Modelowanie i analiza systemów informatycznych (Informatyka st. II sem. 2)

Dokumentacja projektu nr 2

Dawid Bitner, Daniel Broczkowski, Damian Kwaśniok

18 grudnia 2021

Część I

Opis programu

Zaprojektowano system wspomagania lekarza (poprzez użycie sieci neuronowej) poprzez automatyczną analizę danych medycznych. System posiada następujące funkcjonalności:

a) ma dwa tryby – uczenie algorytmu/klasyfikacja nowej próbki
b) dane medyczne są szyfrowane i bezpieczne
c) dane lekarzy są szyfrowane
d) możliwość dodania nowego pacjenta/lekarza.

Moduły zostały przetestowane, wykonano testy automatyczne i regresyjne. Sieć neuronowa została przeanalizowana pod względem ilości neuronów/warstw.

Instrukcja obsługi

0.0.1 Instalacja środowiska, wdrożenie projektu

Aby uruchomić backend należy:

a) przejść do folderu ./src/
b) wykonać kolejno polecenia: dotnet restore
c) dotnet ef database update –project medic-decision-support-system
d) uruchomienie testów: dotnet test
e) dotnet run –project medic-decision-support-system

Aby uruchomić część kliencką w trybie deweloperskim należy:

a) w folderze z plikami frontendowymi wykonać polecenie npm install
b) uruchomić serwer: ng serve –open

Dzięki wbudowanym funkcjom .NET baza danych, w przypadku zainstalowanego systemu bazodanowego *PostgreSQL*, odpowiednia baza wraz z tabelami, oraz domyślnymi użytkownikami automatycznie tworzy się w systemie.

0.1 Instrukcja wdrożenia

W celu uruchomienia całego systemu aplikacji zawierającego część frontendową jak i backendową wymagane jest by na komputerze był zainstalowany system operacyjny pozwalający na instalację i kompilację programów .NET w wersji > 3.5 oraz Node.Js w wersji > 12.0. W systemie powinny być zainstalowane właśnie te dwa środowiska.

Podział prac

Dawid Bitner - projekt bazy danych, dokumentacja, część frontendowa,
Daniel Broczkowski - część frontendowa, dokumentacja,
Damian Kwaśniok - część backendowa, dokumentacja.

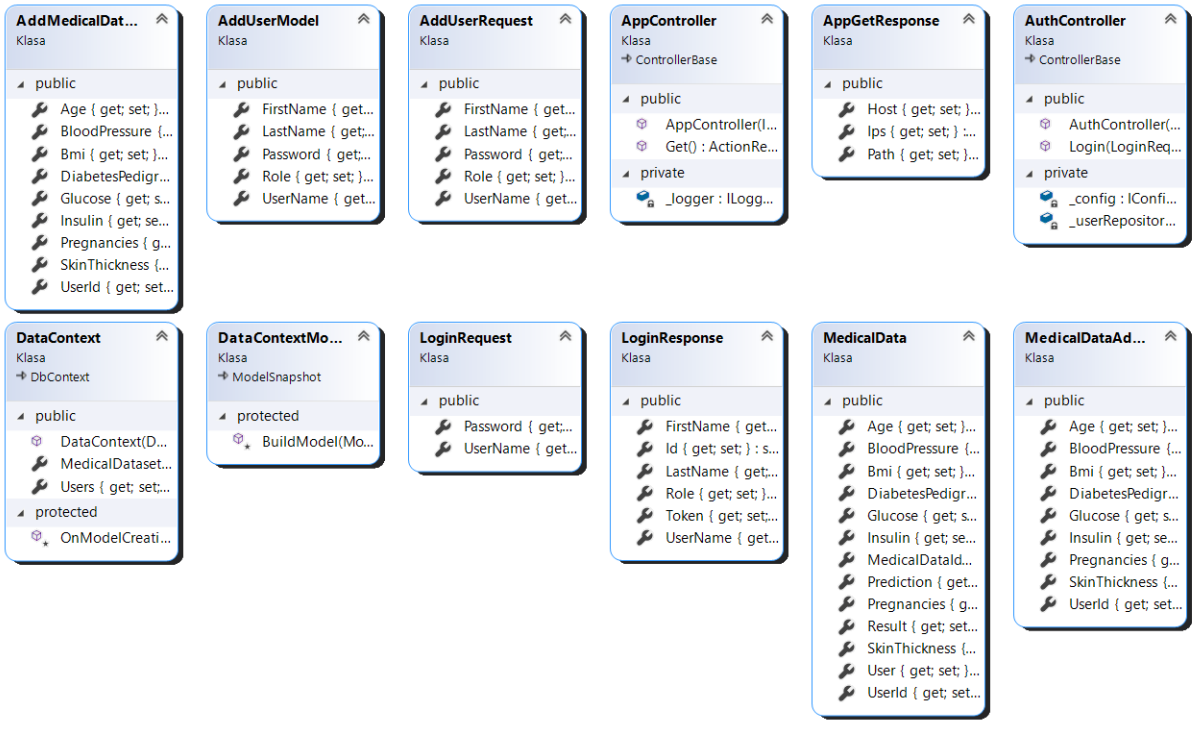
Część II

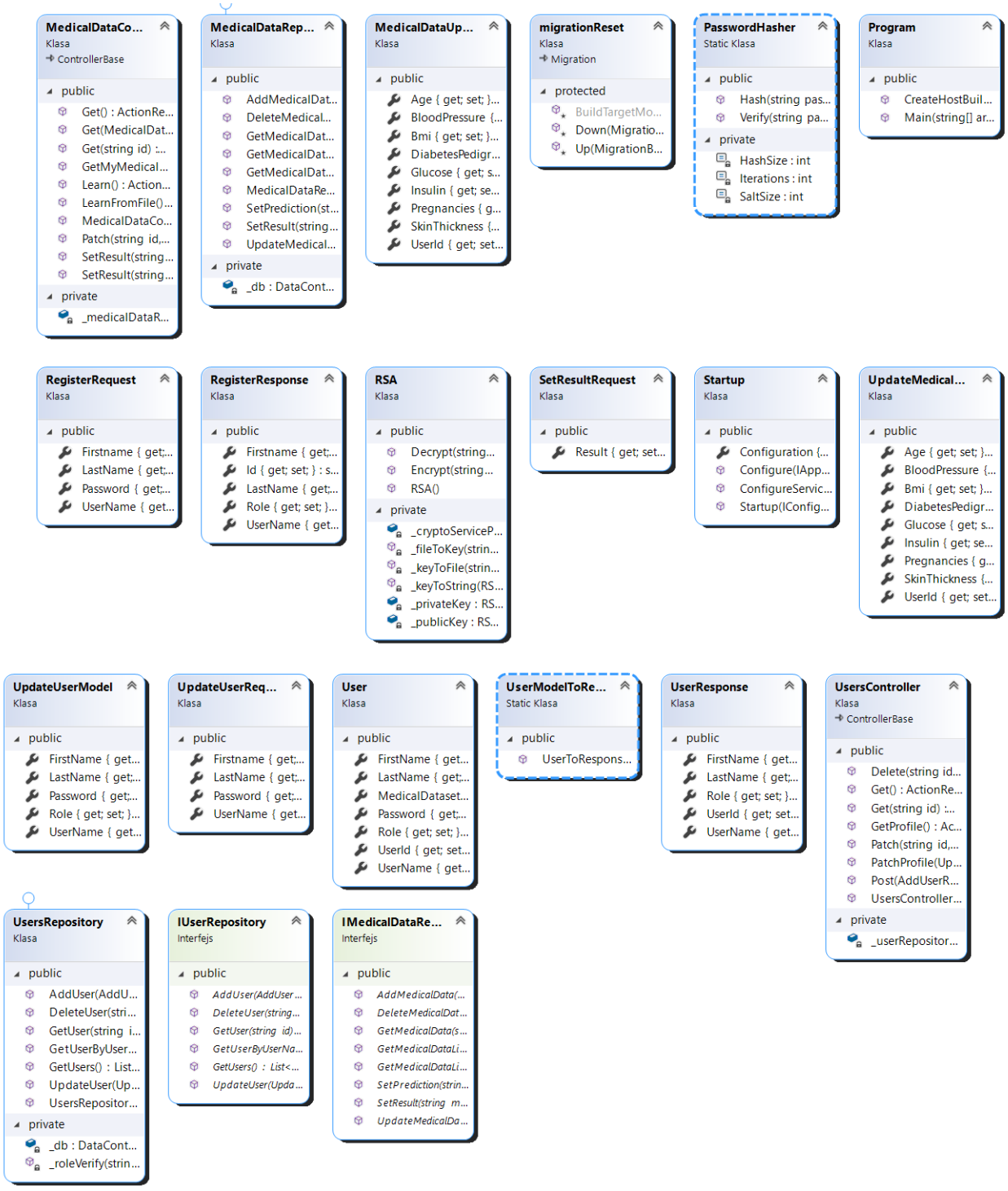
Opis działania

Pacjenci po zalogowaniu posiadają możliwość przeglądania wyników badań należących do nich. Administrator może dodać dwa rodzaje użytkowników: Lekarz i Pacjent. Administrator, czy też lekarz ma możliwość wprowadzenia danych do systemu (ręcznie). Może również wykonać predykcję, sprawdzić za pomocą sieci neuronowej czy wyniki danego badania wskazują na istnienie cukrzycy u pacjenta, czy też nie. Posiada również możliwość przetrenowania modelu decyzyjnego.

