POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Piotr Jałowiecki
Nr albumu 16147

Aplikacja webowa wspomagająca projektowanie i zarządzanie bazami danych

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, wrzesień 2018

Streszczenie

Praca dotyczy problemu projektowania relacyjnych baz danych. Mimo ogromnego rozwoju w dziedzinie baz danych i powstawania coraz to nowych nierelacyjnych modeli bazodanowych, ciągle najpopularniejsze są bazy relacyjne. Odpowiednie zaprojektowanie bazy danych wymaga trzymania się pewnych reguł i wytycznych, o których niedoświadczeni użytkownicy często zapominają. Takie działania objawiają się złym działaniem bazy, trudnym rozwijaniem a czasami nawet koniecznością przeprojektowania całej architektury bazodanowej w systemie. Ponadto skrypty do zakładania tabel są dosyć skomplikowane i trudne do zapamiętania. Istnieją oczywiście na rynku rozwiązania ułatwiające to zadanie, ale są zazwyczaj kierowane do zaawansowanych użytkowników i są mało intuicyjne. Często zdarza się sytuacja, że programista back-end potrzebuje stworzyć prostą bazę danych, na potrzeby tworzonego przez siebie rozwiązania, a nie ma doświadczenia w tym zakresie, co przekłada się na stratę czasu.

Z tego powodu powstał pomysł stworzenia narzędzia ułatwiającego to zadanie. Rozwiązanie to powinno być intuicyjne, łatwe w użytkowaniu i nowoczesne. Dlatego też opiera się o technologie webowe. Prosty dla użytkownika interfejs graficzny stworzony został w oparciu o technologie HTML, CSS, JavaScript i bibliotekę jQuery. Strona serwerowa także została stworzona w technologii Javascript, ale z zastosowaniem biblioteki serwerowej NodeJS, a dane zapisywane są na serwerze w chmurze opartym na nierelacyjnej bazie danych MongoDB. Interfejs jest bardzo prosty, a tabele przedstawione zostały jako encje, które można przesuwać i modyfikować, co w łatwy sposób obrazuje architekturę stworzonej bazy danych. Po zaprojektowaniu można wygenerować skrypt w intersującym nas standardzie SQL (MS SQL lub Oracle). Narzędzie zostało stworzone w sposób umożliwiający jego dalszy rozwój i łatwe rozszerzanie o nowe możliwości.

Spis treści

1 Wstęp

W ostatnich czasach powstało wiele nowych systemów nierelacyjnych baz danych, ale mimo wszystko dalej najczęściej stosowane są bazy danych relacyjne. Swoją popularność zawdzięczają szybkości działania, przejrzystości i łatwości rozwoju. Przy projektowaniu bazy należy trzymać się pewnych zasad, które często są ignorowane, z czego rodzą się potem problemy. O ile w przypadku źle stworzonych indeksów sytuacja jest do opanowania, to w przypadku złego zaprojektowania bazy i stworzenia złych relacji między tabelami, po pewnym czasie, gdy wypełnione są danymi usprawnienie działania takiej bazy może okazać się niemożliwe. Poniższa praca ma przybliżyć ten problem i spróbować przedstawić rozwiązanie.

1.1 Aplikacja webowa do graficznego projektowania bazy danych

W dzisiejszych czasach większość aplikacji tworzy się tak, aby mogły być opublikowane w Internecie, ponieważ jest to najprostsza i najmniej wymagająca od użytkownika forma uzyskiwania dostępów do zasobów. W ostatnim czasie bardzo popularne stały się aplikacje webowe, czyli takie do których dostęp mamy bezpośrednio z poziomu przeglądarki.

„Aplikacja internetowa [1] - (ang. web application), nazywana też webową – program komputerowy, pracujący na serwerze i komunikujący się przez sieć z hostem użytkownika komputera z wykorzystaniem przeglądarki internetowej użytkownika, będącego w takim przypadku interaktywnym klientem aplikacji internetowej. W pracy aplikacji internetowej musi pośredniczyć serwer WWW. Do przygotowania samej aplikacji używa się różnych mechanizmów (np. *CGI, JSP, ASP.NET*) i języków (np. *PHP, Java, C#*), jak również serwerów aplikacji. Mechanizm prezentacji danych w przeglądarce określa się czasem mianem cienkiego klienta.”

Jest to wygodne i bezpieczne rozwiązanie, ponieważ nie wymaga od użytkownika pobierania żadnych plików, które następnie należałoby zainstalować i narazić swój komputer na zainfekowanie złośliwym oprogramowaniem. Tendencja do rozwoju tych rozwiązań jest tak ogromna, że nawet największe firmy takie jak Google czy Microsoft tworzą skomplikowane aplikacje webowe, zastępujące popularne desktopowe aplikacje takie jak *Word* czy *Excel*, przez webowe odpowiedniki platformę *Microsoft 365* czy *Arkusze i dokumenty Google*.

Najbardziej naturalnym dla człowieka sposobem przedstawienia struktury bazy danych jest model graficzny, gdzie tabele przedstawione są jako encje, a relacje między tabelami są opisanymi połączeniami. Architektura bazy danych przedstawiona w ten sposób, daje od razu dobry pogląd na to, jak dane w tabelach są ze sobą powiązane i w jaki sposób robić złączenia, aby jak najprostszą i najszybszą drogą dostać się do danych. Oprócz samego przedstawienia struktury, dochodzi problem

napisania skryptów zakładających tabele, indeksy i referencje między nimi, co niedoświadczonemu użytkownikowi może zająć dużo czasu. Dodatkowym zastosowaniem graficznego zobrazowania struktury bazy, jest możliwość pokazania klientowi w jaki sposób będą ustrukturyzowane dane w tworzonej dla niego aplikacji.

Bazując na stanie sztuki w tym zakresie i doświadczeniach autora w zakresie relacyjnych baz danych, w pracy przedstawiona została propozycja rozwiązania tego problemu.

1.2 Cel pracy

Celem pracy jest przeanalizowanie problematyki graficznego projektowania relacyjnych baz danych i znalezienie wymagań jakie powinny spełniać stworzone do tego aplikacje.

1.3 Rezultaty pracy

Rezultatem pracy jest wynik analizy wymagań i problematyki graficznego projektowania baz danych i zaproponowanie prototypu aplikacji, który przy odpowiednim rozwoju spełni założone wymagania. Prototyp zostanie podzielony na trzy moduły:

- Moduł logowania/rejestracji
- Moduł projektowania schematu bazy danych
- Moduł generowania skryptów SQL

1.4 Organizacja pracy

Pierwsze trzy rozdziały skupiają się na przedstawieniu problemu projektowania relacyjnych systemów bazodanowych, zrobiony został przegląd i ocena rozwiązań dostępnych na rynku.

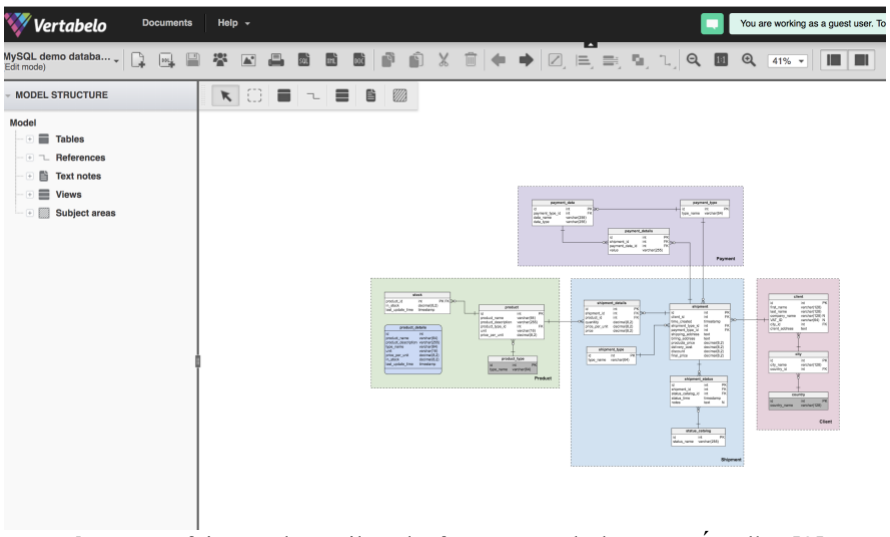
W kolejnych rozdziałach przedstawiono zastosowane technologie, pokazano aspekty techniczne projektu, a na końcu przedstawione zostało działanie prototypu.

2 Istniejące rozwiązania

Rozdział ten ma na celu zaprezentowanie istniejących na rynku rozwiązań problemu opisanego w pracy, przedstawienia ich wad i zalet, oraz oceny użyteczności pod względem wymagań postawionych w pracy. W kolejnych podrozdziałach przedstawione zostały najciekawsze projekty dostępne na rynku.

2.1 Platforma vertabelo.com

Platforma vertabelo.com [2] oferuje użytkownikom możliwość graficznego projektowania systemów bazodanowych i zapis projektów na ich serwerach w chmurze. Rozwiązanie jest bardzo rozbudowane i dostosowane do wielu standardów SQL. Platforma może być użytkowna tylko przez zarejestrowanych użytkowników i przez swoje skomplikowanie jest skierowana raczej do użytku przez profesjonalistów. Wartym zaznaczenia jest fakt, że projekt został stworzony przez Polski zespół programistów. Interfejs rozwiązania widoczny jest na rysunku 1.



Rysunek 1. Interfejs użytkownika platformy vertabelo.com, Źródło: [2]

Zalety:

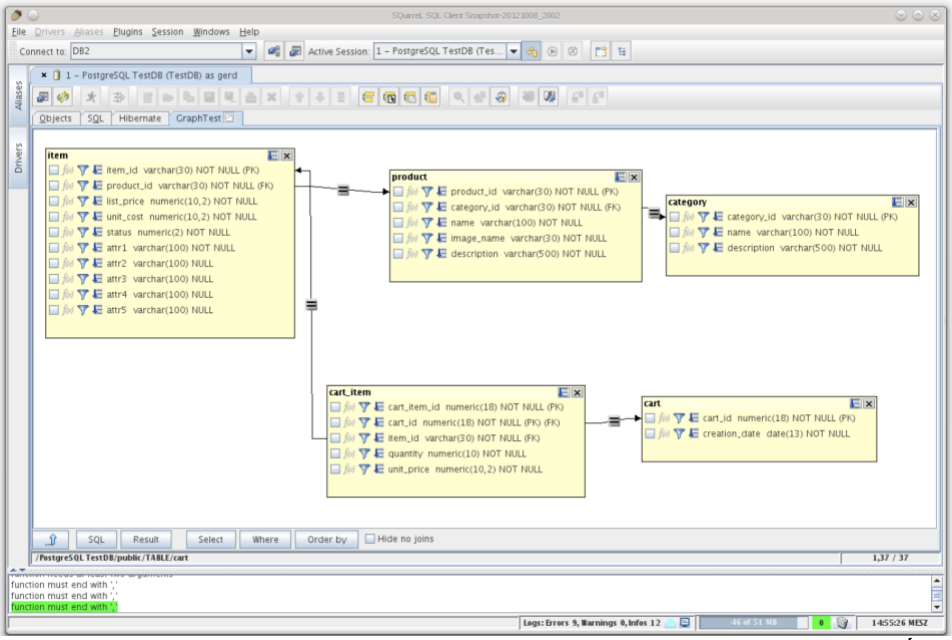
- Bardzo rozbudowana platforma dająca praktycznie nieograniczone możliwości projektowania systemów bazodanowych.
- Aplikacja webowa z zapisem projektów w chmurze.
- Dostępne licencje edukacyjne, dla studentów uczelni wyższych.

Wady:

- Przez stopień skomplikowania, platforma jest przeznaczona raczej dla użytkowników profesjonalnych.
- Drogie plany taryfowe, przy tańszych opcjach znaczące ograniczenia.
- Interfejs jest już trochę przestarzały i mało intuicyjny.

2.2 Aplikacja SquirreL SQL

Aplikacja SquirreL SQL [3] jest desktopową aplikacją, będącą jednocześnie klientem SQL. W celu stworzenia bazy danych, musimy zdefinować połączenie z serwerem SQL. Projektowanie tabel odbywa się w formie graficznej, a tabele przedstawione są w postaci encji, system nie generuje skryptów do tabel, a zakłada je na serwerze na podstawie stworzonego schematy. Ciekawym elementem rozwiązania jest generator zapytań. Aplikacja widoczna jest na rysunku 2.



Rysunek 2. Aplikacja SquirreL SQL i zaprojektowany w niej schemat bazy danych. Źródło: [3]

Zalety:

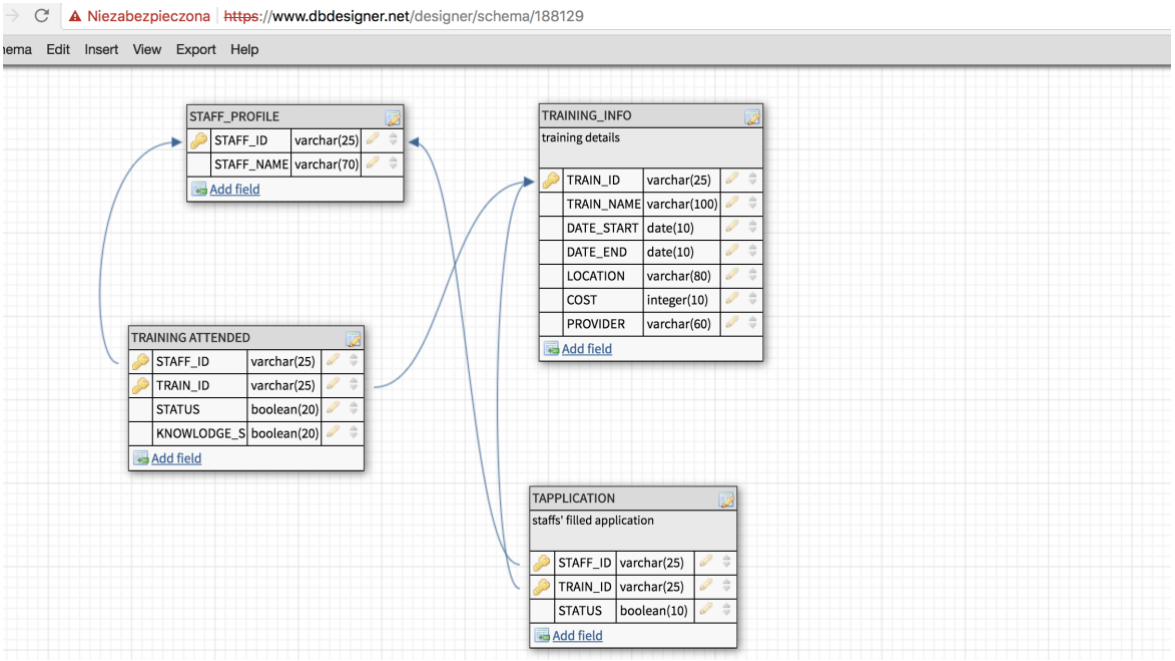
- Aplikacje jednocześnie jest klientem SQL, umożliwiającym wykonanywanie zapytań do bazy.
- Rozbudowany mechanizm generowania tabel SQL bezpośrednio na serwerze.
- Aplikacja jest darmowa.

Wady:

- Wymagane połączenie z serwerem SQL
- Aplikacja desktopowa
- Przestarzały interfejs, aplikacja nie jest już rozwijana.

2.3 Platforma dbdesigner.net

Platforma dbdesigner.net [4] jest aplikacją webową umożliwiającą projektowanie struktury bazy danych. Aplikacja jest łatwa w użyciu, tabele przedstawione są za pomocą encji, które mogą być łączone, co symbolizuje klucze obce. Dla podstawowych użytkowników korzystanie z aplikacji jest darmowe, a dla bardziej zaawansowanych plan subskrypcji jest niedrogi. Przez mały stopień skomplikowania i prosty interfejs skierowana jest raczej do mniej zaawansowanych użytkowników. Aplikacja po narysowaniu schematu, umożliwia wygenerowanie skryptu SQL. Rozwiązanie widoczne jest na rysunku 3.



Rysunek 3. Interfejs platformy dbdesigner.net, Źródło: [4]

Zalety:

- Aplikacja webowa, oparta o JavaScript
- Przyjemny, mało skomplikowany interfejs, dobry dla niezaawansowanych użytkowników

- Użytkowanie aplikacji darmowe do dwóch schematów i 30 tabel na schemat, kolejne plany taryfowe też bardzo korzystne cenowo.

