusza siłownię do wdrożenia nowych restrykcji w postaci limitów osób na siłowni. Pomocą w realizowaniu narzuconych obostrzeń byłoby stworzenie systemu, który ustalałby jaka jest maksymalna liczba osób w obiekcie na podstawie wprowadzonych danych. System powinien również zapewniać możliwość kontrolowania liczby osób, które znajdują się w danej chwili w obiekcie siłowni.

Kolejnym problemem jest liczba użytkowników korzystających z system informatycznego. Sieć siłowni w przeciwieństwie do pojedynczego obiektu musi obsługiwać żądania dużej liczby klientów. Aby zapewnić ciągłość działania systemu ważne jest aby stworzone oprogramowanie było skalowalne. Skalowanie może zostać wykonywane *w górę* (poprzez zmodyfikowanie serwera w celu poprawy jego wydajności) lub *w szerz* (rozdzielenie obciążenia pomiędzy większą liczbą serwerów)[14]. W wielu przypadkach skalowanie w górę jest zbyt kosztowne lub niemożliwe ze względu na ograniczenia technologiczne. W takim przypadku z pomocą przychodzi skalowanie wszerz, które pozwala na rozdzielenie obciążenia pomiędzy różnymi maszynami.

Poza skalowalnością, baza danych powinna posiadać mechanizmy, które zapewnią szybkie przywrócenie działania w przypadku awarii. Klienci, którzy będą doświadczać częstych problemów nie będą chętnie korzystać z proponowanego systemu. W sieci siłowni liczba klientów jest zdecydowanie większa niż w przypadku pojedynczego obiektu, co może generować ewentualne problemy.

Rozwiązaniem problemu skalowalności i awaryjności może być wykorzystanie bazy *NoSQL*. Jedną z takich baz jest *MongoDB*, która pozwala na rozdzielenie obciążenia pomiędzy różnymi maszynami. Podział ten, w przypadku *MongoDB*, realizowany jest poprzez wykorzystanie mechanizmu partycjonowania danych (ang.

*sharding*). Idea tego rozwiązania polega na podzieleniu danych na podzbiory, które są następnie umieszczane na różnych serwerach. Podział ten może zostać skonfigurowany w taki sposób, aby dane częściej wykorzystywane znajdowały się na maszynach o większej wydajności. Inną zaletą partycjonowania danych jest możliwość stworzenia zbioru replik. W przypadku awarii jednej z maszyn, inna przejmuje jej obowiązki, aby zapewnić ciągłość w działaniu[14].

Kolejną zaletą wykorzystania dokumentowej bazy danych jest elastyczność schematu[13]. System, zwłaszcza w początkowej fazie rozwoju, będzie poddawany dużej liczbie zmian w schemacie bazy. *NoSQL*, a w szczególności baza *MongoDB* charakteryzuje się łatwością wprowadzenia zmian w tym aspekcie[1].

Aktualnie w sieci Internet możemy znaleźć wiele dostępnych rozwiązań [10] [4], które pokrywają wymagania funkcjonalne systemu. Duża część oferowanych aplikacji posiada możliwość zarządzania bazą klientów, wyposażeniem oraz opcję tworzenia bogatej oferty karnetów. Największe aplikacje posiadają w swoim zakresie funkcjonalnym rozwiązania wymienionych problemów, jednak żadne z nich nie posiada modułu odpowiedzialnego za zarządzanie dostępem do siłowni pod kątem obowiązujących przepisów sanitarnych. W celu zaproponowania systemu informatycznego, który będzie konkurencyjny na rynku, system powinien posiadać taki moduł.

Tworzony system powinien zapewnić wsparcie pracownikom w spełnianiu aktualnych norm i obostrzeń sanitarnych oraz dostarczyć podstawową funkcjonalność potrzebną w prowadzeniu działalności sieci siłowni. W celu uniknięcia problemów z wydajnością, aplikacja powinna cechować się skalowalnością w celu zapewnienia szybkości i ciągłości działania nawet w przypadku dużej liczby aktywnych użytkowników. Aby to osiągnąć system powinien zostać utworzony przy pomocy infrastruktury, która pozwoli na skalowanie wszerz. Warstwa serwerowa (ang. *backend*) aplikacji powinna zostać utworzona w technologii i przy pomocy metodyk, które pozwolą na łatwe dostosowanie aplikacji do architektury mikro-usług.



# Rozdział 3

## Wymagania i narzędzia

W tym rozdziale opisane zostaną wymagania funkcjonalne systemu, które muszą zostać spełnione. W dalszej części znajdują się wymagania niefunkcjonalne oraz diagramy przypadków użycia. Rozdział kończy się opisem wykorzystanych narzędzi, zewnętrznych rozwiązań oraz metodyk pracy nad projektem.

### 3.1 Wymagania funkcjonalne

Wymienione punkty stanowią listę wszystkich funkcji jakie powinna oferować aplikacja. Opisują one zakres funkcjonalności udostępnionej użytkownikowi poprzez warstwę prezentacji oraz interfejs REST API.

- Tworzenie konta trenera personalnego - w systemie powinna istnieć możliwość dodawania kont dla pracowników siłowni w celu zarządzania zasobami.
- Rejestracja konta klienta - system powinien pozwalać na samodzielne zarejestrowanie się użytkowników w celu skorzystania z funkcjonalności serwisu internetowego.
- Logowanie się na wcześniej utworzone konto użytkownika - w celu weryfikacji użytkownika, system powinien pozwalać na autoryzację poprzez formularz logowania.
- Przeglądanie dostępnej oferty karnetów na siłownię - użytkownik w celu zakupu karnetu musi mieć możliwość przeglądania dostępnej oferty.

- Przedłużenie karnetu przypisanego do konta użytkownika - użytkownik posiadający już karnet może go przedłużyć.
- Zakup karnetu - użytkownik powinien mieć możliwość zakupu karnetu w celu wejścia do obiektu.
- Utworzenie nowego typu karnetu - pracownicy siłowni powinni mieć możliwość stworzenia oferty karnetów.
- Usunięcie istniejącego rodzaju karnetu - pracownicy siłowni powinni mieć możliwość usunięcia z oferty wcześniej utworzonych karnetów.
- Edycja istniejącego typu karnetu - pracownicy siłowni powinni mieć możliwość dokonania zmian w istniejącej ofercie karnetów.
- Edycja informacji o koncie użytkownika - użytkownik serwisu powinien mieć możliwość edycji informacji zawartych w swoim profilu.
- Archiwizacja konta użytkownika - użytkownik serwisu powinien mieć możliwość zarchiwizowania swojego konta, gdy nie jest już zainteresowany dalszym korzystaniem z serwisu.
- Przeglądanie listy dostępnych wydarzeń - użytkownicy powinni mieć możliwość przeglądania listy dostępnych wydarzeń.
- Utworzenie nowego wydarzenia - trenerzy personalni powinni mieć możliwość dodawania wydarzeń, które będą miały miejsce na terenie siłowni.
- Odwołanie wydarzenia - trener personalny powinien mieć możliwość odwołania zaplanowanego wydarzenia.
- Edycja wydarzenia - trener personalny powinien mieć możliwość wprowadzenia zmian w zaplanowanym wydarzeniu.
- Możliwość zapisania się na udział w wydarzeniu - klient powinien mieć możliwość zadeklarowania chęci udziału w wydarzeniu.

- Rezygnacja z udziału w wydarzeniu - klient powinien mieć możliwość zrezygnowania z udziału w wydarzeniu, w którym wcześniej zadeklarował chęć udziału.
- Utworzenie nowego obiektu siłowni - pracownicy powinni mieć możliwość utworzenia nowych obiektów sieci siłowni.
- Możliwość zdefiniowania ograniczeń dostępu do siłowni - pracownicy powinni mieć możliwość zdefiniowania limitu osób jaki może się znajdować w obiekcie siłowni.
- Możliwość dodania nowych pomieszczeń dla obiektu siłowni - pracownicy powinni mieć możliwość dodania nowych pomieszczeń do obiektu siłowni.
- Możliwość usunięcia pomieszczeń z obiektu siłowni - pracownicy powinni mieć możliwość usunięcia pomieszczenia, które zostało utworzone w danym obiekcie siłowni.
- Możliwość zdefiniowania akcesoriów i sprzętu do ćwiczeń - pracownicy powinni mieć możliwość zdefiniowania akcesoriów dostępnych w danym obiekcie siłowni.
- Edycja zdefiniowanych akcesoriów i sprzętu do ćwiczeń - pracownicy powinni mieć możliwość edycji wcześniej zdefiniowanych akcesoriów i sprzętu do ćwiczeń.
- Możliwość uruchomienia aplikacji w trybie bramki wejściowej - aplikacja powinna pozwolić użytkownikowi na uruchomienie aplikacji w trybie bramki wejściowej w celu kontroli liczby klientów znajdujących się w siłowni.
- Możliwość uruchomienia aplikacji w trybie bramki wyjściowej - aplikacja powinna pozwolić użytkownikowi na uruchomienie aplikacji w trybie bramki wyjściowej w celu kontroli liczby klientów wychodzących z siłowni.
- Przeglądanie listy dostępnych treningów indywidualnych - użytkownicy powinni mieć możliwość przeglądania listy dostępnych treningów osobisty.

- Możliwość zapisania się na udział w treningu indywidualnym - klient powinien mieć możliwość zapisania się na wcześniej utworzony trening indywidualny z trenerem personalnym.
- Utworzenie treningu indywidualnego - trener personalny powinien mieć możliwość utworzenia sesji treningów indywidualnych.
- Odwołanie treningu indywidualnego - trener personalny powinien mieć możliwość odwołania wcześniej zaplanowanych treningów personalnych.
- Rezygnacja z udziału w treningu personalnym - klient powinien mieć możliwość zrezygnowania z udziału w treningu personalnym.
- Odmowa wejścia na siłownię - bramka wejściowa powinna mieć możliwość odmówienia wejścia klientowi na siłownię w przypadku braku ważnego kartetu lub osiągnięciu maksymalnej liczby klientów na terenie obiektu siłowni.
- Sprawdzenie czy jest możliwość wejścia na siłownię - bramka wejściowa powinna informować o limicie osób, jaki może znajdować się w obiekcie siłowni.
- Otwarcie bramki wejściowej - bramka wejściowa powinna pozwolić na wejście na teren siłowni po spełnieniu warunków.
- Otwarcie bramki wyjściowej - bramka wyjściowa powinna pozwolić na wyjście z terenu siłowni.

## 3.2 Wymagania нефunkcjonalne

- Wyświetlanie elementów interfejsu użytkownika powinno zostać dostosowane do standardowych rozdzielczości ekranów komputerów osobistych.
- Poprawny sposób działania w nowoczesnych przeglądarkach: Mozilla Firefox, Google Chrome, Microsoft Edge.
- System powinien pozwolić na obsłużenie równoczesnego dostępu do aplikacji dla co najmniej trzydziestu użytkowników równocześnie.



- Aplikacja powinna być dostępna dla użytkowników przez siedem dni w tygodniu w godzinach 3-22.

### 3.3 Diagram przypadków użycia (UML)

Diagram przypadków użycia pozwala na zaprezentowanie interakcji użytkowników z aplikacją i przedstawienie wymogów funkcjonalnych systemu w formie diagramów. Wymagania funkcjonalne z podrozdziału 3.1 przeniesione zostały na schemat, a następnie przydzielone do wcześniej zdefiniowanych aktorów. Rezultaty pracy nad schematem można zobaczyć na rysunku 3.1.

W systemie wydzielono aktorów: *trener personalny* i *klient sieci siłowni*, którzy są szczególnymi przypadkami aktora *zarejestrowany użytkownik*. Poza tym wydzielono również aktorów takich jak *bramka wejściowa* i *bramka wyjściowa*, którzy będą odpowiedzialni za kontrolowanie liczby osób na siłowni. Ostatnim aktorem obecnym w systemie jest *użytkownik niezarejestrowany*, którego rola sprowadza się do możliwości zarejestrowania się w systemie.

### 3.4 Narzędzia

Do napisania projektu wykorzystano IDE (ang. *Integrated development environment*) *Visual Studio 2019*, które oferuje wiele narzędzi ułatwiających pisanie kodu w języku *C#*. Głównymi zaletami tego środowiska jest bliska współpraca z środowiskiem *.NET*, zapewnienie narzędzi w postaci odpluskwiacza (ang. *debugger*) czy integracja z systemem kontroli wersji *GIT*. Do pracy wykorzystano również IDE *WebStorm* firmy *JetBrains*, które pozwala na tworzenie aplikacji z platformą programistyczną *Angular*. Do przeglądania kolekcji i dokumentów w bazie danych *MongoDB* wykorzystano narzędzie *MongoDB Compass*. Diagramy zostały utworzone dzięki wykorzystaniu narzędzi *Astah UML* i *diagrams.net*.



## 3.5 Wykorzystane zewnętrzne rozwiązania

Do projektu zostały wykorzystane nowoczesne platformy programistyczne i biblioteki, które posiadają aktualne wsparcie twórców. Wybrano narzędzia w taki sposób, aby pokryć wymagania funkcjonalne projektu oraz wykorzystać technologie, które są aktualnie wykorzystywane w sposób komercyjny.

Językiem programowania użytym do stworzenia części serwerowej został *C#* firmy *Microsoft*. Bliskie podobieństwo do języka *Java* oraz wiele usprawnień względem niego pozwala na tworzenie oprogramowania przy wykorzystaniu technik programowania obiektowego, funkcyjnego czy generycznego [12].

Platformą programistyczną warstwy serwerowej została technologia *.NET 5*, która pozwala na tworzenie aplikacji wieloplatformowych z pomocą języka *C#*. Dzięki użyciu następcy *.NET Core'a*, aplikacja może zostać uruchomiona zarówno na urządzeniach z systemem *Microsoft* jak i dystrybucjach systemu Linux co pozwala na elastyczność względem wyboru środowiska, w którym aplikacja będzie pracować [17].

Do stworzenia interfejsu *REST API* została wykorzystana biblioteka *ASP.NET Core*, która pozwala na dodawanie metod interfejsu *REST API*, które będą następnie konsumowane przez warstwę prezentacji. *ASP.NET Core* wspiera wykorzystanie kontenerów wstrzykiwania zależności, obsługę autoryzacji przy pomocy żetonu *JWT* jak i komunikację z wykorzystaniem protokołu *HTTPS*[16].

W celu wykorzystania wzorca projektowego *CQRS* (ang. *command query responsibility separation*) zamiast domyślnego kontenera zależności wykorzystano zewnętrzną bibliotekę *Autofac*, która udostępnia możliwość wstrzykiwania obiektów oznaczonych atrybutami oraz określanie cyklu życia serwisu przy pomocy znaczników[8].

Wykorzystanie paradygmatu programowania obiektowego prowadziło do potrzeby mapowania niektórych typów na inne. W celu sprawniejszego przeprowadzania takich mapowań wykorzystano bibliotekę *AutoMapper*, która na podstawie podanego obiektu tworzy inny według schematu określonego w klasie profilu (ang. *Profile*).

Jako bazę danych wykorzystano *MongoDB*. *MongoDB* jest dokumentową bazą danych pozwalającą na przechowywanie danych w formacie *JSON*, który jest for-

matem czytelnym dla człowieka. Baza danych należy do nierelacyjnych baz danych *NoSQL*, której głównymi zaletami są: elastyczny schemat bazy, łatwe skalowanie wszerek oraz możliwość tworzenia zaawansowanych zapytań i raportów[14]. Opcja skalowania bazy wszerek jest istotna w systemach, gdzie trzeba zachować ciągłość działania systemu przy rosnącej bazie użytkowników.

Językiem programowania użytym w części wizualnej aplikacji jest język *TypeScript* zaprojektowany przez firmę *Microsoft*. W odróżnieniu od języka *JavaScript*, *TypeScript* oferuje typowanie w czasie kompilacji, co pozwala na unikanie błędów związanych z niezgodnością typów. Dodanie interfejsów i uogólnień funkcji względem *JavaScriptu* pozwala na pisanie kodu z wykorzystaniem paradygmatów programowania obiektowego i generycznego[18].

Do stworzenia interfejsu użytkownika została wykorzystana platforma programistyczna *Angular*, która pozwala na tworzenie aplikacji internetowych przy wykorzystaniu komponentów wielokrotnego użytku. Cechą charakterystyczną platformy jest natywne wykorzystanie języka *TypeScript* oraz możliwość tworzenia aplikacji typu *SPA*[2][18].

W celu zminimalizowania czasu potrzebnego na tworzenie komponentów warstwy prezentacji wykorzystano bibliotekę *Angular Material* z gotowym zestawem rozwiązań. Kontrolki zawarte w bibliotece implementują filozofię *Google Material* do tworzenia warstw wizualizacji aplikacji internetowych.

W warstwie prezentacji wykorzystano bibliotekę *NgXS*, która pozwala na wykorzystanie wzorca zarządzania stanem. Rozwiązanie implementuje wzorzec projektowy *Redux*, którego główną zaletą jest ułatwienie odpluskwiania warstwy prezentacji oraz zarządzanie danymi poprzez tworzenie stanów (ang. *states*).

## 3.6 Metodyka pracy nad projektowaniem i implementacją

Prace rozpoczęto od utworzenia osobnych repozytoriów kodu z uwzględnieniem reprezentowanej warstwy aplikacji. Te warstwy to:

- *Samson.Web.Application* - warstwa serwerowa aplikacji. Zawiera kod, który będzie uruchamiany po stronie serwera.

- *Samson.Web.Ui* - warstwa prezentacji (ang. *frontend*) aplikacji. Zawiera kod, który będzie uruchamiany na maszynie klienta w celu prezentacji interfejsu użytkownika.

