Modelowanie i analiza systemów informatycznych

dokumentacja projektu:
System wspomagania decyzji

inż. Bartosz Ociepka inż. Beniamin Stecuła

23 listopada 2020

Podział pracy

Podział pracy był płynny, lecz z większym naciskiem na:

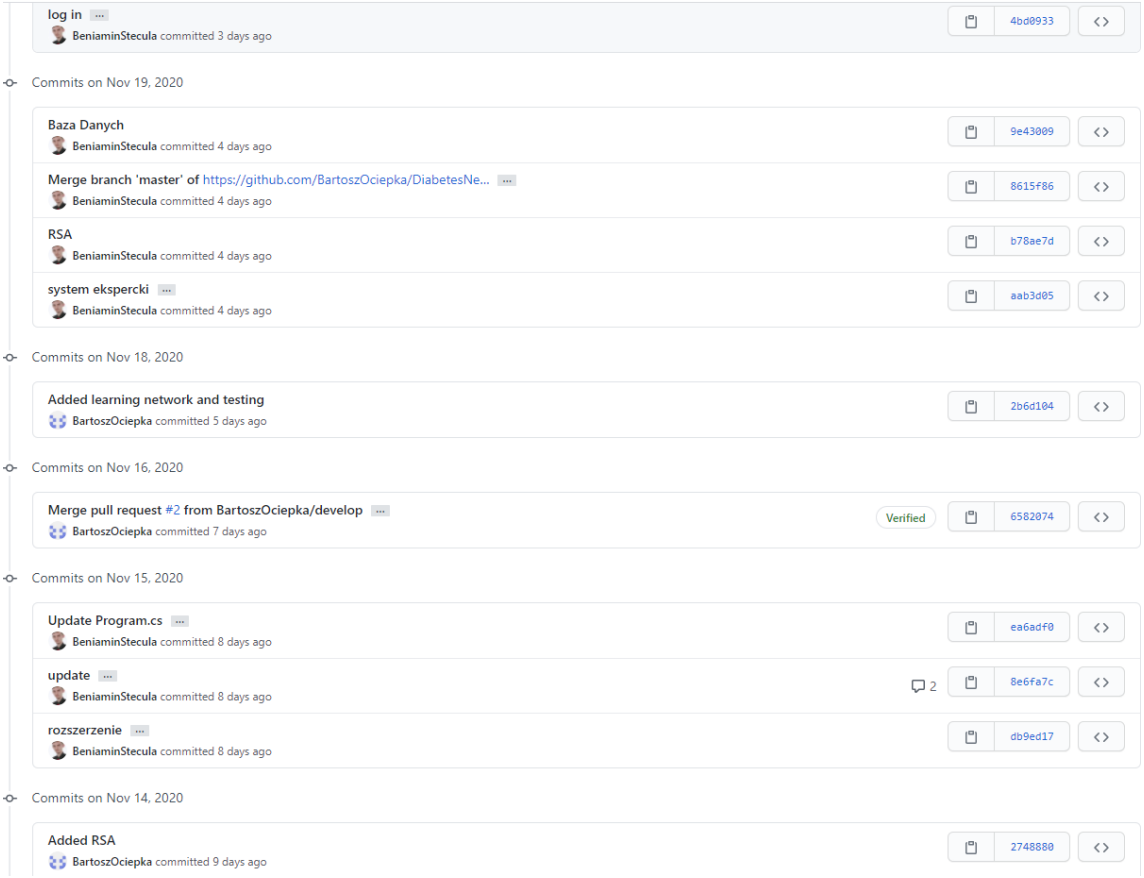
- inż. Bartosz Ociepka – backend, praktyka,
- inż. Beniamin Stecuła – frontend, teoria, dokumentacja.

Udokumentowanie pracy

Dokumentowanie pracy odbyło się na kilka sposobów:

- utworzenie niniejszej dokumentacji,
- zarządzanie podziałem i wykonaniem zadań w serwisie Trello,
- przechowywanie kopii poprzednich wersji programu.