Wady:

- Brak protokołu szyfrowanego HTTPS, co naraża użytkownika na niebezpieczeństwo.
- Wymagana rejestracja w celu stworzenia projektu.

2.4 Podsumowanie rozwiązań

Przedstawione rozwiązania dostępne na rynku, są ciekawymi projektami, których wspólną cechą jest możliwość graficznego zaprojektowania struktury baz danych w postaci encji. Każdy z tych systemów ma jakieś wady i zalety, ale wyciągając wnioski z mocnych i słabych stron tych rozwiązań, można stworzyć narzędzie, które idealnie odpowie na problematykę postawioną w pracy.

3 Propozycja rozwiązania

W tym rozdziale zaprezentowane zostaną cechy jakie powinno posiadać narzędzie na podstawie zalet konkurencyjnych rozwiązań, przedstawiona zostanie motywacja do podjęcia tematu pracy oraz wymagania, które musi spełniać prototyp.

3.1 Wnioski po analizie istniejących rozwiązań

Po wnikliwym przeanalizowaniu istniejących produktów, nasuwają się cechy, którymi powinny charakteryzować się dobre narzędzia do projektowania relacyjnych baz danych:

- Zapis projektów w usłudze chmurowej, nie na komputerze użytkownika
- Dostępność – Najlepsze rozwiązania tego typu są aplikacjami przeglądarkowymi, dzięki czemu dostęp do zasobów jest znacznie uproszczony.
- User-friendly – Przemyślany interfejs jest cechą, która umożliwia szybkie i łatwe znalezienie przez użytkownika pożądanych funkcjonalności.
- Prostota – Najbardziej przystępne dla użytkownika, szczególnie tego bez wiedzy informatycznej okazują się najprostsze aplikacje, nie udostępniające przy podstawowych zastosowaniach bardzo zaawansowanych funkcjonalności.
- Bezpieczeństwo – Dobre i nowoczesne platformy internetowe dbają o bezpieczeństwo przechowywanych danych.
- Generowanie SQL – W dobrych systemach do projektowania baz danych, można od razu wygenerować skrypty SQL, na podstawie których utworzone zostaną tabele w bazie danych użytkownika.
- Technologie umożliwiające rozwój – w najlepszych aplikacjach do projektowania baz danych użyte zostały nowoczesne technologie, które jeszcze przez długi czas będą rozwijane.

Trzymając się przedstawionych wyżej cech i wzorując się na platformach i systemach przedstawionych w poprzednim rozdziale, można stworzyć listę wymagań dla nowego rozwiązania, które wykorzysta najlepsze cechy produktów dostępnych na rynku, ale nie powieli ich błędów, dzięki czemu po narzuceniu odpowiedniego kierunku rozwoju stało by się bardziej funkcjonalne i chętniej wybierane niż prezentowane wcześniej konkurencyjne platformy.

3.2 Wymagania postawione dla nowego rozwiązania

Po przeanalizowaniu cech jakimi charakteryzują się najlepsze, dostępne na rynku aplikacje tego typu wyszczególniono wymagania jakie zostaną narzucone proponowanemu prototypowi:

- Powinno być nowoczesną aplikacją webową, działającą szybko i zapewniającą bezpieczeństwo użytkownikowi.
- Interfejs aplikacji powinien być nieskomplikowany, nowoczesny i zgodny z zasadami projektowania rozwiązań GUI user-friendly.
- Prototyp powinien umożliwiać proste projektowanie struktury relacyjnej bazy danych, przedstawionej w postaci graficznej.
- Platforma powinna umożliwić generowanie skryptów SQL, aby zaprojektowany schemat można było łatwo wykorzystać w praktyce.

3.3 Propozycja rozwiązania

Na podstawie narzuconych wymagań przedstawiona zostanie propozycja rozwiązania. Będzie ono aplikacją internetową, z dostępem poprzez przeglądarkę internetową. Będzie charakteryzowało się prostym interfejsem, umożliwiającym łatwe i szybkie projektowanie relacyjnych baz danych, nawet przez użytkowników bez dużego doświadczenia z bazami danych. Aplikacja podzielona będzie na poniższe moduły:

- Moduł logowania/rejestracji – Umożliwia rejestrację nowych użytkowników i dostęp do utworzonych wcześniej projektów, dla wcześniej zarejestrowanych użytkowników, poprzez logowanie.
- Moduł projektowania – Najważniejszy moduł w aplikacji, umożliwi graficzne projektowanie struktury relacyjnej bazy danych. Tabele przedstawione zostaną jako encje, które można tworzyć, modyfikować, przemieszczać i nadawać im relacje. Głównymi założeniami przy tworzeniu tego modułu będzie interaktywność, prostota użytkowania oraz nowoczesność.
- Moduł generowania SQL – Moduł ten powinien umożliwić użytkownikowi wygenerowanie skryptu SQL, na podstawie utworzonego modelu graficznego, po uruchomieniu którego w kliencie bazodanowym, utworzone zostaną wcześniej zaprojektowane tabele i relacje.

Aplikacja oparta będzie o architekturę klient – serwer, a dane przechowywane będą w usłudze chmurowej, co całkowicie odciąży użytkownika od konieczności pobierania i zapisywania projektów.

Ważnym aspektem prototypu będzie zapewnienie generyczności i podatności na dalszy rozwój prototypu. Platforma stworzona będzie w sposób umożliwiający bezproblemowy rozwój. Udostępnione zostaną przykładowe wtyczki do dialektów SQL, oraz przedstawiona zostanie uniwersalna architektura rozwiązania, która może się przyczynić do powstawania nowych wtyczek/modułów.

3.4 Motywacja i podsumowanie

Motywacją do stworzenia tego typu narzędzia są głównie związane z doświadczeniem zawodowym i zainteresowaniami autora związanymi z relacyjnymi bazami danych i tworzeniem nowoczesnego oprogramowania internetowego. Ważnym powodem motywującym do podjęcia tego tematu jest też chęć ułatwienia i uproszczenia procesu projektowania baz danych i umożliwienia zaoszczędzenia czasu.

Wyciągając wnioski z istniejących rozwiązań można stworzyć bardzo konkurencyjną aplikację, która przy odpowiedniej konstrukcji zapewniającej generyczność i możliwość dalszego rozwoju, może stać się jednym z lepszych rozwiązań tego typu na rynku i ułatwić codzienną pracę programistom bazodanowym oraz wszelkim innym użytkownikom projektującym bazy danych.

4 Zastosowane technologie

W tym rozdziale przedstawione zostały technologie użyte do stworzenia rozwiązania

4.1 HTML

HTML(Hypertext Markup Language) [5]- jest kodem używanym do tworzenia struktury strony i jej zawartości. HTML jest językiem znaczników(tagów), używanych do zamknięcia różnych treści, aby wyglądały i działały w określony sposób. Z pomocą tagów można ze słów czy obrazów zrobić odnośniki do innych stron, pogrubić tekst czy zrobić paragraf.

Elementy HTML składają się z:

- Tagów otwierających np. `<p>`
- Tagów zamykających np. `</p>`
- Zawartości
- Atrybutów – zawierają one dodatkowe informacje o elemencie, które nie są widoczne na stronie internetowej.
    - Atrybut klasy (`class="example-class"`) - klasy najczęściej stosowane są do sterowania wyglądem elementów HTML poprzez style CSS, dobrą praktyką jest opisywanie klasami grup elementów i ich zachowań, a nie konkretnych elementów.
    - Atrybut ID (`id="example-id"`) – atrybut identyfikujący konkretny element, stosowany kiedy potrzeba zlokalizować jednoznacznie element HTML z poziomu języka JavaScript, lub zastosować styl CSS.
    - Atrybut style – służy do opisu stylu bezpośrednio na elemencie, nadmierne używanie tych znaczników nie jest uznawane za dobrą praktykę, ponieważ utrudnia późniejsze zmiany wyglądu strony internetowej.

4.2 CSS

CSS(Cascading Style Sheets) [6] – Kaskadowe arkusze stylów – język służący do opisu formy prezentacji stron WWW. CSS opracowała organizacja W3C w 1996r. Język został stworzony po to, żeby odseparować strukturę dokumentu od jego warstwy prezentacji. Pozwala to na zmniejszenie skomplikowania dokumentu i zwiększenie dostępności strony internetowej, oraz co najważniejsze ułatwia wprowadzanie zmian. Stosowanie arkuszy CSS pozwala na zmianę wielu elementów i stron jednocześnie, bez konieczności definiowania wyglądu na poszczególnych elementach.

Do znalezienia elementów na stronie stosuje się selektory CSS w których po typie lub atrybucie można zlokalizować element.

Przykładowy selektor definiujący wielkość i kolor tekstu dla elementów zawierających klasę *font-small-main-color* widoczny jest na listingu 1.

Listing 1. Przykładowy selektor CSS.

```
div.font-small-main-color
{
      font-size:12px;
      color:grey;
}
```

4.3 Javascript

Javascript [7] - Język programowania który powstał w 1995 roku, na początku umożliwiał pisanie skryptów(programów) do przeglądarki internetowej Netscape. Następnie został przyjęty we wszystkich innych przeglądarkach internetowych. Języka Javascript używa się w celu budowania interaktywnych aplikacji, ale też na zwykłych stronach w celu zapewnienia interaktywności. Kiedy JavaScript zaczął być obsługiwany również przez inne przeglądarki, powstała dokumentacja opisująca jego standard. Standard ten nosi nazwę ECMAScript i pochodzi od organizacji Ecma International, która go wprowadziła.

Cechy charakterystyczne języka Javascript:

- Typowanie dynamiczne – typy zmiennych przypisywane są w trakcie wykonywania skryptu, co oznacza, że nie musimy określać typu podczas deklarowania zmiennej, tylko interpreter sam ustali typ na podstawie przypisanej wartości.
- Jezyk skryptowy – Oznacza, że programy pisane w tym języku nie są niezależną aplikacją, a jedynie kawałkiem kodu wykonywamym niezależnie od innych aplikacji.
- Wieloplatformowość – Programy pisane w języku JavaScript nie są przypisane do konkretnej platformy, mogą być uruchamiane w każdej przeglądarce internetowej zapewniającej obsługę tego języka
- Wieloparadygmatowy- cechujący się wieloma paradygmatami programowania, jest obiektowy, funkcyjny i imperatywny

Język Javascript jest obecnie jednym z najpopularniejszych języków programowania, powstaje do niego ogromna ilość rozszerzeń, bibliotek i frameworków, umożliwiających pisanie aplikacji w tym języku zarówno po stronie klienckiej jak i serwerowej.

4.4 jQuery

jQuery [8] – jest biblioteką języka programowania JavaScript stworzoną w celu ułatwienia programistom i projektantom stron internetowych tworzenie i rozszerzenia interakcji języka JavaScript, oraz udostepnienia zestawu metod ułatwiającego pisanie w języku JavaScript. Składnia tej biblioteki jest łatwa w użyciu i powoduje, że interfejsy API języka JavaScript stają się bardziej przystępne i zrozumiałe, mimo że tak naprawdę biblioteka ta nie oferuje żadnej nowej funkcjonalności.

Najważniejsze zalety jQuery:

- Łatwe wykonywanie żądań Ajax dla danych znajdujących się na serwerze. Upraszcza obsługę i wymianę danych klient-serwer.
- Lokalizowanie elementów na podstawie selektorów CSS
- Prosta obsługa zdarzeń

4.5 Visual Studio Code

Visual Studio Code to darmowy edytor kodu, umożliwiający tworzenie i debugowanie nowoczesnych aplikacji internetowych. Edytor wyposażony jest w nakładki do utrzymywania prawidłowości i oznaczania kolorami składni do wielu języków programowania. Wyposażony jest w system podpowiadania składni IntelliSense. Warty podkreślenia jest fakt, że program ten mimo swoich ogromnych możliwości jest całkowicie darmowy, również na platformę MAC OS.

4.6 HTTP

HTTP (ang. Hypertext Transfer Protocol) [9] – jest protokołem służącym do przesyłania żądań udostępniania dokumentów hipertekstowych WWW, na podstawie odnośników. Wprowadza znormalizowany standard komunikacji. Standaryzuje formę żądań klienta i odpowiedzi serwera.

Protokół HTTP jest bezstanowy, ponieważ nie zapisuje informacji o poprzednich transakcjach z klientem. Takie podejście pozwala na zmniejszenie obciążenia serwera, ale uniemożliwia zapis konkretnego stanu w danej sytuacji. W takim przypadku niezbędne mogą okazać się dodatkowe mechanizmy np. ciasteczka, lub sesje po stronie serwera. Protokół HTTP udostępnia nagłówki, które definiują dane zawarte w żądaniu.

Metody HTTP :

- **GET –** Pobranie danych wskazanych przez URI który może mieć w sobie parametry, żądanie przesłane tą metodą nie powinno mieć wpływu na dane znajdujące się na serwerze, jedynie powinno umożliwić pobranie informacji.
- **HEAD –** Tak jak metoda GET, służy do pobrania danych, stosowana do sprawdzania dostępności zasobu
- **PUT -** Metoda służąca do zapisu danych przesłanych za pomocą żądania na serwerze, najczęściej wykorzystywana do aktualizowania danych
- **POST –** Podobnie jak metoda PUT, służy do zapisu danych na serwerze, ale stosowana jest częściej przy tworzeniu nowych zasobów, lub w celu zapisu zawartości formularzy
- **DELETE –** Metoda ta służy do usuwania zasobów na serwerze
- **OPTIONS –** Metoda informująca o opcjach i wymaganiach możliwych do wykonania na kanale komunikacyjnym
- **TRACE –** Metoda do diagnostyki kanału komunikacyjnego
- **CONNECT –** Metoda która umożliwia połączenie się z serwerami pośrednimi
- **PATCH –** Metoda służąca do aktualizacji części danych

4.7 NodeJS

NodeJS [10] – jest środowiskiem uruchomieniowym języka JavaScript zbudowanym na silniku V8 stworzonego przez Google, stworzonym przez Ryana Dahla w 2009 roku. NodeJS zostało zaprojektowane do tworzenia skalowalnych aplikacji internetowych, w szczególności serwerów WWW napisanych w języku JavaScript. Umożliwia wytwarzanie aplikacji sterowanych zdarzeniami z wykorzystaniem asynchronicznego systemu wejść-wyjść.

Wraz z rozwojem NodeJS powstało wiele pakietów rozszerzających możliwości tego środowiska i ułatwiających pracę z nim. Zarządzanie pakietami odbywa się przez menadżera NPM, wywoływanego z konsoli.

4.8 ExpressJS

ExpressJS [11] – Jest to elastyczny framework do budowania serwerów z wykorzystaniem środkowiska NodeJS i JavaScript. Jest oprogramowaniem open-source pod licencją MIT. Przyspiesza i ułatwia tworzenie API i serwerów napisanych w języku JavaScript

Jego najważniejsze moduły:

- **Routing –** odpowiada w jaki sposób żądania klienta korespondują z endpointami aplikacji. Za pomocą modułu Routing można wskazać jaka metoda ma zostać wykonania w przypadku wywołania odpowiedniej metody HTTP pod konkretnym URI. Handler obsługujący routing zawiera takie informacje jak:
  - **Request –** Żądanie wysłane przez klienta, które zostało przechwycone przez handler.
  - **Response –** Opisuje rezultat, który ma być wysłany klientowi**.**
  - **Next -** funkcja wskazująca na następną metodą, wywoływaną po przechwyceniu żądania.
- **Middleware –** Middleware to funkcje mające dostęp do żądania, odpowiedzi i funkcji next, mogą wykonywać dowolny kod i wprowadzać zmiany w objektach response i request, mogą kończyć cykl request-response**.**

4.9 REST

REST [12] – (ang. Representional State Transfer) – styl architektury tworzenia serwerów www. Jest zbiorem zasad i dobrych praktyk stosowanych przy tworzeniu rozwiązań opartych na komunikacji przez protokół HTTP.

Główne założenia REST:

- **Klient-serwer –** serwer nie powinien być w żadnym stopniu zależny od klienta, możliwe jest stworzenie wielu klientów do jednego serwera np. aplikacja webowa i aplikacja mobilna**.**
- **Bezstanowość –** informacje o kliencie np. Sesja nie są przechowywane na serwerze.
- **Możliwość cache'owania –** Odpowiedzi definiują same siebie, czy mogą być cacheowane.
- **Warstwowość –** Serwer nie informuje klienta o warstwach
- **Jednolity interfejs –** Odpowiedzi same siebie opisują przez odpowiedni nagłówek, zasoby są identyfikowane przez odpowiednie żądania.