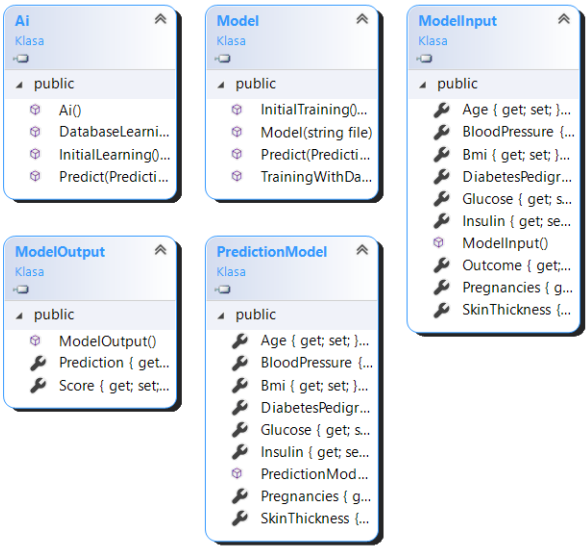
Diagram UML

Medic API:




MedicalDataCo...
Klasa
ControllerBase
public
Get() : ActionRe...
Get(MedicalDat...
Get(string id) :...
GetMyMedical...
Learn() : Action...
LearnFromFile()...
MedicalDataCo...
Patch(string id,...
SetResult(string...
SetResult(string...
private
_medicalDataR...
MedicalDataRep...
Klasa
public
AddMedicalDat...
DeleteMedical...
GetMedicalDat...
GetMedicalDat...
GetMedicalDat...
MedicalDataRe...
SetPrediction(st...
SetResult(string...
UpdateMedical...
private
_db : DataCont...
MedicalDataUp...
Klasa
public
Age { get; set; }...
BloodPressure {...
Bmi { get; set; }...
DiabetesPedigr...
Glucose { get; s...
Insulin { get; se...
Pregnancies { g...
SkinThickness {...
UserId { get; set...
migrationReset
Klasa
Migration
protected
BuildTargetMo...
Down(Migratio...
Up(MigrationB...
PasswordHasher
Static Klasa
public
Hash(string pas...
Verify(string pa...
private
HashSize : int
Iterations : int
SaltSize : int
Program
Klasa
public
CreateHostBuil...
Main(string[] ar...
RegisterRequest
Klasa
public
Firstname { get;...
LastName { get;...
Password { get;...
UserName { get...
RegisterResponse
Klasa
public
Firstname { get;...
Id { get; set; } : s...
LastName { get;...
Role { get; set; }...
UserName { get...
RSA
Klasa
public
Decrypt(string...
Encrypt(string...
RSA()
private
_cryptoServiceP...
_fileToKey(strin...
_keyToFile(strin...
_keyToString(RS...
_privateKey : RS...
_publicKey : RS...
SetResultRequest
Klasa
public
Result { get; set...
Startup
Klasa
public
Configuration {...
Configure(IApp...
ConfigureServic...
Startup(IConfig...
UpdateMedical...
Klasa
public
Age { get; set; }...
BloodPressure {...
Bmi { get; set; }...
DiabetesPedigr...
Glucose { get; s...
Insulin { get; se...
Pregnancies { g...
SkinThickness {...
UserId { get; set...
UpdateUserModel
Klasa
public
FirstName { get...
LastName { get;...
Password { get;...
Role { get; set; }...
UserName { get...
UpdateUserReq...
Klasa
public
Firstname { get;...
LastName { get;...
Password { get;...
UserName { get...
User
Klasa
public
FirstName { get...
LastName { get;...
MedicalDataset...
Password { get;...
Role { get; set; }...
UserId { get; set...
UserName { get...
UserModelToRe...
Static Klasa
public
UserToRespons...
UserResponse
Klasa
public
FirstName { get...
LastName { get;...
Role { get; set; }...
UserId { get; set...
UserName { get...
UsersController
Klasa
ControllerBase
public
Delete(string id...
Get() : ActionRe...
Get(string id) :...
GetProfile() : Ac...
Patch(string id,...
PatchProfile(Up...
Post(AddUserR...
UsersController...
private
_userRepositor...
UsersRepository
Klasa
public
AddUser(AddU...
DeleteUser(stri...
GetUser(string i...
GetUserByUser...
GetUsers() : List...
UpdateUser(Up...
UsersRepositor...
private
_db : DataCont...
_roleVerify(strin...
IUserRepository
Interfejs
public
AddUser(AddUser...
DeleteUser(string...
GetUser(string id)...
GetUserByUserNa...
GetUsers() : List<...
UpdateUser(Upda...
IMedicalDataRe...
Interfejs
public
AddMedicalData(...
DeleteMedicalDat...
GetMedicalData(s...
GetMedicalDataLi...
GetMedicalDataLi...
SetPrediction(strin...
SetResult(string m...
UpdateMedicalDa...

Medic AI:



0.2 Model sieci neuronowej

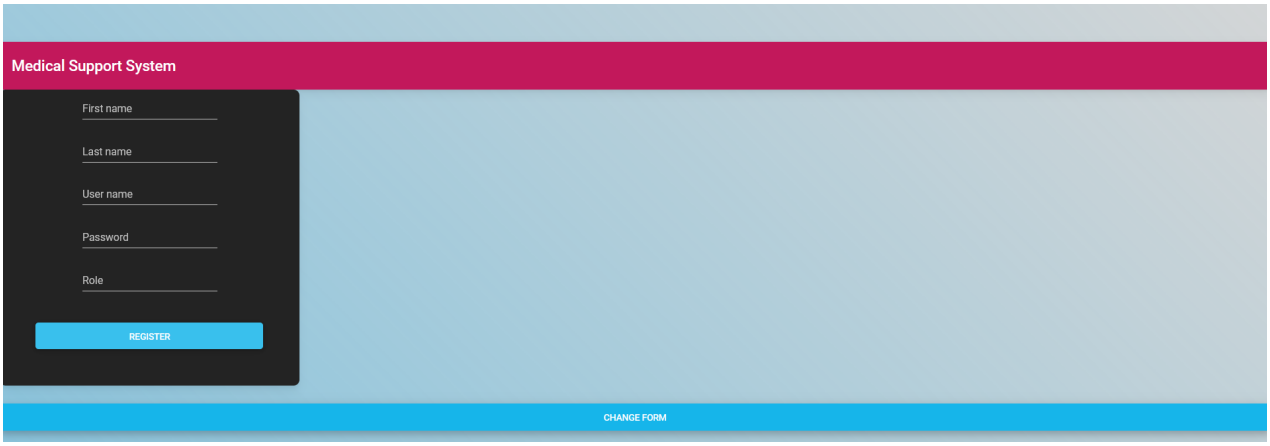
```
1    private double Training(IDataView data, MLContext ctx)
2         {
3              var split = ctx.Data.TrainTestSplit(data, testFraction:
                   0.18);
4
5              var features = split.TrainSet.Schema
6                   .Select(col => col.Name)
7                   .Where(col => col != "Label")
8                   .ToArray();
9              var trainer =  new LbfgsLogisticRegressionBinaryTrainer.
                   Options()
10             {
11                  MaximumNumberOfIterations = 100,
12             };
13             var pipeline = ctx.Transforms.Concatenate("Features",
                   features)
14                  .Append(ctx.BinaryClassification.Trainers.Gam(
                        learningRate: 0.052, numberOfIterations: 25000));
15
16             var model = pipeline.Fit(split.TrainSet);
17
18             var predictions = model.Transform(split.TestSet);
19
20             var metrics = ctx.BinaryClassification.Evaluate(predictions)
                   ;
21
22             ctx.Model.Save(model, data.Schema, "model.zip");
23             _model = model;
24             return metrics.Accuracy;
25        }
```

```
public ModelOutput Predict(PredictionModel input)
        {
            var ctx = new MLContext();
            var model = ctx.Model.Load("model.zip", out _);
            PredictionEngine<ModelInput, ModelOutput> predictionEngine =
                ctx.Model.CreatePredictionEngine<ModelInput, ModelOutput
                    >(model);
            ModelInput data = new ModelInput()
            {
                Age = input.Age,
                Bmi = input.Bmi,
                Glucose = input.Glucose,
                Insulin = input.Insulin,
                Pregnancies = input.Pregnancies,
                BloodPressure = input.BloodPressure,
                SkinThickness = input.SkinThickness,
                DiabetesPedigreeFunction = input.
                    DiabetesPedigreeFunction,
            };
            ModelOutput prediction = predictionEngine.Predict(data);
            return prediction;
        }
```

Model sieci neuronowej, który został użyty w aplikacji składa się z dwóch warstw. Sieć neuronowa jest oparta na klasyfikacji binarnej przy użyciu klasyfikatora opartego na binarnej regresji logistycznej o maksymalnej ilości 100 iteracji. Ze względu na specyfikę problemu, chcąc otrzymać wynik przewidywania, czy pacjent jest zdrowy lub chory została użyta klasyfikacja binarna. Jako wartość tempa uczenia sieci neuronowej (learningRate) została przyjęta wartość 0.052, natomiast za łączną liczbę przebiegów danych treningowych przyjęto liczbę 25000. Dane treningowe stanowiły 18% zbioru danych.