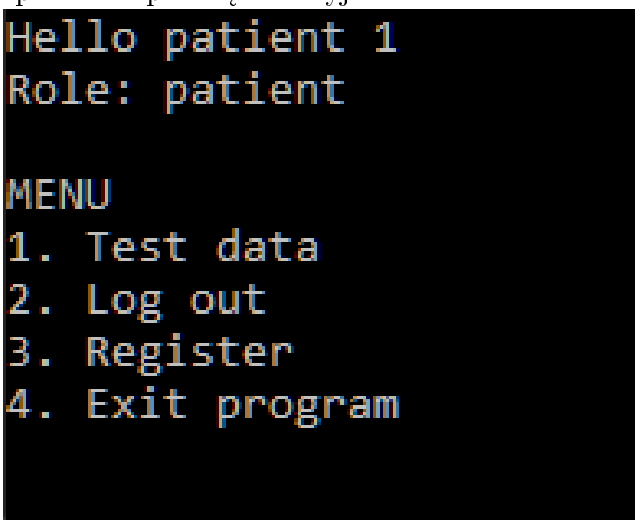
W ramach repozytorium każdy z nas wrzucał commity do swoim zadań



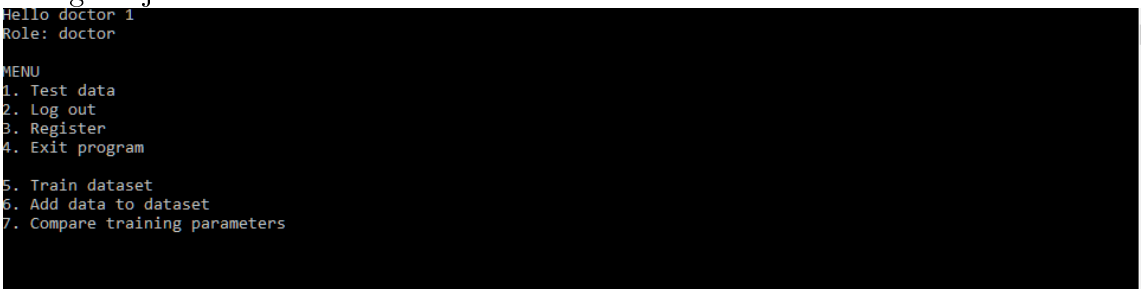
Całe repozytorium jest dostępne pod adresem
https://github.com/BartoszOciepka/DiabetesNeuralNetwork

Instrukcja obsługi

Niezalogowany użytkownik na start widzi menu, z którego może się zalogować, zarejestrować, sprawdzić próbkę lub wyjść.



Po zalogowaniu jako doktor do menu dochodzi opcja dodania próbki, trenowania zbioru lub uruchomienia programu testującego, który sprawdza i pokazuje wyniki uczenia dla różnych konfiguracji sieci.



Wchodząc w każdą opcję użytkownik jest prowadzony za rękę przez cały proces. W razie błędu lub podania niepoprawnych danych wyświetlany jest komunikat błędu.

Instrukcja wdrożenia

Aby wdrożyć projekt należy wykonać poniższą listę kroków:

1. zaimportowanie projektu w Visual Studio 2015,
2. import danych do bazy danych MySql (dołączono plik dump.sql zawierający potrzebne tabele),
3. zmiana connectionString w kodzie na odpowiadające używanej bazie, danych (dokonanie zmiany klas gdy używana jest inna baza niż MySql),
4. uruchomienie projektu.

