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

konsultant: dr inż. Imię Nazwisko

Gliwice, grudzień 2021

Spis treści

Streszczenie

Streszczenie pracy -odpowiednie pole w systemie APD powinno zawierac kopie tego streszczenia. Streszczenie, wraz ze slowami kluczowymi, nie powinno przekroczyc jednej strony.

Slowa kluczowe: 2-5 slow (fraz) kluczowych, oddzielonych przecinkami

Rozdział 1

Wstęp

- wprowadzenie w problem/zagadnienie
- osadzenie problemu w dziedzinie
- cel pracy
- zakres pracy
- zwięzła charakterystyka rozdziałów
- jednoznaczne określenie wkładu autora, w przypadku prac wieloosobowych – tabela z autorstwem poszczególnych elementów pracy

Rozdział 2

Analiza tematu

Na rynku siłowni widać coraz większą konkurencję, a właściciele szukają sposobów jak zachęcić nowych klientów do uczęszczania na ich siłownie. Wiele z sieci pozwala na całodobowe korzystanie z ich obiektów, aby sprostać wymaganiom jak największej grupie klientów. Stawia to nowe wyzwania w organizacji pracy pracowników i sposobie działania obiektów. Dużym usprawnieniem byłaby możliwość zrezygnowania z recepcji na rzecz bramek wejściowych i wyjściowych, które kontrolowałby ważność karnetu oraz pilnowałby, 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 aplikacje internetową może zwiększyć atrakcyjność oferty danej sieci siłowni w obliczu rosnącej konkurencji na rynku sieci siłowni.

Sieć siłowni staje w obliczu innych wyzwań, które nie są znane małym sieciom siłowni lub 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 to 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.

Ważnym 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 ilość 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)[11]. 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.

Aktualnie w sieci Internet możemy znaleźć wiele dostępnych rozwiązań [8] [2], które pokrywają wymagania funkcjonalne systemu. Duża część oferowanych aplikacji posiada możliwość zarządzania bazą klientów, prowadzenie inwentarza wyposażenia 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 podstawowe funkcjonalności potrzebne w prowadzeniu działalności sieci siłowni. W celu uniknięcia problemów z wydajnością, a tym samym złym doświadczeniom płynącym z korzystania z serwisu internetowego, system musi być podatny na zmiany. Aplikacja powinna cechować się skalowalnością w celu zapewnienia szybkości 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 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

3.1 Wymagania funkcjonalne

Wymienione punkty stanowią listę wszystkich funkcjonalności jakie powinna oferować aplikacja. Opisują one jakie funkcje zostały udostępnione 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 siłowni.

- Rejestracja konta klienta - system powinien pozwalać na samodzielne zarejestrowanie się użytkownika w celu skorzystania z funkcjonalności serwisu internetowego sieci siłowni.

- Logowanie się na wcześniej utworzone konto użytkownika - w celu weryfikacji użytkownika, system powinien pozwalać na autoryzację poprzez formularz logowania.

- Przeglądanie dostępnej oferty karnetów na siłownie - 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 siłowni.
- 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 dostępny 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 zdeklarowania chęci udziału w wydarzeniu.
- Rezygnacja z udziału w wydarzeniu - klient powinien mieć możliwość zrezygnowania z udziału w wydarzeniu, w którym wcześniej zdeklarował 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.

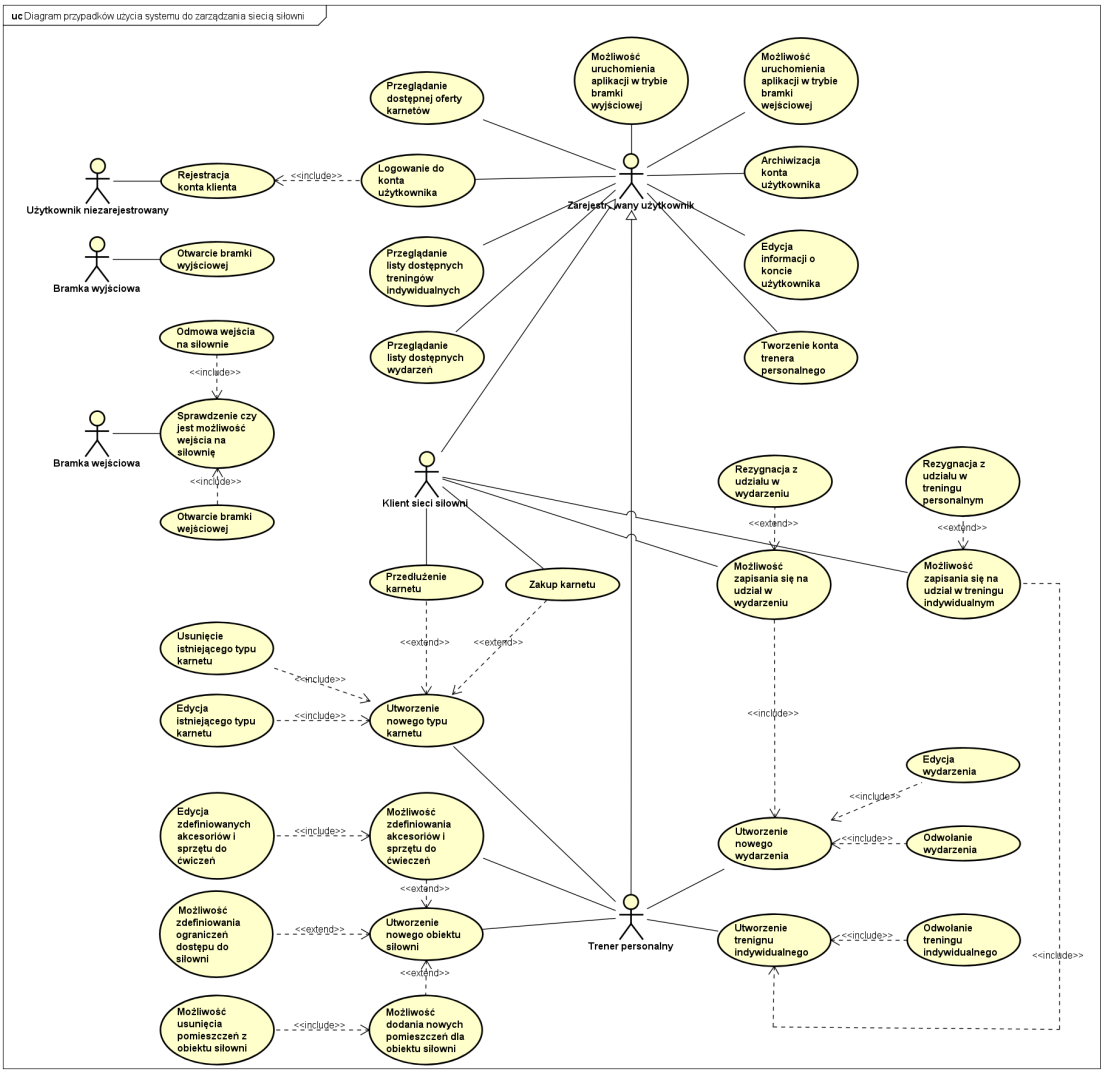
- Utworzenie treningu indywidualnego - trener personalny powinien mieć możliwość utworzenia sesji treningów indywidualnych.
- Odwołanie treningu indywidualnego - trener personalny powinien mieć możliwość odwołania wcześniej zaplanowanych treningów personalnych.
- Rezygnacja z udziału w treningu personalnym - klient powinien mieć możliwość zrezygnowania z udziału w treningu personalnym.
- Odmowa wejścia na siłownie - bramka wejściowa powinna mieć możliwość odmówienia wejścia klientowi na siłownie w przypadku braku ważnego karnetu 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 niefunkcjonalne