Opis komponentów systemu

Panel rejestracji

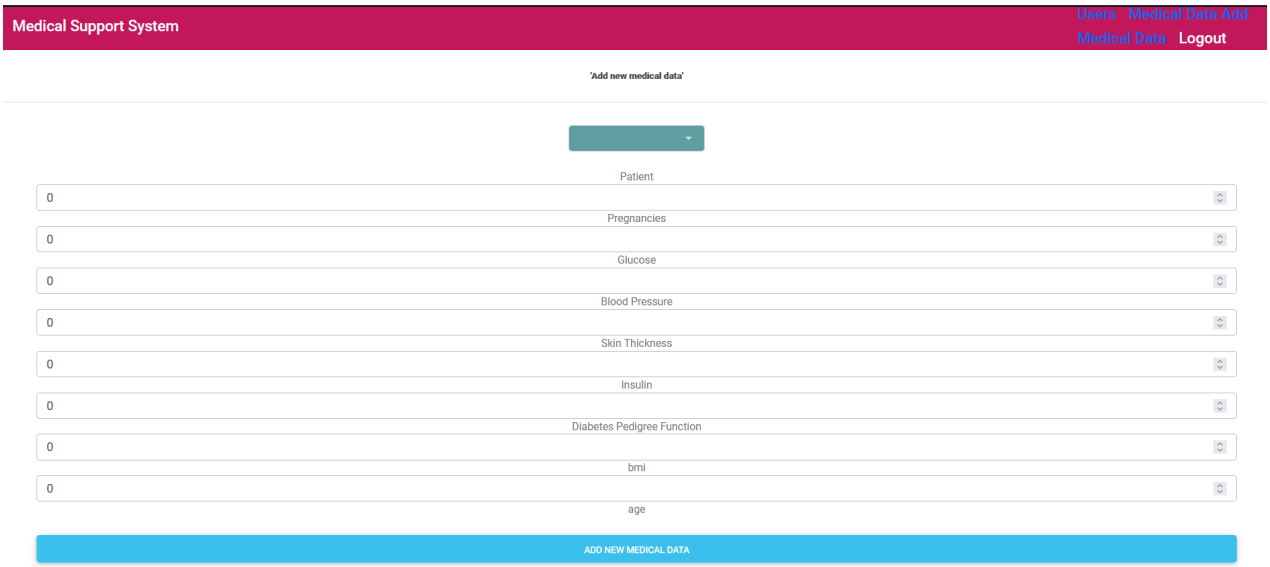


Do rejestracji wymagane jest podanie imienia, nazwiska, loginu oraz hasła i typu konta.

Panel logowania

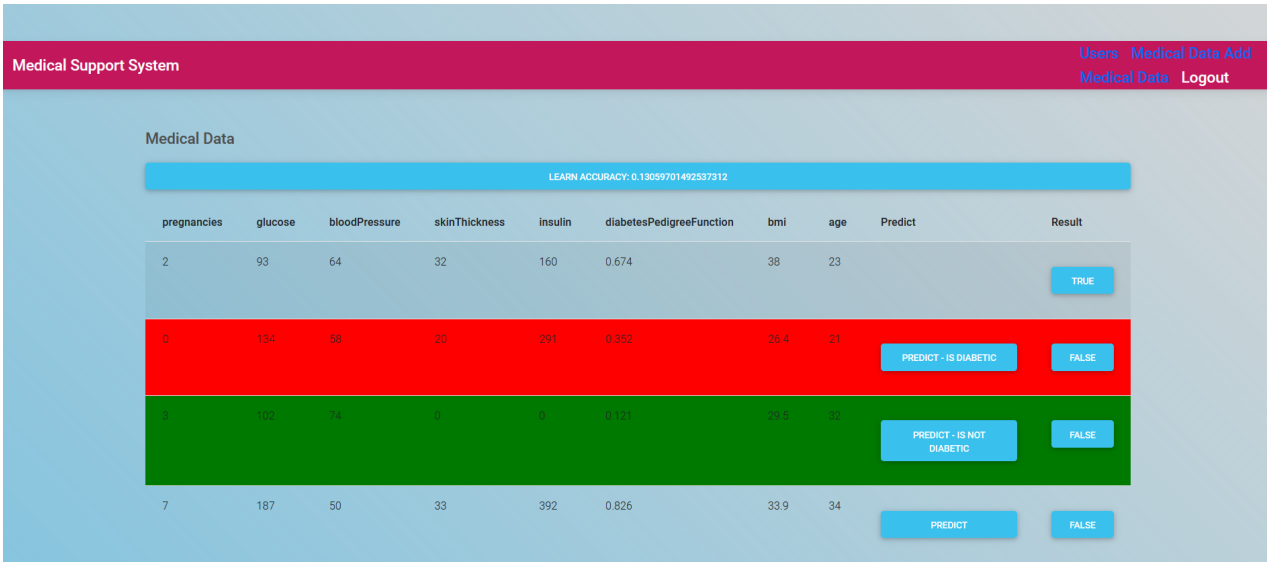
W celu zalogowania należy wprowadzić właściwe dane logowania.

Panel dodawania danych medycznych

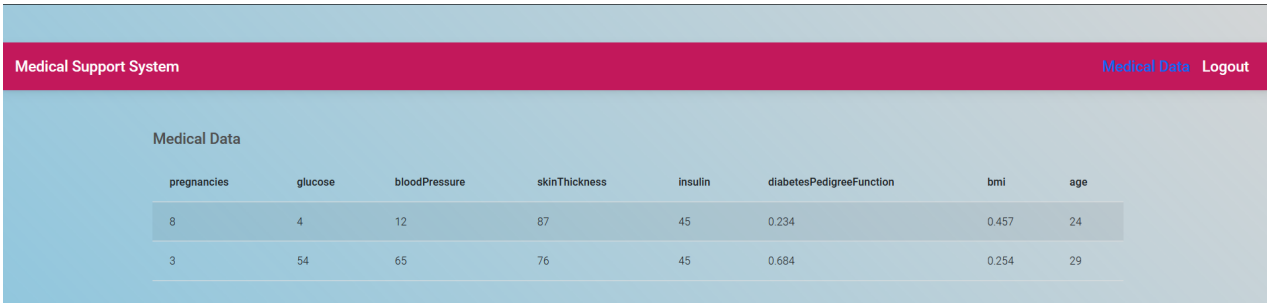


Należy wybrać pacjenta z listy oraz uzupełnić dane reprezentujące jego dane medyczne.

Panel widoku danych medycznych



Panel ten składa się z kilku części i funkcjonalności.
Na górze panelu znajduje się przycisk **Learn**. Po kliknięciu tego przycisku zostanie uruchomione uczenie sieci, a następnie wynik tego uczenie zostanie dopisany w treści przycisku.
Główną cześć widoku stanowi tabela wyświetlająca dane medyczne wprowadzone do systemu. Każdy wiersz posiada dodatkowo dwie kolumny z akcjami do wykonania:
Predict - służy to klasyfikacji próbki. Klasyfikacja może zostać uruchomiona tylko dla próbek nieoznaczonych jako *TRUE* (nieprzeznaczonych do uczenia sieci) albo świeżych próbek wprowadzonych do systemu. Po dokonaniu klasyfikacji w przycisku zostanie zaprezentowany rezultat klasyfikacji oraz dodatkowo wiersz zostanie odpowiednio pokolorowany
Result - służy do oznaczania próbki jako próbki, która ma zostać użyta do uczenia sieci.



Widok pacjenta ogranicza się do widoczności próbek, które należą tylko do niego. Nie może uruchomić uczenia sieci oraz klasyfikacji próbek,

Algorytm szyfrowania

Algorytm RSA składa się z trzech podstawowych kroków:

a) Generacja klucza publicznego i tajnego. Klucz publiczny jest przekazywany wszystkim zainteresowanym i umożliwia zaszyfrowanie danych. Klucz tajny umożliwia rozszyfrowanie danych zakodowanych kluczem publicznym. Jest trzymany w ścisłej tajemnicy.

b) Użytkownik po otrzymaniu klucza publicznego, np. poprzez sieć Internet, koduje za jego pomocą swoje dane i przesyła je w postaci szyfru RSA do adresata dysponującego kluczem

tajnym, np. do banku, firmy komercyjnej, tajnych służb. Klucz publiczny nie musi być chroniony, ponieważ nie umożliwia on rozszyfrowania informacji - proces szyfrowania nie jest odwracalny przy pomocy tego klucza. Zatem nie ma potrzeby jego ochrony i może on być powierzany wszystkim zainteresowanym bez ryzyka złamania kodu.

c) Adresat po otrzymaniu zaszyfrowanej wiadomości rozszyfrowuje ją za pomocą klucza tajnego.