4.10 Mongo DB

MongoDB [16] – jest otwartym, nierelacyjnym systemem zarządzania bazami danych, stworzonym w języku C++. W odróżnieniu od relacyjnych systemów bazodanowych, nie posiada zdefiniowanej struktury, a dane są przetrzymywane w formacie JSON. Dzięki takiemu podejściu znacznie upraszcza się proces przetwarzania danych przez aplikacje, a jednocześnie zachowana jest możliwość indeksowania i tworzenia hierarchii. Bazy MongoDB są często krytykowane jest za słabą obsługę kodowania UTF-8 co sprawia problemy w przypadku przechowywania języków innych niż angielski. Za wadę MongoDB uważa się też spójność danych, z która bywają problemy ze względu na zastosowanie modelu asynchronicznych zapisów, co oznacza, że w przypadku żądania zapisu danych klient otrzymuje jedynie informacje, że dane zostaną zapisane w przyszłości. Zapytania do MongoDB są definiowane w języku JavaScript.

Najważniejsze możliwości MongoDB:

- Wsparcie standardu Unicode
- Inne kodowania danych obsłużone w formacie binarnym
- Bardzo dużo, różnych typów danych
- Możliwość stosowania kursorów
- Możliwość stosowania zapytań ad-hoc
- Tworzenie zapytań do pól zagnieżdżonych
- Indeksowanie
- Agregacja danych
- Składowanie plików w bazie
- Łatwa replikacja

5 Projekt i implementacja

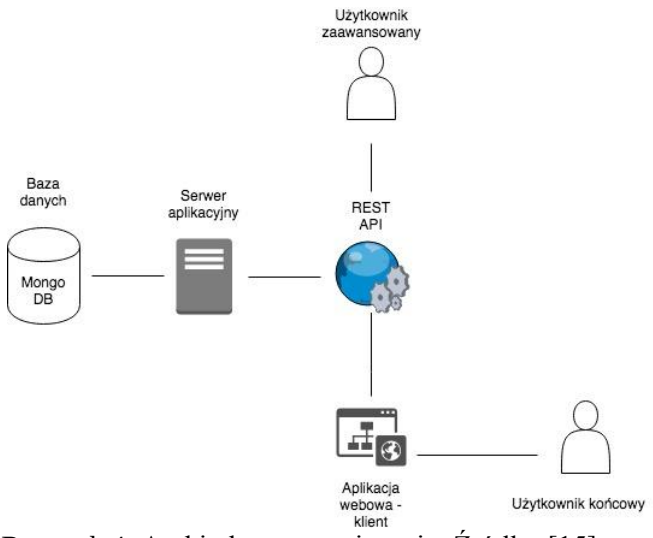
W tym rozdziale pokazana zostanie propozycja rozwiązania podzielona na ogólną architekturę rozwiązania, bazę danych, część serwerową API, oraz aplikację webową. Za pomocą aplikacji webowej użytkownik końcowy może się zalogować lub zarejestrować, celem utworzenia projektu bazy danych. API jest serwerową częścią rozwiązania, umożliwiającą dostęp do danych użytkownika przetrzymywanych w bazie danych.

5.1 Architektura rozwiązania

Architektura rozwiązania składa się bazy danych, serwera aplikacyjnego, udostępnionego API, oraz z aplikacji klienckiej – serwisu internetowego. Serwer aplikacyjny wymienia dane z bazą danych stworzoną w nierelacyjnym systemie bazodanowym MongoDB, który przechowuje zapisane dane w usłudze chmurowej. Aby umożliwić komunikację z serwerem aplikacyjnym, udostępnione zostało API w standardzie REST, z którym za pomocą metod HTTP następuje komunikacja. Komunikacja z API odbywa się na dwa sposoby.

- Wysyłanie bezpośrednich żądań do udostępnionego API- metoda dostępu do danych dla zaawansowanych użytkowników.
- Poprzez aplikację internetową – kliencką z prostym interfejsem, przeznaczoną dla zwykłych użytkowników

Architektura rozwiązania pokazana jest na rysunku 4.



Rysunek 4. Architektura rozwiązania. Źródło: [15]

5.2 Baza danych

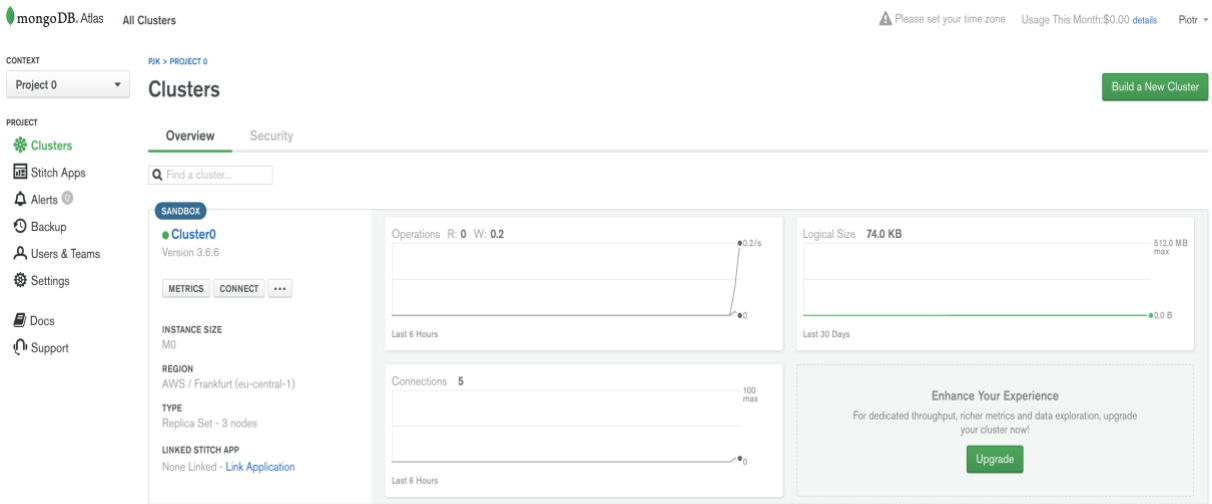
W tym podrozdziale przedstawiona została struktura bazy danych, oraz integracja usługi MongoDB Atlas z aplikacją serwerową.

5.2.1 Integracja usługi MongoDB Atlas z aplikacją JavaScript

Baza danych stworzona w systemie nierelacyjnym MongoDB, postawiona została w usłudze chmurowej MongoDB Atlas. Usługa ta jest darmowa do pojemności logicznej bazy danych 512MB.

Najważniejszą zaletą tego rozwiązania jest łatwość integracji z aplikacją i prosta administracja bazą danych. Każdy użytkownik zakładający konto w usłudze Atlas, ma dostęp do dedykowanego Dashboardu, na którym ma podgląd do wszystkich najważniejszych informacji. Znajdują się tu informacje o klastrach - czyli instancjach bazodanowych, o aktualnym i historycznym zapisie i odczycie danych, o pozostałej pojemności do następnego planu taryfowego. Z poziomu dashboardu możemy zarządzać użytkownikami, którzy mają do niej dostęp, możemy zarządzać backupami, czy poprosić o wsparcie.

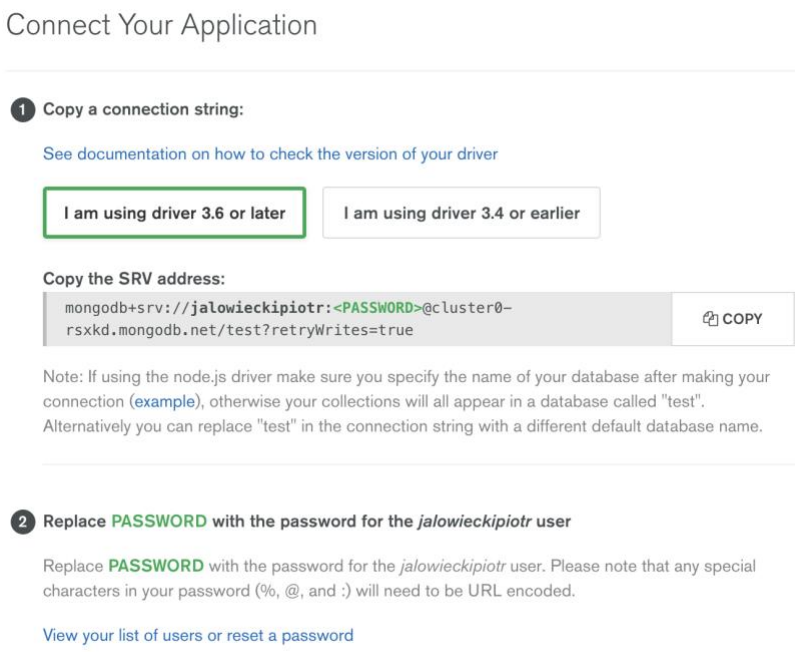
W płatnych planach taryfowych udostępniony został mechanizm do graficznego przedstawienia kolekcji, pisania zapytań bezpośrednio z panelu użytkownika czy sprawdzenia użycia indeksów, w celu optymalizacji bazy danych. Dashboard aplikacji pokazany jest na rysunku 5.



Rysunek 5. Dashboard użytkownika usługi MongoDBAtlas. Źródło: [16]

Aby przeprowadzić integrację bazy danych z istniejącą aplikacją wystarczy z poziomu panelu użytkownika wcisnąć przycisk „Connect”, co doprowadzi nas do kreatora połączenia. Zostaniemy poproszeni o stworzenie Whitelist’y adresów IP, co da usłudze informację, z jakich adresów możliwe

jest połącznie się ze stworzoną bazą danych. Po przejściu przez kreatora wyświetlony zostanie komunikat z adresem serwera, z którego należy skorzystać w celu integracji z aplikacją JavaScriptową. Wygenerowany adres serwera widoczny jest na rysunku 6.



Rysunek 6. Wygenerowany adres serwera przez kreator połączenia Mongo DB Atlas. Źródło: [16]

W celu zintegrowania aplikacji z bazą danych MongoDB utworzoną w usłudze MongoDB Atlas, w serwerze aplikacyjnym zbudowanym z użyciem środowiska NodeJS i biblioteki Express, należy zainstalować pakiet Mongoose, czyli bibliotekę JavaScriptową umożliwiającą integrację z bazą MongoDB. Aby to zrobić należy w uruchomić menedżer pakietów NPM za pomocą komendy widocznej na listingu 2:

Listing 2. Komenda do uruchomienia NPM

```
$npm start
```

A następnie zainstalować bibliotekę Mongoose za pomocą komendy widocznej na listingu 3:

Listing 3. Komenda do zainstalowania Mongoose.

```
$ npm install mongoose
```

Po zainstalowaniu pakietu Mongoose, za pomocą wcześniej wygenerowanego adresu, należy utworzyć połączenie z bazą danych. W tym celu należy stworzyć obiekt mongoose za pomocą funkcji require z pakietu Expres, a następnie na stworzonym obiekcie użyć funkcji connect, co utworzy połączenie z podanym jako parametr adresem, użycie widoczne na listingu 4.

Listing 4. Podłączenie usługi Mongo DB Atlas.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb+srv://jalowieckipiotr:PASSWORD@cluster0-
rsxkd.mongodb.net/test??retryWrites=true');
```

5.2.2 Struktura bazy danych

Dane w bazie MongoDB wymieniane i przechowywane są w formacie JSON, czego naturalnym następstwem jest fakt, że struktury danych są obiektami Javascriptowymi. Zdefiniowane kolekcje/schematy w MongoDB są odpowiednikami tabel w relacyjnych bazach danych, tak samo jak w przypadku relacyjnych baz danych na zdefiniowanych polach można zakładać indeksy. Wszystkie znane z relacyjnych baz operacje modyfikacji danych Insert/Update/Delete, także są dostępne w MongoDB, z tym że są odpowiednimi metodami obiektów JavaScriptowych.

Schematy w MongoDB definiuje się z użyciem klasy Schema z bilblioteki Mongoose i utworzonego z niej obiektu. W projekcie wyróżniono dwa schematy, odpowiadające za przechowywanie danych użytkowników na serwerze.

Schemat User – dane zapisane w bazie za pomocą tego schematu, mają przechowywać informację o zarejestrowanych użytkownikach, hasła użytkowników przed zapisem są szyfrowane, w celu zapewnienia bezpieczeństwa. Opis pól widoczny jest w tabeli 1.

Tabela 1. Schemat User – opis pól.

Nazwa pola	**Typ**	**Opis**
_id	Mongoose Object ID	Unikalny identyfikator użytkownika(wymagane)
name	String	Nazwa użytkownika
email	String	Unikalny e-mail użytkownika(wymagane)
password	String	Hasło użytkownika, zaszyfrowane(wymagane)

Każdy użytkownik powinien posiadać unikalny e-mail, który jednocześnie jest jego loginem, używanym podczas logowania do aplikacji. Na listingu 5 przedstawiono definicję schematu User w języku JavaScript.

Listing 5. Definicja schematu User.

```
const mongoose = require('mongoose');
const user = mongoose.Schema({
    _id: mongoose.Schema.Types.ObjectId,
    name:String,
    email:{type:String, required:true},
    password:{type:String, required:true}
});
module.exports = mongoose.model('User',user);
```

Schemat Entity – odpowiada za przechowywanie informacji o encji, czyli tabeli utworzonych przez użytkowników w aplikacji internetowej. Encje są przypisane do użytkownika, dlatego w definicji schematu Entity utworzone zostało pole userId, które jest unikalnym identyfikatorem użytkownika i pozwala odfiltrować encje należące do konkretnego użytkownika. Opis pól widoczny jest w tabeli 2.

Nazwa pola	**Typ**	**Opis**
_id	Mongoose Object ID	Unikalny identyfikator encji
userId	Mongoose Object ID	Unikalny identyfikator użytkownika
name	String	Nazwa tabeli(encji)
left	Number	Położenie encji - poziome
top	Number	Położenie encji - pionowe
columns	Array	Informacja o kolumnach tabeli(encji)
references	Array	Referencje do innych tabel(encji)
sourceReferences	Array	Informacja o referencjach pochodzących z innych tabel.

Tabela 2. Schemat Entity – opis pól.

Pola left i top przechowują informacje o położeniu encji w aplikacji internetowej, są to kluczowe parametry, dzięki którym encje, po ponownym otwarciu aplikacji znajdują się w tym samym miejscu co przed zamknięciem. Na listingu 6 przedstawiono definicję schematu Entity w języku JavaScript.

Listing 6. Zdefiniowany w JavaScript schemat Entity.

```
const mongoose = require('mongoose');
const entity = mongoose.Schema({
    _id: mongoose.Schema.Types.ObjectId,
    userId: mongoose.Schema.Types.ObjectId,
    name:String,
    left:Number,
    top:Number,
    columns:[],
    references:[],
    sourceReferences:[]
}
);
module.exports = mongoose.model('Entity',entity);
```

5.3 API

W celu zapewnienia generyczności rozwiązania i łatwego dostępu do danych, utworzono otwarte API. Dzięki udostępnionemu API kliencka aplikacja internetowa może wymieniać dane z serwerem aplikacyjnym, a co za tym idzie zapisywać i odczytywać je z bazy danych zlokalizowanej w usłudze chmurowej MongoDB Atlas. API jest otwarte, co oznacza, że do udostępnianych w nim zasobów i metod, mogą dostać się również zewnętrzne aplikacje, czy developerzy. Wymieniane z API dane są w formacie JSON, a samo API jest zbudowane zgodnie z zasadami architektury REST, co zapewnia między innymi jego otwartość i bezstanowość.

Poprzez API mamy dostęp do dwóch zasobów:

- User
- Entity

5.3.1 User

Moduł User udostępnia metody do zarządzania użytkownikami. Z wykorzystaniem tego modułu możliwa jest rejestracja nowych użytkowników, logowanie, usuwanie użytkowników.

Poniżej przedstawiona zostanie specyfikacja i przykładowe wywołania wszystkich metod w module User.

POST user/login

Metoda służąca do logowania, w żądaniu należy podać login(e-mail) i hasło, a w odpowiedzi otrzymujemy userId, czyli unikalny identyfikator użytkownika, na podstawie którego mamy dostęp do pozostałych zasobów. W tabeli 3 przedstawione zostały parametry które należy przekazać w żądaniu. Na listingu 7 przedstawione zostało przykładowe żądanie w JSON wysłane za pomocą programu Postman, służącego do wysyłania żądań do API.

Tabela 3. Parametry żądania POST user/login.

Nazwa parametru	**Opis**
email	e-mail użytkownika, będący jednocześnie loginem. (Wymagany)
Password	Hasło użytkownika.(Wymagany)

Listing 7. Przykładowe żądanie do metody POST user/login w programie Postman.

