P O L I T E C H N I K A Ś W I Ę T O K R Z Y S K A
Wydział Elektrotechniki, Automatyki i Informatyki

Mariusz Mularczyk, Hubert Ptaszek
Grupa: 1ID21B
Studia stacjonarne II stopnia

Technologie obiektowe

Projekt – Narzędzie odwzorowania schemat - skrypt

Kielce, 17.06.2021

Spis treści

1. Wstęp

Tematem projektu było połączenie dwóch istniejących już tematów („Narzędzia odwzorowania obiektowo – relacyjnego” oraz „Wizualizacja zawartości składu lub bazy danych”) czego rezultatem jest projekt pod tytułem „Narzędzie odwzorowania schemat - skrypt”.

Projekt składa się z dwóch części, którymi są: opis teoretyczny problemu oraz aplikacja pozwalająca na graficzne budowanie bazy danych z poziomu przeglądarki internetowej. Aplikacja umożliwia przetwarzanie rezultatów pracy do skryptu SQL zapisywanego w formie pliku lub uruchamianego na serwerze zdalny.

2. Problematyka

2.1. Wstęp

W bazie danych przechowujemy tylko niektóre informacje o świecie rzeczywistym. Wybór właściwych wycinków rzeczywistości i dotyczących ich danych jest bardzo istotny — od niego zależy prawidłowe działanie bazy. Aby ten wybór był właściwy, należy wskazać informacje, które powinny być przechowywane w bazie danych, oraz określić ich strukturę.
Cały proces projektowania bazy danych możemy podzielić na kilka etapów:

- planowanie bazy danych,
- tworzenie modelu konceptualnego (diagramu ERD),
- transformacja modelu konceptualnego na model relacyjny,
- proces normalizacji bazy danych,
- wybór struktur i określenie zasad dostępu do bazy danych.

Z punktu widzenia relacyjnej bazy danych świat rzeczywisty widzimy i analizujemy jako zestaw encji i związków zachodzących między nimi.

- Encja - encją jest każdy przedmiot, zjawisko, stan lub pojęcie, czyli każdy obiekt, który potrafimy odróżnić od innych obiektów (na przykład: osoba, samochód, książka, stan pogody).
  Encje podobne do siebie (opisywane za pomocą podobnych parametrów) grupujemy w zbiory encji. Projektując bazę danych, należy precyzyjnie zdefiniować encje i określić parametry, przy użyciu których będą opisywane.
- Atrybut - encje mają określone cechy wynikające z ich natury. Cechy te nazywamy atrybutami. Zestaw atrybutów, które określamy dla encji, zależy od potrzeb bazy danych.
- Dziedzina - atrybuty encji mogą przyjmować różne wartości. Projektując bazę danych, możemy określić, jakie wartości może przyjmować dany atrybut. Zbiór wartości atrybutu nazywamy dziedziną (domeną).

Ponieważ trudno uniknąć błędów podczas wprowadzania danych, należy odpowiednio zabezpieczyć się przed nimi na etapie projektowania bazy danych.
Należy pamiętać, że prawidłowa klasyfikacja danych jest podstawą dobrego projektu bazy danych. Najgorsze efekty otrzymujemy, próbując zaprojektować bazę danych, która będzie gromadziła wszystkie możliwe dane i przetwarzała je na wszystkie możliwe sposoby.
Każda encja powinna mieć przynajmniej jeden atrybut lub kombinację kilku atrybutów, które identyfikują ją jednoznacznie. Ten atrybut to klucz podstawowy encji.
Jeśli klucz podstawowy składa się z kilku atrybutów, dobrym rozwiązaniem jest zastąpienie go kluczem sztucznym. Najczęściej zawiera on unikatowe liczby przypisane kolejnym encjom.

2.2. Diagramy związków encji

Konceptualne projektowanie bazy danych to konstruowanie schematu danych niezależnego od wybranego modelu danych, docelowego systemu zarządzania bazą danych, programów użytkowych czy języka programowania.
Do tworzenia modelu graficznego schematu bazy danych wykorzystywane są diagramy związków encji, z których najpopularniejsze są diagramy ERD (ang. Entity Relationship Diagram). Pozwalają one na modelowanie struktur danych oraz związków zachodzących między tymi strukturami. Nadają się szczególnie do modelowania relacyjnych baz danych, ponieważ umożliwiają prawie bezpośrednie przekształcenie diagramu w schemat relacyjny.

Diagramy ERD składają się z trzech rodzajów elementów:

- zbiorów encji,
- atrybutów encji,
- związków zachodzących między encjami.

Encja to reprezentacja obiektu przechowywanego w bazie danych.

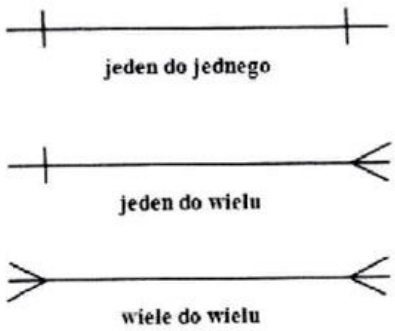
Rysunek 2.1 Graficzna reprezentacja encji

Atrybut opisuje encję. Może on być liczbą, tekstem lub wartością logiczną.

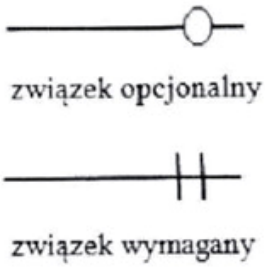
Rysunek 2.2 Graficzna reprezentacja atrybutów dla encji Klient

Związek jest to powiązanie między dwiema lub kilkoma encjami. Każdy związek ma dwa końce, do których są przypisane następujące atrybuty:

- nazwa,
- stopień związku,
- uczestnictwo lub opcjonalność związku. Atrybut ten określa, czy związek jest opcjonalny, czy wymagany.