Zalety:
- Algorytm RSA jest bezpieczny dla swoich użytkowników dzięki zastosowaniu złożonych algorytmów matematycznych.
- Algorytm RSA jest trudny do złamania, ponieważ obejmuje faktoryzację liczb pierwszych, które są trudne do faktoryzacji.

Wady:
-Algorytm RSA może działać bardzo wolno w przypadkach, gdy duże dane muszą być szyfrowane przez ten sam komputer.
-Wymaga weryfikacji wiarygodności kluczy publicznych przez stronę trzecią. -Dane przesyłane za pomocą algorytmu RSA mogą zostać naruszone przez pośredników w trakcie wymiany danych, którzy mogą spreparować system klucza publicznego, tak by był przez nich znany.

Pomimo wad, RSA jest uważany obecnie za jeden z najbezpieczniejszych sposobów szyfrowania danych.

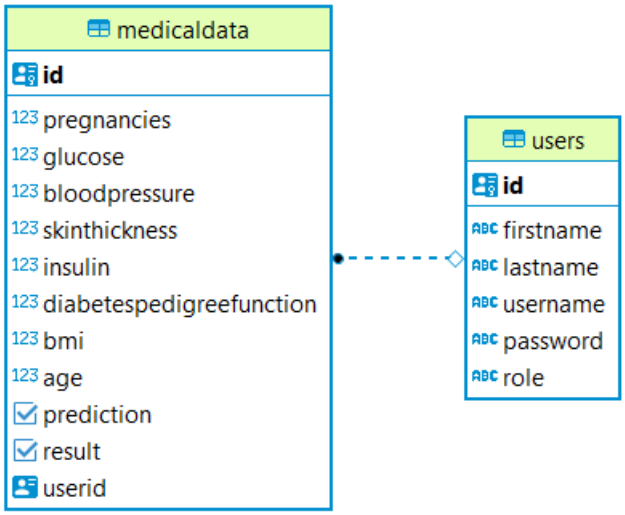
Implementacja:

```
1  using System;
2  using System.IO;
3  using System.Security.Cryptography;
4  using System.Text;
5  using System.Xml;
6  using System.Xml.Serialization;
7
8  namespace medic_api.Helpers
9  {
10     public class RSA
11     {
12         private static RSACryptoServiceProvider _cryptoServiceProvider =
               new RSACryptoServiceProvider(2048);
13         private readonly RSAParameters _privateKey;
14         private readonly RSAParameters _publicKey;
15
16         public RSA()
17         {
18             _privateKey = _fileToKey("privateKey.xml");
19             _publicKey = _fileToKey("publicKey.xml");
20         }
21
22         private RSAParameters _fileToKey(string filename)
23         {
24             try
25             {
26                 XmlDocument xmlDocument = new XmlDocument();
```

```
27                  xmlDocument.Load(filename);
28                  XmlSerializer xmlSerializer = new XmlSerializer(typeof(
                        RSAParameters));
29                  XmlReader xmlReader = new XmlNodeReader(xmlDocument.
                        DocumentElement);
30                  var key = (RSAParameters) xmlSerializer.Deserialize(
                        xmlReader);
31                  return key;
32              }
33              catch
34              {
35                  var key =_cryptoServiceProvider.ExportParameters(true);
36                  _keyToFile(_keyToString(key), filename);
37                  return key;
38              }
39          }
40
41          private void _keyToFile(string key, string filename)
42          {
43              XmlDocument xmlDocument = new XmlDocument();
44              xmlDocument.LoadXml(key);
45              xmlDocument.Save(filename);
46          }
47
48          private string _keyToString(RSAParameters key)
49          {
50              var stringWriter = new StringWriter();
51              var xmlSerializer = new XmlSerializer(typeof(RSAParameters))
                    ;
52              xmlSerializer.Serialize(stringWriter, key);
53              return stringWriter.ToString();
54          }
55
56          public string Encrypt(string plainText)
57          {
58              _cryptoServiceProvider = new RSACryptoServiceProvider();
59              _cryptoServiceProvider.ImportParameters(_publicKey);
60              var data = Encoding.UTF8.GetBytes(plainText);
61              var cypher = _cryptoServiceProvider.Encrypt(data, true);
62              return Convert.ToBase64String(cypher);
63          }
64
65          public string Decrypt(string cypher)
66          {
67              var dataBytes = Convert.FromBase64String(cypher);
68              _cryptoServiceProvider.ImportParameters(_privateKey);
69              var plainText = _cryptoServiceProvider.Decrypt(dataBytes,
                    true);
70              return Encoding.UTF8.GetString(plainText);
71          }
72      }
73  }
```

Bazy danych

Struktura bazy danych projektu prezentuje się następująco:



Baza danych w projekcie została ograniczona do minimum. Stworzona jest w Postgresie. Posiada dwie tabele. Z badaniami i użytkownikami. Tabela badań zawiera parametry badań, oraz kolumnę z identyfikatorem użytkownika do którego należą badania. Natomiast tabela użytkowników zawiera informacje o nich, w tym przypisaną rolę w systemie.

Zalety i wady systemów eksperckich

System ekspertowy to system komputerowy zawierający w sobie wyspecjalizowaną wiedzę na temat określonego obszaru ludzkiej działalności, przy czym wiedza ta jest tak zorganizowana, że umożliwia systemowi wejście w interakcyjny dialog z użytkownikiem, w wyniku czego system może oferować rady lub podpowiadać decyzje, jak również objaśniać proces prowadzonego wnioskowania. System ekspercki może służyć m.in.: do diagnozowania chorób, a raczej dopomocy przy ich diagnozowaniu, jak na przykład ma to miejsce w tym projekcie.

Do głównych zalet systemów eksperckich można zaliczyć:
- mniejszy koszt pojedynczej ekspertyzy
- automatyczne wyjaśnianie decyzji
- szybkość uzyskania ekspertyzy
- stała, niewrażliwa na emocje i czynniki zewnętrzne ekspertyza
- uczenie metodą prób i błędów
- inteligentny interfejs człowiek-komputer

Naszym zdaniem do głównych wad systemów eksperckich należą m.in.:
- Konieczność wykonania treningu na najlepiej dużym zbiorze danych, gdzie możemy nie mieć pewności że rezultaty są poprawne
- Wiedza przetwarzana jest w sposób mechaniczny

- Wąski zakres
- Zasugerowany wynik przez system może wpływać na decyzję człowieka w przypadku gdy użytkownik systemu zbyt mocno ufa wynikowi podanemu przez system.

Implementacja systemu eksperckiego

```
        public ActionResult<double> Learn()
        {
            Ai ai = new Ai();
            var data = _medicalDataRepository.GetMedicalDataList().Where
                (d => d.Result != null).ToList();
            var preparedList = new List<ModelInput>();
            foreach (var medicalData in data)
            {
                var item = new ModelInput()
                {
                    Age = medicalData.Age,
                    Bmi = (float)medicalData.Bmi,
                    Glucose = medicalData.Glucose,
                    Insulin = medicalData.Insulin,
                    Outcome = medicalData.Result != null && (bool)
                        medicalData.Result,
                    Pregnancies = medicalData.Pregnancies,
                    BloodPressure = medicalData.BloodPressure,
                    SkinThickness = medicalData.SkinThickness,
                    DiabetesPedigreeFunction = (float) medicalData.
                        DiabetesPedigreeFunction,
                };
                preparedList.Add(item);
            }

            if (preparedList.Count < 15)
            {
                return Problem("Not enough data");
            }
            var accuracy = ai.DatabaseLearning(preparedList);
            return Ok(accuracy);
        }

        public ActionResult<string> SetResult(string id, [FromBody]
            SetResultRequest body)
        {
            var resultId = _medicalDataRepository.SetResult(id, body.
                Result);
            return Ok(resultId);
        }
```

Testy

Wszystkie testy w aplikacji na ten moment kończą się sukcesem:

```
  medic-test -> C:\Users\damia\OneDrive\Pulpit\Studia\MIASI\Projekt 2\medic-decision-support-system\backend\medic-test\b
in\Debug\netcoreapp3.1\medic-test.dll
Przebieg testu dla: C:\Users\damia\OneDrive\Pulpit\Studia\MIASI\Projekt 2\medic-decision-support-system\backend\medic-te
st\bin\Debug\netcoreapp3.1\medic-test.dll (.NETCoreApp,Version=v3.1)
Narzędzie wiersza polecenia firmy Microsoft (R) służące do wykonania testów (wersja 16.9.1)
Copyright (c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

Trwa rozpoczynanie wykonywania testu, czekaj...
Łączna liczba plików testowych dopasowanych do określonego wzorca: 1.

Powodzenie!   - niepowodzenie:     0, powodzenie:    11, pominięto:     0, łącznie:    11, czas trwania: 439 ms - medic
-test.dll (netcoreapp3.1)

C:\Users\damia\OneDrive\Pulpit\Studia\MIASI\Projekt 2\medic-decision-support-system\backend\medic-test>
```