Następnie podzielono kod warstwy serwerowej na projekty platformy programistycznej .NET. Każdy z projektów został utworzony w taki sposób, aby mógł być zastąpiony w przypadku zmian szczegółów implementacji aplikacji. Podział wygląda następująco:

- *Samson.Web.Application.Api* - projekt zawierający kod kontrolerów platformy ASP.NET, modele wykorzystywane do zapytań HTTP oraz klasy obiektów reprezentujące odpowiedzi serwera na żądania HTTP.
- *Samson.Web.Application.Identity* - projekt, który zawiera konfigurację oraz serwisy odpowiedzialne za autoryzację użytkownika.
- *Samson.Web.Application.Infrastructure* - projekt posiadający w swojej strukturze utworzone atrybuty, rozszerzenia klas, kod oprogramowania pośredniczącego (ang. *middleware*) oraz interfejsy wykorzystywane pomiędzy, różnymi projektami rozwiązania .NET.
- *Samson.Web.Application.Persistence* - projekt zawierający kod klas implementujących wzorzec repozytorium. Odpowiada za dodawania i modyfikację encji w bazie danych.
- *Samson.Web.Application.ReadModels* - projekt platformy .NET implementujący odczyt z bazy danych.
- *Samson.Web.Application.UnitTests* - projekt biblioteki NUnit zawierający kod testów jednostkowych.
- *Samson.Web.Application.WebHost* - główny projekt aplikacji. Zawiera konfigurację warstwy serwerowej aplikacji
- *Samson.Web.Models* - projekt zawierający modele, wyliczenia (ang. *enum*) oraz modele domenowe aplikacji

W następnych krokach implementowano aplikację tworząc w pierwszej kolejności kontrolery platformy *ASP.NET*, a w krokach późniejszych dodawano szczegóły implementacji w postaci domeny, warstwy zapisu, warstwy odczytu oraz serwisów aplikacyjnych. Najważniejsze klasy domenowe zostały uzupełnione testami jednostkowymi w celu zapewnienia poprawności działania. Gdy tworzenie części serwerowa aplikacji zostało zakończone, rozpoczęto prace nad warstwą interfejsu użytkownika. Warstwa prezentacji była implementowana w następującej kolejności:

1. Opakowanie komponentów z bibliotek zewnętrznych
2. Utworzenie widoków i formularzy
3. Integracja z warstwą serwerową
4. Dodanie obsługi żetonu JWT w komunikacji z warstwą serwerową

# Rozdział 4

## Specyfikacja zewnętrzna

W tym rozdziale znajduje się spis wymagań sprzętowych, których spełnienie jest wymagane do poprawnego działania systemu. W dalszej części znajduje się instrukcja instalacji systemu oraz opis sposobu jego aktywacji. Kolejne podrozdziały opisują kategorie użytkowników i metody, które zostały wykorzystane do zabezpieczenia wrażliwych danych. Rozdział kończy się przykładami działania aplikacji w postaci zrzutów ekranu i opisów.

### 4.1 Wymagania sprzętowe

Aby aplikacja działała w poprawny sposób spełnione muszą być poniższe wymagania:

- System operacyjny w wersji [7][6][5][11][9]:
  - *Windows* w wersjach 11/10/8.1
  - *Windows 7 z Service Pack 1*
  - *Windows Server Core 2012 R2*
  - Nano Server w wersji nowszej niż 1809
  - *MacOS 12.0 Monterey*
  - *MacOS 11.0 Big Sur*
  - *MacOS 10.15 Catalina*

- *Linux Alpine* w wersji nowszej niż 3.12 włącznie
  - *Linux CentOS* w wersjach 8/7
  - *Linux Debian* w wersjach 11/10/9
  - *Linux Fedora* w wersjach 35/34/33/32
  - *Linux OpenSuse* w wersji 15
  - *Linux RedHat* w wersjach 8/7
  - *Linux SLES* w wersji 15
  - *Linux Ubuntu* w wersjach 21.10/21.04/20.04
- Procesor z więcej niż jednym rdzeniem o taktowaniu nie mniejszym niż 1 GHz
  - Pamięć operacyjna większa niż 4 GB RAM
  - Minimalna przestrzeń na dysku to 4,5 GB pamięci
  - Zainstalowana nowoczesna przeglądarka WWW taka jak:
    - *Google Chrome* w najnowszej wersji
    - *Mozilla Firefox* w najnowszej wersji
    - *Microsoft Edge* w najnowszej wersji
    - *Opera* w najnowszej wersji

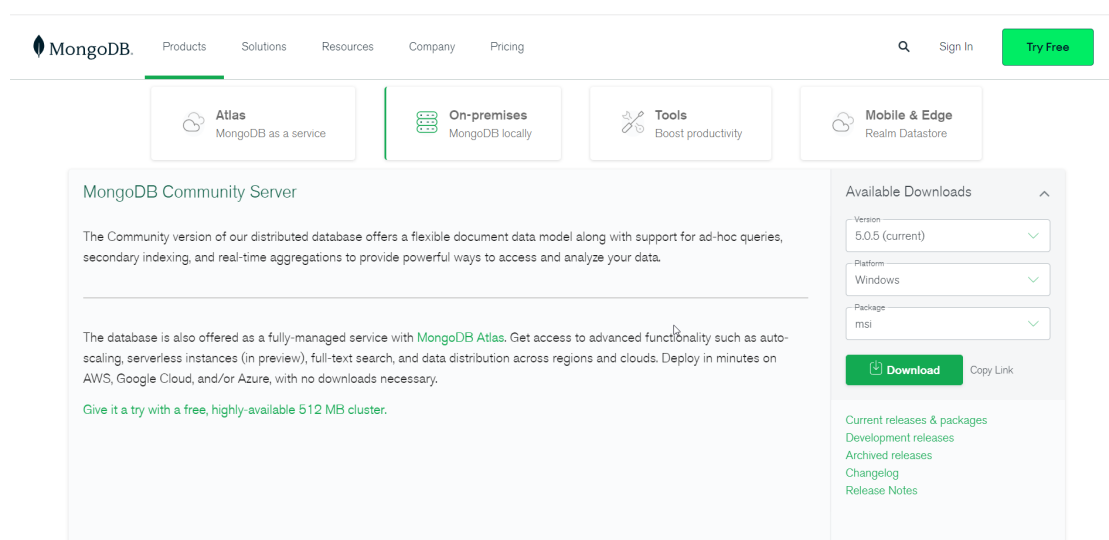
Są to minimalne wymagania do uruchomienia bazy danych *MongoDB* oraz aplikacji z wykorzystaniem platformy uruchomieniowej *.NET 5*. Ponadto wymagane są dodatkowe zasoby potrzebne do uruchomienia przeglądarki, w której wyświetlony zostanie interfejs użytkownika.

## 4.2 Instalacja

### 4.2.1 Instalacja bazy danych MongoDB

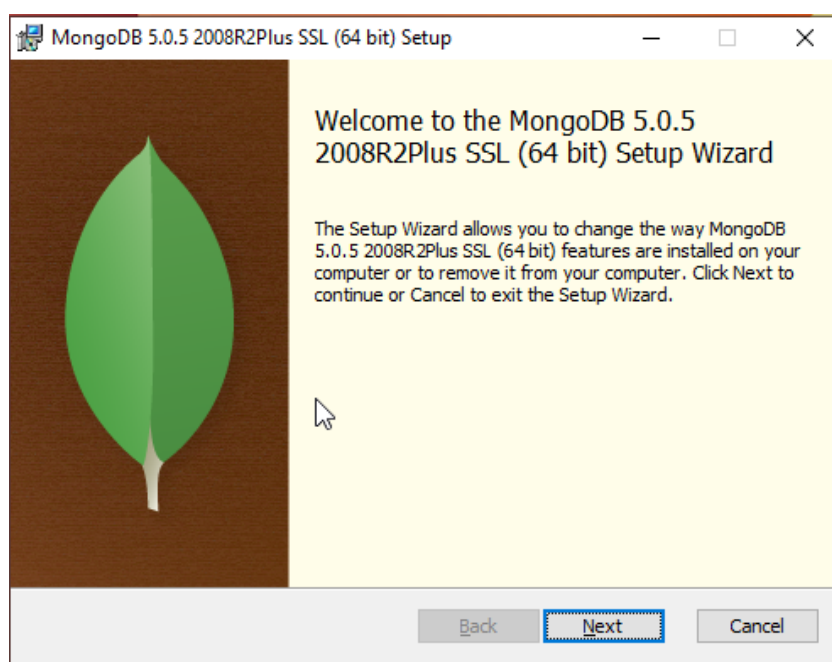
W celu zainstalowania aplikacji, trzeba spełnić wymagania wymienione w poprzednim punkcie. Instalacja zostaje rozpoczęta od pobrania i zainstalowania serwera bazy danych *MongoDB* w wersji Community z strony twórców.





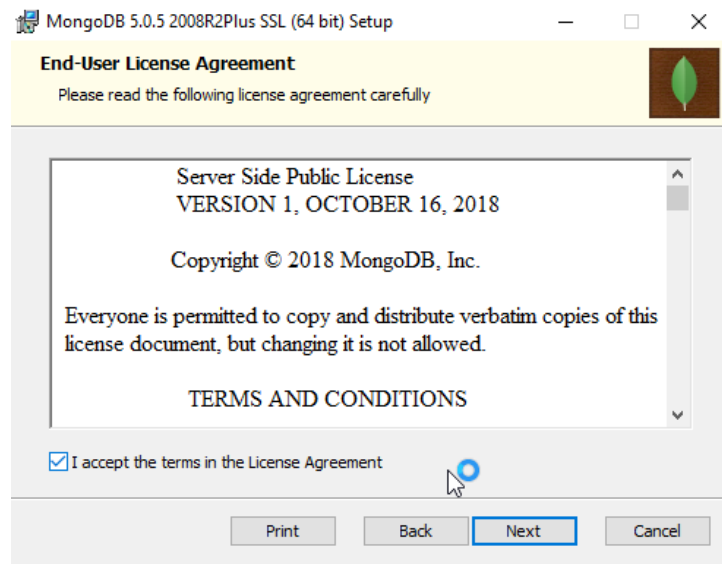
Rysunek 4.1. Strona, z której można pobrać instalator serwera bazy danych MongoDB

Instalacja oprogramowania do zarządzania bazą danych rozpoczyna się od uruchomienia wcześniej pobranego kreatora.



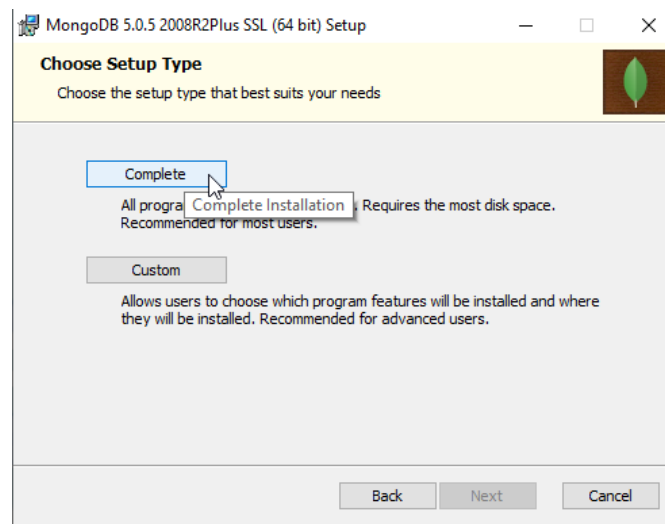
Rysunek 4.2. Widok powitalny instalatora MongoDB

Po zapoznaniu się z warunkami licencyjnymi, aby kontynuować użytkownik musi zaznaczyć pole z podpisem *I accept the terms in the License Agreement*, a następnie przycisnąć *Next*.



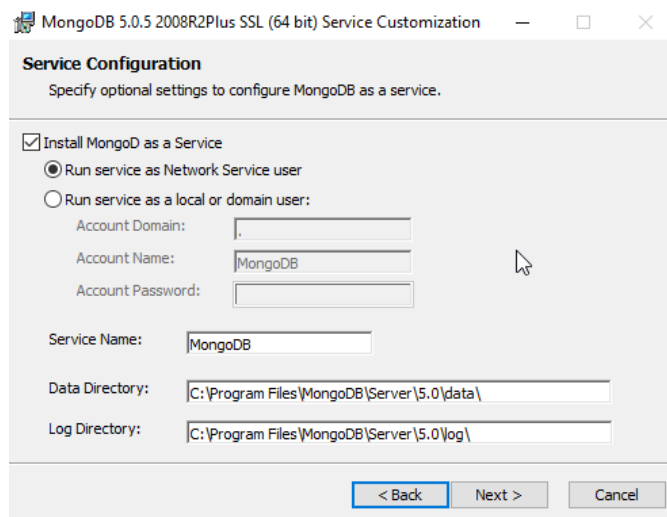
Rysunek 4.3. Umowa licencyjna *MongoDB Community Edition*

W widoku formularza wyboru typu instalacji trzeba wybrać opcję *Completed*, aby zainstalować wszystkie potrzebne narzędzia do działania serwera.



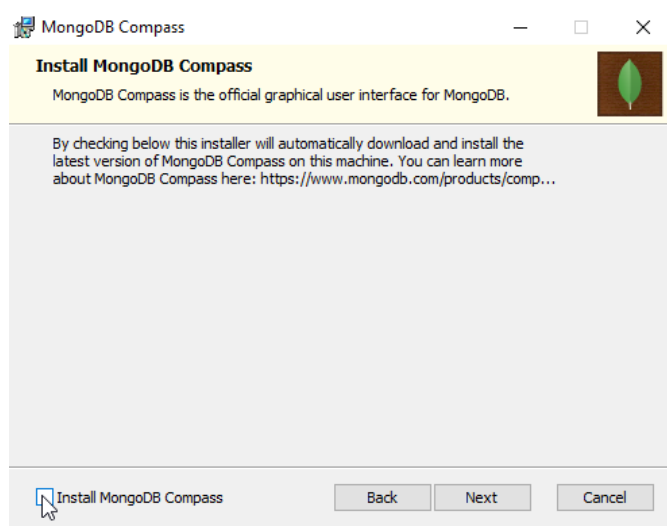
Rysunek 4.4. Typ instalacji w instalatorze *MongoDB Community Edition*

W następnym widoku, gdzie widać konfigurację serwera, ustawienia powinny zostać domyślne.



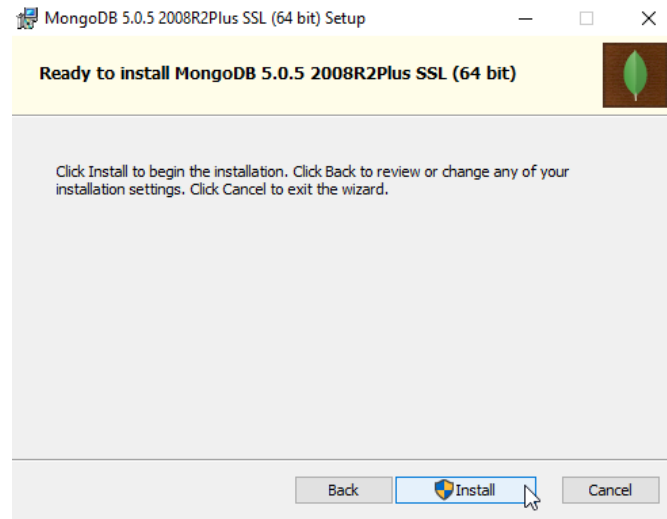
Rysunek 4.5. Początkowa konfiguracja serwera bazy danych

Po naciśnięciu przycisku *Next* powinna być widoczna opcja zainstalowania *MongoDB Compass*. Oprogramowanie *MongoDB Compass* jest opcjonalne, więc opcja instalacji może zostać odznaczona.



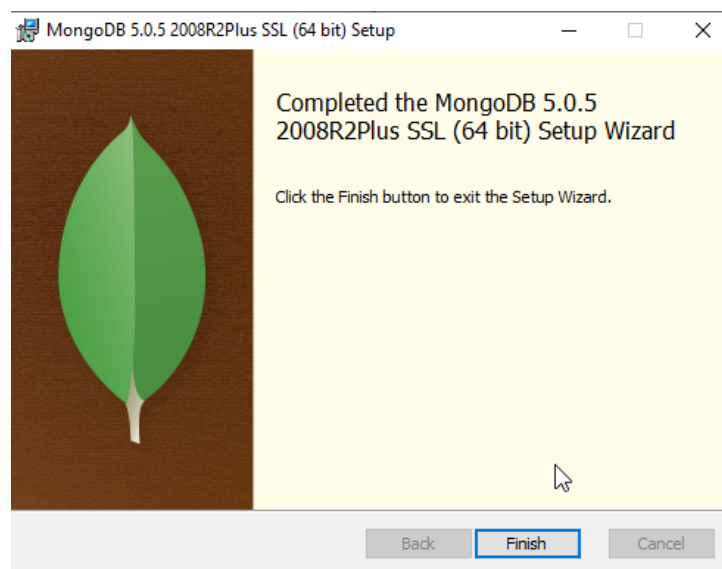
Rysunek 4.6. Informacja o możliwości instalacji klienta bazy danych *MongoDB Compass*

W następnym kroku, aby zainstalować serwer bazy danych trzeba nacisnąć przycisk *Install*.



Rysunek 4.7. Rozpoczęcie instalacji *MongoDB*

Aby zakończyć działanie kreatora po zakończeniu się procesu instalacji trzeba przycisnąć przycisk *Finish*. W tym momencie na maszynie użytkownika powinna znajdować się zainstalowana baza danych *MongoDB*.



Rysunek 4.8. Zakończenie instalacji *MongoDB*

## 4.2.2 Instalacja środowiska .NET 5

Podobnie jak w przypadku *MongoDB*, aby zainstalować środowisko *.NET 5* trzeba przejść na stronę, gdzie możliwe jest pobranie *SDK .NET 5*. W pierwszej kolejności wymagane jest wybranie odpowiedniej wersji instalatora dla zainstalowanego systemu operacyjnego oraz architektury maszyny.

The screenshot shows the Microsoft .NET 5.0.404 download page. It includes a navigation bar with links like 'Why .NET', 'Features', 'Learn', 'Docs', 'Downloads', and 'Community'. The main content area is divided into sections for 'Build apps - SDK 5.0.404' and 'Run apps - Runtime'. The SDK section provides a table of download links for Linux, macOS, and Windows, categorized by architecture (x64, x86, ARM64). The Runtime section provides information about ASP.NET Core Runtime and .NET Desktop Runtime, including their versions and download links.