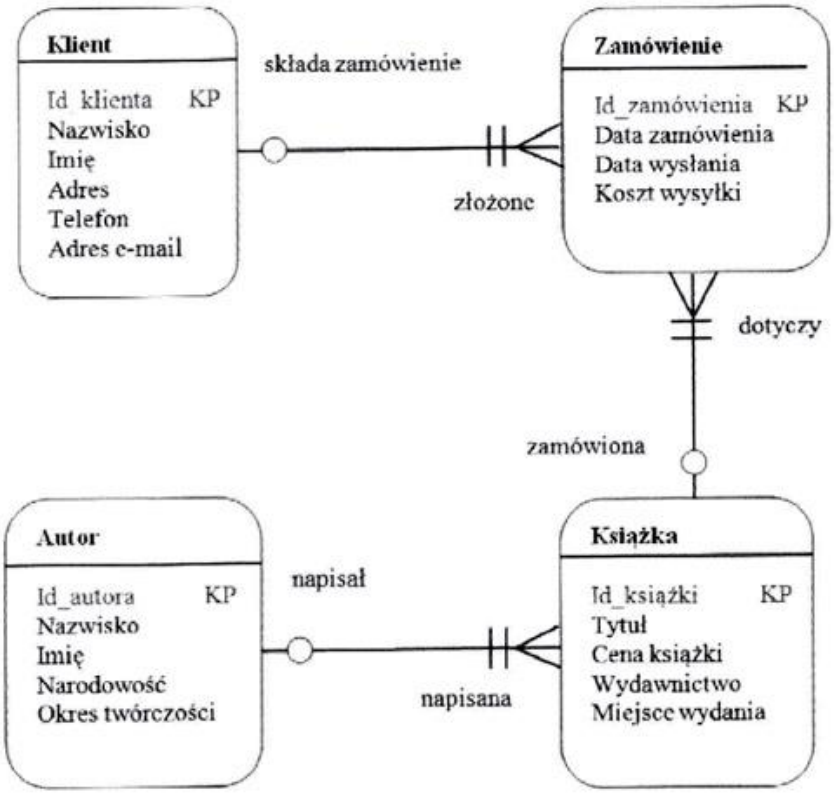


Rysunek 2.3 Graficzna reprezentacja związków zachodzących między encjami



Rysunek 2.4 Graficzna reprezentacja związków zachodzących między encjami

Diagramy ERD można spotkać w różnych notacjach, na przykład: Bachmana, Chena, IDEFIX. Istnieje wiele narzędzi wspomagających rysowanie diagramów ERD, ale jedynie w przypadku narzędzi klasy CASE (ang. Computer Aided Software Engineering) można mówić o określonej notacji.



Rysunek 2.5 Prosty przykład diagramu ERD w notacji Martina dla księgarni internetowej

Rysunek 2.5 przedstawia schemat encji (przedstawione za pomocą zaokrąglonych prostokątów zawierających listę atrybutów). Klucze główne zostały oznaczone jako KP. Stopień związku i uczestnictwo z kolei są oznaczone liniami łączącymi z odpowiednimi symbolami opisującymi stopień oraz opcjonalność związku. Należy zwrócić uwagę na to, że w encjach nie umieszcza się kluczy obcych. Tak przygotowany diagram ERD pozwała na późniejszą weryfikację i optymalizację bazy danych, a także stanowi podstawową dokumentację projektowanej bazy danych. Można go również wykorzystać w jednym z narzędzi CASE do wygenerowania fizycznej struktury bazy danych.

2.3 Reguły projektowania tabel

- Do opisu każdego zbioru podobnych encji stosuje się oddzielną tabelę. jednej encji odpowiada jeden wiersz. Atrybutowi odpowiada kolumna. Dla każdego atrybutu określa się typ informacji.
- Do opisu każdego dwustronnego związku między encjami można użyć oddzielnej tabeli. Kolumny tabeli są tworzone z kluczy encji należących do związku.
- Zapis związku jeden do jednego lub wiele do jednego może być umieszczony w dodatkowych kolumnach tabel pozostających w związku (nie trzeba tworzyć oddzielnej tabeli do opisu tego związku). W przypadku związku jeden do jednego kolumna ta może znaleźć się w dowolnej tabeli, w przypadku związku wiele do jednego musi znaleźć się w tabeli ze strony „wiele". Dołączona kolumna zawiera klucz encji, z którą zachodzi związek.
- Związek wiele do wielu opisuje się w oddzielnej tabeli, której kolumny tworzone są z kluczy encji należących do związku.
- Jeśli klucze w tabeli opisującej związek składają się z wielu atrybutów lub są długie, należy zastąpić je kluczami sztucznymi.

3. Technologie

3.1. ASP.NET Framework v4.7.2

Jest to platforma programistyczna, za pośrednictwem której tworzone są aplikacje pod systemy z rodziny Windows. Framework udostępnia środowisko uruchomieniowe oraz biblioteki klas, które dostarczają zbiór narzędzi i funkcji.

Podczas pracy z .NET Framework wykorzystywany jest pakiet Microsoft Visual Studio, w którym to jest możliwość programowania w jednym z obsługiwanych języków: C++, C#, F#, Visual Basic .NET. Zadaniem platformy .NET Framework jest zarządzanie różnymi elementami systemu: kodem aplikacji, pamięcią i zabezpieczeniami. W środowisku tym można tworzyć oprogramowanie działające po stronie serwera internetowego (IIS) oraz pracujące na systemach, na które istnieje działająca implementacja tej platformy.
Biblioteki składowe platformy .NET to:

- **CLR** (ang. Common Language Runtime) odpowiedzialny za lokalizowanie, wczytywanie oraz zarządzanie typami .NET. To trzon całej platformy .NET ponieważ to właśnie do CLR należy zadanie kompilowania i uruchamiania kodu zapisanego językiem kodu pośredniego (CIL).
- **CTS** (ang. Common Type System) jest odpowiedzialny za opis wszystkich danych udostępnianych przez środowisko uruchomieniowe.
- **CLS** (ang. Common Language Specification) to zbiór zasad definiujących podzbiór wspólnych typów precyzujących zgodność kodu binarnego z dostępnymi kompilatorami .NET

3.2. Entity Framework

Entity Framework jest narzędziem mapowania obiektowo-relacyjnego, tj. ORM. Generuje obiekty biznesowe oraz encje zgodnie z tabelami baz danych. Obiekt biznesowy jest encją (entity) w wielowarstwowej aplikacji, która działa w połączeniu z dostępem do bazy danych oraz warstwą logiki biznesowej służącą do przesyłania danych. Z kolei nasze entity odnosi się do czegoś co jest unikatowe i istnieje oddzielnie, np. tabela bazy danych. Entity Framework pozwala na:

- Wykonywanie podstawowych operacji CRUD (Create, Read, Update, Delete).
- Łatwe zarządzanie relacjami 1 do 1, 1 do wielu oraz wiele do wielu.
- Tworzenie relacji dziedziczenia pomiędzy encjami.