Zostały przeprowadzone następujące kilka testów dotyczących pobierania danych medycznych:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Net.Http;
4  using System.Security.Claims;
5  using System.Security.Principal;
6  using medic_api.Controllers.MedicalData;
7  using medic_api.DAL.Models;
8  using medic_api.DAL.Repository.MedicalData;
9  using Microsoft.AspNetCore.Http;
10 using Microsoft.AspNetCore.Mvc;
11 using Moq;
12 using Xunit;
13
14 namespace medic_test.Controllers
15 {
16     public class MedicalDataControllerTest
17     {
18         [Fact]
19         public void MedicalDataControllerGet()
20         {
21             var newId = new Guid();
22             var mock = new Mock<IMedicalDataRepository>();
23             var controller = new MedicalDataController(mock.Object);
24             var data = controller.Get(newId.ToString());
25             mock.Verify(x => x.GetMedicalData(newId.ToString()), Times.
                   Once);
26             Assert.IsType<ActionResult<MedicalData>>(data);
27         }
28
29         [Fact]
30         public void MedicalDataControllerGetList()
31         {
32             var mock = new Mock<IMedicalDataRepository>();
33             var controller = new MedicalDataController(mock.Object);
34             var data = controller.Get();
35             mock.Verify(x => x.GetMedicalDataList(), Times.Once);
36             Assert.IsType<ActionResult<List<MedicalData>>>(data);
37         }
38
39         [Fact]
40         public void MedicalDataControllerGetListByUser()
```

```
41          {
42              var newId = new Guid();
43              var mock = new Mock<IMedicalDataRepository>();
44              var user = new ClaimsPrincipal(new ClaimsIdentity(new Claim
                    []
45              {
46                  new Claim(ClaimTypes.Sid, newId.ToString()),
47              }, "mock"));
48
49              var controller = new MedicalDataController(mock.Object);
50              controller.ControllerContext = new ControllerContext()
51              {
52                  HttpContext = new DefaultHttpContext() { User = user }
53              };
54              var data = controller.GetMyMedicalData();
55              mock.Verify(x => x.GetMedicalDataListByUser(newId.ToString()
                    ), Times.Once);
56              Assert.IsType<ActionResult<List<MedicalData>>>(data);
57          }
58      }
59  }
```

Przy pomocy biblioteki Xunit zostały przeprowadzone testy szyfrowania algorytmem RSA.

```
1  using System;
2  using medic_api.Helpers;
3  using Xunit;
4  using Xunit.Abstractions;
5
6  namespace medic_test
7  {
8      public class Encryptor
9      {
10         [Fact]
11         public void Encrypt()
12         {
13             RSA rsa = new RSA();
14             RSA rsa2 = new RSA();
15             string testCase = "Admin123 jakis test /n\n";
16             var encrypted= rsa.Encrypt(testCase);
17             var decrypted = rsa2.Decrypt(encrypted);
18             Assert.Equal(testCase, decrypted);
19         }
20     }
21 }
```

```
1  using medic_api.Helpers;
2  using Xunit;
3
4  namespace medic_test
5  {
6      public class Hasher
7      {
8          [Theory]
9          [InlineData("", "")]
```

```
10          public void HasherTest(string expected, string input)
11          {
12              Assert.Equal(expected, PasswordHasher.Hash(input));
13          }
14
15          [Theory]
16          [InlineData("some test", "some test", true)]
17          [InlineData("some test", "some test 2", false)]
18          [InlineData("some test 2", "some test", false)]
19          [InlineData("", "", false)]
20          [InlineData("a", "", false)]
21          [InlineData("", "a", false)]
22          public void Verify(string str, string str2, bool equal)
23          {
24              var hash = PasswordHasher.Hash(str);
25              Assert.Equal(equal, PasswordHasher.Verify(str2, hash));
26          }
27      }
28  }
```

Pod koniec pisania projektu, pokrycie kodu testami wynosiło ok. 2 procent.

Pełen kod programu

0.2.1 AppGetResponse.cs

```
using System.Collections.Generic;

namespace medic_api.Controllers.DTO
{
    public class AppGetResponse
    {
        public string Host { get; set; }
        public string Path { get; set; }
        public IList<string> Ips { get; set; }
    }
}
```

0.2.2 AppController.cs

```
using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System.Net;
using medic_api.Controllers.DTO;
using Microsoft.AspNetCore.Authorization;

namespace medic_api.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    [Authorize(Policy = "Patient")]
    public class AppController : ControllerBase
    {
        private readonly ILogger<AppController> _logger;

        public AppController(ILogger<AppController> logger)
        {
            _logger = logger;
        }

        [HttpGet]
        public ActionResult<AppGetResponse> Get()
        {
            IPAddress[] localIPs = Dns.GetHostAddresses(Dns.GetHostName
                ());
            List<string> ipList = new List<string>();
            foreach (var ipAddress in localIPs)
            {
                ipList.Add(ipAddress.ToString());
            }

            AppGetResponse response = new AppGetResponse()
            {
```

```
35                  Host = Environment.MachineName,
36                  Ips = ipList,
37                  Path = Request.Path,
38              };
39              return Ok(response);
40          }
41      }
42  }
```

0.2.3 LoginRequest.cs

```
1  using System.ComponentModel.DataAnnotations;
2
3  namespace medic_api.Controllers.Auth.DTO
4  {
5      public class LoginRequest
6      {
7          [Required]
8          public string UserName { get; set; }
9          [Required]
10         public string Password { get; set; }
11     }
12 }
```

0.2.4 LoginResponse.cs

```
1  namespace medic_api.Controllers.Auth.DTO
2  {
3      public class LoginResponse
4      {
5          public string UserName { get; set; }
6          public string FirstName { get; set; }
7          public string LastName { get; set; }
8          public string Role { get; set; }
9          public string Id { get; set; }
10         public string Token { get; set; }
11     }
12 }
```

0.2.5 RegisterRequest.cs

```
1  using System.ComponentModel.DataAnnotations;
2
3  namespace medic_api.Controllers.Auth.DTO
4  {
5      public class RegisterRequest
6      {
7          [Required]
8          public string UserName { get; set; }
```

```
9           [Required]
10          public string Firstname { get; set; }
11          [Required]
12          public string LastName { get; set; }
13          [Required, MinLength(4), MaxLength(16)]
14          public string Password { get; set; }
15      }
16  }
```

0.2.6 RegisterResponse.cs

```
1  namespace medic_api.Controllers.Auth.DTO
2  {
3      public class RegisterResponse
4      {
5          public string UserName { get; set; }
6          public string Firstname { get; set; }
7          public string LastName { get; set; }
8          public string Role { get; set; }
9          public string Id { get; set; }
10     }
11 }
```

0.2.7 AuthController.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.IdentityModel.Tokens.Jwt;
4  using System.Security.Claims;
5  using System.Text;
6  using System.Threading.Tasks;
7  using medic_api.Controllers.Auth.DTO;
8  using medic_api.DAL.Repository;
9  using medic_api.Helpers;
10 using Microsoft.AspNetCore.Mvc;
11 using Microsoft.Extensions.Configuration;
12 using Microsoft.IdentityModel.Tokens;
13
14 namespace medic_api.Controllers.Auth
15 {
16     [ApiController]
17     [Route("api/[controller]")]
18     public class AuthController : ControllerBase
19     {
20         private readonly IUserRepository _userRepository;
21         private readonly IConfiguration _config;
22
23         public AuthController(IUserRepository userRepository,
               IConfiguration config)
24         {
25             _userRepository = userRepository;
```

```
26              _config = config;
27          }
28
29          [HttpPost, Route("login")]
30          public async Task<ActionResult<LoginResponse>> Login([FromBody]
                LoginRequest body)
31          {
32              var dataEncryptor = new RSA();
33              var user = _userRepository.GetUserByUserName(body.UserName);
34              if (user == null) return Unauthorized("Wrong username or
                    password!");
35              if(!PasswordHasher.Verify(body.Password, user.Password))
                    return Unauthorized("Wrong username or password!");
36              var secretKey = new SymmetricSecurityKey(Encoding.UTF8.
                    GetBytes(_config.GetValue<string>("Security:SecretKey")))
                    ;
37              var signingCredentials = new SigningCredentials(secretKey,
                    SecurityAlgorithms.HmacSha256);
38              var tokenOptions = new JwtSecurityToken(
39                  claims: new List<Claim>
40                  {
41                      new Claim(ClaimTypes.Name, user.UserName),
42                      new Claim(ClaimTypes.Role, user.Role),
43                      new Claim(ClaimTypes.Sid, user.UserId.ToString()),
44                  },
45                  expires: DateTime.Now.AddHours(8),
46                  signingCredentials: signingCredentials
47              );
48              var tokenString = new JwtSecurityTokenHandler().WriteToken(
                    tokenOptions);
49              LoginResponse response = new LoginResponse()
50              {
51                  FirstName = dataEncryptor.Decrypt(user.FirstName),
52                  Id = user.UserId.ToString(),
53                  Role = user.Role,
54                  LastName = dataEncryptor.Decrypt(user.LastName),
55                  UserName = user.UserName,
56                  Token = tokenString,
57              };
58              return Ok(response);
59          }
60      }
61  }
```