OS	Installers	Binaries
Linux	<a href="#">Package manager instructions</a>	<a href="#">Arm32</a>   <a href="#">Arm32 Alpine</a>   <a href="#">Arm64</a>   <a href="#">Arm64 Alpine</a>   <a href="#">x64</a>   <a href="#">x64 Alpine</a>
macOS	<a href="#">x64</a>	<a href="#">x64</a>
Windows	<a href="#">Arm64</a>   <a href="#">x64</a>   <a href="#">x86</a>	<a href="#">Arm64</a>   <a href="#">x64</a>   <a href="#">x86</a>
All	<a href="#">dotnet-install scripts</a>	

**Visual Studio support**  
Visual Studio 2019 (v16.11)  
Visual Studio 2019 for Mac (v8.10)

**Included in**  
Visual Studio 16.11.8

**Included runtimes**  
NET Runtime 5.0.13  
ASP.NET Core Runtime 5.0.13  
NET Desktop Runtime 5.0.13

**Language support**  
C# 9.0  
F# 5.0

<https://dotnet.microsoft.com/en-us/download/dotnet/thank-you/sdk-5.0.404-windows-x64-installer>

**Run apps - Runtime**  
**ASP.NET Core Runtime 5.0.13**  
The ASP.NET Core Runtime enables you to run existing web/server applications. **On Windows, we recommend installing the Hosting Bundle, which includes the .NET Runtime and IIS support.**

**IIS runtime support (ASP.NET Core Module v2)**  
15.0.21326.13

OS	Installers	Binaries
Linux	<a href="#">Package manager instructions</a>	<a href="#">Arm32</a>   <a href="#">Arm32 Alpine</a>   <a href="#">Arm64</a>   <a href="#">Arm64 Alpine</a>   <a href="#">x64</a>   <a href="#">x64 Alpine</a>
macOS		<a href="#">x64</a>
Windows	<a href="#">Hosting Bundle</a>   <a href="#">x64</a>   <a href="#">x86</a>	<a href="#">Arm64</a>   <a href="#">x64</a>   <a href="#">x86</a>

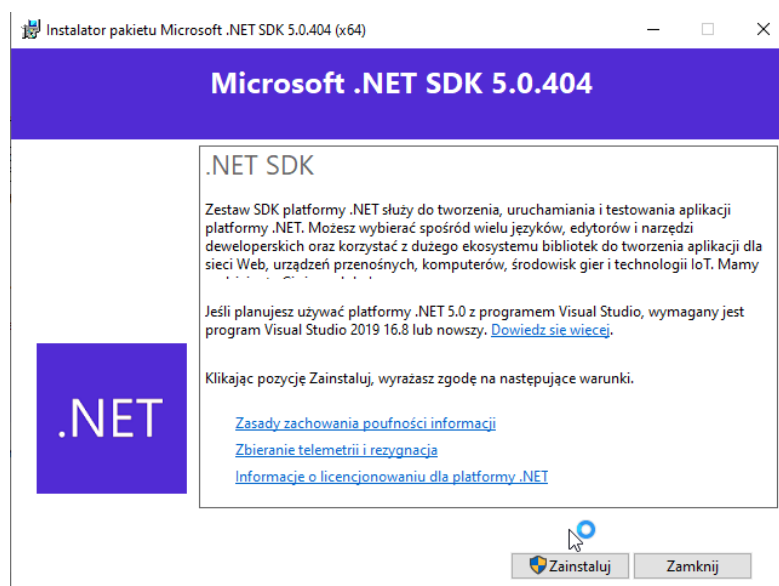
**.NET Desktop Runtime 5.0.13**  
The .NET Desktop Runtime enables you to run existing Windows desktop applications. **This release includes the .NET Runtime; you don't need to install it separately.**

OS	Installers	Binaries
Windows	<a href="#">Arm64</a>   <a href="#">x64</a>   <a href="#">x86</a>	

**.NET Runtime 5.0.13**

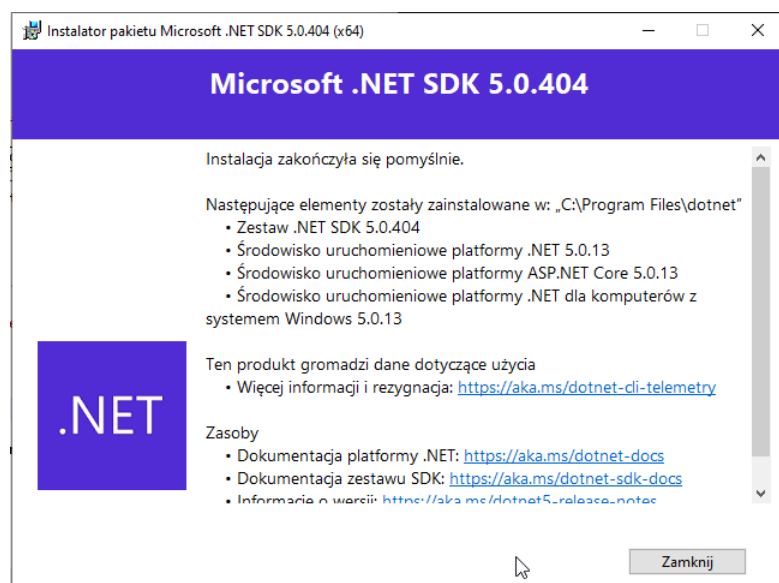
Rysunek 4.9. Strona twórców środowiska *.NET*

W kolejnym kroku trzeba uruchomić pobrany kreator instalacji *SDK .NET 5*. Pierwszym widokiem jest formularz informujący o dostępnej dokumentacji oraz składnikach instalacji. Po zapoznaniu się z informacjami trzeba przycisnąć przycisk *Zainstaluj* w celu instalacji środowiska uruchomieniowego.



Rysunek 4.10. Rozpoczęcie instalacji środowiska .NET 5

Następnie kreator zainstaluje na maszynie wymagane składniki. Kiedy działanie instalatora dobiegnie końca, użytkownik powinien zamknąć kreator przy pomocy przycisku *Zamknij*.



Rysunek 4.11. Zakończenie instalacji środowiska .NET 5

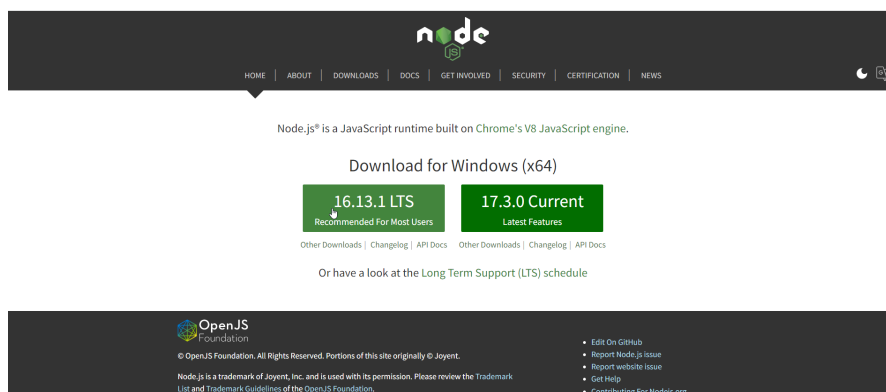
Aby sprawdzić czy instalacja przebiegła pomyślnie trzeba uruchomić konsolę

*Windows Powershell* dla systemów rodziny *Windows* lub odpowiedniego terminala dla posiadanej dystrybucji *Linux*. Następnie trzeba wpisać komendę *dotnet -version*. Jeśli instalacja przebiegła pomyślnie użytkownik powinien zobaczyć w konsoli informację o zainstalowanej wersji środowiska *.NET*.

### 4.2.3 Instalacja środowiska Node.js

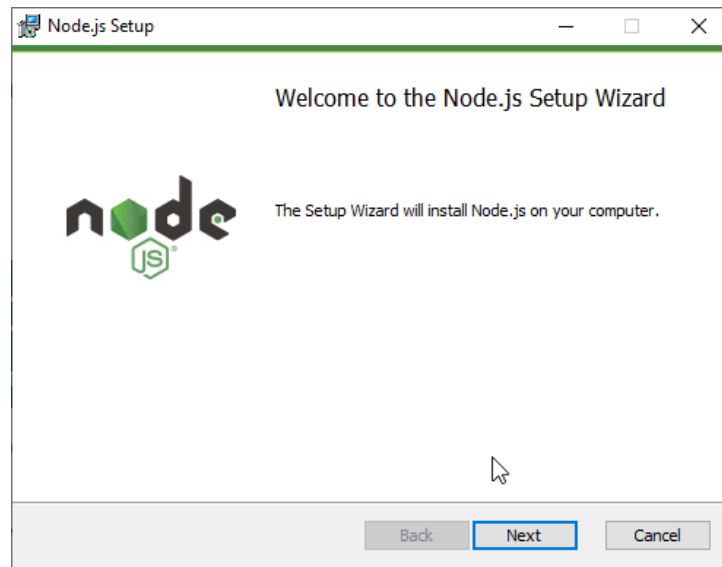
Przedostatnim krokiem, jaki trzeba wykonać, aby zainstalować projekt jest instalacja środowiska, które pozwoli na uruchomienie serwera WWW dzięki, któremu będzie można uruchomić aplikację stworzoną przy pomocy platformy programistycznej Angular. Serwer WWW musi mieć możliwość przekierowania wszystkich zapytań do wybranego adresu URL. Jest to istotne w przypadku aplikacji typu *SPA*, gdzie to aplikacja odpowiada za nawigację pomiędzy widokami. W tej pracy zaprezentowana zostanie instalacja serwera *http-server*, który obsługuje takie przekierowania.

W pierwszej kolejności zainstalowana musi być platforma uruchomieniowa *Node.js*, która pozwala na wykonywanie skryptów napisanych w języku *JavaScript*. Kreator instalacji można pobrać z strony twórców środowiska.

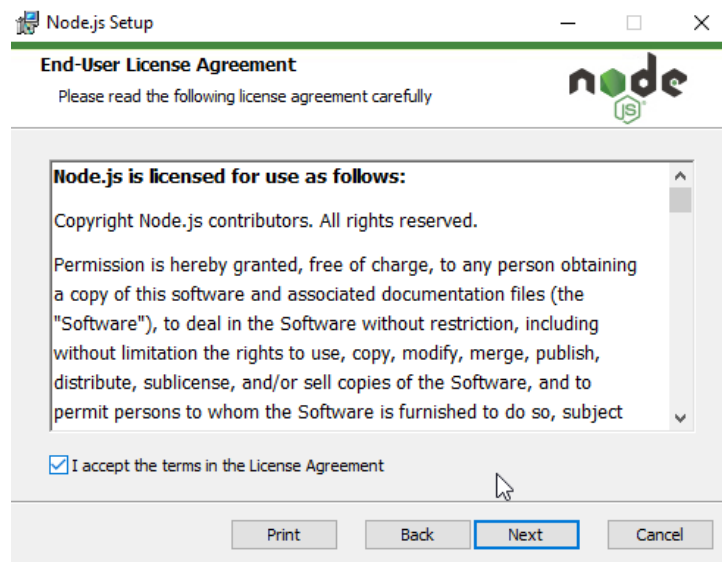


Rysunek 4.12. Strona Node.js

Kolejnym krokiem jest uruchomienie programu instalującego środowisko uruchomieniowe. Aby to zrobić trzeba przejść do lokalizacji na dysku gdzie pobrano plik instalatora a następnie uruchomić go. Wyświetlony powinien zostać następujący widok.

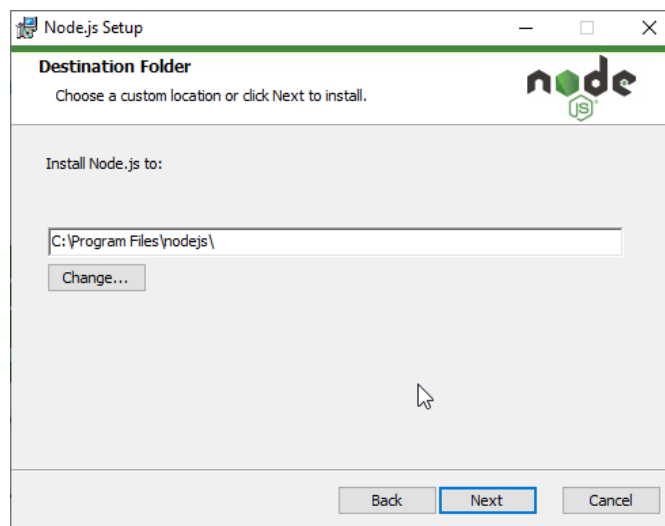
Rysunek 4.13. Rozpoczęcie instalacji środowiska *Node.js*

Po przyciśnięciu przycisku *Next* oczom użytkownika powinien ukazać się formularz z informacjami o licencji oprogramowania. Aby przejść dalej użytkownik powinien zapoznać się z licencją a następnie zaakceptować ją. Po zaakceptowaniu licencji użytkownik musi przejść do kolejnego widoku za pomocą przycisku *Next*.

Rysunek 4.14. Licencja środowiska *Node.js*

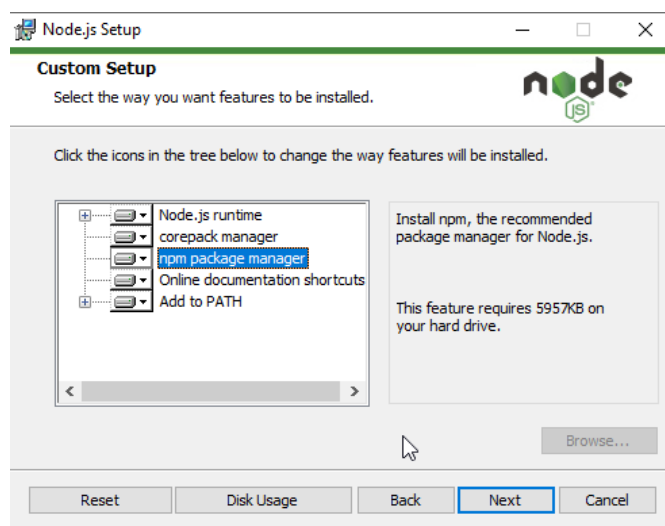


Następnym etapem instalacji jest wybór ścieżki do folderu, gdzie ma zostać zainstalowane oprogramowanie. Po wyborze dogodnej lokalizacji trzeba przejść do kolejnego widoku.



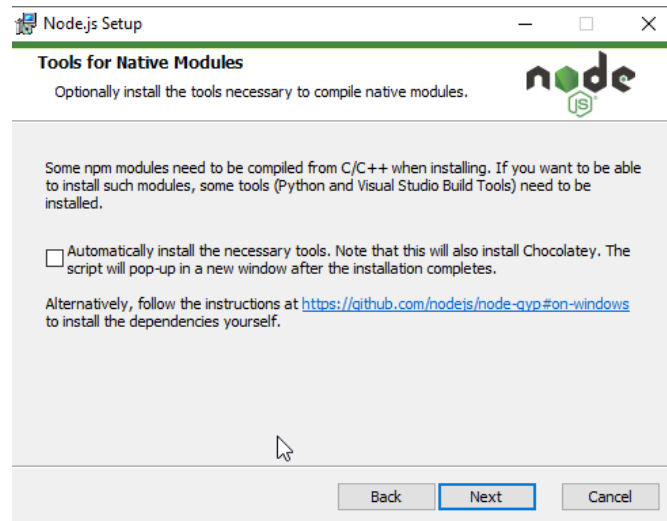
Rysunek 4.15. Ścieżka do folderu docelowego *Node.js*

W kolejnym kroku użytkownik musi wybrać składniki instalacji. W tym przypadku pozostać powinny domyślne opcje. Istotne jest aby zainstalowane zostało narzędzie *NPM*, które pozwoli na instalację serwera WWW.



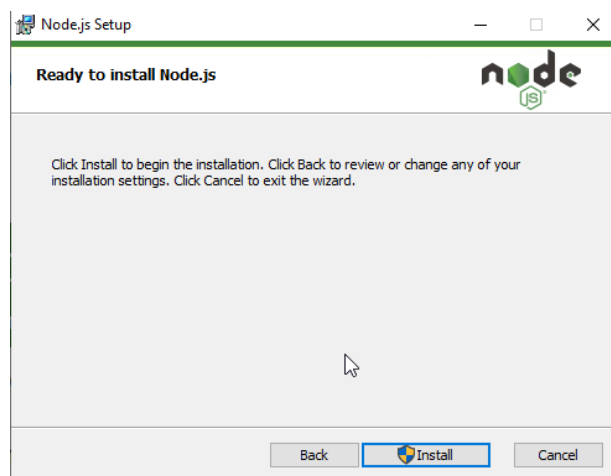
Rysunek 4.16. Składniki instalatora *Node.js*

Jednym z ostatnich kroków instalatora jest widok z informacją o możliwości zainstalowania dodatkowego narzędzia *Chocolatey*. Narzędzie to nie jest potrzebne do działania, więc można wyłączyć opcję instalacji narzędzia.



Rysunek 4.17. Formularz umożliwiający instalację dodatkowego narzędzia *Chocolatey*

W kolejnych etapach instalacji użytkownik powinien uruchomić instalację za pomocą przycisku *Install*. Po zakończonej instalacji instalator powinien wyświetlić informację o zakończeniu pracy.



Rysunek 4.18. Potwierdzenie instalacji środowiska *Node.js*

Aby sprawdzić czy instalacja przebiegła pomyślnie, można uruchomić polecenie `node -v`, które powinno zwrócić informację o zainstalowanej wersji *Node.js*.

#### 4.2.4 Instalacja *http-server*

Ostatnim etapem instalacji systemu jest instalacja serwera WWW, który obsługuje możliwość przekierowań do pliku *index.html*. Aby zainstalować serwer *http-server* w terminalu trzeba uruchomić polecenie zaprezentowane na Kod 4.1.

---

```
1  npm install -g http-server
```

---

Kod 4.1. Instalacja paczki *http-server*

Polecenie to zainstaluje paczkę *npm* do użytku w obrębie całego systemu. Dzięki instalacji globalnej użytkownik będzie mógł uruchomić serwer WWW z każdej lokalizacji na dysku.

### 4.3 Sposób aktywacji

#### 4.3.1 Uruchomienie warstwy serwerowej

Aby uruchomić aplikację stworzoną z wykorzystaniem platformy *.NET* istnieje potrzeba skompilowania jej do pliku wykonywalnego. W celu utworzenia pliku *exe* użytkownik musi przejść do folderu, w którym znajduje się projekt *Samson.Web.Application*, a następnie z poziomu terminala wywołać komendę pokazaną na Kod 4.2.

---

```
1  dotnet restore ;  
2  dotnet build --configuration Release
```

---

Kod 4.2. Kompilacja projektu *ASP.NET*

Pierwsza z komend wymusza instalację zewnętrznych zależności, które są zawarte w postaci paczek *NuGet*’owych. Paczki te pozwalają na udostępnianie kodu napisanego przy wykorzystaniu platformy *.NET*, a następnie mogą zostać wykorzystane do innych projektów przy pomocy narzędzia *NuGet*. Druga komenda buduje pro-

jekt i zwraca wynik w postaci dynamicznych bibliotek *dll* i pliku wykonywalnego *exe*.

W folderze `./Samson.Web.Application.WebHost/bin/Release/net5.0` powinien powstać folder zawierający wyniki kompilacji. Następnie w celu uruchomienia pliku wykonywalnego użytkownik powinien przejść do folderu `./Samson.Web.Application.WebHost/bin/Release/net5.0` przy pomocy komendy zaprezentowanej na Kod 4.3.

