Modelowanie i analiza systemów informatycznych

dokumentacja projektu:
System wspomagania decyzji

inż. Bartosz Ociepka inż. Beniamin Stecuła

5 grudnia 2020

Podział pracy

Podział pracy był płynny, lecz z większym naciskiem na:

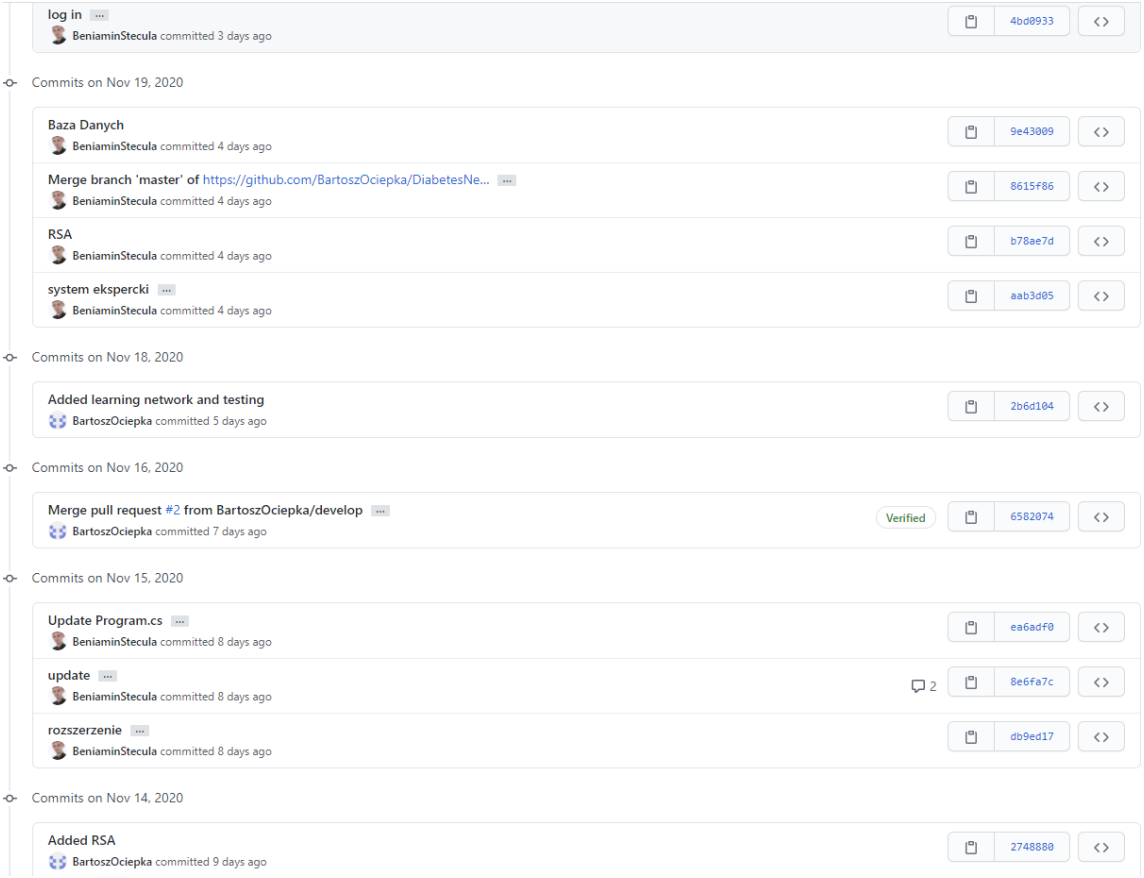
- inż. Bartosz Ociepka – backend, praktyka,
- inż. Beniamin Steculа – frontend, teoria, dokumentacja.

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.