```
{
"email":"test",
"password":"test"
}
```

W tabeli 4 przestawiono możliwe odpowiedzi na wysłane żądanie.

Tabela 4. Parametry odpowiedzi POST user/login

Kod	**Treść odpowiedzi**	**Opis**

200	{
	"message": "Authorization successful",
	"userId": "Identyfikator użytkownika"
	}
		Prawidłowe uwierzytelnienie użytkownika, w odpowiedzi userId, czyli unikalny identyfikator.

401	{
	"message": "Authorization failed"
	}
		Błąd autentykacji, błędny login lub hasło

500	{
	"error": {
	"message": "Opis błędu"
	}
	}
		Błąd serwera, spowodowany błędnym requestem, lub innym wewnętrznym problemem, w obiekcie „error”, zwracany jest komunikat błędu.

POST user/signup

Metoda służąca do założenia nowego użytkownika, w żądaniu należy podać login(e-mail) i hasło, a w odpowiedzi otrzymujemy userId, czyli unikalny identyfikator użytkownika, na podstawie którego mamy dostęp do pozostałych zasobów. W tabeli 5 przedstawione zostały parametry które należy przekazać w żądaniu. Na lisitingu 8 przedstawione zostało przykładowe żądanie w JSON.

Tabela 5. Parametry żądania POST user/signup

Nazwa parametru	**Opis**
email	e-mail użytkownika, będący jednocześnie loginem. (Wymagany)
name	Nazwa użytkownika
password	Hasło użytkownika. (Wymagany)

Listing 8. Przykładowe żądanie do metody POST user/signup w programie Postman.

```
{
"email":"tester",
"name":"tester",
"password":"tester"
}
```

W tabeli 6 przestawiono możliwe odpowiedzi na wysłane żądanie.

Tabela 6. Parametry odpowiedzi POST user/signup.

Kod	**Treść odpowiedzi**	**Opis**

200	{
	"userId": "Identyfikator użytkownika"
	}
		Użytkownik prawidłowo utworzony w odpowiedzi userId, czyli unikalny identyfikator.

409	{
	"message": "Mail exists"
	}
		Błąd zakładania użytkownika, e-mail już istnieje.

500	{
	"error": {
	"message": "Opis błędu"
	}
	}
		Błąd serwera, spowodowany błędnym requestem, lub innym wewnętrznym problemem, w obiekcie „error”, zwracany jest komunikat błędu.

DELETE user

Metoda służąca do usunięcia wszystkich użytkowników, metoda nie jest wykorzystywana w aplikacji klienckiej i powinna być udostępniona jedynie administratorowi. W tabeli 7 przestawiono możliwe odpowiedzi na wysłane żądanie. Żądanie to nie wymaga podawania żadnych parametrów wejściowych.

Tabela 7. Parametry odpowiedzi DELTE user.

Kod	**Treść odpowiedzi**	**Opis**
200	{"message": "all users removed" }	Wszyscy użytkownicy zostali prawidłowo usunięci.

500	{
	"error": {
	"message": "Opis błędu"
	}
	}
		Błąd serwera, spowodowany błędnym requestem, lub innym wewnętrznym problemem, w obiekcie „error”, zwracany jest komunikat błędu.

DELETE user/:userId

Metoda służąca do usunięcia pojedynczego użytkownika, metoda nie jest wykorzystywana w aplikacji klienckiej. W tabeli 8 przestawiono możliwe odpowiedzi na wysłane żądanie.

Tabela 8. Parametry odpowiedzi DELETE user/:userId.

Kod	**Treść odpowiedzi**	**Opis**

200	{
	"message": "User deleted"
	}
		Użytkownik usunięty prawidłowo.

500	{
	"error": {
	"message": "Opis błędu"
	}
	}
		Błąd serwera, spowodowany błędnym requestem, lub innym wewnętrznym problemem, w obiekcie „error”, zwracany jest komunikat błędu.

5.3.2 Entity

Moduł Entity udostępnia metody do zarządzania encjami, czyli tabelami stworzonymi przez użytkowników w aplikacji klienckiej. Z wykorzystaniem tego modułu możliwe jest tworzenie nowych encji, aktualizowanie istniejących, usuwanie oraz wyświetlanie informacji o encjach. Poniżej przedstawiona zostanie specyfikacja i przykładowe wywołania wszystkich metod w module Entity.

GET entities/:userId

Metoda zwracająca informacje o wszystkich utworzonych przez użytkownika encjach. Możliwe odpowiedzi przedstawiono w tabeli 9.

Tabela 9. Możliwe odpowiedzi metody GET entities/:userId

Kod	**Treść odpowiedzi**	**Opis**
200	{ "count": 1, "entities": [{ "columns": ["[{\"id\":1,\"name\":\"DocumentsId\",\"type\":\"bigint\",\"nullable\":false,\"pk\":true},{\"id\":2,\"name\":\"DocumentNo\",\"type\":\"nvarchar(max)\",\"nullable\":false,\"pk\":false}]"], "references": ["[]"], "sourceReferences": ["[]"	W odpowiedzi informacja o ilości wszystkich encji użytkownika, oraz kolekcja zawierająca dane wszystkich zapisanych encji z takimi informacjami jak : columns – kolumny w tabeli, references – klucze obce do innych tabel, sourceReferences- klucze obce pochodzące z innych tabel, _id –

	], "_id": "5b6597a919ac3e0e356492cf", "userId": "5b65979719ac3e0e356492cd", "name": "Documents", "left": 271, "top": 257 }] }	unikalny identyfikator encji, userId – unikalny identyfikator użytkownika, name – nazwa tabeli(encji), left – położenie poziome encji, top – położnie pionowe encji.

500	{
	"error": {
	"message": "Opis błędu"
	}
	}
		Błąd serwera, spowodowany błędnym requestem, lub innym wewnętrznym problemem, w obiekcie „error”, zwracany jest komunikat błędu.

POST entities

Metoda służąca do utworzenia nowej encji, w żądaniu należy podać informacje o tworzonej encji, a w odpowiedzi otrzymamy utworzoną encję. W tabeli 10 przedstawione zostały parametry które należy przekazać w żądaniu. Na listingu 9 przedstawione zostało przykładowe żądanie w JSON.

Tabela 10. Parametry żądania POST entities.

Nazwa parametru	**Opis**
userId	Unikalny identyfikator użytkownika, do którego ma zostać przypisana encja.
name	Nazwa encji(tabeli).
columns	Informacja o kolumnach

Listing 9. Przykładowe żądanie POST entities w JSON.

```
{
"userId": "5b65979719ac3e0e356492cd",
            "name": "Documents",
            "columns":
["[{\"id\":1,\"name\":\"DocumentsId\",\"type\":\"bigint\",\"nullable\":false,\
"pk\":true},{\"id\":2,\"name\":\"DocumentNo\",\"type\":\"nvarchar(max)\",\"nul
lable\":false,\"pk\":false}]"
            ]
```

```
}
```

W tabeli 11 przedstawione zostały możliwe odpowiedzi na żądanie w metodzie POST entities.

Tabela 11. Możliwe odpowiedzi metody POST entities.

Kod	**Treść odpowiedzi**	**Opis**
200	{ "createdEntity": { "columns": ["[{\"id\":1,\"name\":\"DocumentsId\",\"type\":\"bigint\",\"nullable\":false,\"pk\":true},{\"id\":2,\"name\":\"DocumentNo\",\"type\":\"nvarchar(max)\",\"nullable\":false,\"pk\":false}]"], "references": [], "sourceReferences": [], "_id": "5b66d28519ac3e0e356492d0", "userId": "5b65979719ac3e0e356492cd", "name": "Documents", } }	W odpowiedzi informacja mówiąca o pomyślnie utworzonej encji i parametry takie jak: columns – kolumny w tabeli, references – klucze obce do innych tabel, sourceReferences- klucze obce pochodzące z innych tabel, _id – unikalny identyfikator encji, userId – unikalny identyfikator użytkownika, name – nazwa tabeli(encji)
500	{ "error": { "message": "Opis błędu" } }	Błąd serwera, spowodowany błędnym requestem, lub innym wewnętrznym problemem, w obiekcie „error”, zwracany jest komunikat błędu.

PATCH entities/:entityId

Metoda służąca do aktualizacji danych istniejącej encji, w żądaniu należy podać informacje o aktualizowanej encji. W tabeli 12 przedstawione zostały parametry które należy przekazać w żądaniu. Na listingu 10 przedstawione zostało przykładowe żądanie w JSON. W parametrze URL należy podać unikalny identyfikator encji entityId

Tabela 12. Możliwe odpowiedzi metody PATCH entities/:entityId.

Nazwa parametru	**Opis**
name	Nazwa encji(tabeli).
columns	Informacja o kolumnach

left	Położenie poziome encji
top	Położenie pionowe encji
references	klucze obce do innych tabel
sourceReferences	klucze obce pochodzące z innych tabel

Listing 10. Przykładowe żądanie PATCH entities/:entityId w JSON.

```
{"name": "Documents",
            "columns":
["[{\"id\":1,\"name\":\"DocumentsId\",\"type\":\"bigint\",\"nullable\":false,\
"pk\":true}]"
            ],
            "top":20,
            "left":20
}
```

W tabeli 13 przedstawione zostały możliwe odpowiedzi na żądanie w metodzie PATCH entities/:entityId.

Tabela 13. Parametry odpowiedzi PATCH entities/:entityId.

Kod	**Treść odpowiedzi**	**Opis**

200	{
	"message": "entity updated"
	}
		Encja zaktualizowana prawidłowo.

500	{
	"error": {
	"message": "Opis błędu"
	}
	}
		Błąd serwera, spowodowany błędnym requestem, lub innym wewnętrznym problemem, w obiekcie „error”, zwracany jest komunikat błędu.

DELETE entities/:entityId

Metoda służąca do usuwania istniejącej encji. W tabeli 14 przedstawione zostały możliwe odpowiedzi na żądanie w metodzie DELETE entities/:entityId. W parametrze URL należy podać unikalny identyfikator encji entityId.

Tabela 14. Parametry odpowiedzi DELETE entities/:entityId.

Kod	**Treść odpowiedzi**	**Opis**

200	{
	"message": "entity removed"
	}
		Encja usunięta prawidłowo.

500	{
	"error": {
	"message": "Opis błędu"
	}
	}
		Błąd serwera, spowodowany błędnym requestem, lub innym wewnętrznym problemem, w obiekcie „error”, zwracany jest komunikat błędu.

DELETE entities

Metoda służąca do usuwania wszystkich encji, nie jest wykorzystywana w aplikacji internetowej, powinna być udostępniona tylko jedynie administratorowi. W tabeli 15 przedstawione zostały możliwe odpowiedzi na żądanie w metodzie DELETE entities.

Tabela 15. Parametry odpowiedzi DELETE entities.

Kod	**Treść odpowiedzi**	**Opis**
200	{ "message": "all entities removed" }	Wszystkie encje usunięte prawidłowo.

500	{
	"error": {
	"message": "Opis błędu"
	}
	}
		Błąd serwera, spowodowany błędnym requestem, lub innym wewnętrznym problemem, w obiekcie „error”, zwracany jest komunikat błędu.

5.4 Aplikacja internetowa

Aplikacja internetowa, będąca klientem w tym rozwiązaniu charakteryzuje się poniższymi możliwościami:

- Rejestracja - Zakładanie nowych użytkowników.
- Logowanie – Dostęp do wcześniej utworzonego projektu, poprzez zalogowanie, projekt zapisuje się automatycznie w bazie danych, przy każdej zmianie jego zawartości.
- Tworzenie encji(tabel) – Użytkownik ma możliwość stworzenia encji, odpowiadających tabelom w relacyjnym systemie bazodanowym.
- Usuwanie encji – Użytkownik ma możliwość usuwania stworzonych wcześniej encji.
- Dodawanie kolumn – Do stworzonych encji użytkownik może dodać kolumny, przy dodawaniu kolumn, użytkownik musi zdecydować jakiego jest ona typu i czy jest kluczem głównym (Primary key).
- Usuwanie kolumn – Użytkownik ma możliwość usuwania wcześniej stworzonych kolumn
- Dodawanie kluczy obcych(Foreign Key) - Użytkownik ma możliwość wskazania na tabelę z którą ma zostać utworzona referencja, może nazwać tę referencję, a system sam utworzy w tabeli źródłowej kolumny, które wchodzą w skład klucza głównego tabeli docelowej i na ich podstawie utworzy klucz.
- Usuwanie kluczy obcych – Użytkownik ma możliwość usunięcia, wcześniej utworzonych kluczy obcych.
- Graficzne przedstawienie tabel, interakcja – Tabele przedstawione jako Encje, umożliwiona jest interakcja, poprzez przesuwanie encji.
- Generowanie skryptu SQL – Po utworzeniu modelu i wyborze dialektu SQL, wygenerowany zostanie skrypt.

Aplikacja internetowa utworzona została w technologii JavaScript, z dodatkową biblioteką jQuery ułatwiającą i usprawniającą tworzenie aplikacji. Wartwa prezentacji aplikacji stworzona została w HTML z wykorzystaniem arkuszy styli CSS, dzięki takiemu podejściu nie ma potrzeby stosowania bezpośredniego stylowania elementów HTML w kodzie, co wpływa na zwiększenie czytelności i prostoty kodu HTML. Na listingu 11 przedstawiony został kod HTML aplikacji.

Listing 11. Kod HTML aplikacji internetowej.

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>Entity designer</title>
  <link rel="stylesheet" href="style.css">
  <script src="https://code.jquery.com/jquery-1.12.4.js"></script>
  <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
  <script src="jQueryRotate.js" type="text/javascript"></script>
  <script src="jquery.inherit.js"></script>
  <script src="Entity.js"></script>
  <script src="Line.js"></script>
  <script src="EntityService.js"></script>
  <script src="EntityRenderer.js"></script>
  <script src="EntityView.js"></script>
  <script src="SQL_genrator.js"></script>
</head>
<body>
    <div class="main-container anonymous" >
        <header>
            <h1>Entity designer</h1>
        </header>
        <div class="toolbar">
            <input type="text" class="entity-input" placeholder="Nazwa encji" />
            <button type="submit" class="btn add-entity"
onclick="EntityView.addEntity()" >Dodaj</button>
            <select class="sql-languages" ></select>
            <button type="submit" class="btn generate-script">Generuj
skrypt</button>

        </div>
        <div class="entities-container">
        </div>
        <div class="login-container">
                <span class="header">Zaloguj się</span>
                <input type="text" class="email" placeholder="Email" />
                <input type="text" class="password" placeholder="Hasło" />
                <button type="submit" class="btn login"
onclick="EntityView.login()" >Zaloguj</button>
                <span class="header">Nie masz konta? Zarejestruj się!</span>
                <input type="text" class="email-register" placeholder="Email"
/>
                <input type="text" class="password-register"
placeholder="Hasło" />
                <button type="submit" class="btn register"
onclick="EntityView.register()" >Zarejestruj się</button>
        </div>
        <footer>
        <span class="sign">Created by Piotr Jałowiecki</span>
    </footer>
    </div>
</body>
</html>
```

Na znaczne uproszczenie i skrócenie kodu HTML, wpływa także fakt, że kod HTML przedstawiający encje, generowany jest dynamicznie podczas użytkowania aplikacji. Na podstawie zdefiniowanego schematu HTML, tworzony jest kod HTML i wklejany do kontenera div oznaczonego klasą „entities-container”. Na listingu 12 widać schemat HTML encji.

Listing 12. Schemat HTML encji.

```
var EntityTemplate = `<div class="entity">
                      <div class="entity-header">
                          <span class="label">{name}</span>
                          <button type="submit" class="delete">x</button>
                       </div>
                       <table class="entity-body">
                         <tr>
                           <th>L. p.</th>
                           <th>Nazwa kolumny</th>
                           <th>Typ</th>
                           <th>Null</th>
                           <th>PK</th>
                           <th/>
                         </tr>
                         <tr class="add-column-container">
                              <td>
                                  <span>-</span>
                              </td>
                              <td>
                                  <input      class="column-name"      type="text"
placeholder="Wpisz nazwę kolumny"/>
                              </td>
                              <td>
                                <select class="column-type" />
                              </td>
                              <td>
                                <input class="column-nullable" type="checkbox" />
                              </td>
                              <td>
                                <input class="column-pk" type="checkbox" />
                              </td>
                              <td>
                                <button      type="submit"      class="add-column"
>+</button>
                              </td>
                         </tr>
                       </table>
                       <div class="entity-footer">
                          <select class="reference-tables" />
                          <input        class="reference-name"        type="text"
placeholder="Nazwa referencji" />
                          <button    type="submit"    class="add-reference">Dodaj
FK</button>
                       </div>
                       </div>`;
```

Na podstawie wzorca `EntityTemplate` i danych w JSON, które odczytujemy z serwera, tworzone są poszczególne encje za pomocą funkcji `_createEntityItem`. Funkcja ta jako parametr wejściowy dostaje obiekt `entity` w JSON. Na początku z wzorca `EntityTemplate`, generowany jest HTML, w którym nadpisywana jest nazwa. Następnie encja jest pozycjonowana, na podstawie odczytanych parametrów `left` i `top`. Dalej następuje generowanie kolumn, oraz referencji do tabel, a na samym końcu do elementu HTML `select` dodawane są elementy `option`, oznaczające pozostałe tabele do których można utworzyć klucz obcy. Na listingu 13 pokazano funkcję `createEntityItem`.