3.3. LINQ

LINQ, czyli Language INtegrated Query jest częścią technologii Microsoft .NET, która umożliwia zadawanie pytań na obiektach. Składnia języka LINQ jest prosta i przypomina SQL. LINQ stanowi warstwę abstrakcji nad różnymi źródłami danych. Baza danych i jej elementy traktowane są również jak obiekty. Przestrzenie, które obsługuje LINQ, to:

- obiekty implementujące interfejs IEnumerable<T>
- bazy danych
- język XML

3.4. Preprocecors sass

SASS jest to preprocesor CSS pozwalający tworzyć arkusze stylów szybciej i dużo czytelniej. Plik zawierający kod SASS zapisywany jest z rozszerzeniem .sass bądź .scss i wzbogaca napisanie arkuszy stylów o dodatkowe funkcje takie jak zmienne, albo obliczenia. Dzięki temu pozwala skupić się na najważniejszych aspektach pracy, czyli tworzeniu. Wszystko kompilowane jest na lokalnym komputerze developera do postaci CSS'a gotowego do podpięcia na stronę. Dzięki wielostopniowym zagnieżdżaniu się elementów, wstawiania fragmentów kodu za pomocą funkcji mixin i wielu innych przydatnych funkcjonalności w SASS, unikamy powielania fragmentów kodu, co pozwala zaoszczędzić czas oraz czytelność.

3.5. jQuery

Jest to biblioteka JavaScript, która ułatwia manipulację elementami i zdarzeniami w dokumencie HTML. Upraszcza wybieranie elementów z drzewa DOM, manipulacje nimi (np. kopiowanie, usuwanie ukrywanie, zmianę kolejności i właściwości) oraz animację i upraszcza korzystanie z AJAX (ang. Asynchronous JavaScript and XML).
W praktyce, jQuery dostarcza gotowych, często wykorzystywanych funkcji w skryptach JavaScript dzięki czemu:

- upraszcza/ułatwia/przyspiesza pracę programiście,
- zwiększa kompatybilność kodu pod kątem wielu platform i wersji przeglądarek.

3.6. Ajax

AJAX nie jest językiem programowania, tylko technologią, która wykorzystuje inne języki. Wykorzystuje się go wtedy, gdy nie chcemy odświeżać całej strony tylko jej część, czyli asynchronicznie z całą stroną. Oczywiście w takim przypadku dane pobierane są z pliku txt lub XML umieszczonego na serwerze za pomocą jakiś skryptów np. JavaScript. Takim to sposobem rozszyfrowaliśmy dziwną nazwę AJAX, czyli Asynchroniczny JavaScript i XML.

3.7. Razor

Jest to silnik renderujący wprowadzony w MVC3, pozwalający na bardzo łatwe oddzielenie kodu HTML od kodu aplikacji. Posiada prostą składnię. W porównaniu do starszych silników renderujących, aby uzyskać taki sam efekt, wymaga napisania mniejszej ilości kodu. Aby wyświetlić wartość zmiennej, wystarczy postawić przed nią znak „@”, analogicznie postępujemy w przypadku pętli oraz innych elementów nienależących do składni języka HTML, czy JavaScript. Dodatkowo RAZOR obsługuje klamry, które są bardzo pomocne, gdy mamy więcej niż jedną linię kodu wymagającego użycia Razor-a.

3.8. Ninject

Ninject, podobnie jak Autofac, jest rozwijany jako open source, dzięki czemu uzyskujemy łatwy dostęp do kodu źródłowego tego kontenera. Aby skorzystać z tej biblioteki, wystarczy zainstalować paczkę nugetową. Można również skorzystać z gotowych rozwiązań przygotowanych pod konkretne wdrożenia. Mamy tutaj choćby dedykowane paczki do MVC czy WCF. Szerszą dokumentację można znaleźć na stronie wiki na Githubie.

3.9. Rete.js

Rete to modułowa platforma do programowania warstwy wizualnej aplikacji. Rete umożliwia tworzenie edytora opartego na węzłach z możliwością osadzenia go w aplikacji internetowej. Posiada możliwość definiowania węzłów i procesów roboczych, które umożliwiają użytkownikowi tworzyć instrukcje przetwarzania danych w edytorze bez konieczności worzenia jakiegokolwiek wersu kodu.

4. Narzędzia

Podczas tworzenia projektu zostały wykorzystane takie narzędzia jak:

- Visual Studio 2019,
- Microsoft SQL Server Management Studio 18,
- SQL Server 2019,
- IIS 10 (Internet Information Services),
- Przeglądarka internetowa

5. Wymagania systemowe

Minimalne wymagania serwera, które umożliwiają uruchomienie aplikacji to:

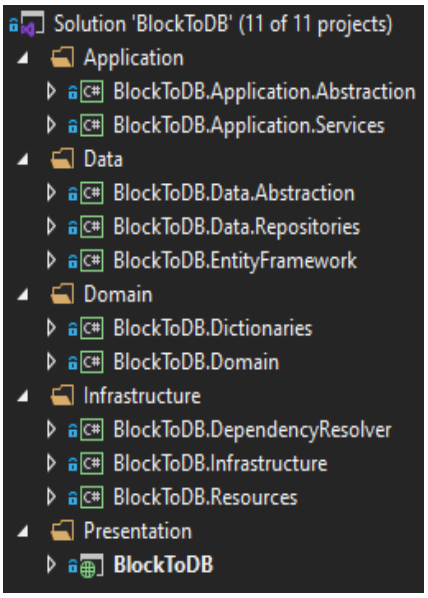
- System operacyjny z rodziny Windows nowszy niż wersja 7
- Skonfigurowany SQL Server
- Zainstalowane środowisko uruchomieniowe Microsoft .NET Framework w wersji 4.7
- Skonfigurowana usługa IIS