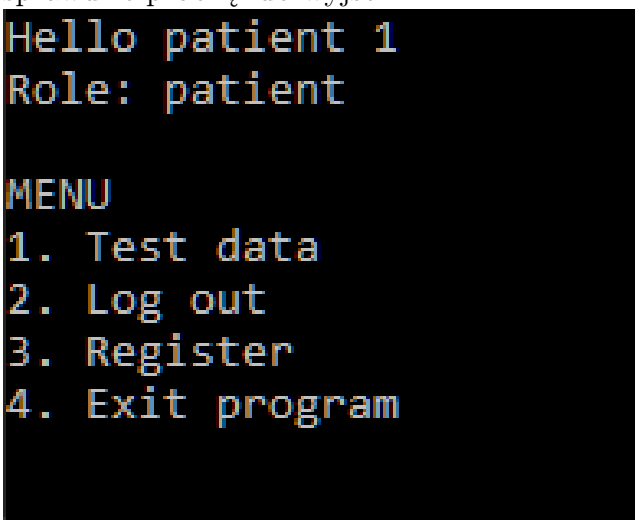
W ramach repozytorium każdy z nas wrzucał commity do swoim zadań



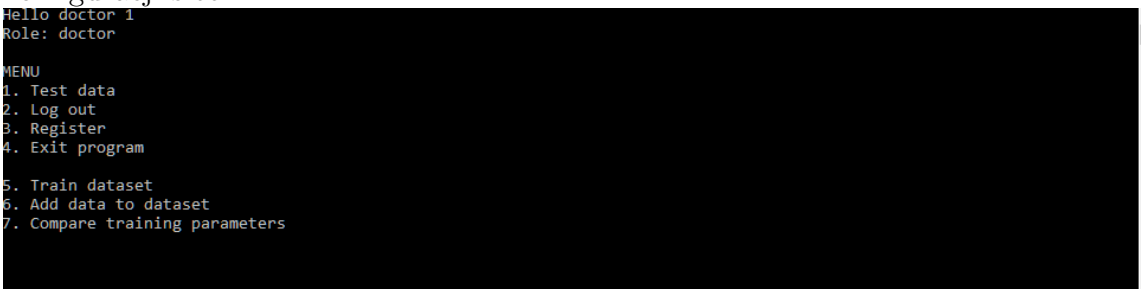
Całe repozytorium jest dostępne pod adresem
https://github.com/BartoszOciepka/DiabetesNeuralNetwork

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:

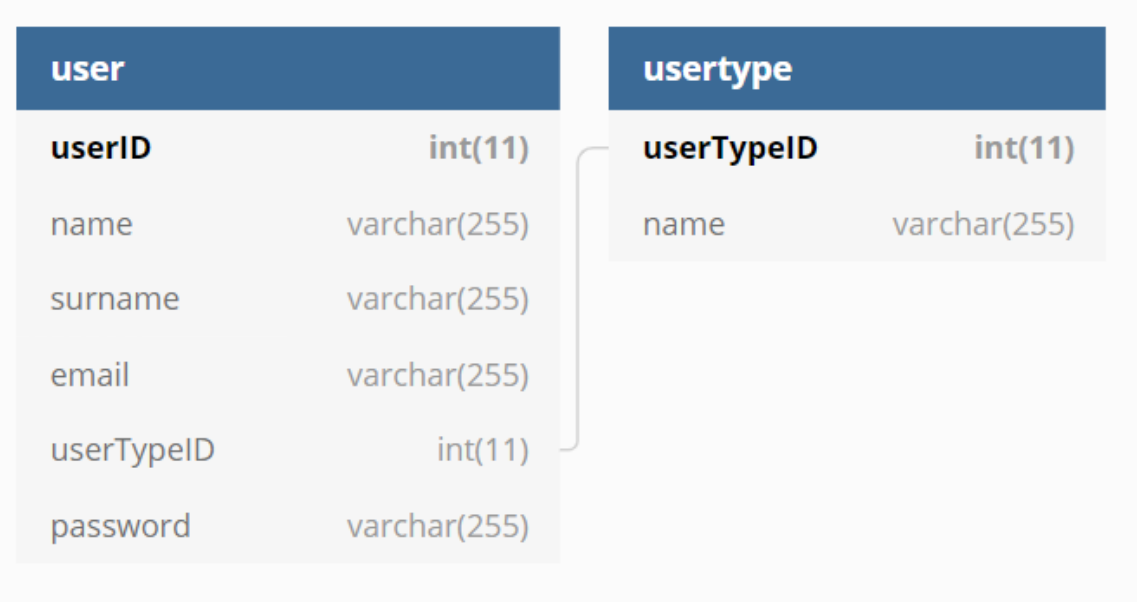
1. zaimportowanie projektu w Visual Studio 2015,
2. import danych do bazy danych MySql (dołączono plik dump.sql zawierający potrzebne tabele),
3. zmiana connectionString w kodzie na odpowiadające używanej bazie, danych (dokonanie zmiany klas gdy używana jest inna baza niż MySql),
4. uruchomienie projektu.