- 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 stu 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 przedstawienie interakcji użytkowników z aplikacją i przedstawienie wymogów funkcjonalnych systemu w formie diagramów.



Rysunek 3.1: Diagram przypadków użycia.

3.4 Narzędzia

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 [10].

Platformą programistyczną warstwy serwerowej został .NET 5, który 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 [13].

Do stworzenia interfejsu REST API została wykorzystana biblioteka ASP.NET Core, która pozwala na dodawanie metod HTTP, 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[12].

W celu wykorzystania wzorca projektowego CQRS zamiast domyślnego kontenera zależności wykorzystano zewnętrzną bibliotekę Autofac, która udostępnia możliwość wstrzykiwania obiektów oznaczonych atrybutami oraz określanie cyklu życia serwisu przy pomocy znaczników[6].

Jako bazę danych wykorzystano MongoDB. MongoDB jest dokumentową bazą danych pozwalająca na przechowywanie danych w formacie JSON, który jest formatem czytelnym dla człowieka. Baza danych należy do nierelacyjnych baz danych NoSQL, której głównymi zaletami są: elastyczny schemat bazy, łatwe skalowanie wszerz oraz możliwość tworzenia zaawansowanych zapytań i raportów[11]. Opcja skalowania bazy wszerz 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 Java-

Script, 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[14].

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[1][14].

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 tworzenie stanów, które są deterministyczne.

3.5 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 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. *M*iddleware) oraz interfejsy wykorzystywane pomiędzy, różnymi projektami rozwiązania .NET.
- *Samson.Web.Application.Persistence* - projekt zawierający kod klas implementujących wzorzec repozytorium. Odpowiada za dodawania i modyfikacje modeli 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, enumy oraz domenę aplikacji

W następnych krokach implementowano aplikacje tworząc w pierwszej kolejności kontrolery platformy ASP.NET, a w krokach późniejszych dodawano szczegóły implementacji w postaci domeny, warstwy zapisu, warstwy odczytu, modeli. Najważniejsze klasy domenowe zostały uzupełnione testami jednostkowymi w celu zapewnienia poprawności działania. Gdy część serwerowa aplikacji została zakończona, 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

4.1 Wymagania sprzętowe

Aby aplikacja działała w poprawny sposób wymagany są:

- System operacyjny w wersji [5][4][3][9][7]:
  - *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 werjsach 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

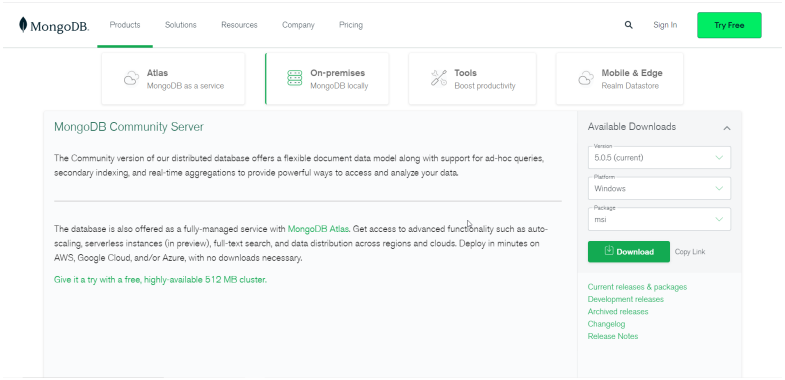
- Procesor z więcej niż jednym rdzeniem o taktowaniu nie mniejszym niż 1GHZ
- Pamięć operacyjna większa niż 4GB RAM
- Minimalna przestrzeń na dysku to 4,5 GB pamięci
- Zainstalowana nowoczesna przeglądarka WWW taka jak:
  - *Google Chrome* w najnowszej wersji
  - *Mozilla Firefox* w najnowszej wersji
  - *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

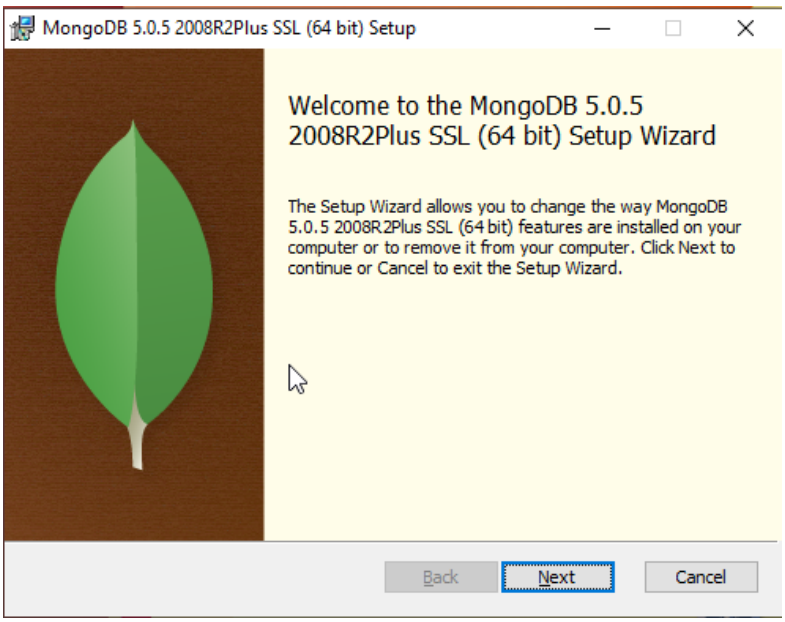
4.2.1 Instalacja bazy danych MongoDB

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.