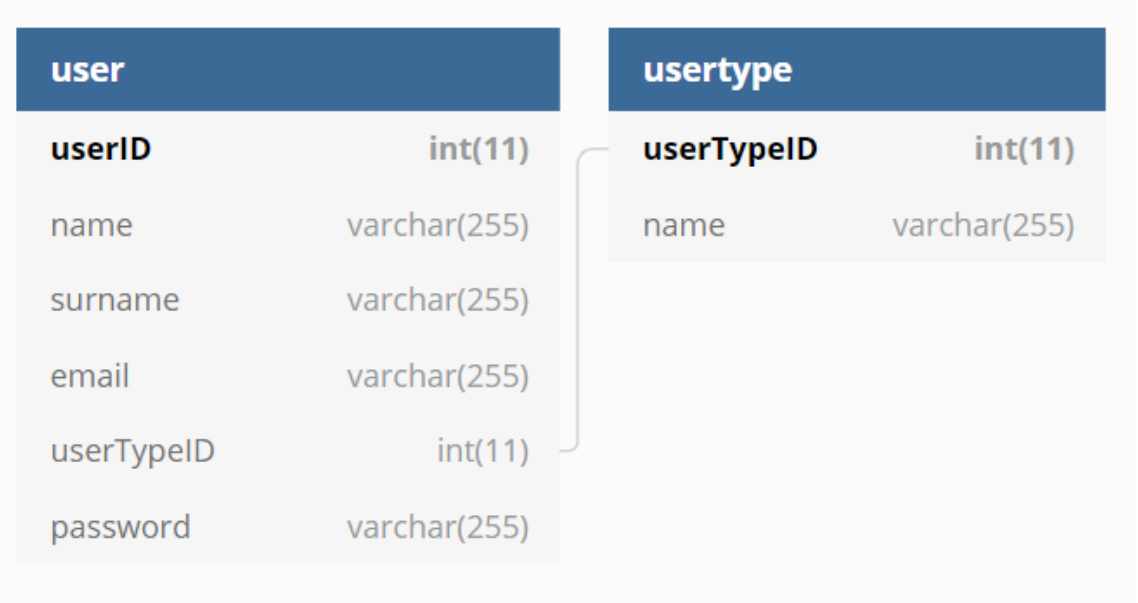
Część testowa

Baza danych

Łączenie z bazą danych ma na stałe ustawione własności, które mogą być w razie potrzeby edytowane w klasie *LinqToDbSettings.cs* we fragmencie:

```
new ConnectionStringSettings
{
  Name = "DiabetesDatabase",
  ProviderName = "MySql",
  ConnectionString = @"Server=localhost;Database=diabetes;Uid=;Pwd=;"
};
```

Diagram UML bazy danych został przedstawiony na rysunku 1.



Rysunek 1: Diagram klas.

Komponenty systemu

W naszym programie komponenty systemu uporządkowane są w odpowienich folderach o nazwach pokazujących jakie pliki tam znajdziemy. I tak mamy:

- Core - z głównymi elementami systemu, w naszym wypadku sieć neuronową
- Authorization - pliki do logowania/rejestracji oraz trzymania ich stanu

- Controllers - tutaj są kontrolery używane jako most między bazą danych a modelami
- Helpers - klasy pomocnicze do walidacji i szyfrowania
- Models - modele używane w programie
- ORM - konfiguracja ORMa, którego używamy do komunikacji z bazą

Użyliśmy wzorca projektowego Singleton, dzięki temu nie musimy tworzyć obiektu gdy chcemy zwalidować dane lub je zaszyfrować. Można też uznać, że w ramach używania biblioteki AForge do sieci neuronowej używamy wzorca Fasada.

Nasza sieć neuronowa jest utworzona przy wykorzystaniu biblioteki AForge (http://www.aforgenet.com/framework/docs/). Nie jest to zbyt znana biblioteka, ale do jej wykorzystania przekonała nas bardzo dokładna dokumentacja.
Biblioteka udostępnia nam obiekt ActivationNetwork, który w konstruktorze bierze funkcję aktywacyjną, ilość wejść sieci oraz tablicę neuronów w poszczególnych warstwach. W naszym wypadku za funkcję aktywacyjną wzieliśmy funkcję sygmoidalną z alfą równą 1.8 (taką wartość uznaliśmy za najlepszą w ramach testowania). Gdy już mamy utworzony obiekt sieci musimy ją nauczyc. Służy do tego klasa BackPropagationLearning (jak i inne, ale my będziemy uczyć sieć używając propagacji wstecznej).