Minimalne wymagania maszyny klienta to:

- Przeglądarka internetowa

6. Architektura

W skład solucji wchodzą następujące projekty:



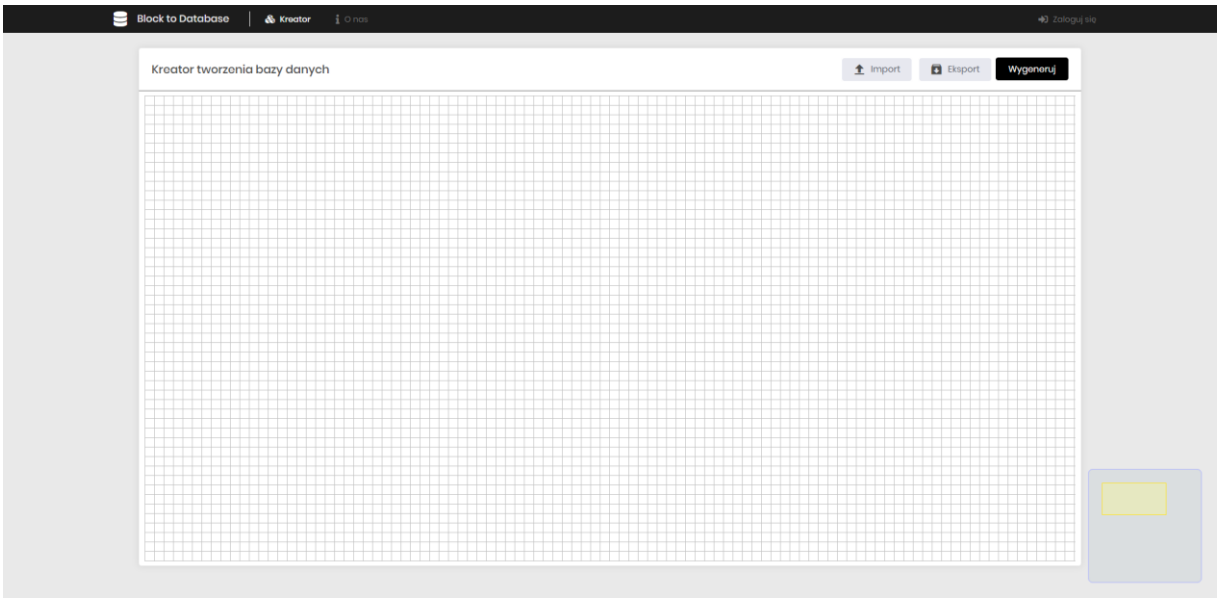
Rysunek 6.1 Architektura aplikacji

- **BlockToDB**– główny projekt, rozdzielony na foldery: App_Data, App_Start, Content, Controlers, Scripts, Views.
- BlockToDB.Application.Services, BlockToDB.Application.Abstraction – Głównym zadaniem tego projektu jest przechowywanie serwisów
- BlockToDB.Data.Abstraction, BlockToDB.Data.Repositories – Głównym zadaniem tego projektu jest przechowywanie repozytoriów
- BlockToDB.EntityFramework – projekt odpowiedzialny za konfigurację i obsługę bazy danych podłączonej do projektu.
- BlockToDB.Domain – encje bazodanowe.
- BlockToDB.Resources, BlockToDB.Dictionaries – projekty odpowiedzialne za definicję typów wyliczeniowych oraz plików resource.
- BlockToDB.Infrastructure, BlockToDB.DependencyResolver – głownie klasa odpowiedzialna za konfiguracje wstrzykiwania klas przez ninject ale także metody rozszerzające dla MVC

7. Interfejs użytkownika

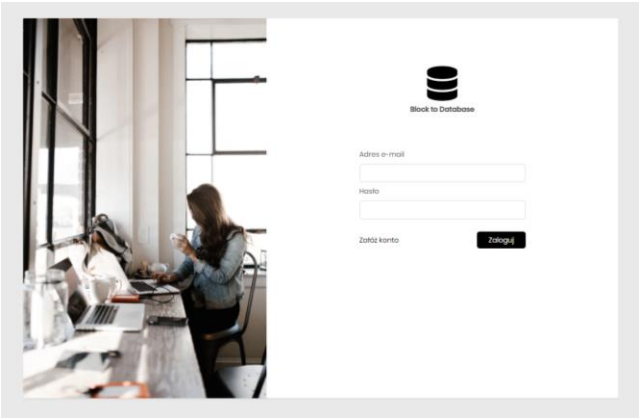
Interfejs użytkownika został zbudowany przy pomocy widoków HTML, wspomaganych przez bibliotekę jQuery oraz technologię Razor. W zależności od statusu autoryzacji interfejs może się różnić. Użytkownik zautoryzowany posiada dostęp do dodatkowych funkcji aplikacji. Autoryzacja odbywa się w oparciu o formularz logowania, do którego potrzebne jest uprzednie utworzenie konta w procesie rejestracji.

Rysunek 7.1 przedstawia główny widok aplikacji, zawierający kreator graficzny, dla użytkownika niezautoryzowanego.



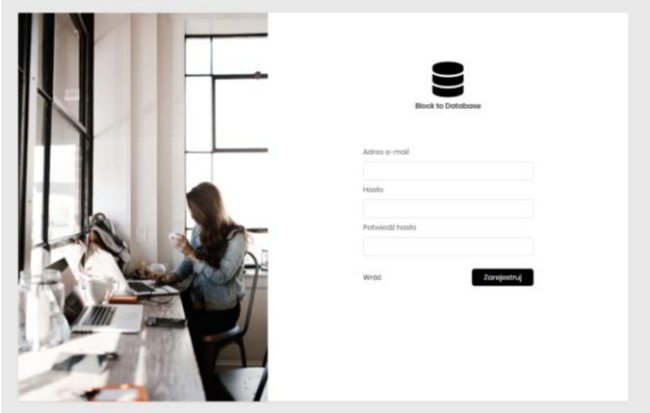
Rysunek 7.1 Kreator - użytkownik niezautoryzowany

Panel logowania został przedstawiony na rysunku 7.2.