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 sigmoidalną z alfą równą 1.8 (taką wartość uznaliśmy za najlepszą w ramach testowania). Funkcja ta ma postać:

$$f(x) = \frac{1}{1 + E^{-\alpha * x}} \tag{1}$$

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.

Generowanie kluczy

Generowanie pary kluczy:

- losujemy duże liczby pierwsze p i q odległe, lecz o zbliżonej długości,
- $n = pq$,
- obliczamy funkcję Eulera dla n: $\varphi(n) = (p-1)(q-1)$,
- wybieramy liczbę e, gdzie: $(1 < e < \varphi(n))$, która jest względnie pierwsza z $\varphi(n)$,
- $d \equiv e - 1(mod\varphi(n))$, czyli: $d \cdot e \equiv 1(mod\varphi(n))$,
- kluczem publicznym jest para liczb (n, e), a prywatnym (n, d).

Szyfrowanie i deszyfrowanie

Korzystamy z dwóch wzorów:

- wiadomość dzielimy na bloki m o wartości nie większej niż n, a następnie: $c \equiv m^e \pmod{n}$,
- zaszyfrowane bloki c deszyfrujemy wzorem: $m \equiv c^d \pmod{n}$.

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.

Sieć neuronowa

Siecią neuronową nazywa się struktury matematyczne z ich programowymi implementacjami. Realizują obliczenia i przetwarzają sygnały drobnymi elementami nazywanymi sztucznymi neuronami, które wykonują proste operacje na otrzymanych danych wejściowych.

Zasada działania opiera się na mózgu składającym się z naturalnych neuronów, łączących je synaps oraz układów nerwowych.

Typy sieci neuronowych:

- jednokierunkowe – bez sprzężenia zwrotnego,
- rekurencyjne – ze sprzężeniem zwrotnym,
- samoorganizujące się mapy – sieci Kohonena, gdzie neurony są stowarzyszone ze współrzędnymi na prostej lub przestrzeni wyższego rzędu.