0.2.8 MedicalDataAddRequest.cs

```
1  using System.ComponentModel.DataAnnotations;
2  using Microsoft.AspNetCore.Mvc.ModelBinding;
3  using Newtonsoft.Json;
4
5  namespace medic_api.Controllers.MedicalData.DTO
6  {
7      public class MedicalDataAddRequest
```

```
8       {
9           [JsonRequired]
10          public int Pregnancies { get; set; }
11          [JsonRequired]
12          public int Glucose { get; set; }
13          [JsonRequired]
14          public int BloodPressure { get; set; }
15          [JsonRequired]
16          public int SkinThickness { get; set; }
17          [JsonRequired]
18          public int Insulin { get; set; }
19          [JsonRequired]
20          public double DiabetesPedigreeFunction { get; set; }
21          [JsonRequired]
22          public double Bmi { get; set; }
23          [JsonRequired]
24          public int Age { get; set; }
25          [JsonRequired]
26          public string UserId { get; set; }
27      }
28  }
```

0.2.9 MedicalDataUpdateRequest.cs

```
1  using System.ComponentModel.DataAnnotations;
2  using System.Runtime.InteropServices;
3  using Microsoft.AspNetCore.Mvc.ModelBinding;
4  using Newtonsoft.Json;
5
6  namespace medic_api.Controllers.MedicalData.DTO
7  {
8      public class MedicalDataUpdateRequest
9      {
10         public int? Pregnancies { get; set; }
11         public int? Glucose { get; set; }
12         public int? BloodPressure { get; set; }
13         public int? SkinThickness { get; set; }
14         public int? Insulin { get; set; }
15         public double? DiabetesPedigreeFunction { get; set; }
16         public double? Bmi { get; set; }
17         public int? Age { get; set; }
18         public string? UserId { get; set; }
19     }
20 }
```

0.2.10 SetResultRequest.cs

```
1  using Newtonsoft.Json;
2
3  namespace medic_api.Controllers.MedicalData.DTO
4  {
```

```
5       public class SetResultRequest
6       {
7           [JsonRequired]
8           public bool Result { get; set; }
9       }
10  }
```

0.2.11 MedicalDataController.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Security.Claims;
5   using medic_ai;
6   using medic_api.Controllers.MedicalData.DTO;
7   using medic_api.DAL.Repository.MedicalData;
8   using Microsoft.AspNetCore.Authorization;
9   using Microsoft.AspNetCore.Mvc;
10
11  namespace medic_api.Controllers.MedicalData
12  {
13      [ApiController]
14      [Route("api/[controller]")]
15      [Authorize(Policy = "Patient")]
16      public class MedicalDataController : ControllerBase
17      {
18          private readonly IMedicalDataRepository _medicalDataRepository;
19
20          public MedicalDataController(IMedicalDataRepository
                medicalDataRepository)
21          {
22              _medicalDataRepository = medicalDataRepository;
23          }
24
25          [HttpGet]
26          public ActionResult<List<DAL.Models.MedicalData>> Get()
27          {
28              var medicalDataset = _medicalDataRepository.
                    GetMedicalDataList();
29              return Ok(medicalDataset);
30          }
31
32          [HttpGet, Route("me")]
33          public ActionResult<List<DAL.Models.MedicalData>>
                GetMyMedicalData()
34          {
35              var userId = this.User.Claims.FirstOrDefault(c => c.Type ==
                    ClaimTypes.Sid)?.Value;
36              var medicalDataset = _medicalDataRepository.
                    GetMedicalDataListByUser(userId);
37              return Ok(medicalDataset);
38          }
39
```

```
40          [HttpGet, Route("{id}")]
41          public ActionResult<DAL.Models.MedicalData> Get(string id)
42          {
43              var medicalData = _medicalDataRepository.GetMedicalData(id);
44              return Ok(medicalData);
45          }
46
47          [HttpPost]
48          [Authorize(Policy = "Doctor")]
49          public ActionResult<string> Get([FromBody] MedicalDataAddRequest
                body)
50          {
51              AddMedicalDataModel model = new AddMedicalDataModel()
52              {
53                  Age = body.Age,
54                  Bmi = body.Bmi,
55                  Glucose = body.Glucose,
56                  Insulin = body.Insulin,
57                  Pregnancies = body.Pregnancies,
58                  BloodPressure = body.BloodPressure,
59                  SkinThickness = body.SkinThickness,
60                  UserId = new Guid(body.UserId),
61                  DiabetesPedigreeFunction = body.DiabetesPedigreeFunction
                        ,
62              };
63              var newId = _medicalDataRepository.AddMedicalData(model);
64              return Ok(newId);
65          }
66
67          [HttpPatch, Route("{id}")]
68          [Authorize(Policy = "Doctor")]
69          public ActionResult<string> Patch(string id, [FromBody]
               MedicalDataUpdateRequest body)
70          {
71              UpdateMedicalDataModel model = new UpdateMedicalDataModel()
72              {
73                  Age = body.Age,
74                  Bmi = body.Bmi,
75                  Glucose = body.Glucose,
76                  Insulin = body.Insulin,
77                  Pregnancies = body.Pregnancies,
78                  BloodPressure = body.BloodPressure,
79                  SkinThickness = body.SkinThickness,
80                  UserId = body.UserId,
81                  DiabetesPedigreeFunction = body.DiabetesPedigreeFunction
                        ,
82              };
83
84              var updatedId = _medicalDataRepository.UpdateMedicalData(
                    model, id);
85
86              return Ok(updatedId);
87          }
88
89          [HttpPost, Route("{id}/result")]
```

```
90          [Authorize(Policy = "Doctor")]
91          public ActionResult<string> SetResult(string id, [FromBody]
                SetResultRequest body)
92          {
93              var resultId = _medicalDataRepository.SetResult(id, body.
                    Result);
94              return Ok(resultId);
95          }
96
97          [HttpPost, Route("{id}/prediction")]
98          [Authorize(Policy = "Doctor")]
99          public ActionResult<string> SetResult(string id)
100         {
101             Ai ai = new Ai();
102             var data = _medicalDataRepository.GetMedicalData(id);
103             PredictionModel model = new PredictionModel()
104             {
105                 Age = data.Age,
106                 Bmi = (float) data.Bmi,
107                 Glucose = data.Glucose,
108                 Insulin = data.Insulin,
109                 Pregnancies = data.Pregnancies,
110                 BloodPressure = data.BloodPressure,
111                 SkinThickness = data.SkinThickness,
112                 DiabetesPedigreeFunction = (float) data.
                        DiabetesPedigreeFunction,
113
114             };
115             var result = ai.Predict(model);
116             _medicalDataRepository.SetPrediction(id, result.Prediction);
117             return Ok(result);
118         }
119
120         [HttpPost]
121         [Route("learn")]
122         [Authorize(Policy = "Admin")]
123         public ActionResult<double> Learn()
124         {
125             Ai ai = new Ai();
126             var data = _medicalDataRepository.GetMedicalDataList().Where
                    (d => d.Result != null).ToList();
127             var preparedList = new List<ModelInput>();
128             foreach (var medicalData in data)
129             {
130                 var item = new ModelInput()
131                 {
132                     Age = medicalData.Age,
133                     Bmi = (float)medicalData.Bmi,
134                     Glucose = medicalData.Glucose,
135                     Insulin = medicalData.Insulin,
136                     Outcome = medicalData.Result != null && (bool)
                            medicalData.Result,
137                     Pregnancies = medicalData.Pregnancies,
138                     BloodPressure = medicalData.BloodPressure,
139                     SkinThickness = medicalData.SkinThickness,
```

```
                    DiabetesPedigreeFunction = (float) medicalData.
                        DiabetesPedigreeFunction,
                };
                preparedList.Add(item);
            }

            if (preparedList.Count < 15)
            {
                return Problem("Not enough data");
            }
            var accuracy = ai.DatabaseLearning(preparedList);
            return Ok(accuracy);
        }

        [HttpPost]
        [Route("learnfromfile")]
        [Authorize(Policy = "Admin")]
        public ActionResult<double> LearnFromFile()
        {
            Ai ai = new Ai();
            var accuracy = ai.InitialLearning();
            return Ok(accuracy);
        }
    }
}
```

0.2.12 AddUserRequest.cs

```
using System.ComponentModel.DataAnnotations;
using Newtonsoft.Json;

namespace medic_api.Controllers.Users
{
    public class AddUserRequest
    {
        [JsonRequired]
        public string Role { get; set; }
        [JsonRequired]
        public string UserName { get; set; }
        [JsonRequired]
        public string FirstName { get; set; }
        [JsonRequired]
        public string LastName { get; set; }
        [JsonRequired, MinLength(4), MaxLength(16)]
        public string Password { get; set; }
    }
}
```

0.2.13 UpdateUserRequest.cs

```
using System.ComponentModel.DataAnnotations;
```

```
2
3  namespace medic_api.Controllers.Users
4  {
5      public class UpdateUserRequest
6      {
7          public string? UserName { get; set; }
8          public string? Firstname { get; set; }
9          public string? LastName { get; set; }
10         [MinLength(4), MaxLength(16)]
11         public string? Password { get; set; }
12     }
13 }
```

0.2.14 UserResponse.cs