---

```
1 cd ./Samson.Web.Application.WebHost/bin/Release/net5.0
```

---

#### Kod 4.3. Przejście do folderu zawierającego plik wykonywalny

Aby uruchomić warstwę serwerową aplikacji wymagane jest dodanie odpowiednich zmiennych środowiskowych przy pomocy polecenia pokazanego na Kod 4.4. Przeznaczenie zmiennych jest następujące:

- *MongoDB:DatabaseName* - nazwa bazy danych jaka zostanie utworzona w środowisku *MongoDB*.
- *Authentication:JWT:Key* - klucz wykorzystywany do funkcji mieszającej, która jest wykorzystywana do zabezpieczenia żetonu *JWT*.
- *ASPNETCORE\_ENVIRONMENT* - oznaczenie typu środowiska. Nie powinna zostać zmieniana.
- *ConnectionString:MongoDB:Atlas* - ciąg znaków określający parametry połączenia z bazą danych. Jeżeli w trakcie instalacji serwera *MongoDB* wartości domyślnie nie zostały zmienione to proponowany ciąg znaków powinien pozwolić na poprawne połączenie z bazą danych.

Wartości parametrów mogą zostać zmodyfikowane, ale zalecane jest aby pozostały w niezmienionej formie ze względu na proponowaną w wcześniejszych krokach konfigurację.

---

```
1 $env:MongoDB:DatabaseName="SamsonDatabase";
2 $env:Authentication:JWT:Key="Top□secret□key□to□provide□
  JWT□token";
```

---

---

```
3 $env:ASPNETCORE_ENVIRONMENT="Release"
4 $env:ConnectionString:MongoDB:Atlas="mongodb://localhost
  :27017/?readPreference=primary&ssl=false";
```

---

Kod 4.4. Ustawienie wartości zmiennych środowiskowych

Ostatnim krokiem w uruchomieniu warstwy serwerowej jest uruchomienie polecenia przedstawionego na Kod 4.5.

---

```
1 start Samson.Web.Application.WebHost.exe
```

---

Kod 4.5. Uruchomienie warstwy serwerowej aplikacji

### 4.3.2 Uruchomienie warstwy prezentacji

Tak jak w przypadku warstwy serwerowej, warstwa prezentacji musi zostać skompilowana do postaci, która może zostać obsługiwana przez serwer WWW. Aby to zrobić trzeba uruchomić terminal a następnie przejść do lokalizacji, w której znajduje się projekt *Samson.Web.Ui*. Kolejnym krokiem jest wywołanie poleceń, które zainstalują wymagane zależności a następnie utworzą paczkę wynikową. Polecenia zostały wypisane na Kod 4.6.

---

```
1 npm install; npm run build
```

---

Kod 4.6. Polecenie, które tworzy paczkę możliwą do uruchomienia przez serwer WWW

Końcowym krokiem uruchamiania warstwy interfejsu użytkownika jest uruchomienie serwera WWW z opcją przekierowania do pliku *index.html*. Polecenie zostało przedstawione na Kod 4.7.

---

```
1 cd dist/samson-web-ui; http-server --proxy http://
  localhost:8080?
```

---

Kod 4.7. Polecenie, które tworzy paczkę możliwą do uruchomienia przez serwer WWW

Po wywołaniu poprzedniego polecenia aplikacja internetowa powinna być dostępna pod adresem: *http://localhost:8080*.

## 4.4 Kategorie użytkowników

Aplikacja internetowa kategoryzuje użytkowników na dwie kategorie:

- Trenerów personalnych - pełnią rolę administratorów w systemie. Posiadają możliwość modyfikacji zasobów siłowni, takich jak sprzęt do ćwiczeń, obiekty siłowni, pomieszczeń oraz możliwość organizacji wydarzeń i treningów indywidualnych.
- Klientów - użytkownicy którzy posiadają możliwość zakupu karnetu , zgłoszenia chęci udziału w treningu indywidualnym lub wydarzeniu.

## 4.5 Kwestie bezpieczeństwa

System wykorzystuje żeton bezpieczeństwa *JWT*. Żeton ten generowany jest w części serwerowej aplikacji w trakcie logowania użytkownika. Kiedy użytkownik poprawnie zaloguje się do systemu, serwer przesyła żeton do warstwy interfejsu użytkownika, gdzie zostaje zapisany w *localStorage* przeglądarki. Warstwa prezentacji dodaje za każdym razem żeton do nagłówka żądania protokołu HTTP, które będzie wymagało uwierzytelnienia użytkownika. W momencie gdy warstwa serwerowa otrzyma taki żeton sprawdza zapisane w nim dane o jego terminie ważności i dostęпах, które posiada użytkownik. Jeśli użytkownik systemu posiada odpowiednie uprawnienia do zasobu to otrzymuje odpowiedź na żądanie, w przeciwnym razie system zwraca informacje o błędzie oznaczone statusem 401 (Unauthorized). Brak przesłania żetonu w nagłówku do metody wymagającej autoryzacji skończy się za każdym razem błędem oraz informacją o próbie nieautoryzowanego dostępu do wrażliwych danych.

## 4.6 Przykład działania

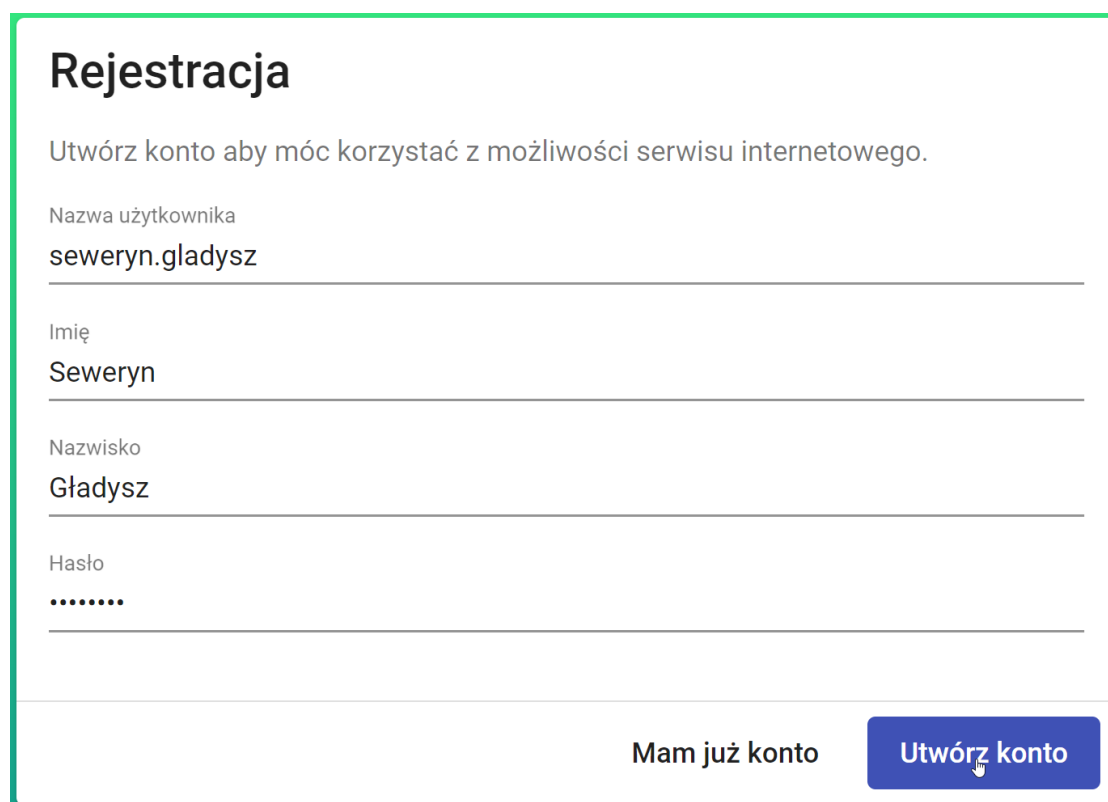
W tej części pracy zaprezentowano przykładowe scenariusze działania systemu. Prezentowane są one z perspektywy różnych użytkowników, ze względu na różnice w posiadanych uprawnieniach.

### 4.6.1 Rejestracja i logowanie

Jako pierwsze zaprezentowane zostanie logowanie i rejestracja w systemie. Są to jedne z początkowych widoków jakie zobaczy użytkownik korzystając z systemu.

#### Rejestracja

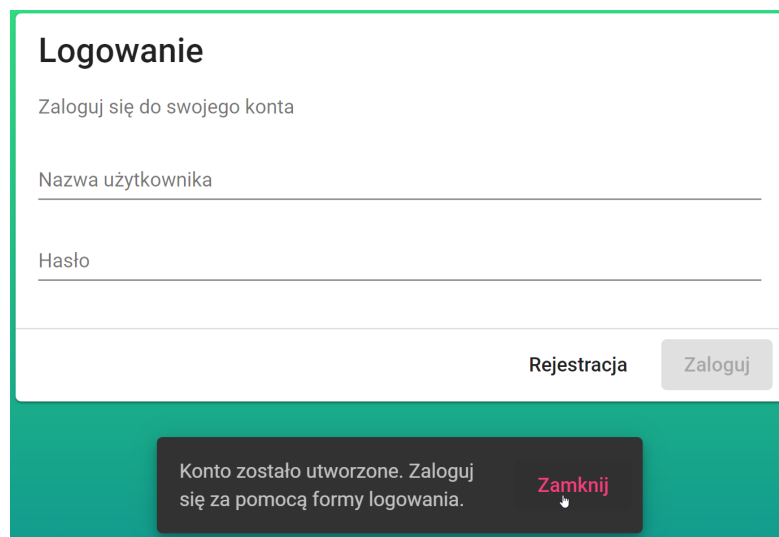
Rejestracja pozwala na utworzenie, konta klienta. Konto daje dostęp do systemu i jego możliwości.



The screenshot shows a web form for registration. At the top, the title 'Rejestracja' is displayed in a large, bold font. Below the title, a subtitle reads: 'Utwórz konto aby móc korzystać z możliwości serwisu internetowego.' The form contains four input fields, each with a label above it: 'Nazwa użytkownika' (containing 'seweryn.gladysz'), 'Imię' (containing 'Seweryn'), 'Nazwisko' (containing 'Gładysz'), and 'Hasło' (containing seven dots). At the bottom right of the form, there are two buttons: a text link 'Mam już konto' and a blue button with white text 'Utwórz konto' and a mouse cursor icon pointing at it.

Rysunek 4.19. Rejestracja klienta

Jeżeli użytkownik wypełni wszystkie wymagane dane oraz wybierze nazwę konta, która nie została wcześniej wykorzystana przez innego użytkownika to powinien zobaczyć informację o poprawnie utworzonym koncie.

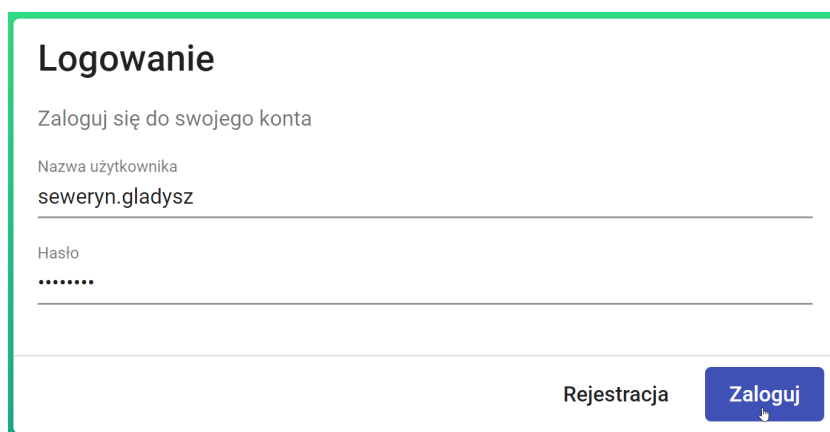


The screenshot shows a web form titled "Logowanie" (Login) with the subtitle "Zaloguj się do swojego konta" (Log in to your account). It contains two input fields: "Nazwa użytkownika" (Username) and "Hasło" (Password). Below the fields are two buttons: "Rejestracja" (Registration) and "Zaloguj" (Login). The "Zaloguj" button is disabled. A dark green banner at the bottom contains a message: "Konto zostało utworzone. Zaloguj się za pomocą formy logowania." (Account created. Log in using the login form.) and a red "Zamknij" (Close) button.

Rysunek 4.20. Klient został zarejestrowany w systemie

## Logowanie

Klient i trener personalny ma możliwość zalogowania się do systemu w celu skorzystania z jego funkcjonalności.

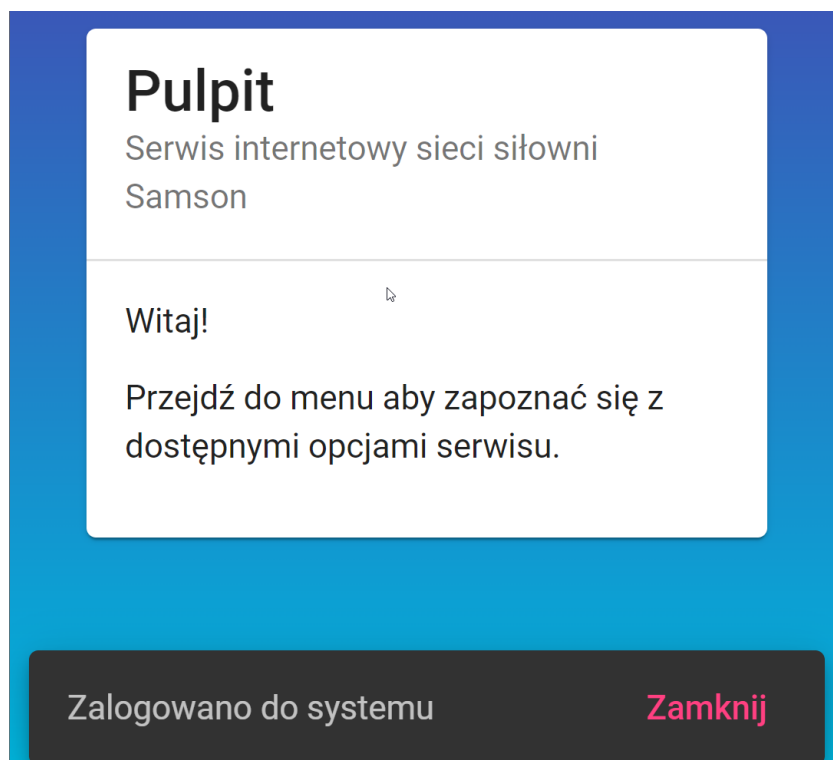


The screenshot shows a web form titled "Logowanie" (Login) with the subtitle "Zaloguj się do swojego konta" (Log in to your account). It contains two input fields: "Nazwa użytkownika" (Username) with the value "seweryn.gladysz" and "Hasło" (Password) with masked characters ".....". Below the fields are two buttons: "Rejestracja" (Registration) and "Zaloguj" (Login). The "Zaloguj" button is active and highlighted in blue.

Rysunek 4.21. Widok logowania do systemu



Użytkownik zostanie zalogowany jeśli wypełni w sposób poprawny dane w formularzu logowania.



Rysunek 4.22. Widok po zalogowaniu do systemu

### 4.6.2 Administracja systemem

System powinien udostępniać możliwość zarządzania systemem oraz zasobami sieci siłowni przy jego pomocy. Aplikacja pozwala na zarządzanie obiektami siłowni, pomieszczeniami, które do tych obiektów zostały przypisane, sprzętem do ćwiczeń, akcesoriami oraz ofertą karnetów.

#### Dodawanie konta trenerów personalnych

Użytkownik ma możliwość utworzenia konta trenera personalnego.

**Zarządzanie siłowniami**  
Zarządzaj obiektami sieci. Dodawaj pomieszczenia i definiuj limity COVIDowe

Trenerzy

Dodaj

Imię  
jan.kowalski

Nazwisko  
Jan

Nazwa użytkownika  
Kowalski

Hasło  
.....

Utwórz

Rysunek 4.23. Formularz tworzenia konta trenera personalnego

## Zarządzanie obiektami siłowni

Trenerzy personalni pełnią rolę administratorów, co pozwala im na zarządzanie obiektami siłowni. W pierwszej kolejności utworzony zostanie nowy obiekt sieci siłowni. W trakcie procesu dodawania obiektu wymagane jest, aby podana została nazwa oraz ilość osób jaka może przebywać na jednym metrze kwadratowym siłowni.

**Zarządzanie siłowniami**  
Zarządzaj obiektami sieci. Dodawaj pomieszczenia i definiuj limity COVIDowe

Siłownie	Trenerzy	Maszyny	Karnety
Edytuj	Dodaj	Dodaj pomieszczenie	Usuń pomieszczenie

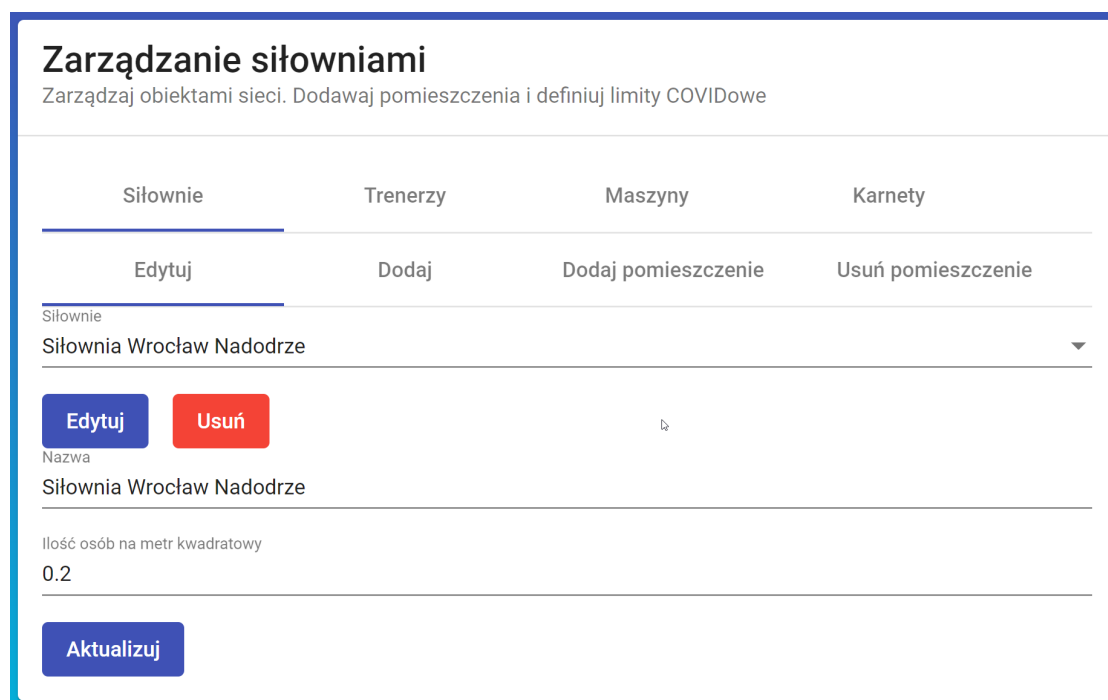
Nazwa

Ilość osób na metr kwadratowy  
5

Aktualizuj

Rysunek 4.24. Formularz tworzenia obiektu siłowni

W formularzu edycji obiektów powinna wyświetlić się możliwość wybrania wcześniej utworzonej siłowni w celu aktualizacji informacji o niej. Widok ten poza edycją pozwala również na zarchiwizowanie obiektu siłowni.