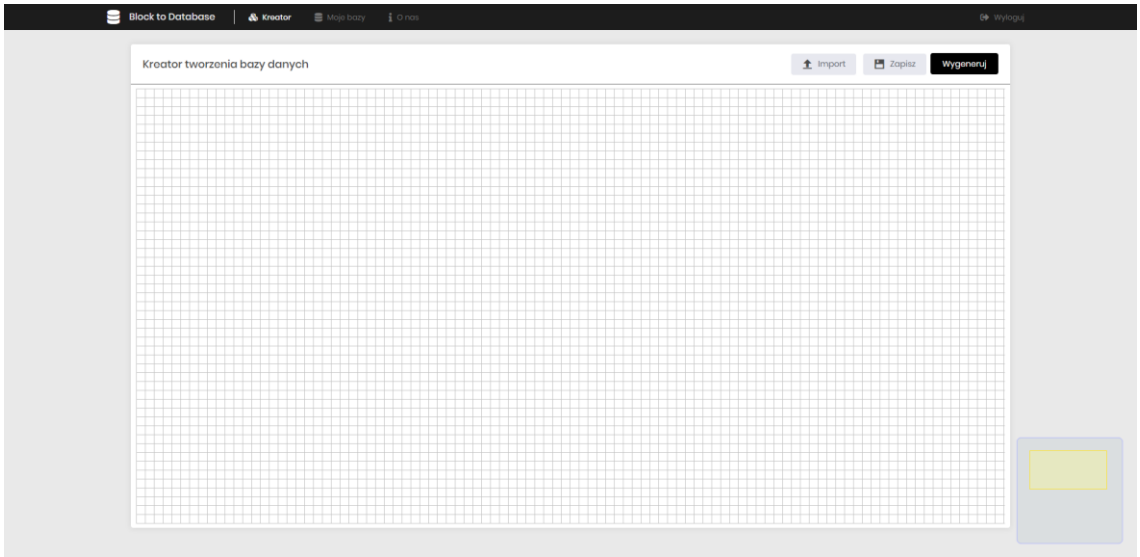
Rysunek 7.2 Panel logowania

Z panelu logowania (rysunek 7.2) użytkowni ma możliwość przejścia do formularza zakładania nowego konta w systemie (rysunek 7.3). W tym celu musi wypełnić formularz takimi danymi jak adres e-mail oraz hasło do konta wraz z jego potwierdzeniem. Jeżeli wprowadzone dane zostaną poprawnie zwalidowane konto zostanie utworzone a użytkownik otrzymy stosowny komunikat oraz będzie posiadała możliwość zalogowanie się do systemu.



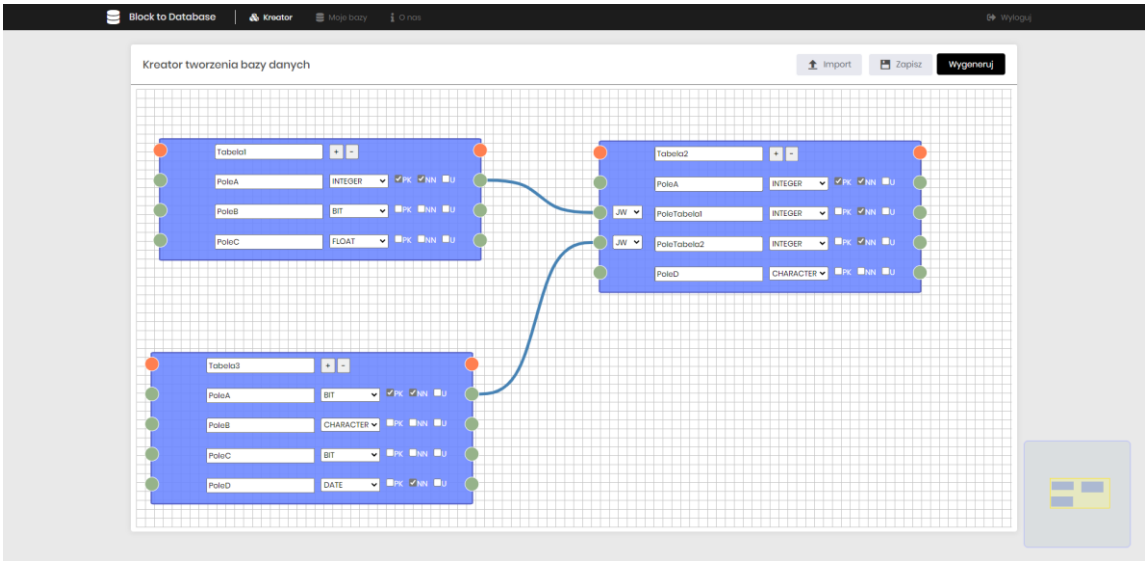
Rysunek 7.3 Formularz rejestracji

Kreator tworzenia nowego schematu bazy danych dla użytkownika zautoryzowanego (rysunek 7.4).