Dobieranie parametrów w programie

W naszym programie do dobierania optymalnych parametrów służy metoda *compareTraining-Parameters()*. Iteruje ona, zgodnie z parametrami po kolejnej liczbie warstw oraz po liczbach neuronów w każdej warstwie. Istnieje jednak warunek, że w kolejnej warstwie nie może być więcej neuronów niż w poprzedniej.

```
1     internal static void compareTrainingParameters()
2     {
3       //Random rand = new Random();
4       maxNeurons = 16;
5       minNeuronsBefore = maxNeurons;
6       int maxLayers = 3;
7
8       Console.WriteLine("Max neurons in a layer:\t" + maxNeurons);
9       Console.WriteLine("Max number of layers:\t" + maxLayers);
10      Console.WriteLine("Alpha:\t" + alpha);
11      Console.WriteLine("Max error accepted:\t" + maxError);
12
13      for (layersNumber = 2; layersNumber<=maxLayers; layersNumber++)
14      {
15        Console.WriteLine("\nCurrent number of layers:\t" + layersNumber
              );
16
17        int[] neuronsTable = new int[layersNumber];
18        neuronsTable[layersNumber - 1] = 1;
19
20        // generate all possibilities for a set of layers
21        //neurons[i] = rand.Next(1, maxNeurons);
22        incrementNeuron(0, neuronsTable, minNeuronsBefore);
23      }
24    }
25    private static void incrementNeuron(int layer, int[] neuronsTable,
          int minNeuronsBefore)
26    {
27      double error;
28      // last layer with neurons
29      if(layer==layersNumber-2)
30      {
31        for(int neu=1; neu<=minNeuronsBefore; neu++)
32        {
33          neuronsTable[layer] = neu;
34
35          int timesEachValuesAreTested = 1;
36
37          Console.Write("Testing: ");
38          foreach (int n in neuronsTable) Console.Write(n + " ");
39
40          error = trainNetwork(neuronsTable, alpha, false,
                timesEachValuesAreTested);
41          if (error <= maxError)
42          {
43            Console.WriteLine("\tError: " + error);
44          }
45          else
```

```
46            {
47              Console.WriteLine("\tError too much");
48            }
49          }
50          //foreach (int n in neuronsTable) Console.Write(n + " ");
51          Console.WriteLine();
52        }
53        // increment and next number
54        else
55        {
56          for (int neu = 1; neu <= minNeuronsBefore; neu++)
57          {
58            neuronsTable[layer] = neu;
59            if (minNeuronsBefore > neu)
60            incrementNeuron(layer+1, neuronsTable, neu);
61            else
62            incrementNeuron(layer + 1, neuronsTable, minNeuronsBefore);
63          }
64          //foreach (int n in neuronsTable) Console.Write(n + " ");
65          //Console.WriteLine();
66          return;
67        }
68      }
```

Przykładowy kod z aplikacji z testami

RSAHelper.cs

```
1  using System;
2  using System.Security.Cryptography;
3  using System.Text;
4
5  namespace DiabetesNeuralNetwork
6  {
7    public class RSAHelper
8    {
9      public static string containerName = "defaultContainerName";
10     public static string encrypt(string message)
11     {
12       var bytesToEncrypt = Encoding.UTF8.GetBytes(message);
13
14       using (var rsa = new RSACryptoServiceProvider(2048))
15       {
16         try
17         {
18           rsa.FromXmlString(RSAHelper.getPublicKeyFromContainer());
19           var encryptedData = rsa.Encrypt(bytesToEncrypt, true);
20           var base64Encrypted = Convert.ToBase64String(encryptedData);
21           return base64Encrypted;
22         }
23         finally
24         {
25           rsa.PersistKeyInCsp = false;
```

```
26          }
27        }
28      }
29
30      public static string decrypt(string message)
31      {
32        var bytesToDescrypt = Encoding.UTF8.GetBytes(message);
33
34        using (var rsa = new RSACryptoServiceProvider(2048))
35        {
36          try
37          {
38            rsa.FromXmlString(RSAHelper.getPrivateKeyFromContainer());
39
40            var resultBytes = Convert.FromBase64String(message);
41            var decryptedBytes = rsa.Decrypt(resultBytes, true);
42            var decryptedData = Encoding.UTF8.GetString(decryptedBytes);
43            return decryptedData.ToString();
44          }
45          finally
46          {
47            rsa.PersistKeyInCsp = false;
48          }
49        }
50      }
51
52      // Run this method before any encryption.
53      // This method adds a new key container and saves it on the machine.
            That way the public and private keys are the same each run.
54      public static void addKeysToContainer()
55      {
56        var parameters = new CspParameters
57        {
58          KeyContainerName = RSAHelper.containerName
59        };
60
61        var rsa = new RSACryptoServiceProvider(parameters);
62
63        Console.WriteLine("Added new key provider to default container");
64      }
65
66      public static string getPublicKeyFromContainer()
67      {
68        var parameters = new CspParameters
69        {
70          KeyContainerName = containerName
71        };
72
73        var rsa = new RSACryptoServiceProvider(parameters);
74        var publicKey = rsa.ExportParameters(false); //Generowanie klucza
              publiczny
75        string publicKeyString = RSAHelper.GetKeyString(publicKey);
76
77        // Display the key information to the console.
78        Console.WriteLine($"Public key retrieved from container");
```

```
79
80        return publicKeyString;
81      }
82
83      public static string getPrivateKeyFromContainer()
84      {
85        var parameters = new CspParameters
86        {
87          KeyContainerName = containerName
88        };
89
90        var rsa = new RSACryptoServiceProvider(parameters);
91        var privateKey = rsa.ExportParameters(true); //Generowanie klucza
              publiczny
92        string privateKeyString = RSAHelper.GetKeyString(privateKey);
93
94        Console.WriteLine("Private key retrieved from container");
95
96        return privateKeyString;
97      }
98      public static string GetKeyString(RSAParameters key)
99      {
100       var stringWriter = new System.IO.StringWriter();
101       var xmlSerializer = new System.Xml.Serialization.XmlSerializer(
              typeof(RSAParameters));
102       xmlSerializer.Serialize(stringWriter, key);
103       return stringWriter.ToString();
104     }
105   }
106 }
```