**Zarządzanie siłowniami**  
Zarządzaj obiektami sieci. Dodawaj pomieszczenia i definiuj limity COVIDowe

Siłownie	Trenerzy	Maszyny	Karnety
Edytuj	Dodaj	Dodaj pomieszczenie	Usuń pomieszczenie

Siłownie

Siłownia Wrocław Nadodrze

Edytuj Usuń

Nazwa

Siłownia Wrocław Nadodrze

Ilość osób na metr kwadratowy

0.2

Aktualizuj

Rysunek 4.25. Formularz edycji i archiwizacji obiektów siłowni

## Zarządzanie pomieszczeniami

System udostępnia możliwość dodawania pomieszczeń do obiektów siłowni. Pomieszczenia są istotne ze względu na możliwość przypisania do nich akcesoriów do ćwiczeń oraz wyliczania maksymalnej liczby osób, która może znajdować się na siłowni. W formularzu muszą zostać podane informacje o siłowni, w której znajduje się pomieszczenie, nazwa pomieszczenia oraz jego szerokość i długość.

### Zarządzanie siłowniami

Zarządzaj obiektami sieci. Dodawaj pomieszczenia i definiuj limity COVIDowe

Siłownie	Trenerzy	Maszyny	Karnety
Edytuj	Dodaj	Dodaj pomieszczenie	Usuń pomieszczenie

Siłownie

Siłownia Wrocław Nadodrze

Nazwa pokoju

Strefa ćwiczeń górnej części pleców

Szerokość pokoju

10

Długość pokoju

5

Dodaj pomieszczenie

Rysunek 4.26. Formularz dodawania pomieszczenia do obiektu siłowni

Wcześniej utworzone pomieszczenie może zostać zarchiwizowane.

### Zarządzanie siłowniami

Zarządzaj obiektami sieci. Dodawaj pomieszczenia i definiuj limity COVIDowe

Siłownie	Trenerzy	Maszyny	Karnety
Edytuj	Dodaj	Dodaj pomieszczenie	Usuń pomieszczenie

Siłownie

Siłownia Wrocław Nadodrze

Pomieszczenie

Strefa ćwiczeń górnej części pleców

Usuń

Rysunek 4.27. Formularz archiwizacji pomieszczenia

### Zarządzaniem sprzętem do ćwiczeń i akcesoriami

System informatyczny powinien udostępniać możliwość stworzenia listy posiadanych akcesoriów do ćwiczeń. Dzięki takiej liście personel ma dostęp do informacji o posiadanym sprzęcie co pozwala na przeprowadzanie inwentaryzacji zasobów. Aby dodać sprzęt wymagane jest podanie informacji o kodzie producenta, nazwie, typie oraz siłowni, w której znajduje się akcesorium.

### Zarządzanie siłowniami

Zarządzaj obiektami sieci. Dodawaj pomieszczenia i definiuj limity COVIDowe

Siłownie

Trenerzy

**Maszyny**

Karnety

Edytuj

**Dodaj**

Kod

W\_11AB\_DIPP

Nazwa

GymSportex poręcz do dipów

Typ maszyny:

☐ Ławeczka

☐ Sztanga

☒ Maszyna do dipów

☐ Hantle

☐ Maszyna do wyciskania nogami

☐ Rack

Siłownia

Siłownia Wrocław Nadodrze

Aktualizuj

Rysunek 4.28. Formularz dodawania sprzętu do ćwiczeń

Wcześniej utworzone informacje o akcesorium mogą zostać zaktualizowane lub zarchiwizowane przy pomocy formularza.

## Zarządzanie siłowniami

Zarządzaj obiektami sieci. Dodawaj pomieszczenia i definiuj limity COVIDowe

Siłownie

Trenerzy

**Maszyny**

Karnety

Edytuj

Dodaj

Maszyny

GymSportex poręcz do dipów

Edytuj

Usuń

Kod

W\_11AB\_DIPP

Nazwa

GymSportex poręcz do dipów

Typ maszyny:

☐ Ławeczka

☐ Sztanga

☒ Maszyna do dipów

☐ Hantle

☐ Maszyna do wyciskania nogami

☐ Rack

Siłownia

Siłownia Wrocław Nadodrze

Aktualizuj

Rysunek 4.29. Formularz edycji i archiwizacji akcesoriów

### Zarządzanie ofertą karnetów

Ostatnią funkcjonalnością udostępnioną przez moduł administracji jest funkcja zarządzania ofertą karnetów. System pozwala na utworzenie nowego typu karnetu, jeśli podane zostaną informacje o jego nazwie, czasie trwania w dniach oraz cenie.

The screenshot shows a web application interface titled "Zarządzanie siłowniami" (Gym Management) with the subtitle "Zarządzaj obiektami sieci. Dodawaj pomieszczenia i definiuj limity COVIDowe" (Manage network objects. Add rooms and define COVID limits). The interface has a top navigation bar with four tabs: "Siłownie", "Trenerzy", "Maszyny", and "Karnety". The "Karnety" tab is selected. Below the tabs, there are two sub-tabs: "Edytuj" and "Dodaj". The "Dodaj" sub-tab is active. The form contains three input fields: "Nazwa" (Name) with the value "Brązowy Karnet", "Czas trwania" (Duration) with the value "15", and "Cena" (Price) with the value "70". At the bottom left of the form is a blue button labeled "Aktualizuj" (Update).

Rysunek 4.30. Formularz dodawania karnetu do oferty

Wcześniej utworzone karnety mogą zostać zaktualizowane lub zarchiwizowane.

The screenshot shows the same web application interface as Figure 4.30, but with the "Edytuj" sub-tab selected. The "Karnety" tab remains selected. The form now includes two buttons at the top left: a blue "Edytuj" (Edit) button and a red "Usuń" (Delete) button. The input fields for "Nazwa", "Czas trwania", and "Cena" are still present and contain the same values as in Figure 4.30. The "Aktualizuj" button is still at the bottom left.

Rysunek 4.31. Formularz edycji i archiwizacji karnetów

### 4.6.3 Wydarzenia

Moduł wydarzeń udostępnia możliwość tworzenia, przeglądania, anulowania i zgłaszania chęci wzięcia udziału w wydarzeniu w zależności od rodzaju konta użytkownika.

#### Tworzenie wydarzenia

Konto trenera personalnego pozwala na tworzenie wydarzeń, w których będą mogli brać udział klienci w określonym czasie.

**Wydarzenia**  
Sprawdź dostępne wydarzenia i zapisz się na najciekawsze

< enia      Moje wydarzenia (trener)      Utwórz wydarzenie >

Nazwa  
Wspólne ćwiczenie górnych partii pleców

Data rozpoczęcia  
28.12.2021

Godzina rozpoczęcia  
5:00 PM

Data zakończenia  
28.12.2021

Godzina rozpoczęcia  
8:00 PM

Maksymalna liczba uczestników  
12

Pomieszczenie  
Strefa ćwiczeń górnej części pleców

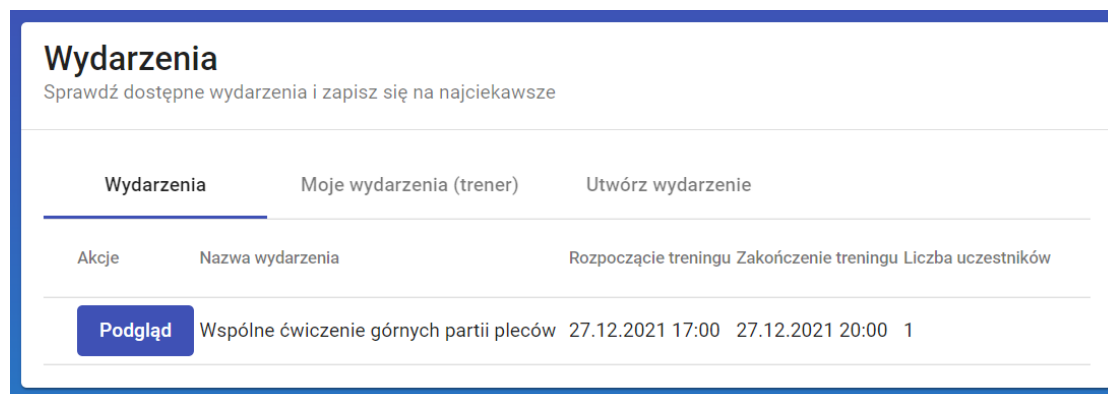
**Utwórz wydarzenie**

Rysunek 4.32. Formularz tworzenia wydarzenia



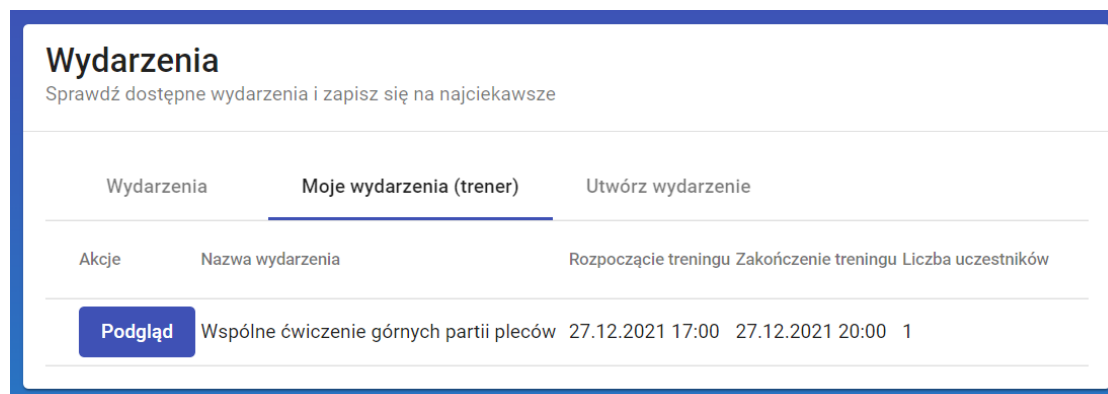
## Listy wydarzeń

Klienci i trenerzy personalni mają dostęp do listy wszystkich wydarzeń. Z poziomu tej listy użytkownicy mogą przejść do poglądu wydarzenia, gdzie znajdują się dodatkowe opcje zarządzania wydarzeniem.



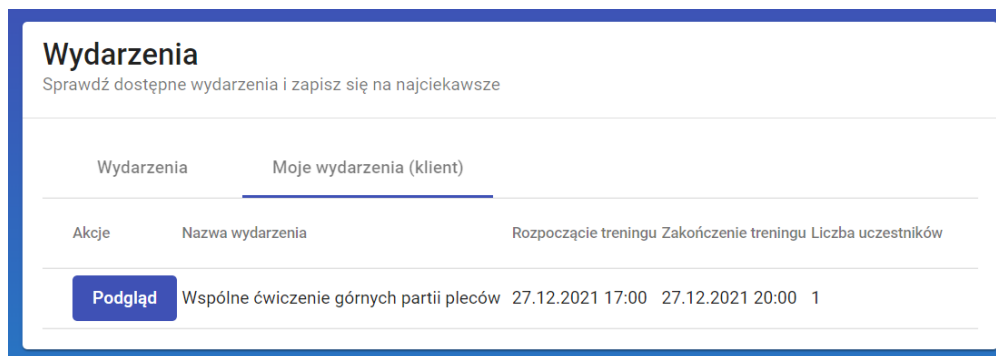
Rysunek 4.33. Lista wszystkich wydarzeń

Trenerzy personalni mają dodatkowo możliwość wyświetlenia listy wydarzeń, które sami utworzyli.



Rysunek 4.34. Lista utworzonych wydarzeń przez trenera personalnego

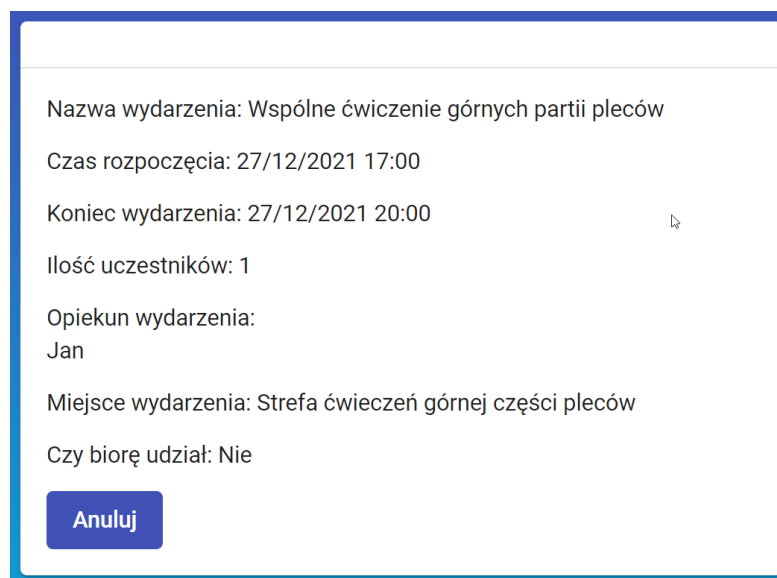
Klienci natomiast mają możliwość wyświetlenia wydarzeń, w których zgłosili chęć wzięcia udziału.



Rysunek 4.35. Lista wydarzeń klienta

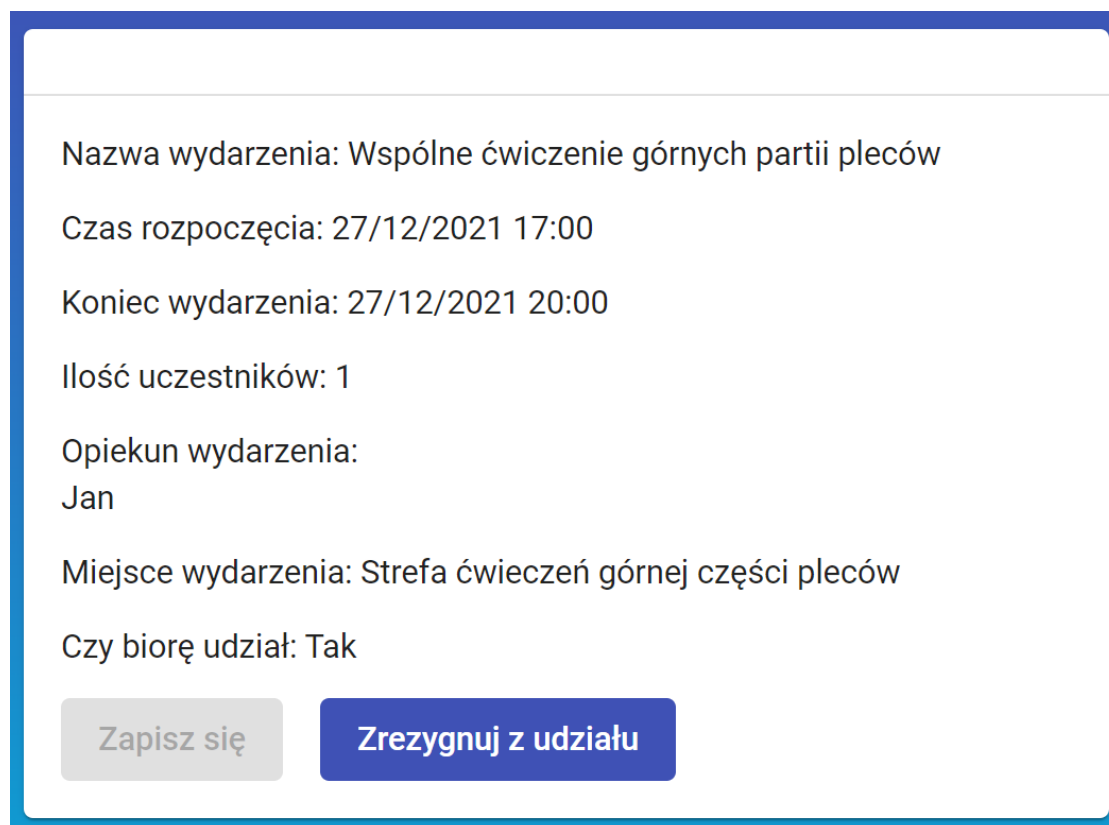
### Podgląd wydarzeń

Systemu udostępnia widok podglądu wydarzenia. Użytkownicy mogą przeglądać wszystkie wydarzenia, ale dostępne opcje są zależne od roli jaką pełnią. Trener personalny w podglądzie wydarzenia zobaczy opcję anulowania wydarzenia.



Rysunek 4.36. Podgląd wydarzenia jako trener personalny

Klient na tym samym widoku zobaczy opcję zgłoszenia chęci udziału w wydarzeniu lub zrezygnowania z niego.



Nazwa wydarzenia: Wspólne ćwiczenie górnych partii pleców

Czas rozpoczęcia: 27/12/2021 17:00

Koniec wydarzenia: 27/12/2021 20:00

Ilość uczestników: 1

Opiekun wydarzenia:  
Jan

Miejsce wydarzenia: Strefa ćwiczeń górnej części pleców

Czy biorę udział: Tak

Zapisz się      Zrezygnuj z udziału

Rysunek 4.37. Pogląd wydarzenia jako klient

#### 4.6.4 Treningi indywidualne

Kolejnym modułem w systemie jest moduł treningów indywidualnych. Moduł ten pozwala na organizowanie treningów, w których brać będzie udział klient pod nadzorem trenera personalnego. Pozwala to na utworzenie oferty dostosowanej do klientów, którzy chcą ćwiczyć pod okiem specjalistów.

## Tworzenie treningów indywidualnych

Trener personalny posiada możliwość utworzenia treningu. Aby dodać trening, użytkownik z rolą trenera musi wskazać w formularzu, gdzie odbędzie się trening oraz w jakim terminie.

The screenshot shows a web form titled "Treningi indywidualne" with the subtitle "Zapisz się na trening z trenerem personalnym". The form has two tabs: "Dodaj trening" (selected) and "Moje treningi (trener)". The form fields are as follows:

Label	Value	Icon
Obiekt siłowni	Siłownia Wrocław Nadodrze	Dropdown arrow
Data rozpoczęcia treningu	11.1.2022	Calendar icon
Godzina rozpoczęcia treningu	8:00 PM	Clock icon
Data zakończenia treningu	11.1.2022	Calendar icon
Godzina zakończenia treningu	9:00 PM	Clock icon

At the bottom of the form is a blue button labeled "Utwórz trening".