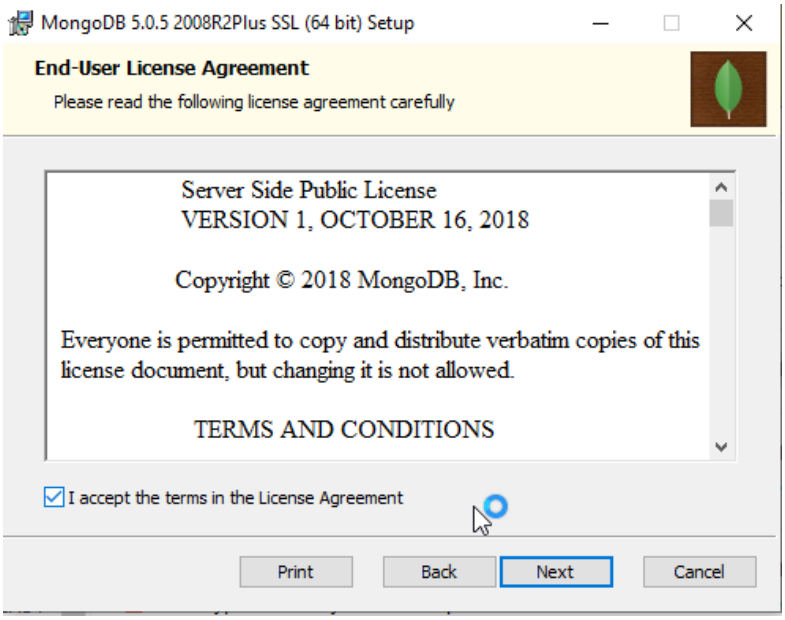
Rysunek 4.1: Strona z której można pobrać instalator serwera bazy danych MongoDB

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



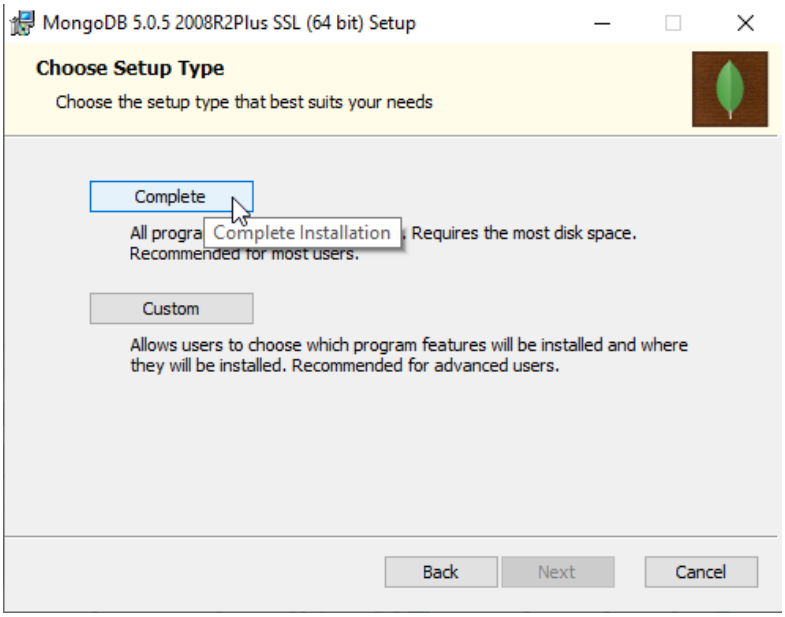
Rysunek 4.2: Rozpoczęcie instalacji serwera

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*.



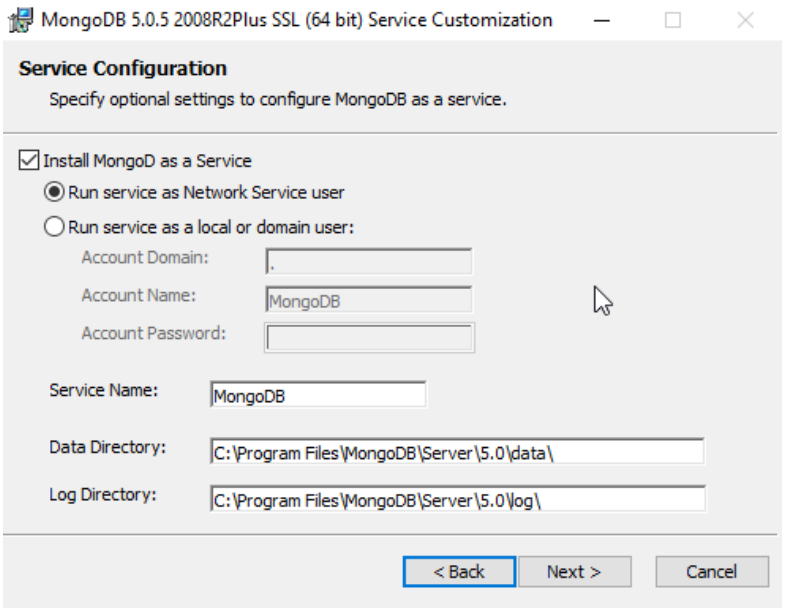
Rysunek 4.3: Umowa licencyjna *MongoDB Community Edition*