Listing 13. Definicja funkcji _createEntityItem.

```
function _createEntityItem(entity) {
        //
        var                                $entity                                =
$(EntityTemplate.replace("{name}",entity.TableName)).addClass("id-
"+entity.Id);
        $_containerRoot.append($entity)
        $entity.css( "left", entity.Left);
        $entity.css( "top", entity.Top);
        var $body = $entity.find(".entity-body");
        if (typeof entity.Columns != 'undefined'){
            for (var i = 0; i < entity.Columns.length; i++) {
                _createEntityColumn(entity.Columns[i],$body, entity.Id);
            }
        }
        var $referencetables = $(".entity.id-" +entity.Id + " select.reference-
tables");
        var entities = EntityView.getEntities();
        ///
        for (var i = 0; i < entity.References.length; i++) {
            var id = entity.References[i].id;
            var targetId = entity.References[i].targetId;
            var targetName = entity.References[i].targetName;
            var referenceName = entity.References[i].referenceName;
            _createEntityReference(id, targetId, targetName, referenceName);
        }
        entity.referenceManage(entity);
        $.each(entities, function() {
            var _entity = this;
            if ( _entity.Id !=entity.Id)
                {
                    $referencetables.append($("<option
/>").val(this.Id).text(this.TableName));
                }
        });
    }
```

Kolumny tabel, też są generowane dynamicznie na podstawie wzorca w HTML. Wzorzec widoczny jest na listingu 14.

Listing 14. Schemat HTML kolumny.

```
var ColumnTemplate = `<tr class="entity-column">
                        <td>
                          <span class="column-id">{column_id}</span>
                        </td>
                        <td>
                          <span class="column-name">{column_name}</span>
                        </td>
                        <td>
                          <span class="column-type">{column_type}</span>
                        </td>
                        <td>
                          <input        class="nullable"        type="checkbox"
disabled="disabled"/>
                        </td>
                        <td>
                          <input           class="pk"           type="checkbox"
disabled="disabled"/>
                        </td>
                        <td>
                        <button type="submit" class="delete-column">x</button>
                        </td>
```

```
                    </tr>`;
```

Funkcja `_createEntityColumn`, wywoływana w opisanej wcześniej `_createEntityItem`, z wzorca `ColumnTemplate` tworzy obiekt HTML symbolizujący kolumnę, a następnie podmienia dane wzorcowe, danymi odczytanymi z serwera. Definicja funkcji `_createEntityColumn` widoczna jest na listingu 15.

Listing 15. Definicja funkcji createEntityItem.

```
function _createEntityColumn(column, $body, entityId) {
        var                          $column_temp                          =
$(ColumnTemplate.replace("{column_id}",column.id).replace("{column_name}",colu
mn.name).replace("{column_type}", column.type));
        $column_temp.find("input.nullable")[0].checked = column.nullable;
        $column_temp.find("input.pk")[0].checked = column.pk;
        $column_temp.find("button.delete-column").click(function() {
            EntityView.deleteColumn(column.id, entityId);
        });
        $body.append($column_temp);
    }
```

Aplikacja internetowa komunikuje się z wystawionym przez serwer aplikacyjny API, poprzez żądania AJAX, na lisitingu 15 przedstawiono wywołanie metody służącej do pobrania wszystkich encji użytkownika, która wywoływana jest po zalogowaniu.

Listing 16. Pobranie listy encji w kontekście użytkownika przez żądanie AJAX.

```
service.loadEntities = function(){
        $.ajax(
            {
                url: "http://localhost:3000/entities/" + userId,
                type:"GET"
        })
        .done(function(response) {

            for (var i = 0; i < response.entities.length; i++)
            {
                var entity = response.entities[i];
                jQuery.parseJSON(JSON.stringify(entity));
                _entities.push(new Entity(entity.name, entity._id, entity.left,
entity.top,                                    jQuery.parseJSON(entity.columns),
jQuery.parseJSON(entity.references),
jQuery.parseJSON(entity.sourceReferences)));
            }
            updateView();
        });
    }
```

W celu zapewnienia możliwości przesuwania elementów wykorzystana została interakcja `draggable` z biblioteki jQuery. Dzięki tej interakcji, możliwe jest przesuwanie elementów po wcześniejszym kliknięciu myszą, a sama interakcja definiuje wiele zdarzeń. Zdarzeniem, które

wykorzystywane jest w tym projekcie, jest zdarzenie `drag`, do obsługi którego zdefiniowana została funkcja `onDrag`. Na listingu 17 pokazana została implementacja tej interakcji.

Listing 17. Implementacja interakcji draggable z biblioteki jQuery.

```
var $entity = $(".entity.id-" +entity.Id);
        $entity.draggable({
            drag: function( event, ui ) {
                var   container_width   =   $_containerRoot[0].clientWidth   -
$entity[0].clientWidth;
                var   container_height   =$_containerRoot[0].clientHeight   -
$entity[0].clientHeight;
                ui.position.left = Math.max( 0, ui.position.left );
                ui.position.left = Math.min( container_width, ui.position.left
);
                ui.position.top = Math.max( 0, ui.position.top );
                ui.position.top = Math.min( container_height, ui.position.top );
                entity.onDrag(ui.position.left, ui.position.top )


            }
   });
```

Na listingu 18 pokazana została definicja funkcji `onDrag`, która jest odpowiedzialna za zmienianie i zapis położenia encji.

Listing 18. Definicja funkcji onDrag.

```
onDrag: function (left, top) {
     var self = this;
     self.Left = left;
     self.Top = top;
     EntityService.changeState(self);
     self.referenceManage(self);
    }
```

Funkcja `changeState` odpowiada za zmianę stanu encji i zapisanie go na serwerze, poprzez żądanie AJAX do udostępnionego REST API, jej definicja widoczna jest na listingu 19.

Listing 19. Definicja funkcji changeState.

```
service.changeState = function(self) {
        var entity = _entities.filter(function(entity) {
            return entity.Id === self.Id;
        })[0];
        entity.Left = self.Left;
        entity.Top = self.Top;
        entity.Columns = self.Columns;
        entity.References = self.References;
        entity.SourceReferences = self.SourceReferences;
        $.ajax(
            {
                url: "http://localhost:3000/entities/"+self.Id,
                type:"PATCH",
                data:{
                    name:self.TableName,
                    left:self.Left,
                    top:self.Top,
                    columns:JSON.stringify(self.Columns),
                    references:JSON.stringify(self.References),
                    sourceReferences:JSON.stringify(self.SourceReferences)
                }
        })
```

```
        .done(function(response) {
            console.log(response);
        });
    }
```

Połączenia między encjami symbolizowane są liniami, w celu wyznaczenia kąta i odległości między encjami, stworzona została funkcja `rotationCalculate`, w której za pomocą twierdzenia Pitagorasa i funkcji trygonometrycznych wyliczane są odległość między encjami, oraz kąt o jaki należy obrócić linie podczas przesuwania którejś z encji. Parametrami wejściowymi funkcji *`rotationCalculate`* są współrzędne połączonych ze sobą encji oraz obiekt jQuery linii, która ma być przesunięta. Na listingu 20 przedstawiona została definicja funkcji `rotationCalculate`.

Listing 20. Definicja funkcji rotationCalculate.

```
rotationCalculate(x1, x2, y1, y2, $line)
    {
        var hypotenuse = Math.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
        var angle = Math.atan2((y1-y2), (x1-x2)) *  (180/Math.PI);
        if(angle >= 90 && angle < 180){
            y1 = y1 - (y1-y2);
        }
        if(angle > 0 && angle < 90){
            x1 = x1 - (x1-x2);
            y1 = y1 - (y1-y2);
        }
        if(angle <= 0 && angle > -90){
            x1 = x1 - (x1-x2);
        }
        if ($line.length >0){
            $line.queue(function(){
                    $(this).offset({top: y1, left: x1});
                    $(this).dequeue();
                }).queue(function(){
                    $(this).width(hypotenuse);
                    $(this).dequeue();
                }).queue(function(){
                    $(this).rotate(angle);
                    $(this).dequeue();
                });
        }
    }
```

5.5 Wtyczka do dialektów SQL

Język SQL [11] został ustandaryzowany w 1986roku w ramach opracowanego przez ANSI standardu SQL:86. Dzięki czemu ogólne standardy i zasady relacyjnych baz danych dla różnych producentów są spójne. Regularnie aktualizowane standardy SQL wprowadzają porządek w nowo powstałych funkcjonalnościach. Mimo ustandaryzowania w zależności, od firm które zajmowały się rozwojem relacyjnych baz danych, powstały różne dialekty SQL, najpopularniejsze z nich to:

- MS SQL – Wprowadzony przez SysBase, rozwijany do dzisiaj przez Microsoft przez SQL Server
- PL/SQL – Wprowadzony przez firmę Oracle.
- SQL/PSM – Bardzo popularny w serwisach WWW jako MySQL.

Różnice między skryptami pisanymi w różnych dialektach są minimalne, ale znaczące na tyle, że skrypt napisany w jednym dialekcie, mógłby nie zadziałać w innym. Dlatego też w generatorze skryptów SQL, będącym tematem tej pracy, nie mogło zabraknąć uwzględnienia dialektów SQL, przed wygenerowaniem skryptu.

Aby zapewnić wysoką generyczność rozwiązania, nowe dialekty mogą być dopisywane do tego rozwiązania jako wtyczki *JavaScriptowe*. W celu napisania własnej wtyczki, wystarczy trzymać się pewnych zasad, a system sam będzie w stanie ją zaimplementować. Wtyczka powinna być:

- Obiektem o nazwie pasującej do wzorca `SQLGenerator_**Dialekt SQL**`
- Mieć publiczną metodę o nazwie `generateScript`.
- Metoda `generateScript` musi zwracać typ String.

Na listingu 21 widać sposób znajdywania funkcji generujących SQL, według podanego wyżej wzorca.

Listing 21. Zastosowanie wzorca do znalezienia funkcji generujących SQL.

```
document.addEventListener("DOMContentLoaded", function(event) {
    var globalVariables  = Object.keys(window);
    var pattern ="SQLGenerator_"
    var $sqlLanguages = $('select.sql-languages');
    var $generateButton = $('button.generate-script');

    for (var i = 0; i < globalVariables.length; i++)    {
        if (globalVariables[i].indexOf(pattern) > -1)
        {
            var f = globalVariables[i];
            var name = f.substring(pattern.length);
            $sqlLanguages.append($("<option />").val(f).text(name));
        }
    }

    $generateButton.click(function() {
        var function_name = $sqlLanguages[0].value

        var SQLScript = eval(function_name+".generateSript()");
        prompt(SQLScript +'\n\nHere is your copiable version:' ,SQLScript);
        download("script.sql", SQLScript);
    });
});
```

W zmiennej `globalVariables` zachowany jest obiekt przechowujący informacje o wszystkich zmiennych globalnych znajdujących się w projekcie. Zmienna `pattern` wskazuje wzorzec według

którego szukane są funkcje. Obiekt jQuery `$sqlLanguages` przechowuje informacje o elemencie HTML `select` z którego wybiera się dialekt SQL. Obiekt jQuery `$generateButton` przechowuje informacje o przycisku, po wciśnięciu którego wygenerowany zostaje skrypt. W pętli `for` zmienne globalne porównywane są ze wzorcem i jeśli zmienna odpowiada wzorcowi, to za pomocą funkcji `append` dopinany jest element `option` do elementu listy wyboru `select`. Następnie następuje podpięcie zdarzenia `click` do przycisku generującego skrypt, w którym wykorzystywana jest funkcja `eval` służąca do wykonywania kodu JavaScript, będącego Stringiem. Na listingu 22 pokazany jest przykład implementacji funkcji generującej skrypt.

Listing 22. Przykład funkcji generującej skrypt SQL.

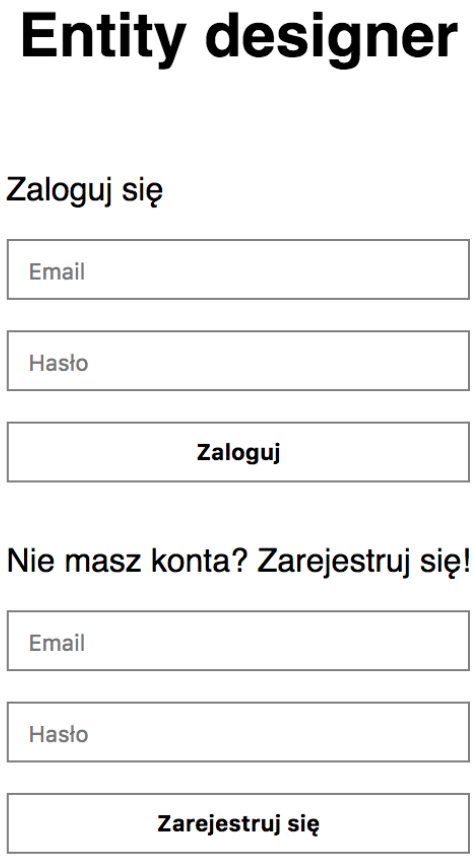
```
var SQLGenerator_TSQL = (function (generator) {

    var SQLScript='';
    // public
    generator.generateSript = function () {
        var entities = EntityService.getEntities();
        return generateTSQL(entities);

    };

    // private
    function generateTSQL(entities) {
        var date= new Date();
        SQLScript='--Script created by Entity Designer at ' +
String(date)+'\n(\n';
        for (var i = 0; i < entities.length; i++)
        {
            var entity = entities[i];
            hasPrimaryKey=false;
            var primaryKey = '\nCONSTRAINT PK_' + entity.TableName +' PRIMARY
KEY(';
            SQLScript += 'CREATE TABLE dbo.' + entity.TableName + '\n(\n';
```

6 Zastosowanie aplikacji

W tym rozdziale przedstawione zostało zastosowanie prezentowanego rozwiązania oraz wygenerowany zostanie przykładowy skrypt, do zakładania tabel utworzony za pomocą aplikacji.

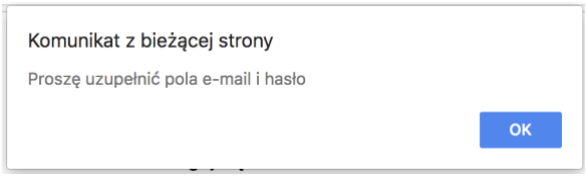
6.1 Użycie aplikacji

Pierwszym ekranem, który widzi użytkownik jest ekran logowania i rejestracji. Ekran ten pozwala na utworzenie nowego użytkownika w systemie lub zalogowanie się, jeśli już wcześniej się rejestrowaliśmy. Na rysunku 7 przedstawiony został ekran logowania.



Rysunek 7. Ekran logowania. Źródło: Opracowanie własne

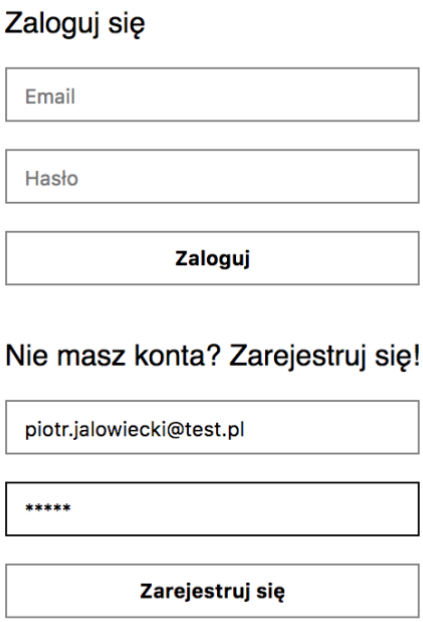
W przypadku niepodania danych lub wpisania błędnych danych aplikacja wyśle nam odpowiedni komunikat, przykłady komunikatów pokazane są na rysunkach 8 i 9.