Rysunek 4.38. Formularz dodawania treningu

## Listy treningów indywidualnych

System pozwala klientom na pogląd listy dostępnych treningów. Treningi w statusie *Open* oznaczają treningi, w których żaden użytkownik nie zgłosił chęci wzięcia udziału.

Dostępne treningi indywidualne		Moje treningi (klient)			
Akcje	Siłownia	Trener	Rozpoczęcie treningu	Zakończenie treningu	Status
	Siłownia Wrocław Nadodrze	Jan	28.12.2021 07:00	28.12.2021 07:45	Oczekujący na potwierdzenie
Zapisz się	Siłownia Wrocław Nadodrze	Jan	10.01.2022 20:00	10.01.2022 21:00	Dostępny

Rysunek 4.39. Lista dostępnych treningów indywidualnych

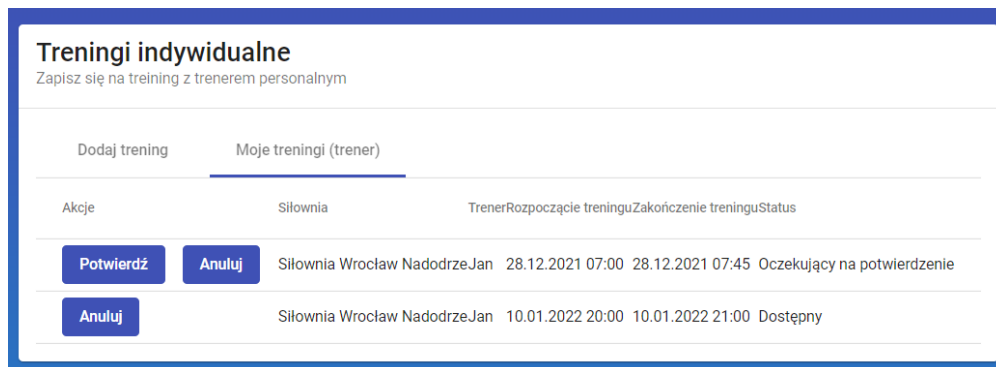
Z perspektywy klienta dostępna jest również lista treningów, na które się zapisał. Dzięki tej liście klient będzie mógł sprawdzić na jakich treningach był oraz jakie go jeszcze czekają.

Dostępne treningi indywidualne		Moje treningi (klient)		
AkcjeSiłownia	Trener	Rozpoczęcie treningu	Zakończenie treningu	Status
Siłownia Wrocław Nadodrze	Jan	28.12.2021 07:00	28.12.2021 07:45	Oczekujący na potwierdzenie

Rysunek 4.40. Lista treningów, na które zapisał się klient

Trener personalny natomiast posiada możliwość poglądu utworzonych przez siebie wydarzeń. Lista pozwala na odwołanie treningu lub potwierdzenie go.

To jakie opcje są dostępne jest zależne od statusu w jakim znajduje się dany trening.



**Treningi indywidualne**  
Zapisz się na trening z trenerem personalnym

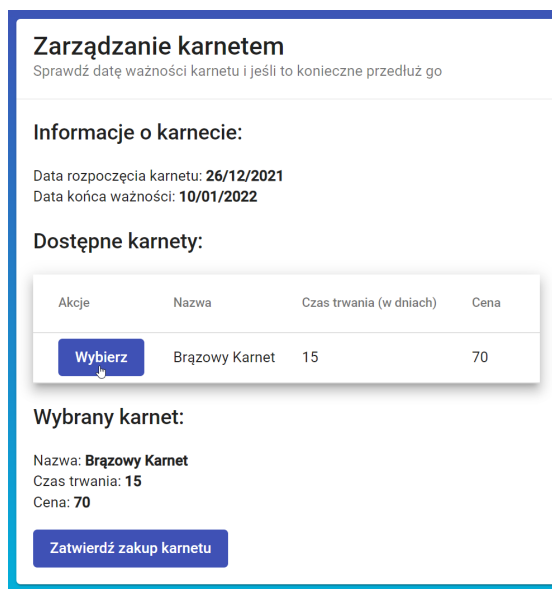
Dodaj trening      Moje treningi (trener)

Akcje	Siłownia	Trener	Rozpoczęcie treningu	Zakończenie treningu	Status
<button>Potwierdź</button> <button>Anuluj</button>	Siłownia Wrocław Nadodrze	Jan	28.12.2021 07:00	28.12.2021 07:45	Oczekujący na potwierdzenie
<button>Anuluj</button>	Siłownia Wrocław Nadodrze	Jan	10.01.2022 20:00	10.01.2022 21:00	Dostępny

Rysunek 4.41. Lista utworzonych treningów przez trenera

#### 4.6.5 Karnety

System posiada moduł karnetów, gdzie klient może sprawdzić ważność karnetu, dostępną ofertę, zakupić lub przedłużyć karnet.



**Zarządzanie karnetem**  
Sprawdź datę ważności karnetu i jeśli to konieczne przedłuż go

**Informacje o karnecie:**  
Data rozpoczęcia karnetu: **26/12/2021**  
Data końca ważności: **10/01/2022**

**Dostępne karnety:**

Akcje	Nazwa	Czas trwania (w dniach)	Cena
<button>Wybierz</button>	Brązowy Karnet	15	70

**Wybrany karnet:**  
Nazwa: **Brązowy Karnet**  
Czas trwania: **15**  
Cena: **70**

Zatwierdź zakup karnetu

Rysunek 4.42. Oferta karnetów oraz informacje o posiadanym karnecie

### 4.6.6 Tryb bramek

Istotnym z punktu widzenia pandemicznych obostrzeń jest moduł bramek. System udostępnia tryb bramek, które pozwalają na podłączenie aplikacji do bramek i wyświetlaczy w celu kontroli liczby osób znajdujących aktualnie na siłowni.

#### Bramka wejściowa

Głównym zadaniem bramki wejściowej jest kontrolowanie ważności posiadanego karnetu oraz kontroli czy na siłowni nie znajduje się zbyt duża liczba osób w stosunku do obowiązujących obostrzeń. Aby wejść na siłownię klient musi podać lub zeskanować za pomocą skanera swój kod identyfikacyjny. Formularz powinien w tym momencie wyświetlić informację o tym czy istnieje możliwość wejścia na siłownię. To czy można wejść na siłownię zależy od liczby osób, która znajduje się na siłowni oraz ważności karnetu klienta.

### Bramka wejściowa

Wprowadź nazwę klienta i wejdź na siłownię

---

Identyfikator użytkownika  
61c3710dbf67178d86eeccc2

---

Sprawdź

Aktualna liczba klientów na siłowni: **2**

Maksymalna liczba klientów na siłowni: **10**

Czy klient ma aktualny karnet: **Tak**

Czy siłownia jest pełna: **Nie**

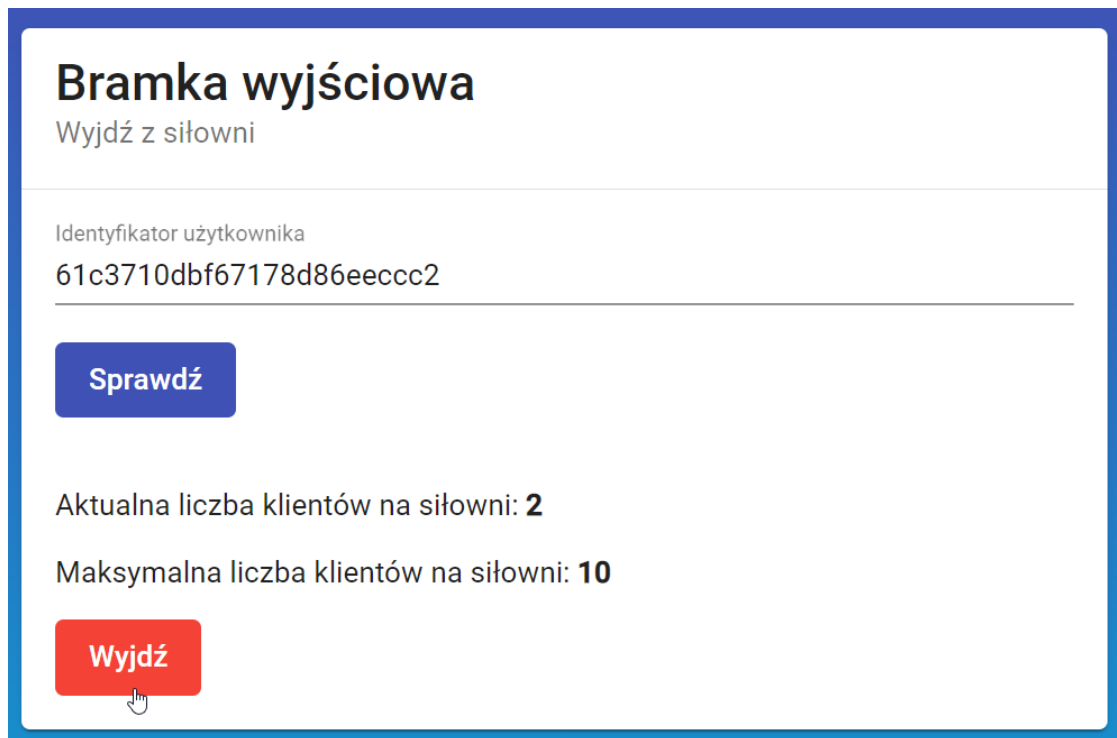
Czy można wejść: **Tak**

Wejdź

Rysunek 4.43. Widok wyświetlany na wyświetlaczu bramki wejściowej

## Bramka wyjściowa

Zadaniem bramki wyjściowej jest usuwanie z listy klientów osób, które już nie korzystają z usług danego obiektu siłowni.



**Bramka wyjściowa**  
Wyjdź z siłowni

---

Identyfikator użytkownika  
61c3710dbf67178d86ecccc2

---

**Sprawdź**

Aktualna liczba klientów na siłowni: **2**

Maksymalna liczba klientów na siłowni: **10**

**Wyjdź**

Rysunek 4.44. Widok wyświetlany na wyświetlaczu bramki wyjściowej



# Rozdział 5

## Specyfikacja wewnętrzna

Rozdział rozpoczyna się od opisanie architektury systemu z podziałem na warstwy z uwzględnieniem komunikacji pomiędzy nimi. W dalszej części rozdziału znajduje się przedstawienie organizacji bazy danych. Następnie w rozdziale przedstawiono przegląd najważniejszych klas i algorytmów. Rozdział zakończony jest opisem zastosowanych wzorców projektowych.

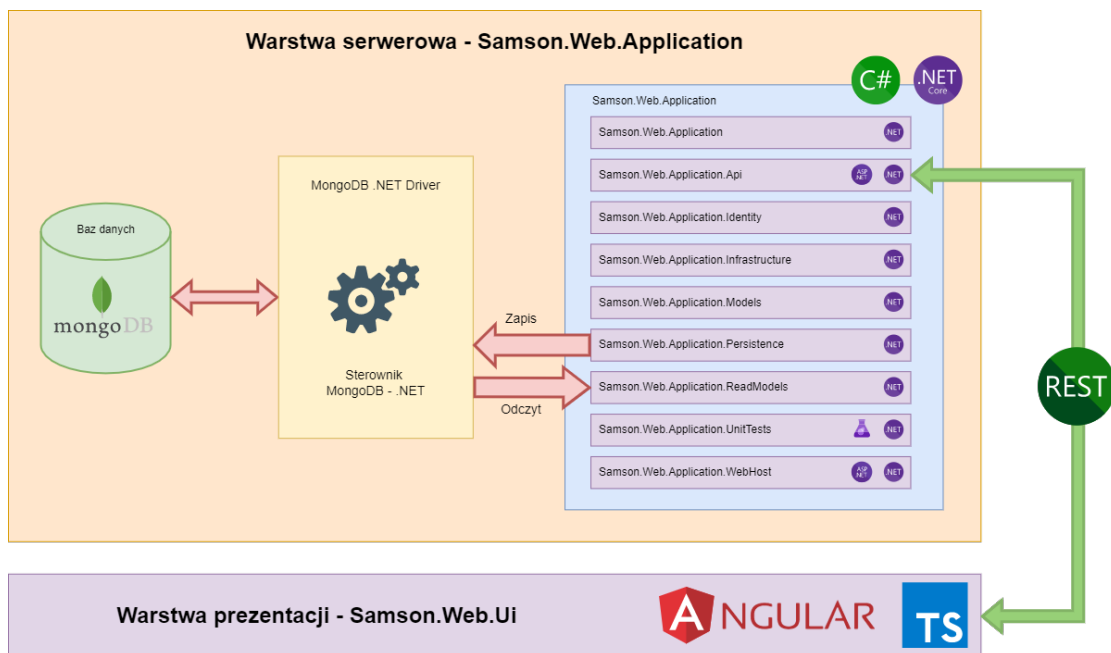
### 5.1 Architektura systemu

System został podzielony na dwie główne warstwy. Architektura została zaprezentowana na rysunku 5.1.

#### Warstwa serwerowa

Warstwa serwerowa to część aplikacji znajdująca się na serwerze. Jej głównym zadaniem jest obsługiwanie żądań wysyłanych przez część kliencką systemu. Warstwa serwerowa składa się z głównego projektu *Samson.Web.Application*, w którego skład wchodzi projekty platformy .NET:

- *Samson.Web.Application* - projekt w skład którego wchodzi serwisy aplikacyjne realizujące logikę biznesową. Ważnymi składowymi projektu są klasy obsługi komend (ang. *command handler*), których zadaniem jest aktualizacja danych oraz klasy obsługi zapytań (ang. *query handler*), które odpowiadają za odczyt danych z bazy.



Rysunek 5.1. Architektura systemu

- *Samson.Web.Application.Api* - projekt, który przy pomocy kontrolerów (ang. *controllers*) obsługuje otrzymane żądania z warstwy klienckiej. Metoda klasy kontrolera tworzy komendę (ang. *command*) lub zapytanie (ang. *query*), które jest następnie przekazywane do mediatora (ang. *mediator*). Mediator rozdziela je do odpowiedniej klasy obsługi komendy lub klasy obsługi zapytań. Takie podejście pozwala na stworzenie skalowalnej architektury podatnej na zmiany. Dzięki takiemu podejściu klasy kontrolerów nie muszą wstrzykiwać zależności w postaci serwisów aplikacyjny lub klas repozytorium (ang. *repository*).
- *Samson.Web.Application.Identity* - to projekt odpowiedzialny za tworzenie żetonów JWT oraz sprawdzanie ich poprawności. *Samson.Web.Application.Identity* pełni kluczową rolę w ochronie systemu przed nieautoryzowanym dostępem.
- *Samson.Web.Application.Infrastructure* - projekt w skład, którego wchodzi atrybuty, oprogramowanie pośrednie (ang. *middleware*) oraz klasy wykorzystywane w pozostałych projektach warstwy serwerowej. Projekt zawiera także klasy globalnej obsługi błędów (ang. *global error handler*), których zadaniem

jest przechwytywanie wyjątków (ang. *exception*), które nie zostały wcześniej obsługane.

- *Samson.Web.Application.Models* - projekt zawierający klasy modeli, obiekty transferu danych (ang. *data transfer object*), struktury danych oraz wyliczenia (ang. *enum*).
- *Samson.Web.Application.Persistence* - projekt pełniący kluczową rolę, jeśli chodzi o zapis danych. Projekt zawiera klasy encji (ang. *entity*) oraz klasy repozytoriów. Klasy repozytoriów tworzą warstwę odpowiedzialną za aktualizację danych przy pomocy sterownika *MongoDB .NET Driver*.
- *Samson.Web.Application.ReadModels* - w przeciwieństwie do poprzedniego projektu, w skład *Samson.Web.Application.ReadModels* wchodzi klasy odpowiedzialne za odczyt danych z bazy. Zadanie to jest realizowane przy wykorzystaniu klas odczytu modeli (ang. *read model*), w których znajduje się kod realizujący zapytania do bazy przy pomocy wstawek *JavaScript* lub składni *LINQ*.
- *Samson.Web.Application.UnitTests* - projekt zawierający kod testów jednostkowych
- *Samson.Web.Application.WebHost* - projekt, który jest punktem początkowym aplikacji. W projekcie można znaleźć pliki konfiguracji z rozszerzeniem *JSON*.

Istotną częścią warstwy serwerowej jest baza danych, która odpowiada za przechowywanie danych w sposób uporządkowany i trwały. Projekt główny realizuje komunikację z bazą danych przy pomocy sterownika, który pozwala na zintegrowanie technologii *.NET* i bazy *MongoDB*.

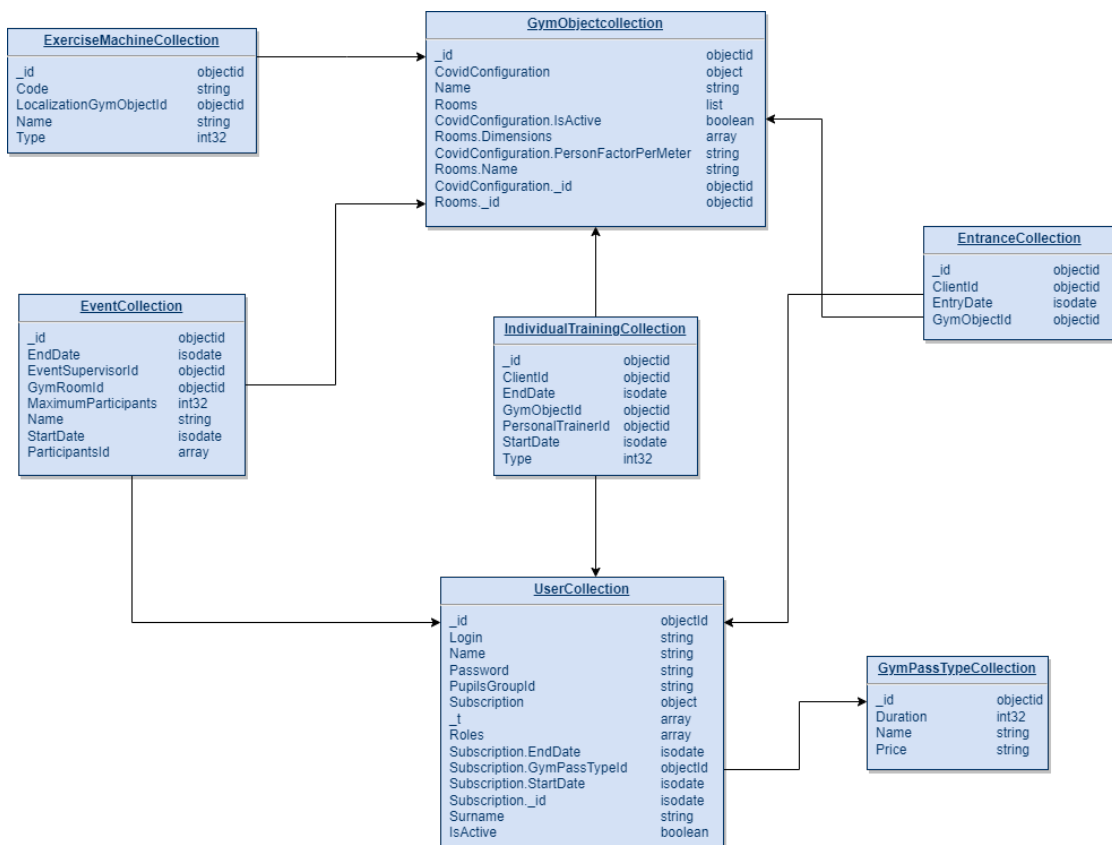
## Warstwa prezentacji

Warstwa prezentacji wysyła żądania do warstwy serwerowej przy pomocy metod *HTTP* i interfejsu *REST API*, który jest punktem łączącym obie warstwy. Podstawowym zadaniem, które realizuje ta warstwa jest graficzna prezentacja funkcjonalności w celu uproszczenia korzystania z systemu.