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



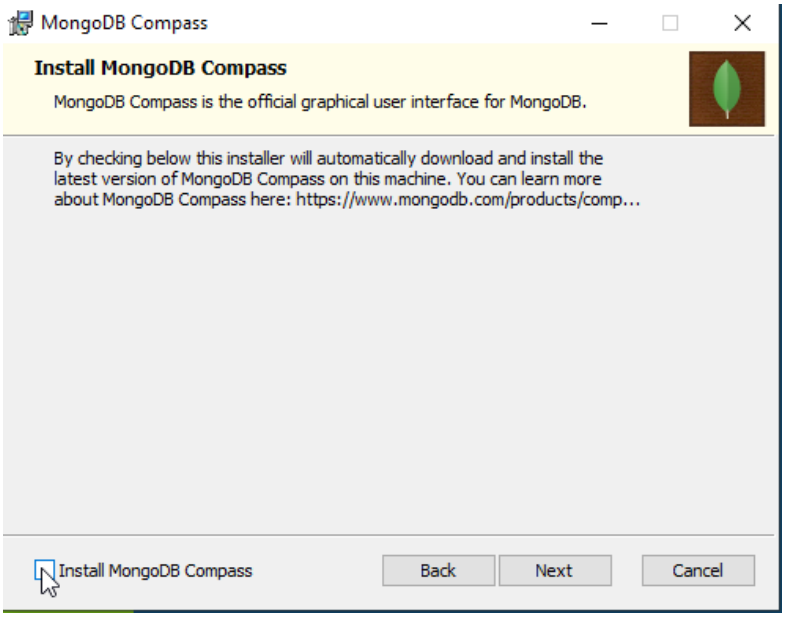
Rysunek 4.4: Typ instalacji w instalatorze *MongoDB Community Edition*

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



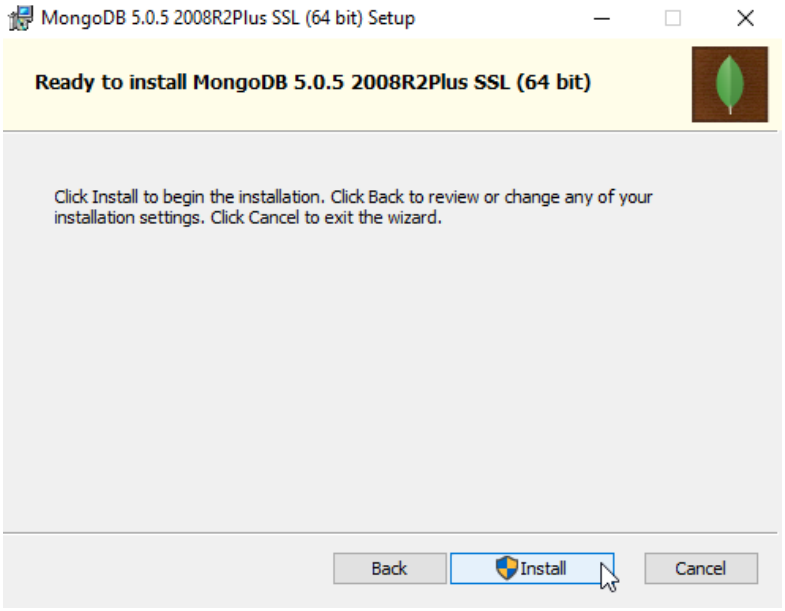
Rysunek 4.5: Początkowa konfiguracja serwera bazy danych

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



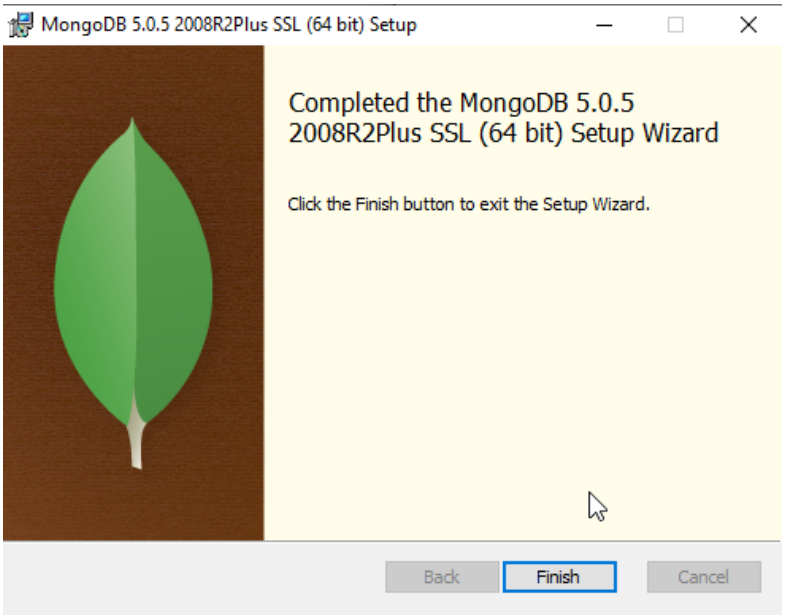
Rysunek 4.6: Informacja o możliwości instalacji klienta bazy danych *MongoDB Compass*

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



Rysunek 4.7: Rozpoczęcie instalacji *MongoDB*

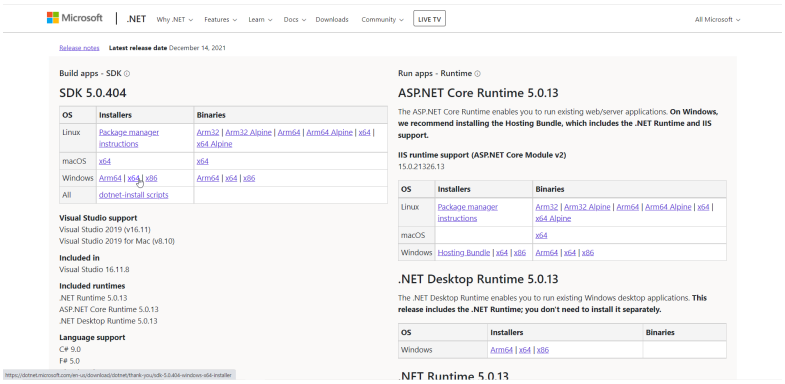
Aby zakończyć instalację po zakończeniu się procesu instalacji trzeba przycisnąć przycisk *Finish*. W tym momencie na maszynie użytkownika powinna znajdować się w pełni sprawny serwer bazy danych MongoDB.