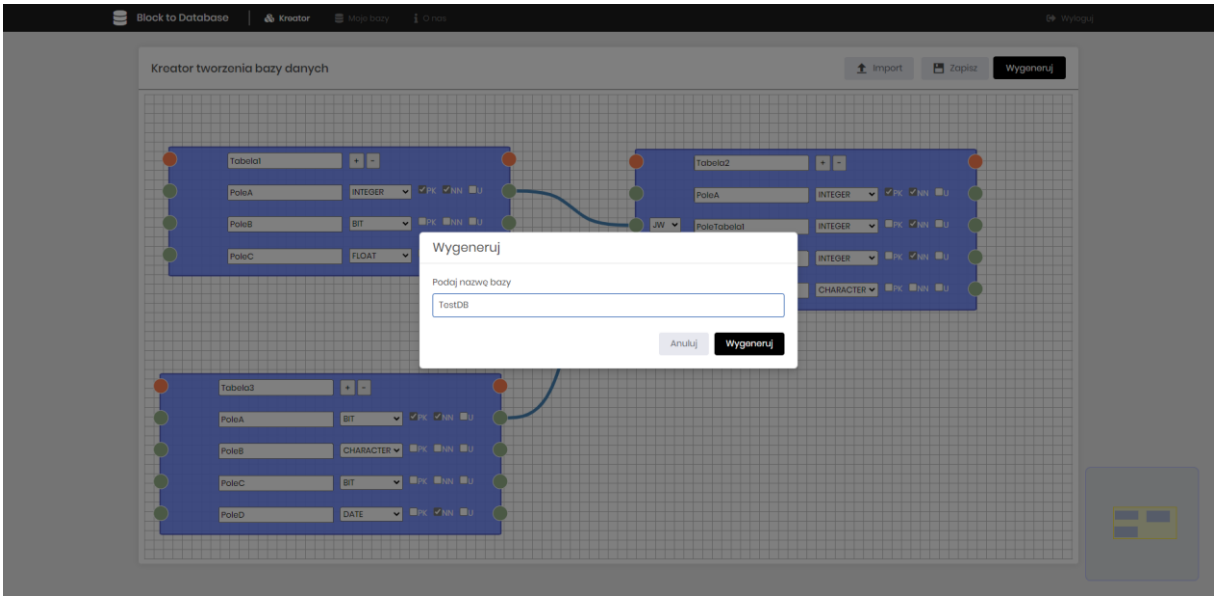
Rysunek 7.4 Kreator - użytkownik zautoryzowany

Przy pomocy kontrolki Rete.js zostało stworzone graficzne narzędzie umożliwiające tworzenie schematów bazy danych (rysunek 7.5). Dodatkową funkcją dostępną z tego poziomu jest zapis kopii roboczej (w przypadku zautoryzowanego użytkownika jest to zapis schematu w bazie danych, w innym pobranie takiej wersji do pliku z rozszerzeniem .json) oraz wgranie wersji w postaci pliku .json.



Rysunek 7.5 Przykładowy schemat bazy danych

Przy pomocy „rozwijanego przycisku Wygeneruj" użytkownik ma możliwość wygenerowania skryptu tworzącego bazę danych, który zostaje następnie pobrany do pliku z rozszerzeniem .sql lub tworzony bezpośrednio na serwerze zdalnym zgodnie z wprowadzonymi parametrami połączenia.



Rysunek 7.6 Generowanie schematu do pliku .sql

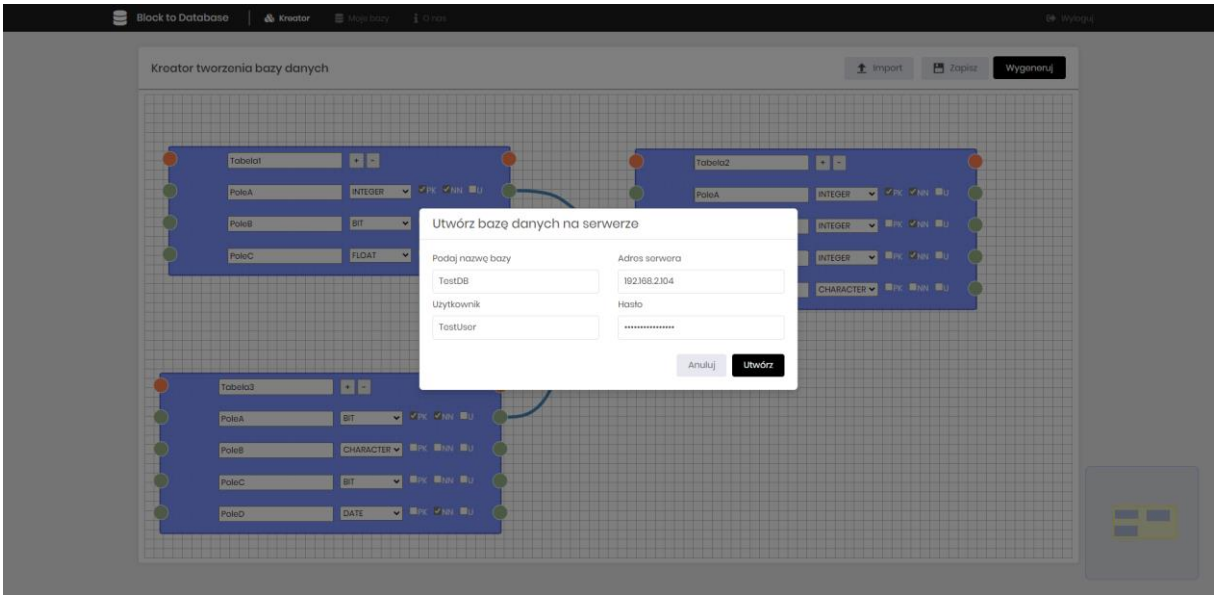
Przykładowa struktura wygenerowanego pliku.

```
CREATE DATABASE [TestDB];
GO
USE [TestDB]
GO

CREATE TABLE Tabela1 (
    PoleA INTEGER NOT NULL,
```

```
    PoleB BIT,
    PoleC FLOAT,
    PRIMARY KEY (PoleA)
);

CREATE TABLE Tabela2 (
    PoleA INTEGER NOT NULL,
    PoleTabela1 INTEGER NOT NULL,
    PoleTabela2 INTEGER NOT NULL,
    PoleD CHARACTER(255),
    PRIMARY KEY (PoleA)
);

CREATE TABLE Tabela3 (
    PoleA BIT NOT NULL,
    PoleB CHARACTER(255),
    PoleC BIT,
    PoleD DATE NOT NULL,
    PRIMARY KEY (PoleA)
);

ALTER TABLE Tabela1 ADD CONSTRAINT FK_2_1_90 FOREIGN KEY(PoleA) REFERENCES
Tabela2(PoleTabela1);
ALTER TABLE Tabela3 ADD CONSTRAINT FK_3_1_48 FOREIGN KEY(PoleA) REFERENCES
Tabela2(PoleTabela2);
```