Algorytm szyfrowania

W naszym programie do rejestracji i logowania użyliśmy stworzonej przez siebie klasy *AuthorizationManager*, która zawiera metody *LogIn(), Register(), LogOut()*. Szyfrowanie zaimplementowane zostało w klasie *RSAHelper*, w której znajduje się cała logika szyfrowania opierająca się na klasie *RSACryptoServiceProvider*. Do szyfrowania użyty został algorytm RSA (definicja oraz dwa wywołania):

```
1    using (var rsa = new RSACryptoServiceProvider(2048))
2    ...
3    User user = UserController.FindUserByEmailAndPassword(email,
         RSAHelper.encrypt(password));
4    ...
5    User user = new User(name, surname, email, RSAHelper.encrypt(
         password), userTypeID);
```

Algorytm Rivesta-Shamira-Adlemana (RSA) to jeden z pierwszych i najpopularniejszych asymetrycznych algorytmów kryptograficznych o kluczu publicznym. Może być stosowany i do szyfrowania, i cyfrowego podpisywania plików.

Polega on na liczeniu funkcji Eulera dla dużych liczb pierwszych, a jego bezpieczeństwo opiera się na trudności faktoryzacji dużych liczb złożonych.

Każdy z rozmówców posiada parę kluczy: prywatny i publiczny. Pierwszy z nich służy do deszyfrowania wiadomości przychodzącej, a drugi do szyfrowania wychodzącej. Aby nawiązać komunikację rozmówcy muszą wymienić się swoimi kluczami publicznymi. Klucze prywatne nigdy nie są ujawniane.

System ekspercki

W dziale sztucznej inteligencji systemem eksperckim (Expert System – ES) nazywa się system komputerowy, który emuluje podejmowanie decyzji przez ludzkiego specjalistę z danej dziedziny. Odzwierciedla procesy myślowe na zasadzie działania ludzkiego mózgu.

Do rozwiązań wykorzystujących ES zaliczamy:

- systemy agentowe,
- agenty programowe,
- eksplorację danych,
- wspomaganie twórczego myślenia,
- ontologie systematyzujące wiedzę.

ES powinien zawierać trzy charakterystyczne, podstawowe cechy:

- bazy wiedzy – schematycznie zapisane informacje uzyskane od ludzkich ekspertów w danej dziedzinie,
- procedury wnioskujące – odzwierciedlające wnioskowanie symboliczne, które jest charakterystyczne dla funkcjonowania ludzkiego mózgu,
- zdolność do poszerzania wiedzy – umożliwienie ciągłego rozwoju systemu w przypadku stale napływających nowych informacji koniecznych do uwzględnienia aby otrzymywać stale aktualne wyniki zgodne z nowym stanem wiedzy w dziedzinie.

Struktura ES składa się z sześciu podstawowych elementów:

- baza wiedzy – składająca się ze zbioru reguł,
- baza danych – zawierająca dane obiektów, wartości, przypadki i hipotezy,
- procedury wnioskowania – jest to maszyna wnioskująca,
- procedury objaśniania – tłumaczą strategię wnioskowania,
- procedury sterowania dialogiem – funkcje wejścia/wyjścia sterujące programem i wyznaczające zadania, które ma wykonać
- procedury zarządzania wiedzą – pozwalają na modyfikację oraz rozszerzanie, pozyskiwanie wiedzy.

Implementując kod programu i tworząc samemu kod ekspertowy doszliśmy do pewnych wniosków na temat zalet i wad systemów eksperckich:
Zalety to na pewno szybkość przetwarzania danych. Człowiekowi zajęło by to nieporównywanie więcej czasu niż komputerowi. Do tego sieć potrafi zauważyć powiązanie, które człowiek pominie albo nie zauważy.
Wady to przede wszystkim brak zdrowego rozsądku. Dla komputera liczby to liczby więc błędne dane czy dane na logikę wskazujące na pewne rozwiązanie nie zwrócą jego uwagi. Dochodzi do tego możliwość przeuczenia sieci, błędne dopasowanie parametrów sieci lub zbyt mały zbiór danych i mamy rozwiązanie, które jest bardzo potężne, ale wymaga uwagi i wiedzy aby poprawnie zaimplementować.

Przykładowy kod z aplikacji z testami

```
1   Tutaj wklejamy pelen kod.
```