Rysunek 4.8: Zakończenie instalacji *MongoDB*

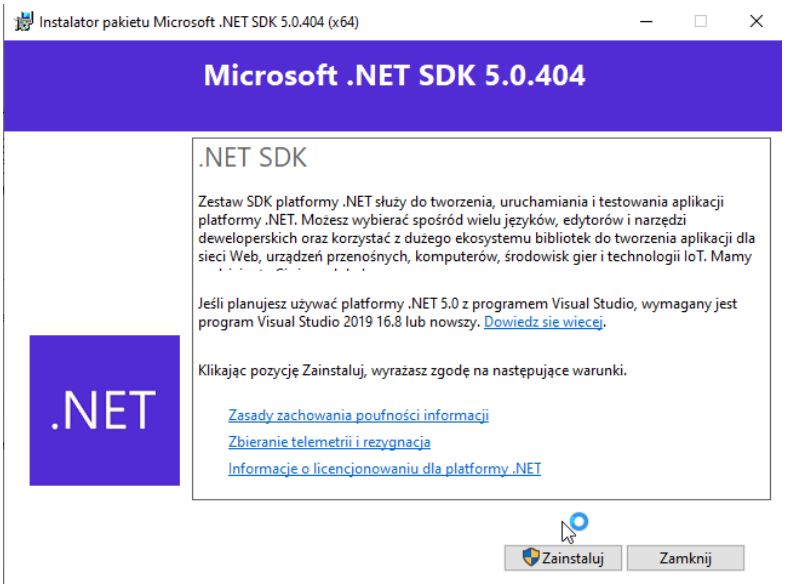
4.2.2 Instalacja środowiska .NET 5

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.



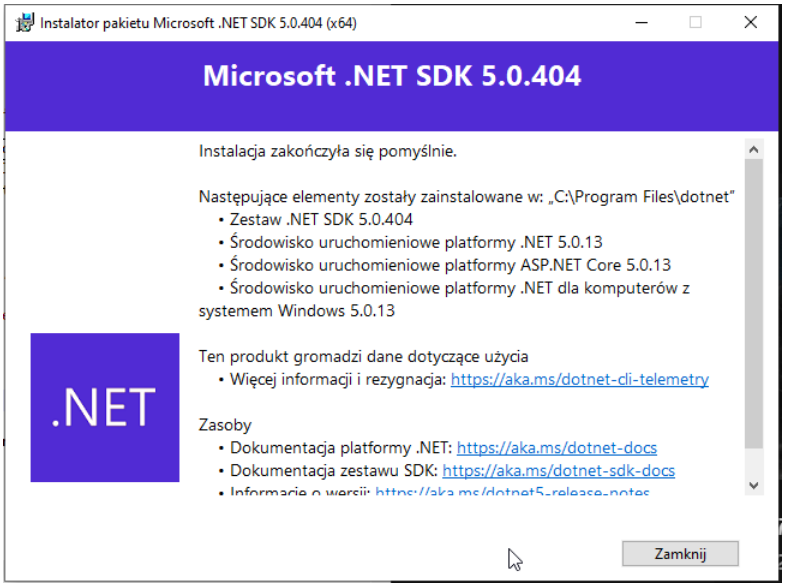
Rysunek 4.9: Strona twórców środowiska *.NET*

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 informacji trzeba przycisnąć przycisk *Zainstaluj* w celu instalacji środowiska uruchomieniowego.



Rysunek 4.10: Rozpoczęcie instalacji środowiska *.NET* 5

W kolejnym kroku kreator zainstaluje na maszynie wymagane składniki. Kiedy instalator zakończy działanie zamknięcie kreatora wymaga przyciśnięcia w przycisk *Zamknij*.



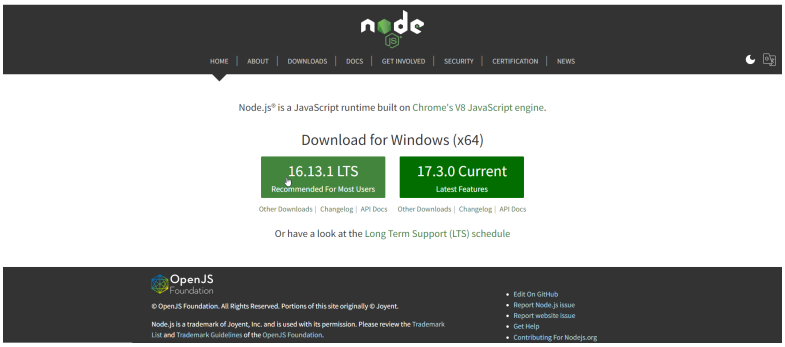
Rysunek 4.11: Zakończenie instalacji środowiska *.NET 5*

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

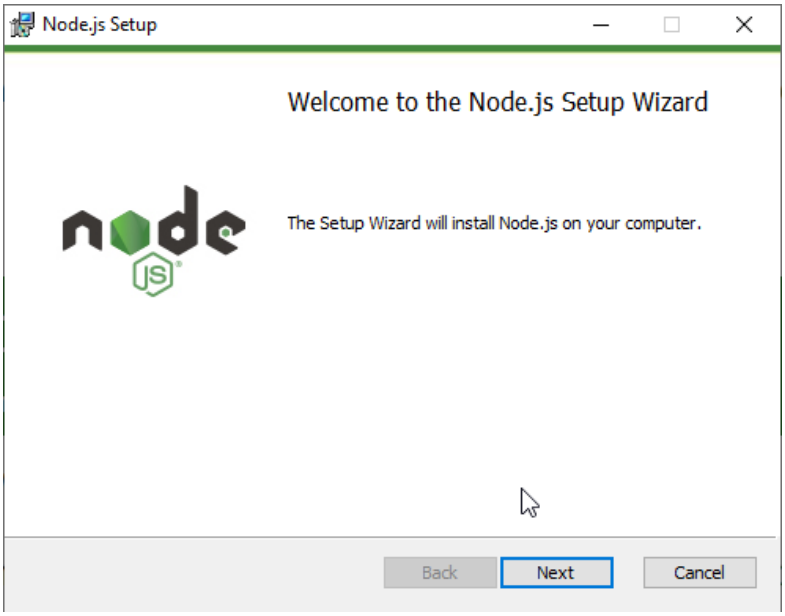
Przedostatnim krokiem, jaki trzeba wykonać, aby zainstalować projekt jest instalacja środowiska, które pozwoli na uruchomienie serwera WWW dzięki, któremu będzie można uruchomić aplikację stworzoną przy pomocy platformy programistycznej Angular. Serwer WWW musi mieć możliwość przekierowania wszystkich zapytań do wybranego adresu URL. Jest to istotne w przypadku aplikacji typu SPA, gdzie to aplikacja odpowiada za nawigacje 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 przejśc w miejsce gdzie pobrano plik instalatora a następnie uruchomić go. Wyświetlony powinien zostać następujący widok.