```
1  namespace medic_api.Controllers.Users
2  {
3      public class UserResponse
4      {
5          public string UserId { get; set; }
6          public string FirstName { get; set; }
7          public string LastName { get; set; }
8          public string UserName { get; set; }
9          public string Role { get; set; }
10     }
11 }
```

0.2.15 UsersController.cs

```
1  using System.Collections.Generic;
2  using System.Linq;
3  using System.Security.Claims;
4  using medic_api.DAL;
5  using medic_api.DAL.Models;
6  using medic_api.DAL.Repository;
7  using medic_api.Helpers;
8  using Microsoft.AspNetCore.Authorization;
9  using Microsoft.AspNetCore.Mvc;
10
11 namespace medic_api.Controllers.Users
12 {
13     [ApiController]
14     [Route("api/[controller]")]
15     public class UsersController : ControllerBase
16     {
17         private readonly IUserRepository _userRepository;
18
19         public UsersController(IUserRepository userRepository)
20         {
21             _userRepository = userRepository;
22         }
```

```
23
24          [HttpGet]
25          [Authorize(Policy = "Patient")]
26          [Route("me")]
27          public ActionResult<UserResponse> GetProfile()
28          {
29              var userId = this.User.Claims.FirstOrDefault(c => c.Type ==
                    ClaimTypes.Sid)?.Value;
30              var user = _userRepository.GetUser(userId);
31              var response = Helpers.UserModelToResponse.UserToResponse(
                    user);
32              return Ok(response);
33          }
34
35          [HttpPatch]
36          [Authorize(Policy = "Patient")]
37          [Route("me")]
38          public ActionResult<string> PatchProfile([FromBody]
                UpdateUserModel body)
39          {
40              var dataEncryptor = new RSA();
41              var userId = this.User.Claims.FirstOrDefault(c => c.Type ==
                    ClaimTypes.Sid)?.Value;
42              var updateUser = new UpdateUserModel()
43              {
44                  Password = Helpers.PasswordHasher.Hash(body.Password),
45                  Role = body.Role,
46                  FirstName = dataEncryptor.Encrypt(body.FirstName),
47                  LastName = dataEncryptor.Encrypt(body.LastName),
48                  UserName = body.UserName,
49              };
50              var user = _userRepository.UpdateUser(updateUser, userId);
51              return Ok(user);
52          }
53
54          [HttpGet]
55          [Authorize(Policy = "Doctor")]
56          public ActionResult<List<UserResponse>> Get()
57          {
58              var users = _userRepository.GetUsers();
59              var userList = new List<UserResponse>();
60              foreach (var user in users)
61              {
62                  var newResponse = Helpers.UserModelToResponse.
                        UserToResponse(user);
63                  userList.Add(newResponse);
64              }
65              return Ok(userList);
66          }
67
68          [HttpGet]
69          [Authorize(Policy = "Doctor")]
70          [Route("{id}")]
71          public ActionResult<UserResponse> Get(string id)
72          {
```

```
73              var user = _userRepository.GetUser(id);
74              var response = Helpers.UserModelToResponse.UserToResponse(
                    user);
75              return Ok(response);
76          }
77
78          [HttpPost]
79          [Authorize(Policy = "Doctor")]
80          public ActionResult<string> Post([FromBody] AddUserRequest body)
81          {
82              var dataEncryptor = new RSA();
83              if (body.Role == "Doctor" || body.Role == "Admin")
84              {
85                  var role = this.User.Claims.FirstOrDefault(c => c.Type
                        == ClaimTypes.Role)?.Value;
86                  if (role == "Doctor") return BadRequest("Only Admin can
                        add new doctor or admin");
87              }
88              var newUser = new AddUserModel()
89              {
90                  Password = Helpers.PasswordHasher.Hash(body.Password),
91                  Role = body.Role,
92                  FirstName = dataEncryptor.Encrypt(body.FirstName),
93                  LastName = dataEncryptor.Encrypt(body.LastName),
94                  UserName = body.UserName,
95              };
96              var user = _userRepository.AddUser(newUser);
97              return Ok(user);
98          }
99
100         [HttpPatch]
101         [Authorize(Policy = "Admin")]
102         [Route("{id}")]
103         public ActionResult<string> Patch(string id, [FromBody]
                UpdateUserModel body)
104         {
105             var dataEncryptor = new RSA();
106             var updateUser = new UpdateUserModel()
107             {
108                 Password = Helpers.PasswordHasher.Hash(body.Password),
109                 Role = body.Role,
110                 FirstName = dataEncryptor.Encrypt(body.FirstName),
111                 LastName = dataEncryptor.Encrypt(body.LastName),
112                 UserName = body.UserName,
113             };
114             var user = _userRepository.UpdateUser(updateUser, id);
115             return Ok(user);
116         }
117
118         [HttpDelete]
119         [Authorize(Policy = "Admin")]
120         [Route("{id}")]
121         public ActionResult<string> Delete(string id)
122         {
123             var user = _userRepository.DeleteUser(id);
```

```
124                 return Ok(user);
125             }
126         }
127 }
```

0.2.16 MedicalData.cs

```
1  using System;
2  using System.ComponentModel.DataAnnotations.Schema;
3
4  namespace medic_api.DAL.Models
5  {
6      [Table("medicaldata")]
7      public partial class MedicalData
8      {
9          [Column("id")]
10         public Guid MedicalDataId { get; set; }
11         [Column("pregnancies")]
12         public int Pregnancies { get; set; }
13         [Column("glucose")]
14         public int Glucose { get; set; }
15         [Column("bloodpressure")]
16         public int BloodPressure { get; set; }
17         [Column("skinthickness")]
18         public int SkinThickness { get; set; }
19         [Column("insulin")]
20         public int Insulin { get; set; }
21         [Column("diabetespedigreefunction")]
22         public double DiabetesPedigreeFunction { get; set; }
23         [Column("bmi")]
24         public double Bmi { get; set; }
25         [Column("age")]
26         public int Age { get; set; }
27         [Column("prediction")]
28         public bool? Prediction { get; set; }
29         [Column("result")]
30         public bool? Result { get; set; }
31         [Column("userid")]
32         public Guid UserId { get; set; }
33         public User User { get; set; }
34     }
35 }
```

0.2.17 CreateTicketDTO.cs

```
1  using System;
2  using cinema_app_api.Models;
3
4  namespace cinema_app_api.DTO {
5      public class CreateTicketDto {
6          public string Showing { get; set; }
```

```
7           public string User { get; set; }
8           public int FieldX { get; set; }
9           public int FieldY { get; set; }
10          public TicketStatus Status { get; set; }
11      }
12  }
```

0.2.18 User.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations.Schema;
4
5  namespace medic_api.DAL.Models
6  {
7      [Table("users")]
8      public partial class User
9      {
10         [Column("id")]
11         public Guid UserId { get; set; }
12         [Column("firstname")]
13         public string FirstName { get; set; }
14         [Column("lastname")]
15         public string LastName { get; set; }
16         [Column("username")]
17         public string UserName { get; set; }
18         [Column("password")]
19         public string Password { get; set; }
20         [Column("role")]
21         public string Role { get; set; }
22
23         [InverseProperty("User")]
24         public List<MedicalData> MedicalDataset { get; set; }
25     }
26 }
```

0.2.19 AddMedicalDataModel.cs

```
1  using System;
2
3  namespace medic_api.DAL.Repository.MedicalData
4  {
5      public class AddMedicalDataModel
6      {
7          public int Pregnancies { get; set; }
8          public int Glucose { get; set; }
9          public int BloodPressure { get; set; }
10         public int SkinThickness { get; set; }
11         public int Insulin { get; set; }
12         public double DiabetesPedigreeFunction { get; set; }
13         public double Bmi { get; set; }
```

```
14          public int Age { get; set; }
15          public Guid UserId { get; set; }
16      }
17  }
```

0.2.20 IMedicalDataRepository.cs

```
1  using System.Collections.Generic;
2
3  namespace medic_api.DAL.Repository.MedicalData
4  {
5      public interface IMedicalDataRepository
6      {
7          public string AddMedicalData(AddMedicalDataModel model);
8          public Models.MedicalData GetMedicalData(string medicalDataId);
9          public List<Models.MedicalData> GetMedicalDataList();
10         public List<Models.MedicalData> GetMedicalDataListByUser(string
               userId);
11         public string UpdateMedicalData(UpdateMedicalDataModel model,
               string medicalDataId);
12         public string DeleteMedicalData(string medicalDataId);
13         public string SetPrediction(string medicalDataId, bool
               prediction);
14         public string SetResult(string medicalDataId, bool result);
15     }
16 }
```

0.2.21 IMedicalDataRepository.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  namespace medic_api.DAL.Repository.MedicalData
6  {
7      public class MedicalDataRepository : IMedicalDataRepository
8      {
9          private readonly DataContext _db;
10
11         public MedicalDataRepository(DataContext db)
12         {
13             _db = db;
14         }
15
16         public string AddMedicalData(AddMedicalDataModel model)
17         {
18             var medical = new Models.MedicalData()
19             {
20                 Age     = model.Age,
21                 Bmi = model.Bmi,
22                 Glucose = model.Glucose,
```