Rysunek 8. Komunikat informujący o nieuzupełnieniu pól. Źródło: Opracowanie własne

Rysunek 9. Komunikat informujący o błędzie autoryzacji użytkownika. Źródło: Opracowanie własne

W celu zarejestrowania nowego użytkownika, należy uzupełnić pole e-mail oraz hasło, a następnie wcisnąć przycisk „Zarejestruj się", tak jak pokazano na rysunku 10.

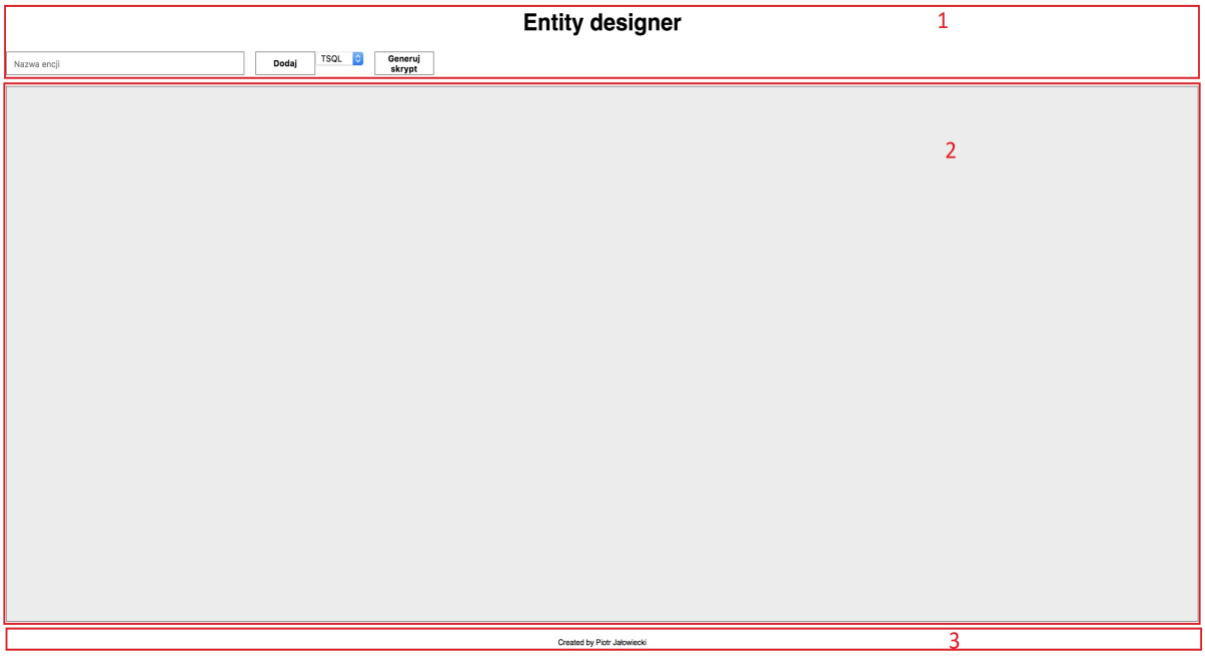


Rysunek 10. Rejestracja użytkownika. Źródło:[Źródło własne]

Na rysunku 11. Przedstawiony został ekran główny aplikacji do projektowania relacyjnych baz danych. Interfejs jest bardzo prosty, a obsługa jest intuicyjna i nie powinna sprawić problemu nawet niezaawansowanym użytkownikom. Przy projektowaniu tej aplikacji skupiono się głównie na funkcjonalności.

Ekran można podzielić na trzy główne części:

- Nagłówek – Nagłówek zawiera baner z nazwą aplikacji oraz toolbar w którym znajdują się
  - Pole „Nazwa encji” - służy do wpisania encji, która zostanie stworzona
  - Przycisk „Dodaj” - służący do dodawania nowej encji.
  - Lista wyboru dialektu SQL – domyślnie TSQL.
  - Przycisk „Generuj skrypt” - Służący do generowania skryptu SQL z utworzonego projektu.
- Pole robocze – Jest to przestrzeń, na której dodawane są nowe encje, w obrębie pola roboczego można przesuwać i przestawiać tworzony model bazy danych. Pole robocze ma szare tło, co pozwala odróżnić je od reszty aplikacji i wyraźnie zaznaczyć obszar projektowania. Nie jest możliwe przesunięcie encji poza pola robocze.
- Stopka – Informacja o autorze

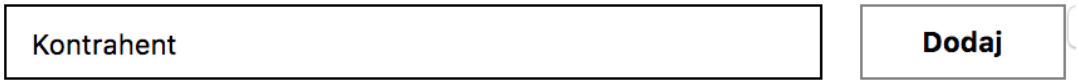


Rysunek 11. Interfejs aplikacji do projektowania baz danych. Źródło: Opracowanie własne

Aby zaprezentować możliwości aplikacji, stworzony zostanie najbardziej popularny schemat bazodanowy wykorzystywany w firmach, czyli tabele:

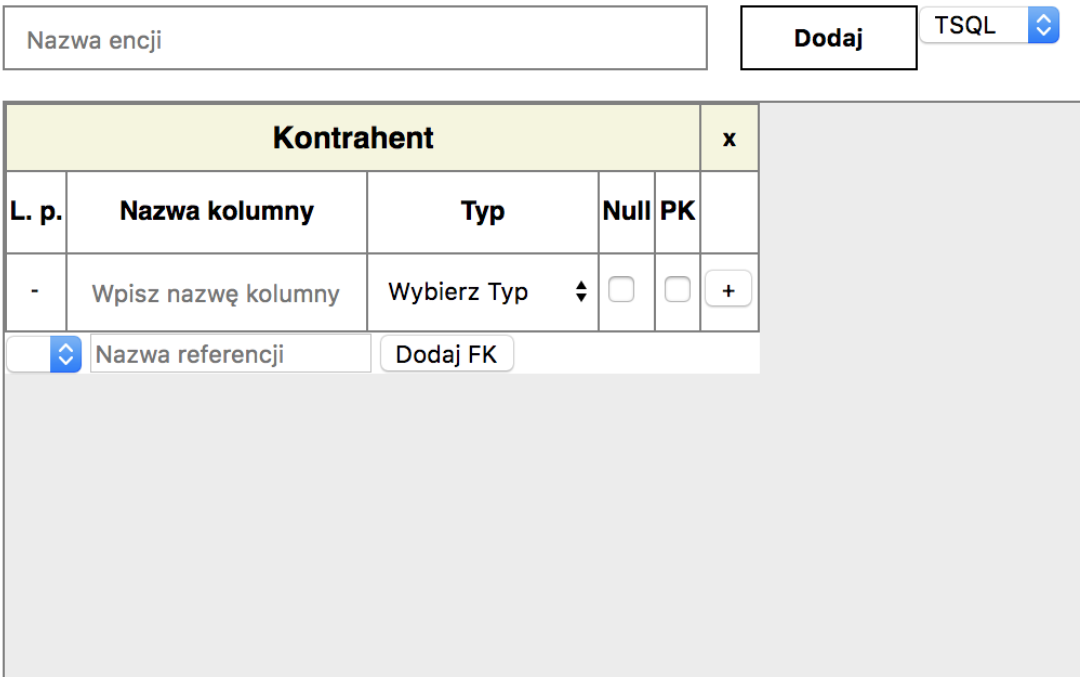
- Nagłówek dokumentu – przechowuje informacje o Numerze dokumentu, Kliencie, Dacie.
- Pozycje dokumentu – przechowuje informacje o pozycjach, cenie oraz ilości.
- Kontrahent- Przechowuje informacje o kontrahencie

W celu stworzenia nowej encji, należy wpisać nazwę, w tym przypadku „Kontrahent" a następnie wcisnąć przycisk „Dodaj", tak jak pokazano na rysunku 12:



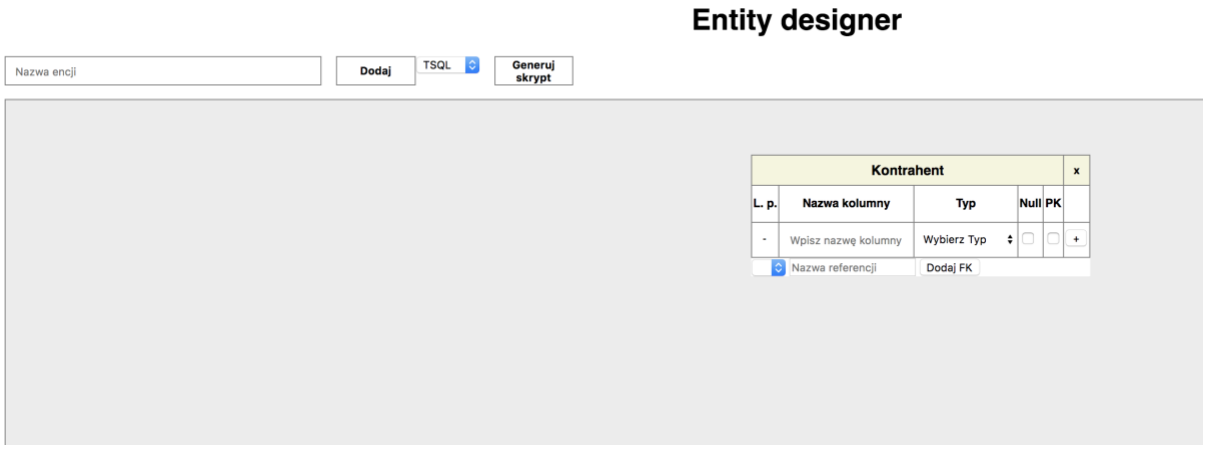
Rysunek 12. Uzupełnienie nazwy encji. Źródło: Opracowanie własne

Po wciśnięciu przycisku „Dodaj", zostaje utworzona nowa encja o nazwie „Kontrahent", efekt widoczny na rysunku 13.



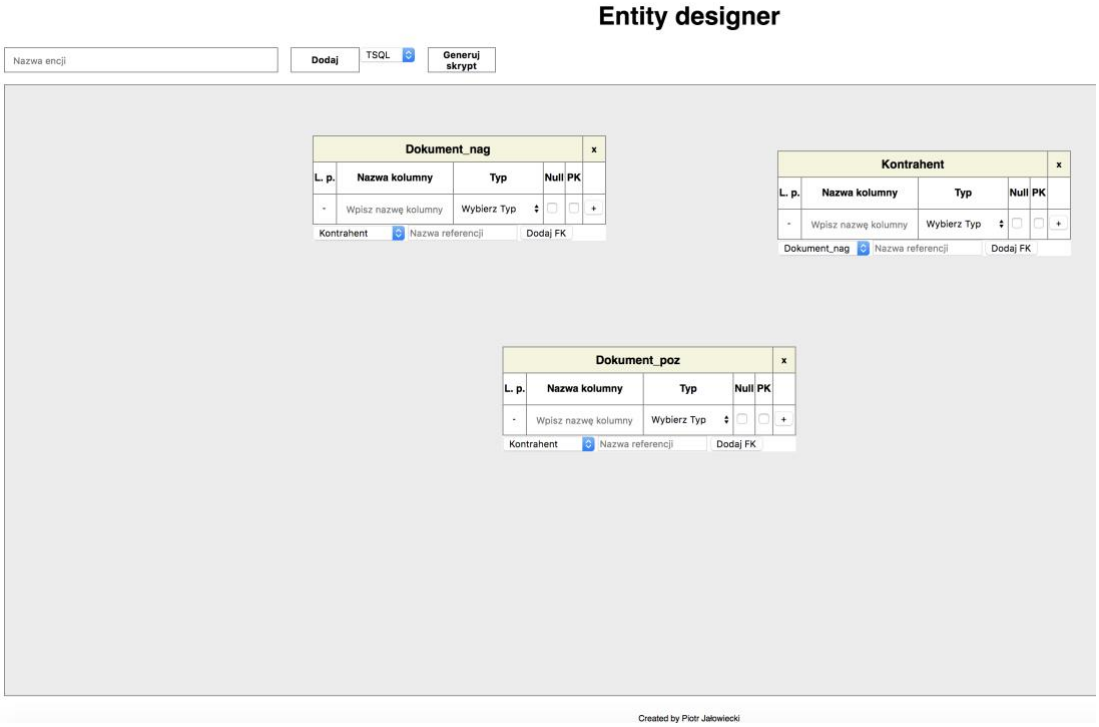
Rysunek 13. Utworzona encja „Kontrahent". Źródło: Opracowanie własne

Utworzoną encję, można przesuwać w inne miejsce na polu roboczym, poprzez kliknięcie i przeciągnięcie myszą, tak jak pokazano na rysunku 14.



Rysunek 14. Przeciąganie elementów po polu roboczym. Źródło: Opracowanie własne

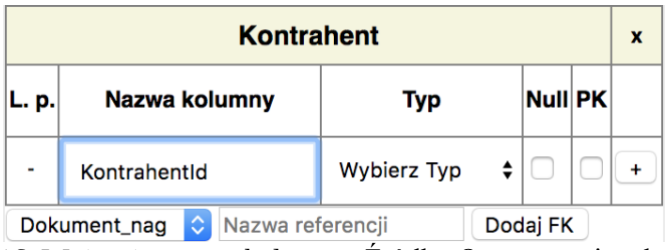
W ten sam sposób dodane zostały dwie kolejne tabele „Dokument_nag" oraz „Dokument_poz" efekty widoczne są na rysunku 15.



Rysunek 15.Utworzenie tabel „Dokument_nag" i „Dokument_poz". Źródło: Opracowanie własne

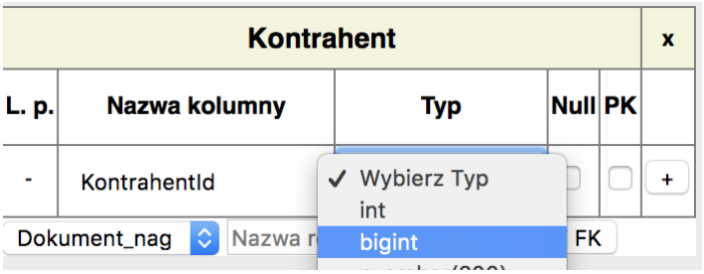
Następnym etapem jest dodanie do tabel kolumn. W tym celu należy:

- Wpisać nazwę kolumny, jak na rysunku 16.



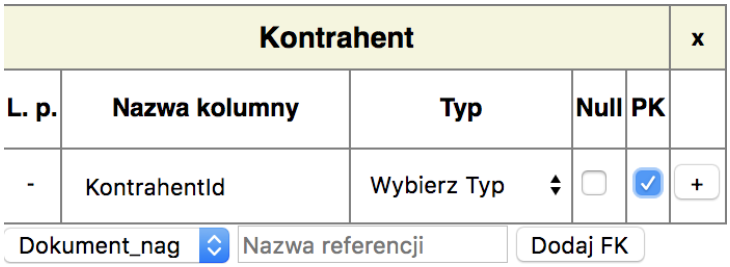
Rysunek 16. Wpisanie nazwy kolumny. Źródło: Opracowanie własne

- Wybrać typ pola SQL z listy, jak na rysunku 17.



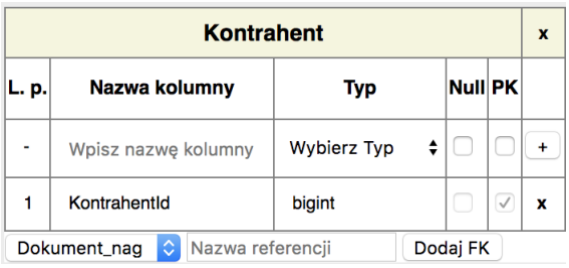
Rysunek 17. Wybór typu SQL. Źródło: Opracowanie własne

- Oznaczyć poprzez checkbox'y czy pole może zawierać wartości puste „null" (znacznik „null") oraz czy ma się zawierać w unikalnym kluczu głównym Primary Key(znacznik „PK"), jak na rysunku 18.



Rysunek 18. Znaczniki „Null" i „PK". Źródło: Opracowanie własne

Następnie należy wcisnąć przycisk „+" w celu utworzenia kolumny. Tak jak pokazano na rysunku 19.