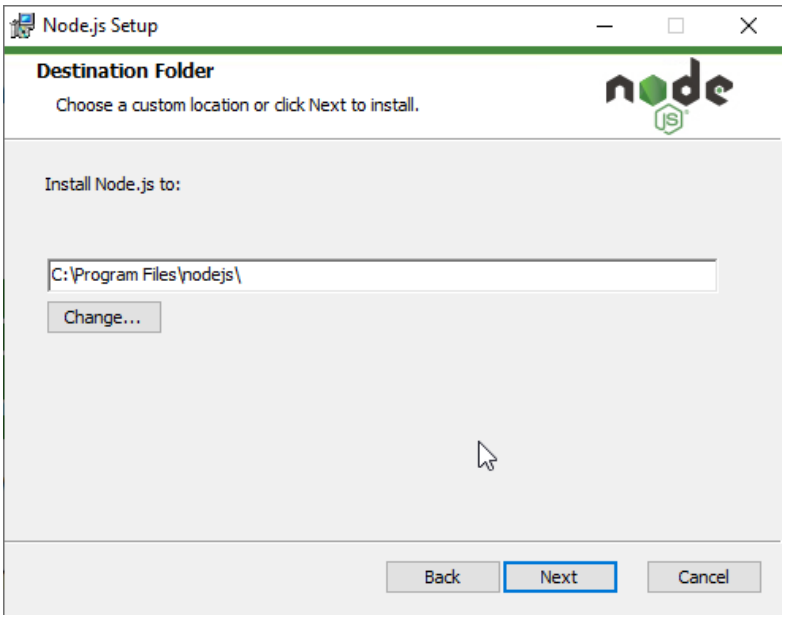


Rysunek 4.13: Rozpoczęcie instalacji środowiska *Node.js*

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

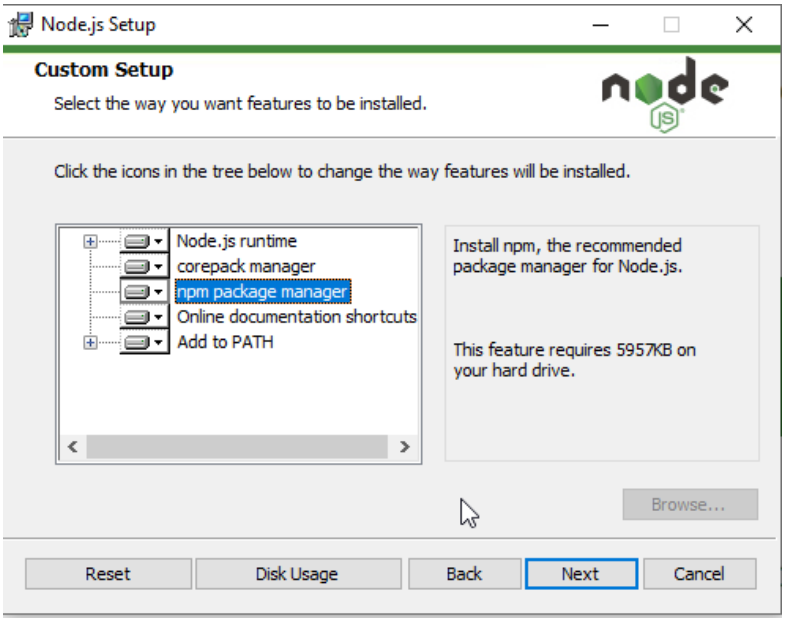
Rysunek 4.14: Licencja środowiska *Node.js*

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



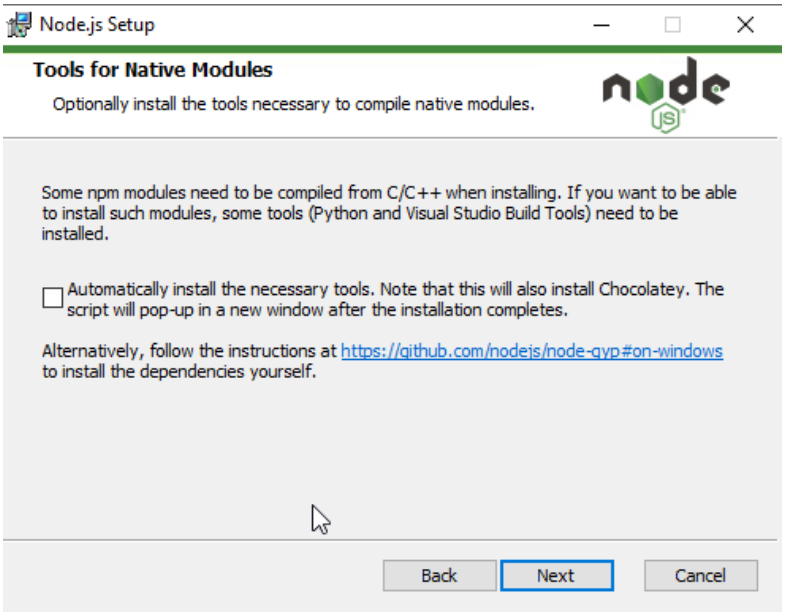
Rysunek 4.15: Ścieżka do folderu docelowego *Node.js*

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 instalacje serwera WWW.