UserType.cs

```
1  using System.Collections.Generic;
2  using LinqToDB.Mapping;
3
4  namespace DiabetesNeuralNetwork.Models
5  {
6    [Table(Name = "UserType")]
7    public class UserType
8    {
9      [PrimaryKey, Identity]
10     public int userTypeID;
11     [Column(Name = "name"), NotNull]
12     public string name { get; set; }
13     List<User> users { get; set; }
14
15     public override string ToString()
16     {
17       return this.userTypeID + " " + this.name;
18     }
19   }
20 }
```

User.cs

```
using LinqToDB.Mapping;

namespace DiabetesNeuralNetwork
{
  [Table(Name = "User")]
  public class User
  {
    [PrimaryKey, Identity]
    public int userID;
    [Column(Name = "name"), NotNull]
    public string name;
    [Column(Name = "surname"), NotNull]
    public string surname;
    [Column(Name = "email"), NotNull]
    public string email;
    [Column(Name = "userTypeID"), NotNull]
    public int userTypeID;
    [Column(Name = "password"), NotNull]
    public string password;

    public User()
    {

    }
    public User(string name, string surname, string email, string
        password, int userTypeID)
    {
      this.name = name;
      this.surname = surname;
      this.email = email;
      this.password = password;
      this.userTypeID = userTypeID;
    }

    public override string ToString()
    {
      return this.name + " " + this.surname + " " + this.email;
    }
  }
}
```

AuthorizationManager.cs

```
using DiabetesNeuralNetwork.Controllers;
using System;

namespace DiabetesNeuralNetwork
{
  class AuthorizationManager
  {
    public static void LogIn()
    {
      Console.WriteLine("Email:");
      string email = Console.ReadLine();
      Console.WriteLine("Password:");
```

```
13
14        var password = readPassword();
15
16        User user = UserController.FindUserByEmailAndPassword(email,
              RSAHelper.encrypt(password));
17        if(user == null) {
18          Console.WriteLine("Incorrect credentials");
19        }
20        else
21        {
22          LogInUser(user);
23          Console.WriteLine("Hi " + user.name + "! You were successfully
                logged in.");
24        }
25      }
26
27      public static void LogInUser(User user)
28      {
29        LoginStatus.IsLoggedIn = true;
30        LoginStatus.LoggedInUser = user;
31      }
32
33      public static void Register()
34      {
35        try
36        {
37          Console.WriteLine("What user type to register:");
38          UserTypeController.GetAll().ForEach(i => Console.WriteLine("{0}\
                t", i));
39          int userTypeID = Int32.Parse(Console.ReadLine());
40          Console.WriteLine("Name:");
41          string name = Console.ReadLine();
42          Console.WriteLine("Surname:");
43          string surname = Console.ReadLine();
44          Console.WriteLine("Email:");
45          string email = Console.ReadLine();
46          Console.WriteLine("Password:");
47          string password = readPassword();
48
49
50          Validator.ValidateUser(name, surname, email, password,
                userTypeID);
51          User user = new User(name, surname, email, RSAHelper.encrypt(
                password), userTypeID);
52          UserController.Insert(user);
53          LogInUser(user);
54          Console.WriteLine("Successfully registered new user");
55        }
56        catch (Exception ex)
57        {
58          Console.WriteLine(ex.Message);
59        }
60      }
61
62          private static string readPassword()
```