## 5.2 Organizacja bazy danych

Baza danych *MongoDB* jest dokumentową, niereleacyjną bazą danych *NoSQL*, więc nie posiada typowej struktury bazy *SQL*. W przeciwieństwie do baz relacyjnych w bazie *MongoDB* często stosuje się denormalizację struktury w celu poprawienia wydajności odczytu danych. Wadą tego rozwiązania jest potrzeba ręcznego aktualizowania powielonych dokumentów oraz mniejsza wydajność zapisu.

Zaprezentowany schemat bazy prezentuje kolekcje dokumentów, jakie zawiera baza. Na schemacie uwzględnione zostały relacje, które utworzono przy wykorzystaniu referencji. Takie podejście pozwala na wprowadzenie normalizacji danych i uniknięcie powielania dokumentów[3].



Rysunek 5.2. Poglądowy schemat bazy danych

Poszczególne kolekcje zostały podzielone według funkcjonalności, którym odpowiadają:

- *ExerciseMachineCollection* - kolekcja zawierająca informacje o akcesoriach do ćwiczeń. Dokumenty w kolekcji zawierają referencję do *GymObjectCollection*. Referencja została zrealizowana przy pomocy klucza *LocalizationGymObjectId*, który pozwala na określenie gdzie znajduje się dany przyrząd do ćwiczeń.
- *GymObjectCollection* - kolekcja przechowująca informacje o obiektach i pomieszczeniach siłowni.
- *EventCollection* - zawiera informacje o wydarzeniach organizowanych przez sieć. Kolekcja ta posiada referencje do kolekcji *GymObjectCollection* i *UserCollection*.
- *IndividualTraniningCollection* - kolekcja odpowiedzialna za przechowywanie informacji o treningach indywidualnych. W celu określenia miejsca treningu posiada referencję do kolekcji *GymObjectCollection*. Organizator i klienci biorący udział w wydarzeniu są odnajdywani w bazie przy użyciu referencji do *UserCollection*.
- *EntranceCollection* - kolekcja zawierająca informacje o klientach, którzy znajdują się wewnątrz siłowni.
- *UserCollection* - kolekcja przechowująca informacje o klientach i trenerach personalnych.
- *GymPassTypeCollection* - kolekcja w ramach, której znajdują się dokumenty opisujące rodzaje karnetów, które są oferowane przez sieć.

## 5.3 Przegląd najważniejszych klas

W trakcie pracy nad systemem zaistniała potrzeba stworzenia klasy pozwalającej na modyfikacje danych w bazie. Głównym założeniem było utworzenie klasy, która będzie wykorzystywać wzorzec repozytorium oraz metodę szablonową. Wzorzec repozytorium miał pomóc w stworzeniu klasy, której głównym zadaniem będzie realizowanie operacji *CRUD*, natomiast wykorzystanie wzorca metody sza-

blonowej miało pomóc w stworzeniu ogólnego algorytmu, który będzie uszczegółowiony w klasach pochodnych. Według tych zasad powstała abstrakcyjna klasa *MongoRepository* reprezentująca ogólne algorytmy wykorzystywane do operacji *CRUD*. Klasa implementuje operacje, które nie dotyczą żadnej, konkretnej kolekcji w bazie danych. To klasy pochodne *MongoRepository*, wykorzystując klasę bazową, tworzą interfejs do zarządzania dokumentami w konkretnej kolekcji. Kod klasy został zaprezentowany na Kod 1

W projekcie można znaleźć wiele klas, które pełnią rolę klas odczytu danych. W takich przypadkach nie stworzono ogólnej klasy bazowej ze względu na niewielkie podobieństwo klas między sobą. Odczyt danych z bazy można osiągnąć na kilka sposobów przy wykorzystaniu sterownika *.NET - MongoDB*. Pierwszy z nich to wykorzystanie wstawek w postaci języka *JavaScript*, co pozwala na pisanie zapytań takich jak w konsoli *MongoDB*. Jedną z takich klas, gdzie wykorzystano możliwości wstawek jest klasa *ExerciseMachineReadModel*. Podstawowym zadaniem tej klasy jest odczyt danych z kolekcji *ExerciseMachineCollection*, gdzie znajdują się informacje o posiadanym sprzęcie do ćwiczeń. Mimo, że *MongoDB* jest nierelacyjną bazą danych, to dzięki wykorzystaniu operatora *lookup* można stworzyć namiastkę złączeń (ang. *join*), które są znane z relacyjnych baz danych. Na Kod 5.1 zaprezentowana została metoda wykorzystująca operator *lookup* i wcześniej wspomniane wstawki kodu.

---

```
1  private IAggregateFluent<BsonDocument>
    GetAllExerciseMachinesQuery()
2  {
3      var client = new MongoClient(_databaseConfiguration.
        ConnectionString);
4      var database = client.GetDatabase(
        _databaseConfiguration.DatabaseName);
5
6      var collection = database.GetCollection<
        ExerciseMachineEntity>("ExerciseMachineCollection");
7
8      return collection
```

---

```

9      .Aggregate()
10     .AppendStage<BsonDocument>("{$_lookup:{$from: '
      GymObjectCollection', _localField: '
      LocalizationGymObjectId', _foreignField: '$_id', _as: '
      LocalizationGymObject'} }")
11     .AppendStage<BsonDocument>("{$_unwind: '
      $LocalizationGymObject' } }");
12 }
13 }
```

---

Kod 5.1. Metoda odczytu danych z kolekcji *ExerciseMachineCollection* przy wykorzystaniu wstawek *JavaScript*

Innym sposobem odczytu danych z bazy jest wykorzystanie składni *LINQ*, która pozwala na m.in. tworzenie zapytań. Pomimo dużej ograniczeń tego rozwiązania, *LINQ* posiada możliwość tworzenia złączeń. Złączenia te są jednak bardzo ograniczone w porównaniu do baz relacyjnych. Na Kod 5.2 przedstawiono fragment klasy *GymObjectReadModel*, która odpowiada za odczyt danych z kolekcji *GymObjectCollection*, gdzie znajdują się informacje o obiektach siłowni i pomieszczeniach. Jak widać składnia *LINQ* pozwala na tworzenie zapytań, które przypominają składnię języka *SQL*.

---

```

1  public Task<GymObjectDto> GetById( ObjectId id )
2  {
3      var client = new MongoClient( _databaseConfiguration .
      ConnectionString );
4      var database = client .GetDatabase(
      _databaseConfiguration .DatabaseName );
5
6      var collection = database .GetCollection<GymObjectEntity>
      >("GymObjectCollection");
7
8      var query = from gymObject in collection .AsQueryable()
9      where gymObject.Id == id
```

```
10     select gymObject;  
11  
12     return query  
13         .SingleOrDefaultAsync()  
14         .ContinueWith(result => _mapper.Map<GymObjectDto>(  
            result.Result));  
15 }
```

---

Kod 5.2. Metoda odczytu danych z kolekcji *GymObjectCollection* przy wykorzystaniu wstawek składni *LINQ*

Ostatnim sposobem odczytu danych z bazy jest wykorzystanie metod przygotowanych w bibliotece sterownika *MongoDB .NET Driver*. Posiadają takie same parametry jak te z języka *JavaScript*, w którym domyślnie tworzone są zapytania dla bazy *MongoDB*. Z powodu braku niektórych metod względem tych z języka *JavaScript*, programista jest zmuszony do umieszczania wstawek kodu takich jak w klasie *ExerciseMachineReadModel*. Na Kod 5.3 zaprezentowano metodę klasy *PersonalTrainerReadModel*, której zadaniem jest odczyt z bazy informacji o koncie trenera personalnego.

---

```
1     public Task<PersonalTrainerDto> GetById(ObjectId id)  
2     {  
3         var client = new MongoClient(_databaseConfiguration.  
            ConnectionString);  
4         var database = client.GetDatabase(  
            _databaseConfiguration.DatabaseName);  
5  
6         var collection = database.GetCollection<  
            PersonalTrainerEntity>("UserCollection");  
7  
8         var query = collection  
9             .Aggregate()  
10            .Match(clientEntity => clientEntity.Id == id)  
11            .As<PersonalTrainerDto>();
```



```
12
13     return query.SingleOrDefaultAsync();
14 }
```

---

Kod 5.3. Metoda odczytu informacji o koncie trenera personalnego przy wykorzystaniu metod sterownika

## 5.4 Przegląd wzorców projektowych

Poza wcześniej wymienionymi wzorcami repozytorium i metoda szablonowa do stworzenia aplikacji wykorzystano wzorzec *CQRS*. Do implementacji wzorca *CQRS* wykorzystuje się klasy komend i zapytań w celu rozdzielenia zadań dotyczących zapisu i odczytu danych. Komendy są obsługiwane przez klasy obsługi komend, gdzie wywoływane są metody serwisów aplikacyjnych do modyfikacji danych w bazie. Każda klasa obsługi komendy posiada metodę *Handle*, która zostaje wywołana gdy znajdzie potrzeba obsłużenia żądania w postaci obiektu typu *IRequest*. Na Kod 5.4 widać metodę *Handle*, która jest fragmentem klasy *CreateGymPassTypeCommandHandler*.

---

```
1  public Task<ObjectId> Handle(CreateGymPassTypeCommand
    request, CancellationToken cancellationToken)
2  {
3      var dataStructure = _mapper.Map<
        CreateGymPassTypeCommand,
        CreateGymPassTypeDataStructure>(request);
4      return _service.Create(dataStructure);
5  }
```

---

Kod 5.4. Metoda *Handle* klasy *CreateGymPassTypeCommandHandler*

Klasy obsługi komend odpowiadają za aktualizację danych w bazie, natomiast klasy obsługi odczytu mają za zadanie odczytać dane z bazy w celu przekazania ich do warstwy prezentacji. Na Kod 5.5 zaprezentowana została klasa *GymPassTypesQueryHandler*, która odpowiada za odczyt informacji o wszystkich typach karnetów w bazie.

---

```
1  public class GymPassTypesQueryHandler : IRequestHandler<
    GetAllGymPassTypesQuery , List<GymPassTypeDto>>
2  {
3      private readonly IGymPassReadModel _readModel;
4
5      public GymPassTypesQueryHandler(IGymPassReadModel
        readModel)
6      {
7          _readModel = readModel ?? throw new
            ArgumentNullException(nameof(readModel));
8      }
9
10     public Task<List<GymPassTypeDto>> Handle(
        GetAllGymPassTypesQuery query , CancellationToken
        cancellationToken)
11     {
12         return _readModel.GetAll();
13     }
14 }
```

---

Kod 5.5. Klasa *GymPassTypesQueryHandler*

Dopełnieniem *CQRS* jest wzorzec projektowy mediator, którego zadaniem jest przekazywanie obiektów klas odczytu i komend do odpowiednich klas obsługi odczytu i obsługi komend. Takie rozwiązanie pozwala na zwiększenie skalowalności systemu i usunięcie potrzeby wstrzykiwania serwisów aplikacyjnych do klas kontrolerów. Klasa mediatora odpowiada za poprawne dostarczenie obiektu żądania do obiektu klasy obsługi tego żądania. Poniżej przedstawiono fragment kodu klasy kontrolera *EventController*, który odpowiada za udostępnienie metod *CRUD* dotyczących obiektów klasy *Event*. Na Kod 5.6 widać jak wykorzystywany jest mediator do przekazania żądania odczytu informacji o wydarzeniu.

---

```
1  public async Task<ActionResult> GetById(string id)
```

---

```
2  {
3      if (id.IsNullOrEmpty())
4      {
5          return BadRequest();
6      }
7
8      var query = _mapper.Map<string, GetEventByIdQuery>(id);
9      var queryResult = await _mediator.Send(query);
10     var result = _mapper.Map<EventDto, EventViewModel>(
11         queryResult);
12     return Ok(result);
13 }
```

---

Kod 5.6. Metoda *GetById* klasy *EventController*



## Rozdział 6

# Weryfikacja i walidacja

W tej części pracy opisano sposoby weryfikacji i walidacji tworzonego projektu. W pierwszej części przedstawiono metody testowania kodu aplikacji oraz jego przykłady. Następnie zaprezentowano sposób w jaki zorganizowano testy. Rozdział zakończony jest paroma przykładami błędów, które zostały wykryte dzięki testom.

### 6.1 Sposoby testowania

W trakcie pisania kodu systemu zaszła potrzeba sprawdzenia poprawności jego działania, aby to zrobić wykorzystano testy jednostkowe i manualne. Kod części serwerowej został przetestowany przy pomocy testów jednostkowych i manualnych, natomiast warstwa prezentacji została przetestowana przy pomocy testów manualnych. W celu utworzenia testów jednostkowych wykorzystano bibliotekę *NUnit*.

Testy jednostkowe pozwoliły na testowanie pojedynczych metod i klas przy pomocy kodu *C#*. Przetestowane zostały metody klas obiektów domenowych, w których znajdowała się logika biznesowa systemu. Przykładowo na Kod 6.1 przedstawiono test jednostkowy, który odpowiada za sprawdzenie poprawności operacji przedłużenia terminu ważności karnetu.

---

```
1  [ Test ]
2  public void Client_SecondGymPassExtendEndDate ()
3  {
```

---

```

4     var gymPass = CreateTestGymPass();
5     var endDate = DateTime.Now;
6     endDate = endDate.AddDays(20);
7
8     _client.ExtendPass(gymPass);
9     _client.ExtendPass(gymPass);
10
11    var compareResult = _client.Subscription.EndDate.
        DayOfYear - endDate.DayOfYear;
12    Assert.Zero(compareResult, "Another_extensions_should_
        add_days_to_subscription_duration");
13 }

```

---

Kod 6.1. Przykładowy test klasy *ClientUnitTest*

Poza testami obiektów domenowych, napisano również testy, których zadaniem była kontrola przestrzegania konwencji jakie zostały narzucone w projekcie. Przyjęte konwencje są istotne ze względu na kontener wstrzykiwania zależności, który został użyty. Instancje niektórych klas powinny zostać dodane do kontenera w celu późniejszego wstrzyknięcia ich, aby tego dokonać system musi rozróżniać, które instancje klas powinny się w tym kontenerze znajdować. Rozwiązaniem tego problemu było dodanie atrybutów, które pełniły rolę znaczników klas, których instancje powinny się w takim kontenerze znajdować. Aby programista nie musiał poświęcać czasu na przeszukiwania projektu w celu odnalezienia klas, które takiego atrybutu nie mają, stworzono klasę *ConventionUnitTest*, gdzie znajduje się kod testów jednostkowych odpowiedzialnych za kontrolę przyjętych konwencji. Na Kod 6.2 przedstawiono test odpowiedzialny za sprawdzenie czy klasy repozytorium posiadają atrybut *RepositoryAttribute*.

---

```

1  [ Test ]
2  public void
        Repositories_AreAnnotatedByAttribute_Repository()
3  {
4      var classEndingInRepository = from type in

```

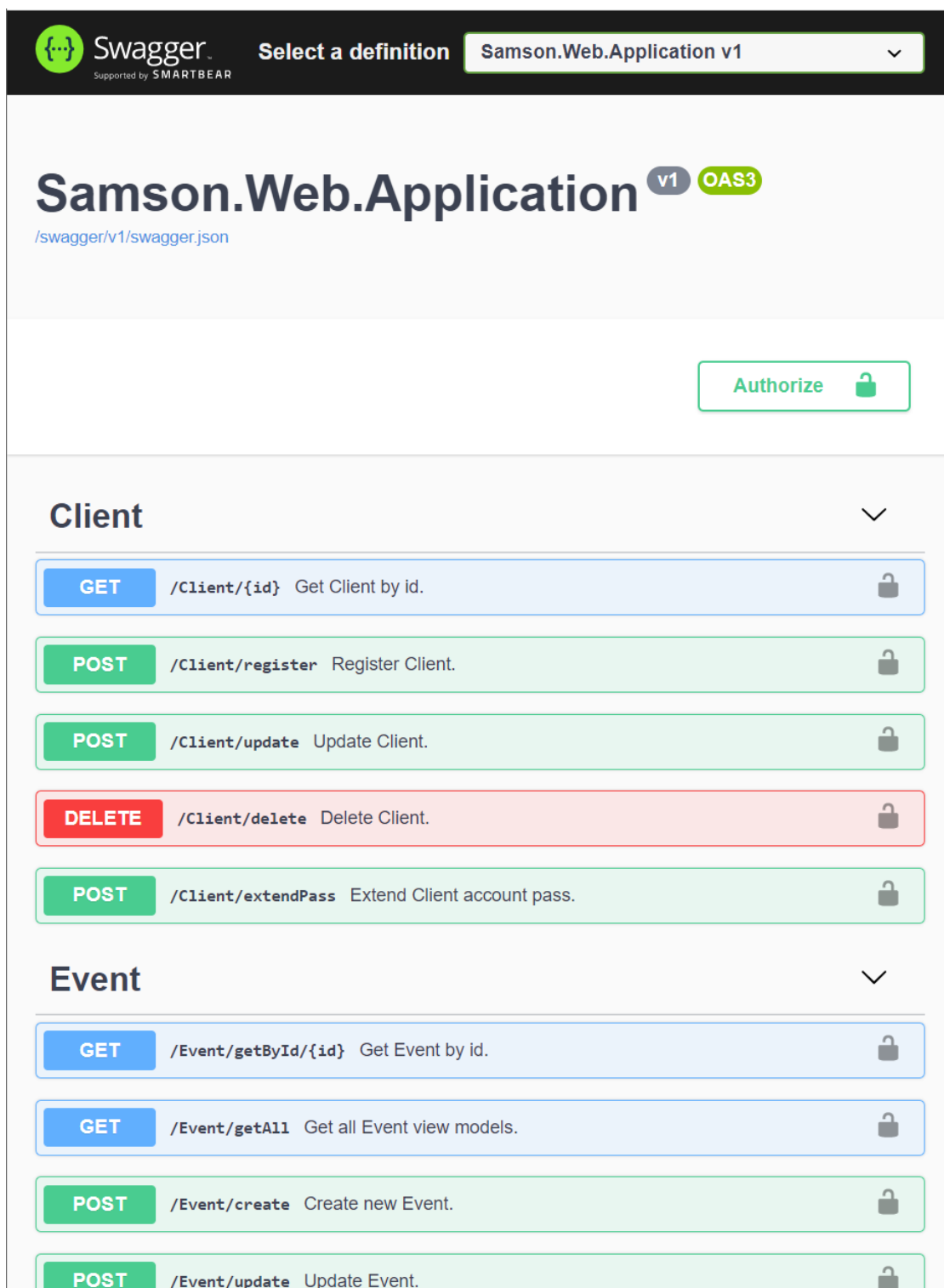
```
    _persistenceTypes
5   where !type.FullName.IsNullOrEmpty() && !type.
      IsInterface
6   where type.FullName.EndsWith("Repository")
7   select type;
8
9   foreach (var repository in classEndingInRepository)
10  {
11      var hasRepositoryAttribute = repository.IsDefined(
          typeof(RepositoryAttribute), false);
12      Assert.True(hasRepositoryAttribute, $"Repository {has_
          to_be_annotated_by_RepositoryAttribute: {
          repository.Name}");
13  }
14  }
```

Kod 6.2. Przykładowy test klasy *ConventionUnitTest*

W celu przetestowania poprawności działania warstwy serwerowej wykorzystano również testy manualne. Projekt *ASP.NET* posiada domyślnie włączone, automatyczne generowanie interaktywnej dokumentacji *Swagger*, która pozwala na manualne testowanie interfejsu *REST API*. Część wygenerowanej dokumentacji przedstawiono na rysunku 6.1. Przy pomocy wcześniej wspominanej dokumentacji przetestowano poprawność działania końcówek (ang. *endpoint*) interfejsu *REST API*. Natomiast w celu całościowego sprawdzenia poprawności działania systemu wykonano szereg testów manualnych przy wykorzystaniu warstwy prezentacji. System był testowany według scenariuszy, które odpowiadają funkcjonalnościom przedstawionym w podrozdziale **Przykład działania**.