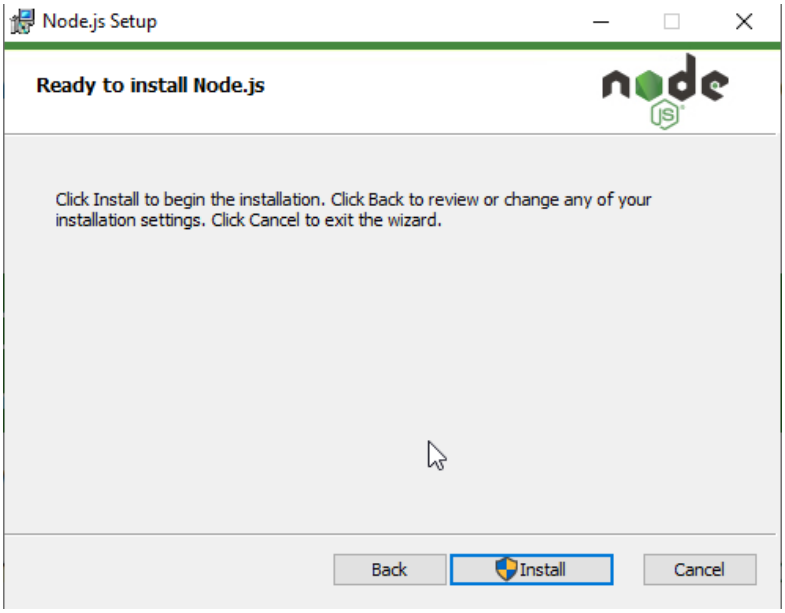
Rysunek 4.16: Składniki instalatora Node.js

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



Rysunek 4.17: Formularz umożliwający instalację dodatkowego narzędzia *Cocolatey*

W kolejnych etapach instalacji użytkownik powinien uruchomić instalacje 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*.

4.3 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 następujące polecenie:

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

Rysunek 4.19: Instalacja paczki *http-server*

Polecenie to zainstaluje paczkę *npm* dla użytku globalnego. Dzięki instalacji globalnej użytkownik będzie mógł uruchomić serwer WWW z każdej lokalizacji.

4.4 Sposób aktywacji

4.4.1 Uruchomienie warstwy serwerowej

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

```
1 dotnet build --configuration Release
```

Rysunek 4.20: Kompilacja projektu *ASP.NET*

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/Relea przy pomocy komendy:

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

Rysunek 4.21: Przejście do folderu zawierającego plik wykonywalny

Aby uruchomić warstwę serwerową aplikacji wymagane jest dodanie odpowiednich zmiennych środowiskowych przy pomocy poniższego polecenia. Przeznaczenie zmiennych jest następujące:

- *MongoDB:DatabaseName* - nazwa bazy danych jaka zostanie utworzona w środowisku *MongoDB*.
- *Authentication:JWT:Key* - klucz wykorzystywany do funkcji mieszające, które jest wykorzystywana do zabezpieczenia żetonu *JWT*.
- *ĄSPNETCORE*$_E$*NVIRONMENT"*$-$*oznaczenietypuśrodowiska.Niepowinnazostaćzmieniana.Conn ciągznakówokreślającyparametrypołączeniazbaządanych.JeżeliwtrakcieinstalacjiserweraMongoDB* Poniższe wartości mogą zostać zmodyfikowane, ale zalecane jest aby pozostały w

niezmienione formie ze względu na proponowaną w wcześniejszych krokach konfigurację.

```
$env:MongoDB:DatabaseName="SamsonDatabase";
$env:Authentication:JWT:Key="Top secret key to provide JWT token";
$env:ASPNETCORE_ENVIRONMENT="Release"
$env:ConnectionString:MongoDB:Atlas="mongodb://localhost:27017/?readPreference=primary&ssl=false";
```

Rysunek 4.22: Ustawienie wartości zmiennych środowiskowych

Ostatnim krokiem w uruchomieniu warstwy serwerowej jest uruchomienie poniższego polecenia.

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

Rysunek 4.23: Uruchomienie warstwy serwerowej aplikacji

4.4.2 Uruchomienie warstwy prezentacji

Tak jak w przypadku warstwy serwerowej, warstwa prezentacji musi zostać skompilowana do postaci, która może zostać obsłużona 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:

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

Rysunek 4.24: 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.

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

Rysunek 4.25: Polecenie, które tworzy paczkę możliwą do uruchomienia przez serwer WWW

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

4.5 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, obiektów, pomieszczeń oraz możliwość organizacji wydarzeń i treningów indywidualnych.
- Klientów - użytkownik który posiadają możliwość zakupu karnetu, zgłoszenia chęci udziału w treningu indywidualnym lub wydarzeniu.

4.6 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 dostępach, 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.

4.7 Przykład działania

4.8 Scenariusze korzystania z systemu

Rozdział 5

Specyfikacja wewnętrzna

Jeśli to Specyfikacja wewnętrzna:

- przedstawienie idei
- architektura systemu
- opis struktur danych (i organizacji baz danych)
- komponenty, moduły, biblioteki, przegląd ważniejszych klas (jeśli występują)
- przegląd ważniejszych algorytmów (jeśli występują)
- szczegóły implementacji wybranych fragmentów, zastosowane wzorce projektowe
- diagramy UML

Krótka wstawka kodu w linii tekstu jest możliwa, np. **descriptor**, a nawet **descriptor_gaussian**. Dłuższe fragmenty lepiej jest umieszczać jako rysunek, np. kod na rysunku 5.1, a naprawdę długie fragmenty – w załączniku.

```
1  class descriptor_gaussian : virtual public descriptor
2  {
3      protected:
4          /** core of the gaussian fuzzy set */
5          double _mean;
6          /** fuzzyfication of the gaussian fuzzy set */
7          double _stddev;
8
9      public:
10         /** @param mean core of the set
11             @param stddev standard deviation */
12         descriptor_gaussian (double mean, double stddev);
13         descriptor_gaussian (const descriptor_gaussian & w);
14         virtual ~descriptor_gaussian();
15         virtual descriptor * clone () const;
16
17         /** The method elaborates membership to the gaussian
               fuzzy set. */
18         virtual double getMembership (double x) const;
19
20 };
```

Rysunek 5.1: Klasa **descriptor_gaussian**.