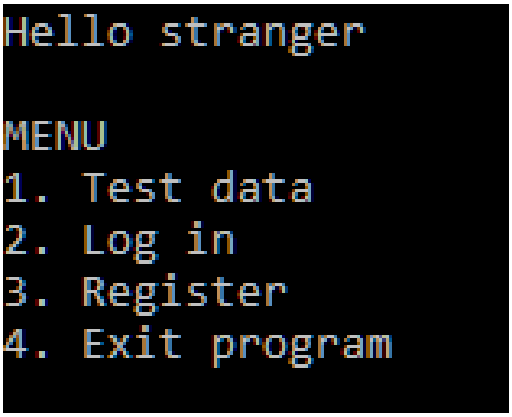
```
63          {
64        var password = string.Empty;
65        ConsoleKey key;
66        do
67        {
68          var keyInfo = Console.ReadKey(intercept: true);
69          key = keyInfo.Key;
70
71          if (key == ConsoleKey.Backspace && password.Length > 0)
72          {
73            Console.Write("\b \b");
74            password = password.Remove(password.Length - 1);
75          }
76          else if (!char.IsControl(keyInfo.KeyChar))
77          {
78            Console.Write("*");
79            password += keyInfo.KeyChar;
80          }
81        } while (key != ConsoleKey.Enter);
82        Console.WriteLine();
83
84        return password;
85      }
86
87          public static void LogOut()
88      {
89        LoginStatus.IsLoggedIn = false;
90        LoginStatus.LoggedInUser = null;
91        Console.WriteLine("Successfully logged out");
92      }
93
94
95
96    }
97  }
```

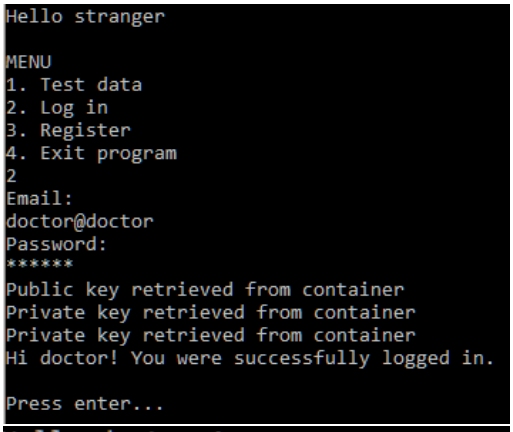
Przykładowe testy:

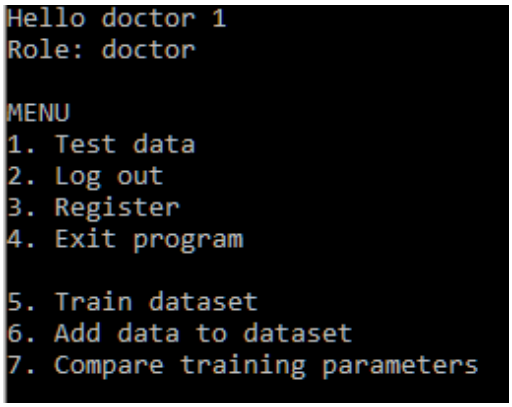
```
1       [TestMethod]
2       public void TestRSACorrectlyEncryptsAndDecrypts()
3       {
4         string testString = "Ala ma kota";
5
6         string encryptedText = RSAHelper.encrypt(testString);
7
8         Assert.AreEqual(testString, RSAHelper.decrypt(encryptedText),
              false, "Original text and same text after encrypt and decrypt
              are not the same");
9       }
10      [TestMethod]
11      public void TestRSAEncryptsTheSameEachTime()
12      {
13        string testString = "Ala ma kota";
14
15        string encryptedText = RSAHelper.encrypt(testString);
```

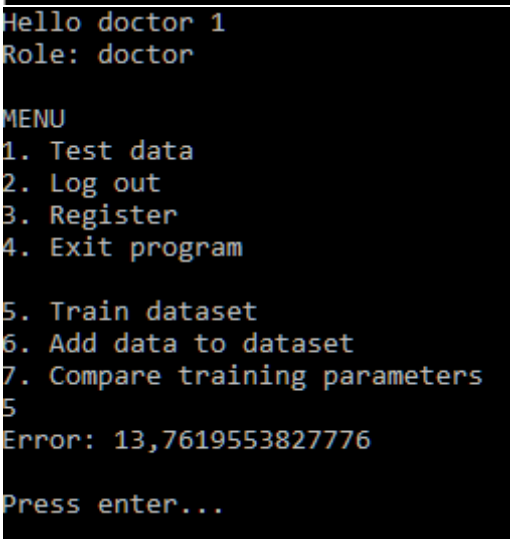
```
16        string anotherEncryptedText = RSAHelper.encrypt(testString);
17
18        Assert.AreEqual(RSAHelper.decrypt(encryptedText), RSAHelper.
              decrypt(anotherEncryptedText), false, "Not the same results
              when encrypting and decrypting the same string");
19      }
20      [TestMethod]
21      public void testUserValidationCorrect()
22      {
23        string name = "test";
24        string surname = "test";
25        string password = "test";
26
27        Validator.ValidateName(name);
28        Validator.ValidateSurname(surname);
29        Validator.ValidatePassword(password);
30      }
31
32      [TestMethod]
33      [ExpectedException(typeof(Exception), "Empty name should not pass
            validation")]
34      public void testUserValidationIncorrect()
35      {
36        Validator.ValidateName("");
37      }
```

Wyglad programu









```
MENU
1. Test data
2. Log out
3. Register
4. Exit program

5. Train dataset
6. Add data to dataset
7. Compare training parameters
6
Pregnancies:
1
Glucose:
123
Blood pressure:
123
Skin thickness:
3
Insulin:
23
BMI:
0,65
Diabetes pedigree function:
0,5
Age:
12
```

```
Testing: 8 1    Error too much
Testing: 9 1    Error too much
Testing: 10 1   Error too much
Testing: 11 1   Error too much
Testing: 12 1   Error too much
Testing: 13 1   Error too much
Testing: 14 1   Error: 44,2429932380331
Testing: 15 1   Error: 47,3197320024758
Testing: 16 1   Error: 42,6153842096591


Current number of layers:       3
Testing: 1 1 1  Error too much

Testing: 2 1 1  Error too much
Testing: 2 2 1  Error too much

Testing: 3 1 1  Error too much
Testing: 3 2 1  Error too much
Testing: 3 3 1  Error too much

Testing: 4 1 1  Error too much
Testing: 4 2 1  Error too much
Testing: 4 3 1  Error too much
Testing: 4 4 1  Error too much

Testing: 5 1 1  Error too much
Testing: 5 2 1  Error too much
Testing: 5 3 1  Error too much
Testing: 5 4 1
```