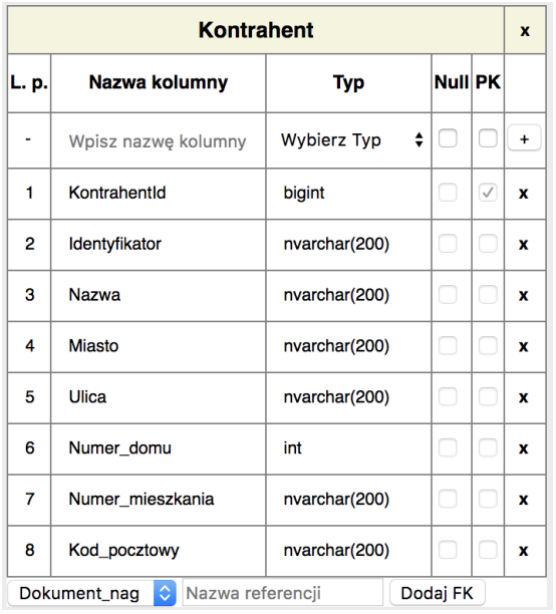
Rysunek 19. Dodawanie kolumny. Źródło: Opracowanie własne

W celu usunięcia wcześniej dodanej kolumny, należy wcisnąć przycisk „X” na konkretnej kolumnie. Jak na rysunku 20.

Rysunek 20. Usuwanie kolumny. Źródło: Opracowanie własne

W analogiczny sposób do tabeli „Kontrahent” dodane zostały następujące kolumny, efekt widoczny na rysunku 21:

- Identyfikator – skrócona nazwa kontrahenta
- Nazwa – Pełna nazwa
- Miasto
- Ulica
- Numer Domu
- Numer Mieszkania
- Kod pocztowy



Kontrahent					x
L. p.	Nazwa kolumny	Typ	Null	PK	
-	Wpisz nazwę kolumny	Wybierz Typ	☐	☐	+
1	KontrahentId	bigint	☐	☑	x
2	Identyfikator	nvarchar(200)	☐	☐	x
3	Nazwa	nvarchar(200)	☐	☐	x
4	Miasto	nvarchar(200)	☐	☐	x
5	Ulica	nvarchar(200)	☐	☐	x
6	Numer_domu	int	☐	☐	x
7	Numer_mieszkania	nvarchar(200)	☐	☐	x
8	Kod_pocztowy	nvarchar(200)	☐	☐	x

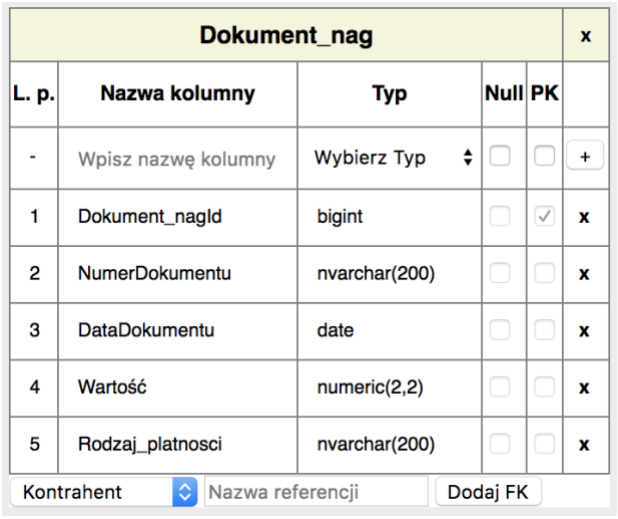
Dokument_nag | Nazwa referencji | Dodaj FK

Rysunek 21. Tabela „Kontrahent” z uzupełnionymi kolumnami. Źródło: Opracowanie własne

Do tabeli „Dokument_nag” dodano kolumny:

- Numer dokumentu
- Data dokumentu
- Wartość
- Rodzaj płatności

Efekt widoczny na rysunku 22.



Dokument_nag					x
L. p.	Nazwa kolumny	Typ	Null	PK	
-	Wpisz nazwę kolumny	Wybierz Typ	☐	☐	+
1	Dokument_nagId	bigint	☐	☑	x
2	NumerDokumentu	nvarchar(200)	☐	☐	x
3	DataDokumentu	date	☐	☐	x
4	Wartość	numeric(2,2)	☐	☐	x
5	Rodzaj_platnosci	nvarchar(200)	☐	☐	x

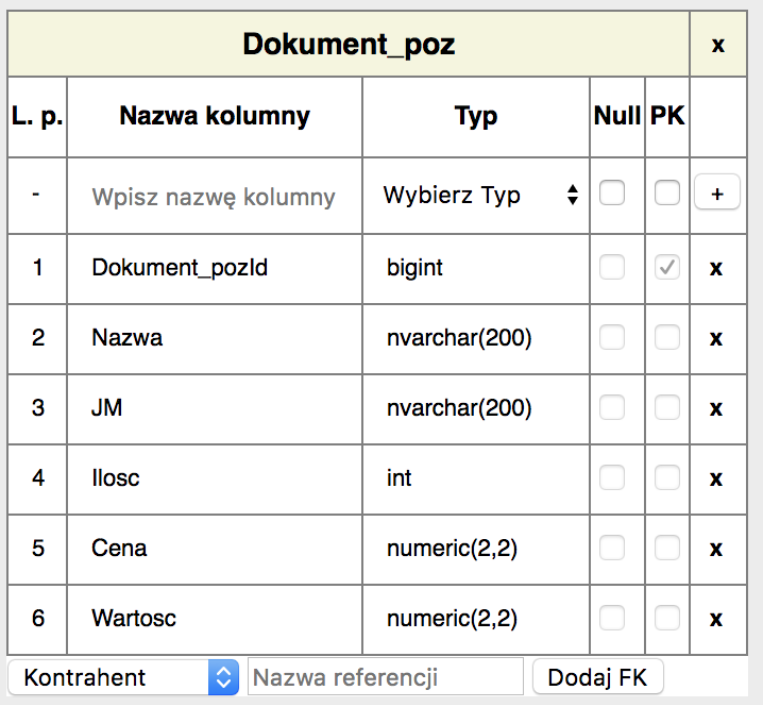
Kontrahent | Nazwa referencji | Dodaj FK

Rysunek 22. Tabela „Dokument_nag” z uzupełnionymi kolumnami. Źródło: Opracowanie własne

Do tabeli „Dokument_poz” dodano:

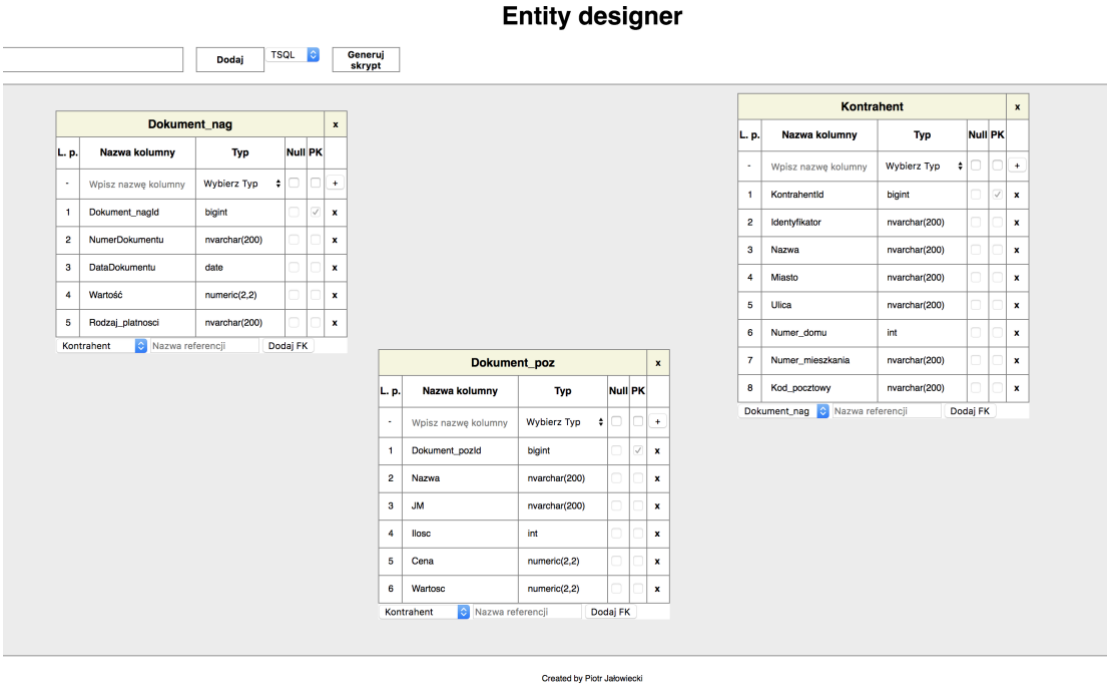
- Nazwa pozycji
- Jednostka miary
- Ilość
- Cena
- Wartość

Efekt widoczny na rysunku 23.



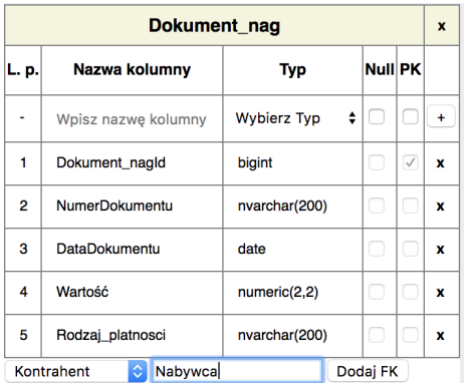
Rysunek 23. Tabela „Dokument_poz” z uzupełnionymi kolumnami. Źródło: Opracowanie własne

Na rysunku 24 przedstawiono schemat projektu po uzupełnieniu kolumn w tabelach, brakującym elementem prawidłowego schematu bazodanowego są relacje między tabelami.



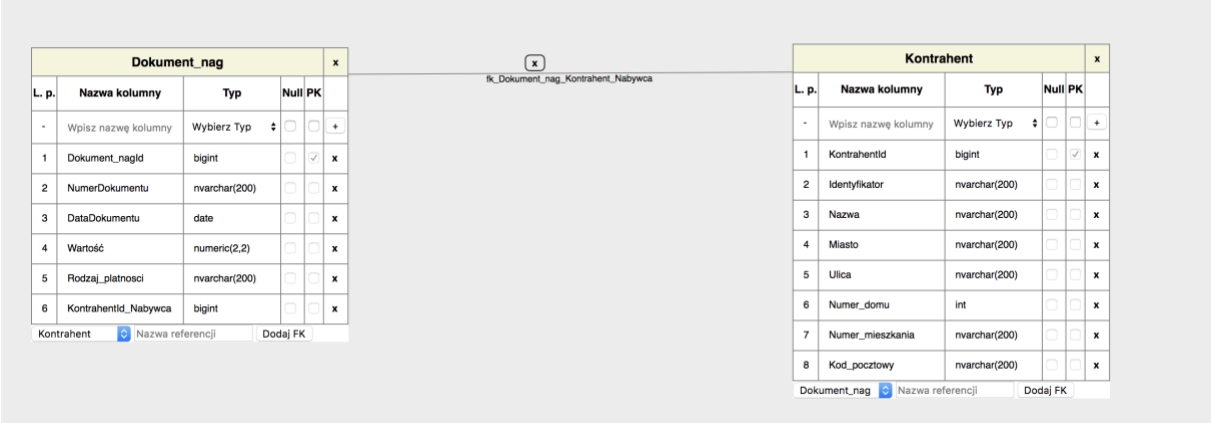
Rysunek 24. Schemat projektu, po dodaniu kolumn. Źródło: Opracowanie własne

W celu utworzenia klucza obcego, należy wybrać tabelę z listy wyboru, następnie należy wpisać nazwę referencji i wcisnąć przycisk „Dodaj FK”, tak jak pokazano na rysunku 25.



Rysunek 25. Dodawanie klucza obcego do tabeli. Źródło: Opracowanie własne

Klucze obce zobrazowane są jako czerwone linie, z nazwą referencji. Po dodaniu relacji, do tabeli źródłowej dodane zostają kolumny będące w kluczu głównym tabeli docelowej. Efekt widoczny na rysunku 27

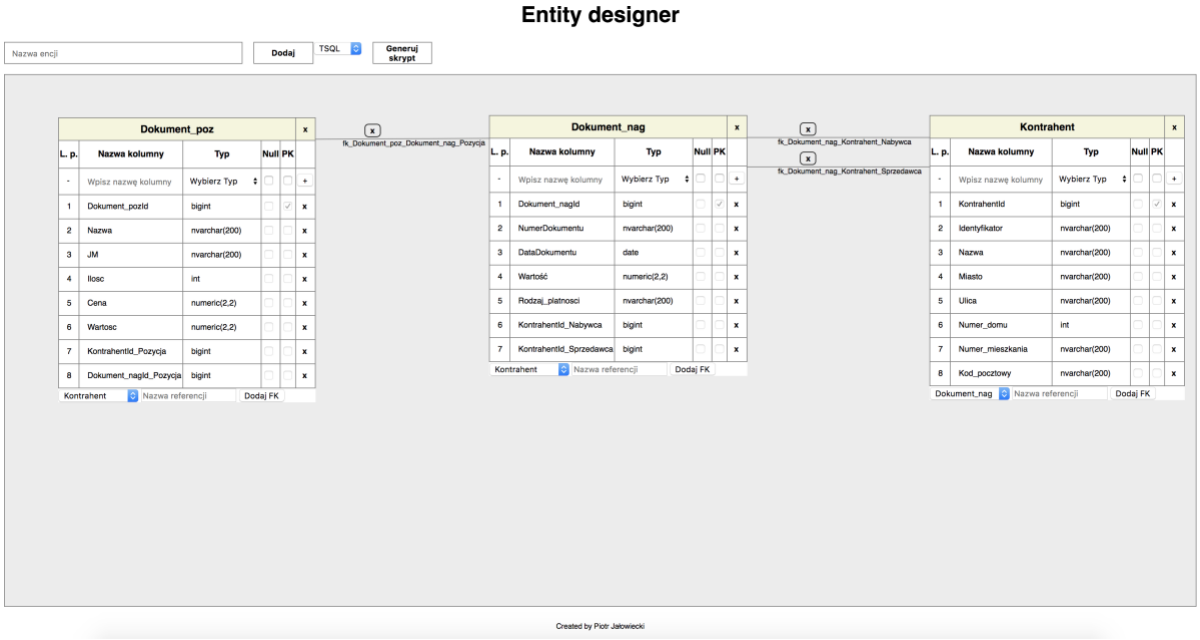


Rysunek 26. Relacja pomiędzy tabelami „Dokument_nag” oraz „Kontrahent”. Źródło: Opracowanie własne

Aby usunąć referencję, należy wcisnąć przycisk „X” znajdujący się nad linią symbolizującą relację, tak jak pokazano na rysunku 27.

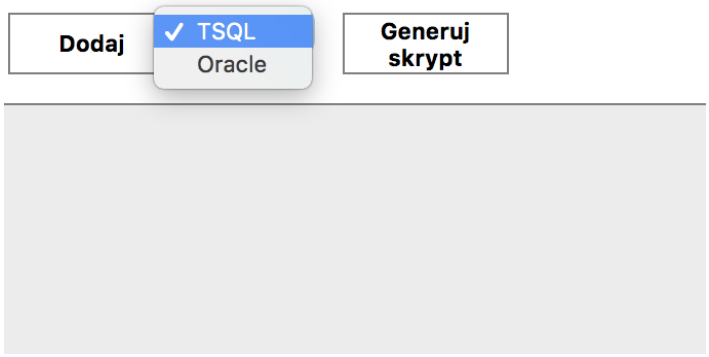
Rysunek 27. Usuwanie relacji. Źródło: Opracowanie własne

Nie ma ograniczenia co do ilości dodawanych referencji do tabel, w przypadku tabeli „Dokument_nag”, oprócz wcześniej utworzonej relacji „Nabywca”, dodana zostanie relacja „Sprzedawca”. Natomiast do tabeli „Dokument_poz”, dodany zostanie klucz obcy do tabeli „Dokument_nag”. Efekty przedstawiono na rysunku 28.