Rozdział 6

Weryfikacja i walidacja

- sposób testowania w ramach pracy (np. odniesienie do modelu V)
- organizacja eksperymentów
- przypadki testowe zakres testowania (pełny/niepełny)
- wykryte i usunięte błędy
- opcjonalnie wyniki badań eksperymentalnych

Tablica 6.1: Opis tabeli nad nią.

	metoda						
			alg. 3			alg. 4, $\gamma = 2$	
ζ	alg. 1	alg. 2	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$	$\beta = 0.1$	$\beta = -0.1$
0	8.3250	1.45305	7.5791	14.8517	20.0028	1.16396	1.1365
5	0.6111	2.27126	6.9952	13.8560	18.6064	1.18659	1.1630
10	11.6126	2.69218	6.2520	12.5202	16.8278	1.23180	1.2045
15	0.5665	2.95046	5.7753	11.4588	15.4837	1.25131	1.2614
20	15.8728	3.07225	5.3071	10.3935	13.8738	1.25307	1.2217
25	0.9791	3.19034	5.4575	9.9533	13.0721	1.27104	1.2640
30	2.0228	3.27474	5.7461	9.7164	12.2637	1.33404	1.3209
35	13.4210	3.36086	6.6735	10.0442	12.0270	1.35385	1.3059
40	13.2226	3.36420	7.7248	10.4495	12.0379	1.34919	1.2768
45	12.8445	3.47436	8.5539	10.8552	12.2773	1.42303	1.4362
50	12.9245	3.58228	9.2702	11.2183	12.3990	1.40922	1.3724

Rozdział 7

Podsumowanie i wnioski

- uzyskane wyniki w świetle postawionych celów i zdefiniowanych wyżej wymagań
- kierunki ewentualnych danych prac (rozbudowa funkcjonalna ...)
- problemy napotkane w trakcie pracy

Bibliografia

[1] Angular: The modern web developer's platform.

[2] Funkcjonalności, które ułatwią zarządzanie twoim klubem.

[3] Install .net on linux.

[4] Install .net on macos.

[5] Install .net on windows.

[6] Instance per lifetime scope.

[7] .net framework system requirements.

[8] Oprogramowanie crm: Maksymalizuj potencjał zdobywania klientów dzięki narzędziom do pozyskiwania leadów i zmieniaj kontakty w płacących klientów.

[9] Production notes.

[10] Joseph Albahari, Ben Albahari. *C# 7.0 w pigułce.* Helion SA, ul. Kościuszki 1c, 44-100 Gliwice, 2018.

[11] Shannon Bradshaw, Eoin Brazil, Kristina Chodorow. *Przewodnik po MongoDB: Wydajna i skalowalna baza danych.* Helion SA, ul. Kościuszki 1c, 44-100 Gliwice, 2021.

[12] Dino Esposito. *Programowanie w ASP.NET Core.* APN Promise SA, ul. Domaniewska 44a, 02-672 Warszawa, 2017.

[13] Richard Lander. Introducing .net 5.

[14] Nathan Rozentals. *Język TypeScript: Tajniki kodu.* Helion SA, ul. Kościuszki 1c, 44-100 Gliwice, 2018.

Dodatki

Spis skrótów i symboli

MVC	model – widok – kontroler (ang. *model–view–controller*)
SPA	jednostronicowa aplikacja internetowa (ang. *Single page application*)
CQRS	(ang. *Command and Query Responsibility Segregation*)
TS	(ang. *TypeScript]*)
JS	(ang. *JavaScript*)
REST	(ang. *Representational state transfer*)
API	aplikacyjny interfejs programistyczny (ang. *Application programming interface*)
JWT	(ang. *JSON Web Token*)
JSON	(ang. *JavaScript Object Notation*)
HTTP	(ang. *Hypertext Transfer Protocol*)
HTTPS	(ang. *Hypertext Transfer Protocol Secure*)
NoSQL	nierelacyjna baza danych
WWW	światowa rozległa sieć komputerowa (ang. *World Wide Web*)

Źródła

Jeżeli w pracy konieczne jest umieszczenie długich fragmentów kodu źródłowego, należy je przenieść do załącznika.

```
1  partition fcm_possibilistic::doPartition
2                                  (const dataset & ds)
3  {
4      try
5      {
6          if (_nClusters < 1)
7              throw std::string ("unknown number of clusters");
8          if (_nIterations < 1 and _epsilon < 0)
9              throw std::string ("You should set a maximal
                   number of iteration or minimal difference —
                   epsilon.");
10         if (_nIterations > 0 and _epsilon > 0)
11             throw std::string ("Both number of iterations and
                   minimal epsilon set — you should set either
                   number of iterations or minimal epsilon.");
12
13         auto mX = ds.getMatrix();
14         std::size_t nAttr = ds.getNumberOfAttributes();
15         std::size_t nX    = ds.getNumberOfData();
16         std::vector<std::vector<double>> mV;
17         mU = std::vector<std::vector<double>> (_nClusters);
18         for (auto & u : mU)
```

```
19          u = std::vector<double> (nX);
20      randomise(mU);
21      normaliseByColumns(mU);
22      calculateEtas(_nClusters, nX, ds);
23      if (_nIterations > 0)
24      {
25          for (int iter = 0; iter < _nIterations; iter++)
26          {
27              mV = calculateClusterCentres(mU, mX);
28              mU = modifyPartitionMatrix (mV, mX);
29          }
30      }
31      else if (_epsilon > 0)
32      {
33          double frob;
34          do
35          {
36              mV = calculateClusterCentres(mU, mX);
37              auto mUnew = modifyPartitionMatrix (mV, mX);
38
39              frob = Frobenius_norm_of_difference (mU, mUnew)
                    ;
40              mU = mUnew;
41          } while (frob > _epsilon);
42      }
43      mV = calculateClusterCentres(mU, mX);
44      std::vector<std::vector<double>> mS =
            calculateClusterFuzzification(mU, mV, mX);
45
46      partition part;
47      for (int c = 0; c < _nClusters; c++)
48      {
49          cluster cl;
```

```
50            for (std::size_t a = 0; a < nAttr; a++)
51            {
52               descriptor_gaussian d (mV[c][a], mS[c][a]);
53               cl.addDescriptor(d);
54            }
55            part.addCluster(cl);
56         }
57         return part;
58      }
59      catch (my_exception & ex)
60      {
61         throw my_exception (__FILE__, __FUNCTION__, __LINE__,
               ex.what());
62      }
63      catch (std::exception & ex)
64      {
65         throw my_exceptionn (__FILE__, __FUNCTION__, __LINE__
               , ex.what());
66      }
67      catch (std::string & ex)
68      {
69         throw my_exception (__FILE__, __FUNCTION__, __LINE__,
               ex);
70      }
71      catch (...)
72      {
73         throw my_exception (__FILE__, __FUNCTION__, __LINE__,
               "unknown expection");
74      }
75   }
```

Zawartość dołączonej płyty

Do pracy dołączona jest płyta CD z następującą zawartością:

- praca (źródła LaTeXowe i końcowa wersja w `pdf`),
- źródła programu,
- dane testowe.

Spis rysunków

Spis tablic