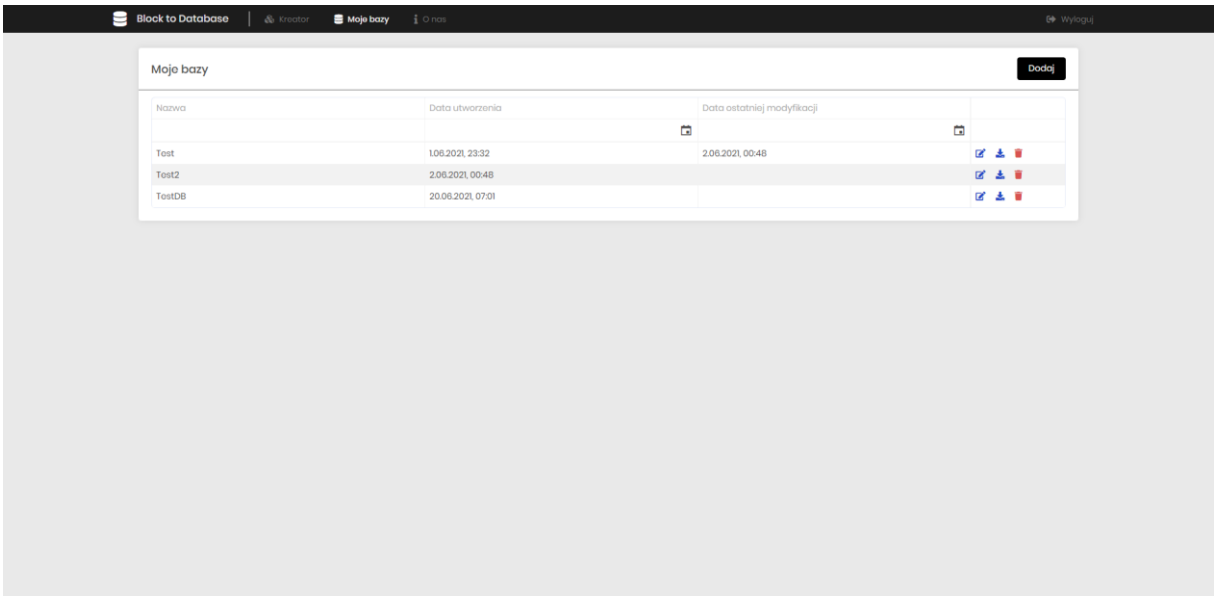
Listing 7.1 Przykładowy skrypt bazy danych

Rysunek 7.7 przedstawia opcję tworzenia bazy danej bezpośrednio na serwerze zdalnym.



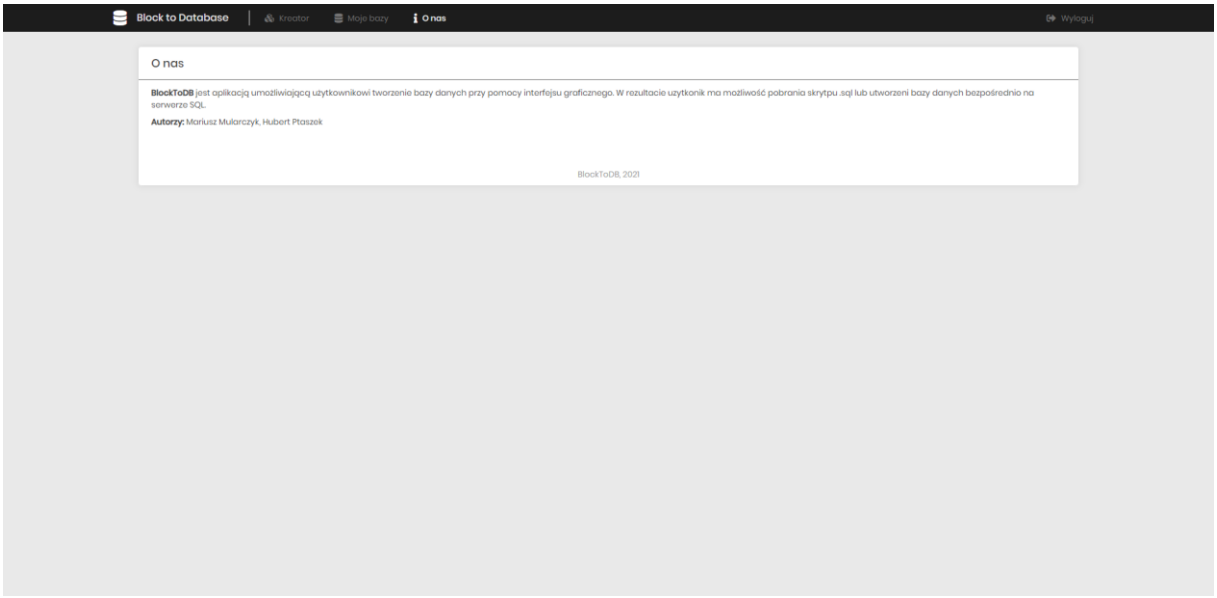
Rysunek 7.7 Tworzenie bazy danych na serwerze zdalnym

Na rysunku 7.8 została przedstawiona strona, odpowiedzialna jest ona za możliwość dostępu do zapisanych baz oraz kopii roboczych użytkownika przypisanych do konta.



Rysunek 7.8 Storna "Moje bazy"

Strona „O nas” (rysunek 7.9) przedstawia podstawowe informacje o projekcie oraz o autorach.



Rysunek 7.9 Strona "O nas"

8. Implementacja serwera

W procesie implementacji części serwerowej aplikacji MVC została wykorzystana technologia APS.NET Framework w wersji 4.7 wraz z bazą danych MS SQL. Zostało również zastosowane podejście „Code First”. Poniżej został przedstawione najważniejsze fragmenty kodu źródłowego.

```
public string CreateRemoteDataBase(BlockToDBGenerateRemoteVM model)
{
    BlockToDBGenerateVM blockToDBGenerate = BlockToDBConverter.FromBlockToDBGenera
teRemoteVM(model);
    int fileId = GenerateScript(blockToDBGenerate);
```

```
    DatabaseSchema databaseSchema = DatabaseSchemaRepository.GetSingle(x =>
      x.Id == fileId);
    string connectionString = string.Format("Data Source={0};Initial Catalog=maste
r;Integrated Security=False; User Id={1};Password={2}", model.Url, model.UserName,
 model.Password);
    Editor editor = JsonConvert.DeserializeObject<Editor>(model.Json);
    List<string> commands = BlockToDBConverter.ToSqlCodeList(editor, model.Name);
    string result = "";
    try
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand(commands[0], conn);
            cmd.ExecuteNonQuery();
            conn.ChangeDatabase(model.Name);
            cmd.CommandText = commands[1];
            cmd.ExecuteNonQuery();
        }
    }
    catch (Exception)
    {
        result = "Error";
    }
    return result;
}
```