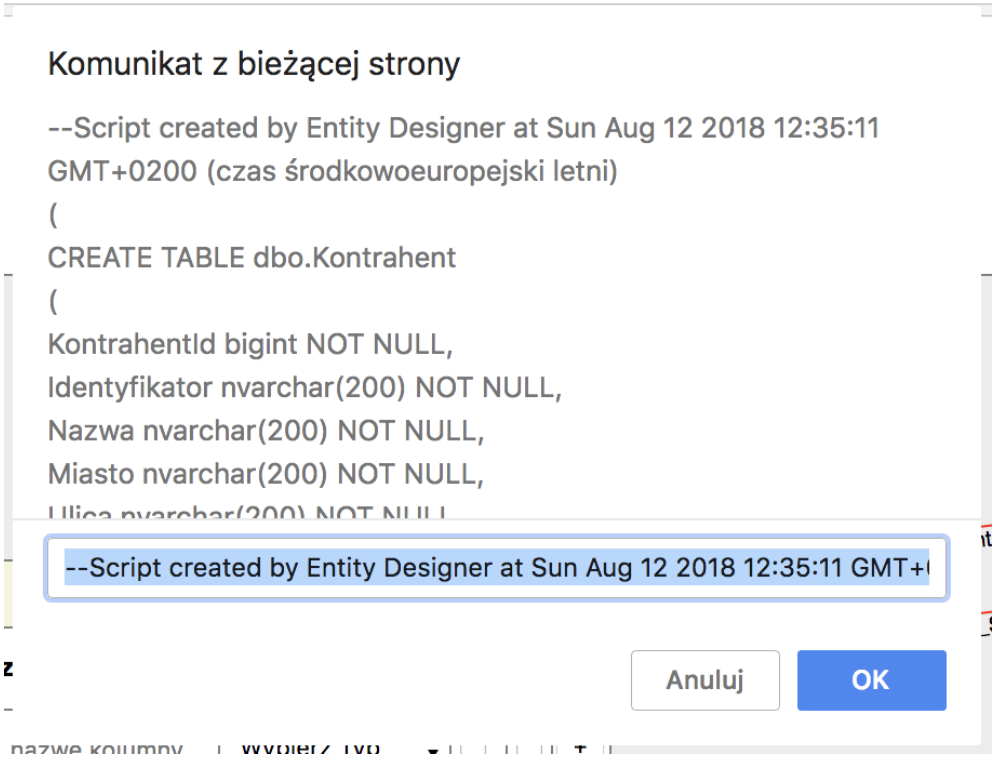
Rysunek 28. Schemat bazy danych, po uzupełnieniu o relacje. Źródło: Opracowanie własne

Po uzupełnieniu projektu o odpowiednie relacje, można przejść do wygenerowania skryptu SQL ze schematu. W tym celu, należy z listy wyboru wybrać odpowiedni dialekt SQL i wcisnąć przycisk „Generuj Skrypt”, tak jak pokazano na rysunku 29.



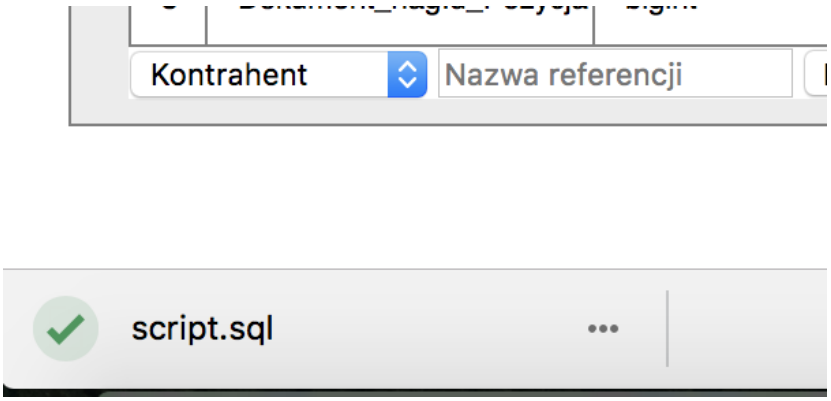
Rysunek 29. Generowanie skryptu SQL. Źródło: Opracowanie własne

Po wciśnięciu przycisku, pojawia się komunikat ze strony, zawierający wygenerowany skrypt, widoczny na rysunku 30.



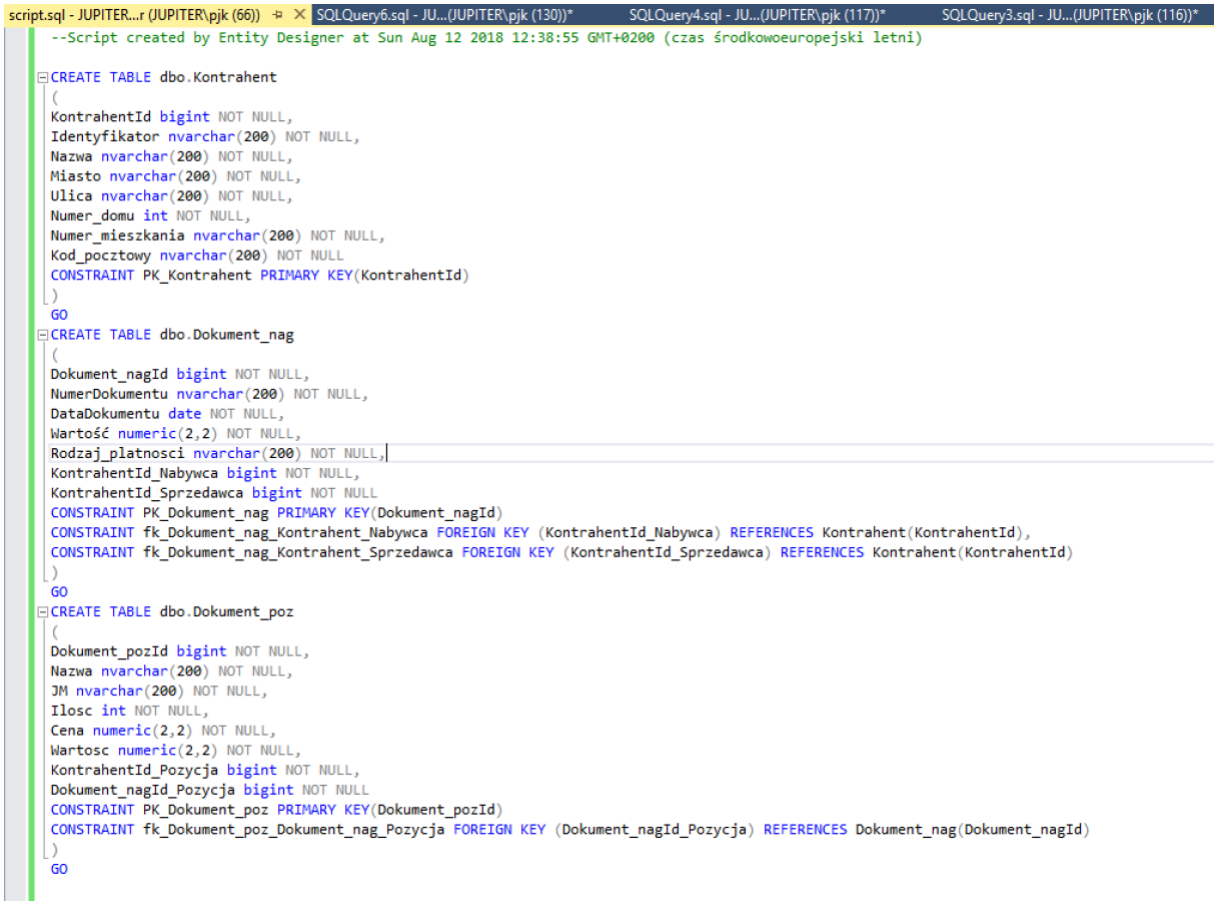
Rysunek 30. Komunikat z wygenerowanym skryptem. Źródło: Opracowanie własne

Skrypt można skopiować z pola edycyjnego lub zamknąć komunikat, co spowoduje automatyczne pobranie skryptu na dysk w postaci pliku z rozszerzeniem *sql*, efekt widoczny na rysunku 31.



Rysunek 31.Pobrany skrypt na dysk. Źródło: Opracowanie własne

Wygenerowany skrypt można następnie otworzyć w kliencie bazodanowym np. Microsoft SQL Server Managment Studio, aby go wykonać i utworzyć tabele. Skrypt otwarty w MSSMS pokazano na rysunku 32.



Rysunek 32. Wygenerowany skrypt otwarty w MSSMS. Źródło: Opracowanie własne

7 Podsumowanie

W tym rozdziale opisane zostały wady i zalety rozwiązania, możliwe kierunki rozwoju oraz podsumowana została zgodność przedstawionego rozwiązania z analizowanym w pracy problemem.

7.1 Wady i zalety

Wady:

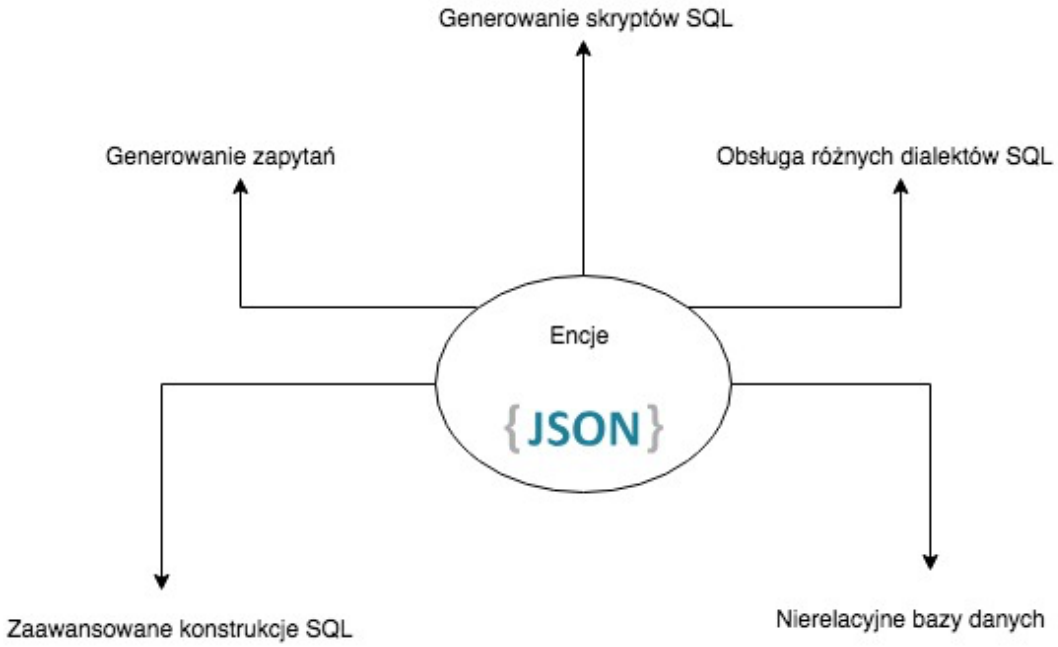
- Brak możliwości jednoczesnej pracy nad kilkoma projektami przez jednego użytkownika, zmusza to do ukończenia pracy nad jednym projektem, przed rozpoczęciem następnego.
- Brak potwierdzenia mailowego rejestracji oraz miejsca do wprowadzania szczegółowych danych przez użytkownika.
- Brak możliwości rejestracji przez Facebook'a lub Google
- Prototyp nie jest dostosowany do pracy na urządzeniach mobilnych
- Brak obsługi ciasteczek i zapisywania plików w pamięci przeglądarki klienta, co przyczyniłoby się do szybszej pracy, przy wolnym łączu.

Zalety:

- Rozwiązanie jest intuicyjne i łatwe w użyciu, nawet niedoświadczony użytkownik nie powinien mieć problemu z obsługą.
- Prototyp został stworzony w oparciu o nowoczesne technologie, dzięki temu przez długi czas zachowana zostanie możliwość rozwoju.
- Rozwiązanie przyczynia się do znacznego uproszczenia pracy, przy projektowaniu relacyjnych baz danych
- Dzięki udostępnionemu API, aplikacja może być rozwijana na różne sposoby, również na urządzenia mobilne.
- Rozwiązanie jest konkurencyjne, dla przedstawionych rozwiązań innych firm
- Prototyp jest zbudowany w sposób umożliwiający dalszy rozwój

7.2 Możliwości rozwoju prototypu

Na rysunku 33 przedstawiono możliwe kierunki rozwoju aplikacji do projektowania schematu baz danych.



Rysunek 33. Możliwe kierunki rozwoju aplikacji Źródło: Opracowanie własne

- **Generowanie skryptów SQL –** Oprócz już udostępnionych możliwości generowania skryptów SQL do zakładania tabel, prototyp mógłby zostać rozbudowany o możliwość zakładania dodatkowych indeksów nieklastrowanych, które mają wpływ na wydajność. Klucze obce można rozbudować o dodawania CASCADE, czyli usuwanie rekordów w przypadku usunięcia rodzica.

- **Generowanie zapytań –** Z danych zgromadzonych w formacie JSON o encjach, można stworzyć generator zapytań. Generator powinien umożliwiać wybór kolumny z pola wyboru, a następnie sam powinien generować zapytanie SQL, tworząc potrzebne przy tym złączenia.

- **Zaawansowane konstrukcje SQL –** Po rozwinięciu aplikacji o wspomniane wcześniej moduły, projekt można rozszerzyć o możliwość pracy nad bardziej zaawansowanymi konstrukcjami SQL. Przykładem tego może być generator indeksów pełnotekstowych do wyszukiwania.

- **Obsługa różnych dialektów SQL** – Aplikacja powinna zostać rozwinięta o możliwość generowania skryptów w reszcie dialektów SQL, do tej pory została udostępniona obsługa T-SQL, oraz Oracle

- **Nierelacyjne bazy danych –** Na podstawie stworzonych encji, możliwe jest przedstawienie wyniku jako interpretacja w nierelacyjnej bazie danych. Przykładem tego mogą być schematy obiektów w bazach JSON np. MongoDB, lub schematy do baz grafowych.

Oprócz kierunków rozwoju przedstawionych powyżej, aplikacja powinna zostać rozwinięta o panel zarządzania projektami, konto użytkownika i rejestracje z użyciem portali społecznościowych.

7.3 Podsumowanie

Praca przedstawiła problematykę tematu projektowania relacyjnych baz danych, pokazała dostępne na rynku rozwiązania. Po przeanalizowaniu konkurencyjnych produktów i wyciągnięciu mocnych stron tych rozwiązań, podjęta została próba zaprojektowania własnego prototypu. Rozwiązanie zaprezentowane w pracy wyróżnia się intuicyjnym interfejsem, prostotą użytkowania. Dzięki temu, że rozwiązanie jest aplikacją internetową, nie ma potrzeby pobierania oprogramowania na dysk komputera, a cała obsługa odbywa się w przeglądarce. Skrypty wygenerowane za pomocą tej aplikacji, mogą zostać umieszczone w kliencie bazodanowym i być wykorzystane do założenia bazy danych.

Rozwiązanie to jest zaprojektowane z myślą o dalszym rozwoju, a mimo to już w tej chwili odpowiada na problem przedstawiony w pracy i może konkurować z innymi rozwiązaniami dostępnymi na rynku.

Bibliografia

[1]. Wikipedia, https://pl.wikipedia.org/wiki/Aplikacja_internetowa Dostęp: 2018-08-18.

[2]. Vertabelo, http://www.vertabelo.com/, Dostęp: 2018-08-18.

[3]. SquirrelSQL, http://squirrel-sql.sourceforge.net/

[4]. DBDesigner, https://www.dbdesigner.net/

[5]. Developers Mozilla, https://developer.mozilla.org/pl/docs/Learn/Getting_started_with_the_web/HTML_basics Dostęp: 2018-08-18.

[6]. Jeffrey Zeldman, Projektowanie serwisów WWW. Standardy sieciowe. Helion, ISBN 83-246-0774-9

[7]. Marijn Haverbeke, *Zrozumieć Javascript*, Strony 25-25, ISBN 978-83-283-0969-2

[8]. Jake Rutter, *Podręcznik jQuery*. Strony 13-14, ISBN 978-83-246-3316-6

[9]. JMarshall, http://www.jmarshall.com/easy/http/ Dostęp: 2018-08-18.

[10]. Tom Hughes-Croucher,Mike Wilson, Up and Running with Node.js, Sebastaspol, ISBN 978-1-4493-9858-3

[11]. ExpressJS, https://expressjs.com/ Dostęp: 2018-08-18.

[12]. Acedemind, Youtube, Dostęp: 2018-08-18, „What is RESTful API”, https://www.youtube.com/watch?v=0oXYLzuucwE

[13]. MongoDB Official Site, https://www.mongodb.com/ Dostęp: 2018-08-18.

[14]. SQLPedia, https://www.sqlpedia.pl/jezyk-sql-historia-standardy/

[15]. Draw.IO, https://draw.io Dostęp: 2018-08-18.

[16]. MongoDB Cloud, https://cloud.mongodb.com/ Dostęp: 2018-08-18.

Spis tabel

Spis rysunków

Spis listingów