## 6.2 Organizacja testów

Podczas pracy nad systemem równolegle tworzone kod testów jednostkowych, aby już w trakcie implementacji funkcjonalności wykrywać błędy w kodzie. Część błędów została wykryta już na etapie uruchamiania kodu testów jednostkowych.

Rysunek 6.1. Interaktywna dokumentacja interfejsu *REST API*



W takim przypadku poprawiono kod implementujący daną funkcjonalność w celu usunięcia błędu. Inne błędy zostały wykryte w trakcie testów manualnych. W takim przypadku potrzebna była dodatkowa analiza kodu aplikacji. Po zidentyfikowaniu błędu, zostawał on usuwany, a następnie w celu uniknięcia przyszłych nieprawidłowości dopisywano test jednostkowych, którego zadaniem było dopilnowanie by taka sytuacja się nie powtórzyła.

## 6.3 Przykłady wykrytych i usuniętych błędów

W trakcie pracy nad systemem wykryto, dzięki testom kilka błędów wynikających z niepoprawnej implementacji. Jednym z przykładów niepoprawnego działania, było usuwanie konta użytkownika zamiast jego archiwizacja. Gdy ten sam użytkownik był zapisany na jakiś trening lub wydarzenie, a konto zostało usunięte całkowicie z bazy to w danych pojawiły się relacje do nieistniejącego konta co prowadziło do licznych błędów.

Poprzedni błąd został wykryty przez testy manualne, natomiast udało się wykryć błąd przy pomocy testu jednostkowego. W systemie istnieje moduł, który odpowiada za kontrolę liczby osób, która znajduje się na siłowni. Maksymalna liczba osób była obliczana w niepoprawny sposób z powodu wykonywania dzielenia zamiast mnożenia. Poprawiona funkcja została zaprezentowana na Kod 6.3

---

```
1  public int CalcMaximumClientsCount ()
2  {
3      if ( CovidConfiguration == null)
4          return -1;
5
6      var availableGymObjectArea = GymObjectArea ();
7      /* (int) Math.Floor(availableGymObjectArea /
8          CovidConfiguration.PersonFactorPerMeter); */
9      return (int) Math.Floor(availableGymObjectArea *
          CovidConfiguration.PersonFactorPerMeter);
10 }
```

Kod 6.3. Metoda licząca jaka jest maksymalna liczba osób, która znajduje się w jednej chwili w obiekcie siłowni

Wykorzystanie interaktywnej dokumentacji *Swagger* pozwoliło na wykrycie błędów we wczesnym etapie tworzenia aplikacji. W systemie wykorzystano bibliotekę *AutoMapper*, która pozwala na mapowanie obiektów z jednej klasy na inną według zdefiniowanego schematu. Gdy taki schemat nie został zdefiniowany, a funkcja *Map* została wywołana to o błędzie użytkownik dowiadywał się dopiero gdy aplikacja została już uruchomiona. W takich przypadkach interaktywna dokumentacja pozwoliła na wykrywanie błędów i ich usuwanie jeszcze przed utworzeniem interfejsu użytkownika.

# Rozdział 7

## Podsumowanie i wnioski

Praca nad projektem przyniosła rezultaty w postaci systemu spełniającego określone wcześniej wymagania. Udało się pokryć zakres funkcjonalny związany z obostrzeniami pandemicznymi, co było główną przesłanką do rozpoczęcia prac nad nim. Wybór aplikacji internetowej pozwolił na utworzenie systemu dostępnego dla większości systemów operacyjnych. Poza spełnieniem wymagań funkcjonalnych, udało się również wykorzystać nowoczesne technologie, które zapewniają dłuższe wsparcie producentów.

W ramach projektu udało się stworzyć dokument, który opisuje sposób działania systemu oraz ideę, która za nim stała. W trakcie tworzenia systemu położono nacisk na architekturę, która pozwoli na skalowanie. Poza tym zwrócono również uwagę na odpowiednią dokumentację kodu, aby rozwiązanie mogło być rozwijane w kilkuosobowym zespole. Zaproponowana architektura pozwala również na zmianę architektury monolitycznej na mikro-usługową, co pozwoliłoby na pracę w większych zespołach.

Mimo, że rozwiązanie posiada wiele cech produktu komercyjnego, wymaga ono dopracowania w warstwie prezentacji. Niektóre z widoków powinny zostać zaprojektowane na nowo, tak aby były bardziej intuicyjne dla użytkownika. Poza poprawieniem przejrzystości formularzy, kolejnym krokiem w rozwoju systemu powinno być dostosowanie aplikacji do zasad RWD. Jest to istotne ze względu na duży udział urządzeń mobilnych w generowaniu ruchu na stronach internetowych[15].

Na rynku komercyjnym znajduje się wiele innych rozwiązań, które oferują więk-

sze możliwości pod względem funkcjonalnym. Naturalnym krokiem byłoby dodanie modułu recepcji oraz barów z odżywkami dla klientów. Poza nowymi modułami rozszerzone powinny być również te istniejące. Moduł karnetów powinien zostać rozszerzony o integrację z systemem płatności. Na rynku znajduje się wielu pośredników, którzy udostępniają API pozwalające na płatność w wygodny dla klienta sposób.

W celu komercjalizacji systemu ważne jest dodanie ról dla pozostały pracowników sieci siłowni. Konto trenera personalnego powinno mieć inne uprawnienia od konta administratora systemu. Natomiast w razie dodania modułu recepcji istotne byłoby dodanie roli recepcjonisty, który nie potrzebowałby dostępu do modułu treningów indywidualnych.

# Bibliografia

- [1] Advantages of MongoDB. <https://www.mongodb.com/advantages-of-mongodb>.
- [2] Angular: The modern web developer's platform. <https://angular.io/>.
- [3] Data Model Design. <https://docs.mongodb.com/manual/core/data-model-design/#std-label-data-modeling-referencing>.
- [4] Funkcjonalności, które ułatwią zarządzanie Twoim klubem. <https://wod.guru/pl/funkcjonalnosci>.
- [5] Install .NET on Linux. <https://docs.microsoft.com/en-us/dotnet/core/install/linux>.
- [6] Install .NET on macOS. <https://docs.microsoft.com/en-us/dotnet/core/install/macos>.
- [7] Install .NET on Windows. <https://docs.microsoft.com/en-us/dotnet/core/install/windows?tabs=net50>.
- [8] Instance Per Lifetime Scope. <https://autofac.readthedocs.io/en/latest/lifetime/instance-scope.html#instance-per-matching-lifetime-scope>.
- [9] .NET Framework system requirements. <https://docs.microsoft.com/en-us/dotnet/framework/get-started/system-requirements>.
- [10] Oprogramowanie CRM: Maksymalizuj potencjał zdobywania klientów dzięki narzędziom do pozyskiwania leadów i zmieniaj kontakty w płacących klientów. <https://www.perfectgym.com/pl/features/gym-crm>.

- 
- [11] Production Notes. <https://docs.mongodb.com/manual/administration/production-notes/>.
  - [12] Joseph Albahari, Ben Albahari. *C# 7.0 w pigułce*. Helion SA, ul. Kościuszki 1c, 44-100 Gliwice, 2018.
  - [13] Benjamin Anderson, Brad Nicholson. SQL vs. NoSQL Databases: What's the Difference? <https://www.ibm.com/cloud/blog/sql-vs-nosql>.
  - [14] Shannon Bradshaw, Eoin Brazil, Kristina Chodorow. *Przewodnik po MongoDB: Wydajna i skalowalna baza danych*. Helion SA, ul. Kościuszki 1c, 44-100 Gliwice, 2021.
  - [15] Eric Enge. Mobile vs. Desktop Usage in 2020. <https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage>.
  - [16] Dino Esposito. *Programowanie w ASP.NET Core*. APN Promise SA, ul. Domaniewska 44a, 02-672 Warszawa, 2017.
  - [17] Richard Lander. Introducing .NET 5. <https://devblogs.microsoft.com/dotnet/introducing-net-5/>.
  - [18] Nathan Rozentals. *Język TypeScript: Tajniki kodu*. Helion SA, ul. Kościuszki 1c, 44-100 Gliwice, 2018.

# Dodatki





# Spis skrótów i symboli

API aplikacyjny interfejs programistyczny (ang. *Application programming interface*)

CRUD utwórz, odczytaj, aktualizuj, usuń - (ang. *Create, Update, Read, Delete*)

CQRS (ang. *Command and Query Responsibility Segregation*)

DTO obiekt transferu danych (ang. *Data transfer object*)

HTTP (ang. *Hypertext Transfer Protocol*)

HTTPS (ang. *Hypertext Transfer Protocol Secure*)

IDE zintegrowane środowisko programistyczne (ang. *Integrated development environment*)

JS (ang. *JavaScript*)

JSON (ang. *JavaScript Object Notation*)

JWT (ang. *JSON Web Token*)

MVC model – widok – kontroler (ang. *model-view-controller*)

NoSQL nierelacyjna baza danych

LINQ (ang. *Language-Integrated Query*)

REST (ang. *Representational state transfer*)

RWD (ang. *Responsive web design*)

SDK zestaw narzędzi dla programistów (ang. *Software development kit*)

SPA jednostronicowa aplikacja internetowa (ang. *Single page application*)

SQL (ang. *Structured Query Language*)

TS (ang. *TypeScript*)

WWW światowa rozległa sieć komputerowa (ang. *World Wide Web*)

# Źródła

---

```
1  public abstract class MongoRepository<TModel, TEntity> :  
    IRepository<TModel> where TModel : IAggregate where  
        TEntity : IEntity  
2  {  
3      protected readonly IMongoCollection<TEntity> Collection  
        ;  
4      protected readonly IMapper Mapper;  
5  
6      public MongoRepository(IDatabaseConfiguration  
        databaseConfiguration , IMapper mapper)  
7      {...}  
8  
9      public MongoRepository(IDatabaseConfiguration  
        databaseConfiguration , IMapper mapper, string  
        collectionName)  
10     {...}  
11  
12     public TModel Get(ObjectId id)  
13     {...}  
14  
15     public List<TModel> Get()  
16     {...}  
17  
18     public Task<ObjectId> Create(TModel model)
```

```
19     {...}
20
21     public Task<ObjectId> Update( ObjectId id , TModel
        updatedModel)
22     {...}
23
24     public Task<ObjectId> Remove(TModel modelToDelete)
25     {...}
26
27     public Task<ObjectId> Remove( ObjectId id ) => ...
28 }
```

---

Kod 1. Klasa *MongoRepository*

# Spis załączników elektronicznych

Do pracy dołączona jest następującą zawartością:

- praca (źródła L<sup>A</sup>T<sub>E</sub>Xowe i końcowa wersja w pdf),
- źródła programu,
- dokumentacja kodu warstwy serwerowej.



# Spis rysunków

3.1	Diagram przypadków użycia . . . . .	14
4.1	Strona, z której można pobrać instalator serwera bazy danych MongoDB . . . . .	21
4.2	Widok powitalny instalatora MongoDB . . . . .	21
4.3	Umowa licencyjna <i>MongoDB Community Edition</i> . . . . .	22
4.4	Typ instalacji w instalatorze <i>MongoDB Community Edition</i> . . . . .	22
4.5	Początkowa konfiguracja serwera bazy danych . . . . .	23
4.6	Informacja o możliwości instalacji klienta bazy danych <i>MongoDB Compass</i> . . . . .	23
4.7	Rozpoczęcie instalacji <i>MongoDB</i> . . . . .	24
4.8	Zakończenie instalacji <i>MongoDB</i> . . . . .	24
4.9	Strona twórców środowiska <i>.NET</i> . . . . .	25
4.10	Rozpoczęcie instalacji środowiska <i>.NET 5</i> . . . . .	26
4.11	Zakończenie instalacji środowiska <i>.NET 5</i> . . . . .	26
4.12	Strona Node.js . . . . .	27
4.13	Rozpoczęcie instalacji środowiska <i>Node.js</i> . . . . .	28
4.14	Licencja środowiska <i>Node.js</i> . . . . .	28
4.15	Ścieżka do folderu docelowego <i>Node.js</i> . . . . .	29
4.16	Składniki instalatora Node.js . . . . .	29
4.17	Formularz umożliwiający instalację dodatkowego narzędzia <i>Cocolatey</i> . . . . .	30
4.18	Potwierdzenie instalacji środowiska <i>Node.js</i> . . . . .	30
4.19	Rejestracja klienta . . . . .	35
4.20	Klient został zarejestrowany w systemie . . . . .	36
4.21	Widok logowania do systemu . . . . .	36

4.22	Widok po zalogowaniu do systemu . . . . .	37
4.23	Formularz tworzenia konta trenera personalnego . . . . .	38
4.24	Formularz tworzenia obiektu siłowni . . . . .	38
4.25	Formularz edycji i archiwizacji obiektów siłowni . . . . .	39
4.26	Formularz dodawania pomieszczenia do obiektu siłowni . . . . .	40
4.27	Formularz archiwizacji pomieszczenia . . . . .	40
4.28	Formularz dodawania sprzętu do ćwiczeń . . . . .	41
4.29	Formularz edycji i archiwizacji akcesoriów . . . . .	42
4.30	Formularz dodawania karnetu do oferty . . . . .	43
4.31	Formularz edycji i archiwizacji karnetów . . . . .	43
4.32	Formularz tworzenia wydarzenia . . . . .	44
4.33	Lista wszystkich wydarzeń . . . . .	45
4.34	Lista utworzonych wydarzeń przez trenera personalnego . . . . .	45
4.35	Lista wydarzeń klienta . . . . .	46
4.36	Podgląd wydarzenia jako trener personalny . . . . .	46
4.37	Pogląd wydarzenia jako klient . . . . .	47
4.38	Formularz dodawania treningu . . . . .	48
4.39	Lista dostępnych treningów indywidualnych . . . . .	49
4.40	Lista treningów, na które zapisał się klient . . . . .	49
4.41	Lista utworzonych treningów przez trenera . . . . .	50
4.42	Oferta karnetów oraz informacje o posiadanym karnecie . . . . .	50
4.43	Widok wyświetlany na wyświetlaczu bramki wejściowej . . . . .	51
4.44	Widok wyświetlany na wyświetlaczu bramki wyjściowej . . . . .	52
5.1	Architektura systemu . . . . .	54
5.2	Poglądowy schemat bazy danych . . . . .	56
6.1	Interaktywna dokumentacja interfejsu <i>REST API</i> . . . . .	68



# Spis źródeł

4.1	Instalacja paczki <i>http-server</i> . . . . .	31
4.2	Kompilacja projektu <i>ASP.NET</i> . . . . .	31
4.3	Przejdźcie do folderu zawierającego plik wykonywalny . . . . .	32
4.4	Ustawienie wartości zmiennych środowiskowych . . . . .	32
4.5	Uruchomienie warstwy serwerowej aplikacji . . . . .	33
4.6	Polecenie, które tworzy paczkę możliwą do uruchomienia przez serwer WWW . . . . .	33
4.7	Polecenie, które tworzy paczkę możliwą do uruchomienia przez serwer WWW . . . . .	33
5.1	Metoda odczytu danych z kolekcji <i>ExerciseMachineCollection</i> przy wykorzystaniu wstawek <i>JavaScript</i> . . . . .	58
5.2	Metoda odczytu danych z kolekcji <i>GymObjectCollection</i> przy wykorzystaniu wstawek składni <i>LINQ</i> . . . . .	59
5.3	Metoda odczytu informacji o koncie trenera personalnego przy wykorzystaniu metod sterownika . . . . .	60
5.4	Metoda <i>Handle</i> klasy <i>CreateGymPassTypeCommandHandler</i> . . . . .	61
5.5	Klasa <i>GymPassTypesQueryHandler</i> . . . . .	62
5.6	Metoda <i>GetById</i> klasy <i>EventController</i> . . . . .	62
6.1	Przykładowy test klasy <i>ClientUnitTest</i> . . . . .	65
6.2	Przykładowy test klasy <i>ConventionUnitTest</i> . . . . .	66
6.3	Metoda licząca jaka jest maksymalna liczba osób, która znajduje się w jednej chwili w obiekcie siłowni . . . . .	69