Listing 8.1 Tworzenie bazy danych na serwerze zdalnym

Funkcja CreateRemoteDataBase (listing 8.1) jest odpowiedzialna za nawiązanie połączenia z serwerem zdalnym na podstawie przekazanych przez użytkownika danych, które przekazywane są w parametrze model.
Pierwszym krokiem jaki jest wykonywany w funkcji jest deserializacja danych z edytora (JSON) do obiektu klasy Editor. Następnie w oparciu o ten obiekt tworzony jest skrypt SQL, który to w kolejnym kroku jest uruchamiany na serwerze zdalnym, w momencie gdy uda się z nim nawiązać połączenie.

```
public string ToSqlCode(Editor model, string name)
{
    StringBuilder content = new StringBuilder();
    List<InheritConn> inheritConns = new List<InheritConn>();
    List<InheritConn> inheritConns2 = new List<InheritConn>();

    content.AppendFormat("CREATE DATABASE [{0}];", name);
    content.AppendLine("");
    content.AppendLine("GO");
    content.AppendFormat("USE [{0}]", name);
    content.AppendLine("");
    content.AppendLine("GO");
    content.AppendLine("");
```

```
    foreach (KeyValuePair<string, Node> node in model.Nodes)
    {
        ...
    }
}
```

Listing 8.2 Generowanie kodu SQL z edytora graficznego

Generowanie skryptu SQL (listing 8.2) odbywa się w głównej mierze w funkcji ToSQLCode, która to w parametrze przyjmuje obiekt klasy Editor oraz nazwę bazy danych. Pierwsze operacje wykonywane w funkcji to utworzenie obiektu typu StringBuilder, a następnie dwóch struktur które będą przechowywały tymczasowo zależności pomiędzy tabelami.
Następnie tworzone są pierwsze linie skryptu SQL, które odpowiedzialne są za tworzenie bazy danych. Po zakończeniu tych operacji funkcja przechodzi do pętli foreach w której to iteruje po poszczególnych węzłach edytora tworząc tabele wraz ich własnościami.

```
public void TPCGenerate(ref StringBuilder nodeContent, ref List<InheritConn>
inheritConns2, ref List<string> primaryKeys, Editor model, Node node, Input
inhetitFrom, int fieldsCount, ref int islast, string firstNodeName)
        {
            Connection conn = inhetitFrom.Connections[0];
            int fieldsCountInherit = model.Nodes.FirstOrDefault(x => x.Key ==
conn.Node.ToString()).Value.Inputs.Count - 1;
            Node inheritNode = model.Nodes.FirstOrDefault(x => x.Key == conn.N
ode.ToString()).Value;

            Input inhetitFrom2 = inheritNode.GetInheritFrom();
            string inhetitType = "0";
            if (inhetitFrom2.Connections.Count > 0)
            {
                inhetitType = inheritNode.GetInheritType();
            }

            if (inhetitType == "TPC")
            {
                int fieldsCount2 = inheritNode.Inputs.Count - 1;
                islast++;
                TPCGenerate(ref nodeContent, ref inheritConns2, ref primaryKey
s, model, inheritNode, inhetitFrom2, fieldsCount2, ref islast, firstNodeName);
                islast--;
            }
            else
            {
                islast--;
            }

            if (inhetitType == "0")
            {
                for (int i = 1; i <= fieldsCountInherit; i++)
                {
```

```
                    nodeContent.AppendLine(inheritNode.GetTableField(i, ref pr
imaryKeys));
                }
            }
            for (int i = 1; i <= fieldsCount; i++)
            {
                nodeContent.AppendLine(node.GetTableField(i, ref primaryKeys))
;
            }
            inheritNode.GetInheritTableConnection(firstNodeName, ref inheritCo
nns2, model);
            if (islast < 0)
            {
                if (primaryKeys.Count != 0)
                {
                    StringBuilder primaryKeysString = new StringBuilder();

                    primaryKeysString.Append("PRIMARY KEY (");
                    int last = primaryKeys.Count;
                    int j = primaryKeys.Count;
                    foreach (string key in primaryKeys)
                    {
                        primaryKeysString.Append(key);
                        if (last == j)
                        {
                            primaryKeysString.Append(")");
                        }
                        else
                        {
                            j++;
                            primaryKeysString.Append(",");
                        }
                    }
                    nodeContent.AppendLine(primaryKeysString.ToString());
                }
            }
        }
```

Listing 8.3 Generowanie kodu SQL dla dziedziczenia TPC

Listing 8.3 przedstawia funkcję TPCGenerate, która to jest uruchamiana wewnątrz pętli w funkcji ToSQLCode i jest odpowiedzialna za generowanie kodu SQL dla dziedziczenia typu TPC (Table per class). Funkcja ta w parametrze otrzymuje między innymi konkretną tabelę oraz węzeł w postaci referencji, a także całą strukturę edytora. Następnie węzeł jest interpretowany i rozdzielany na mniejsze części, a następnie po sprawdzeniu czy istnieje zagnieżdżona dziedziczenie wywoływana jest rekurencyjnie ta sama funkcja, aż do uzyskania „najgłębszego” węzła. W takim przypadku zostaje tworzony stosowny kod tabeli, który z każdą iteracją jest rozbudowywany.

9. Podsumowanie

Celem projektu było opracowanie aplikacji z warstwą interfejsu użytkownika w postaci aplikacji przeglądarkowej, która pozwala na graficzne budowanie bazy danych z poziomu przeglądarki internetowej. Aplikacja umożliwia przetwarzanie rezultatów pracy do skryptu SQL zapisywanego w formie pliku lub uruchamianego na serwerze zdalny. Drugim zagadnieniem z kolei było stworzenie opisu teoretycznego dotyczącego problemu projektowania baz danych.
Cele projektu zostały zrealizowane.