```
23                  Insulin = model.Insulin,
24                  Pregnancies = model.Pregnancies,
25                  UserId = model.UserId,
26                  BloodPressure = model.BloodPressure,
27                  SkinThickness = model.SkinThickness,
28                  DiabetesPedigreeFunction = model.
                        DiabetesPedigreeFunction,
29              };
30              var newMedical = _db.MedicalDataset.Add(medical);
31              _db.SaveChanges();
32              return newMedical.Entity.MedicalDataId.ToString();
33          }
34
35          public Models.MedicalData GetMedicalData(string medicalDataId)
36          {
37              var entity = _db.MedicalDataset.FirstOrDefault(d => d.
                    MedicalDataId == new Guid(medicalDataId));
38              return entity;
39          }
40
41          public List<Models.MedicalData> GetMedicalDataList()
42          {
43              var entities = _db.MedicalDataset.ToList();
44              return entities;
45          }
46
47          public List<Models.MedicalData> GetMedicalDataListByUser(string
                userId)
48          {
49              var entities = _db.MedicalDataset.Where(d => d.UserId == new
                    Guid(userId));
50              return entities.ToList();
51          }
52
53          public string UpdateMedicalData(UpdateMedicalDataModel model,
                string medicalDataId)
54          {
55              var medical = _db.MedicalDataset.FirstOrDefault(d => d.
                    MedicalDataId == new Guid(medicalDataId));
56              if (medical == null) throw new Exception("Medical Data not
                    exists");
57              if (model.Age.HasValue) medical.Age = model.Age.Value;
58              if (model.Bmi.HasValue) medical.Bmi = model.Bmi.Value;
59              if (model.Glucose.HasValue) medical.Glucose = model.Glucose.
                    Value;
60              if (model.Insulin.HasValue) medical.Insulin = model.Insulin.
                    Value;
61              if (model.Pregnancies.HasValue) medical.Pregnancies = model.
                    Pregnancies.Value;
62              if (model.BloodPressure.HasValue) medical.BloodPressure =
                    model.BloodPressure.Value;
63              if (model.SkinThickness.HasValue) medical.SkinThickness =
                    model.SkinThickness.Value;
64              if (model.DiabetesPedigreeFunction.HasValue) medical.
                    DiabetesPedigreeFunction = model.DiabetesPedigreeFunction
```

```
                .Value;
65              if (!string.IsNullOrEmpty(model.UserId)) medical.UserId =
                    new Guid(model.UserId);
66              _db.SaveChanges();
67              return medical.UserId.ToString();
68          }
69
70          public string DeleteMedicalData(string medicalDataId)
71          {
72              var medical = _db.MedicalDataset.FirstOrDefault(d => d.
                    MedicalDataId == new Guid(medicalDataId));
73              if (medical == null) throw new Exception("Medical Data not
                    exists");
74              _db.MedicalDataset.Remove(medical);
75              _db.SaveChanges();
76              return medical.MedicalDataId.ToString();
77          }
78
79          public string SetPrediction(string medicalDataId, bool
                prediction)
80          {
81              var medical = _db.MedicalDataset.FirstOrDefault(d => d.
                    MedicalDataId == new Guid(medicalDataId));
82              if (medical == null) throw new Exception("Medical Data not
                    exists");
83              medical.Prediction = prediction;
84              _db.SaveChanges();
85              return medical.MedicalDataId.ToString();
86          }
87
88          public string SetResult(string medicalDataId, bool result)
89          {
90              var medical = _db.MedicalDataset.FirstOrDefault(d => d.
                    MedicalDataId == new Guid(medicalDataId));
91              if (medical == null) throw new Exception("Medical Data not
                    exists");
92              medical.Result = result;
93              _db.SaveChanges();
94              return medical.MedicalDataId.ToString();
95          }
96      }
97  }
```

0.2.22 UpdateMedicalDataModel.cs

```
1  using System;
2
3  namespace medic_api.DAL.Repository.MedicalData
4  {
5      public class UpdateMedicalDataModel
6      {
7          public int? Pregnancies { get; set; }
8          public int? Glucose { get; set; }
```

```
9           public int? BloodPressure { get; set; }
10          public int? SkinThickness { get; set; }
11          public int? Insulin { get; set; }
12          public double? DiabetesPedigreeFunction { get; set; }
13          public double? Bmi { get; set; }
14          public int? Age { get; set; }
15          public string? UserId { get; set; }
16      }
17  }
```

0.2.23 AddUserModel.cs

```
1  namespace medic_api.DAL.Repository
2  {
3      public class AddUserModel
4      {
5          public string FirstName { get; set; }
6          public string LastName { get; set; }
7          public string UserName { get; set; }
8          public string Role { get; set; }
9          public string Password { get; set; }
10     }
11 }
```

0.2.24 IUserRepository.cs

```
1  using System.Collections.Generic;
2  using medic_api.DAL.Models;
3
4  namespace medic_api.DAL.Repository
5  {
6      public interface IUserRepository
7      {
8          public User GetUser(string id);
9          public User GetUserByUserName(string userName);
10         public List<User> GetUsers();
11         public string AddUser(AddUserModel userModelData);
12         public string UpdateUser(UpdateUserModel userModelData, string
               userId);
13         public string DeleteUser(string userId);
14     }
15 }
```

0.2.25 UpdateUserModel.cs

```
1  namespace medic_api.DAL.Repository
2  {
3      public class UpdateUserModel
4      {
```

```
5           public string? FirstName { get; set; }
6           public string? LastName { get; set; }
7           public string? UserName { get; set; }
8           public string? Role { get; set; }
9           public string? Password { get; set; }
10      }
11  }
```

0.2.26 UsersRepository.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using medic_api.DAL.Models;
5
6  namespace medic_api.DAL.Repository
7  {
8      public class UsersRepository : IUserRepository
9      {
10         private readonly DataContext _db;
11
12         public UsersRepository(DataContext db)
13         {
14             _db = db;
15         }
16
17         public User GetUser(string id)
18         {
19             var entity = _db.Users.FirstOrDefault(user => user.UserId ==
                   new Guid(id));
20             return entity;
21         }
22
23         public User GetUserByUserName(string userName)
24         {
25             var entity = _db.Users.FirstOrDefault(user => user.UserName
                   == userName);
26             return entity;
27         }
28
29         public List<User> GetUsers()
30         {
31             var entities = _db.Users.ToList();
32             return entities;
33         }
34
35         public string AddUser(AddUserModel userModelData)
36         {
37             var existing = _db.Users.FirstOrDefault(u => u.UserName ==
                   userModelData.UserName);
38             if (existing != null) throw new Exception("User already
                   exists");
```

```
39              if (!_roleVerify(userModelData.Role)) throw new Exception("
                    Bad role");
40              User user = new User
41              {
42                  Password = userModelData.Password,
43                  Role = userModelData.Role,
44                  FirstName = userModelData.FirstName,
45                  LastName = userModelData.LastName,
46                  UserName = userModelData.UserName,
47              };
48              var newUser = _db.Users.Add(user);
49              _db.SaveChanges();
50
51              return newUser.Entity.UserId.ToString();
52          }
53
54          public string UpdateUser(UpdateUserModel userModelData, string
                userId)
55          {
56              var user = _db.Users.FirstOrDefault(u => u.UserId == new
                    Guid(userId));
57              if (user == null) throw new Exception("User does not exists"
                    );
58              if (!_roleVerify(userModelData.Role)) throw new Exception("
                    Bad role");
59              if (!string.IsNullOrEmpty(userModelData.Password)) user.
                    Password = userModelData.Password;
60              if (!string.IsNullOrEmpty(userModelData.Role)) user.Role =
                    userModelData.Role;
61              if (!string.IsNullOrEmpty(userModelData.FirstName)) user.
                    FirstName = userModelData.FirstName;
62              if (!string.IsNullOrEmpty(userModelData.LastName)) user.
                    LastName = userModelData.LastName;
63              if (!string.IsNullOrEmpty(userModelData.UserName)) user.
                    UserName = userModelData.UserName;
64              _db.SaveChanges();
65              return user.UserId.ToString();
66          }
67
68          public string DeleteUser(string userId)
69          {
70              var user = _db.Users.FirstOrDefault(u => u.UserId == new
                    Guid(userId));
71              if (user == null) throw new Exception("User does not exists"
                    );
72              _db.Users.Remove(user);
73              _db.SaveChanges();
74              return user.UserId.ToString();
75          }
76
77          private bool _roleVerify(string role)
78          {
79              return role == "Admin" || role == "Doctor" || role == "
                    Patient";
80          }
```

```
81      }
82  }
```

0.2.27 DataContext.cs

```
1  using System;
2  using medic_api.DAL.Models;
3  using medic_api.Helpers;
4  using Microsoft.EntityFrameworkCore;
5
6  namespace medic_api.DAL
7  {
8      public class DataContext: DbContext
9      {
10         public DataContext(DbContextOptions<DataContext> options): base(
               options) { }
11
12         public DbSet<User> Users { get; set; }
13         public DbSet<MedicalData> MedicalDataset { get; set; }
14
15         protected override void OnModelCreating(ModelBuilder
               modelBuilder)
16         {
17             var rsa = new RSA();
18             modelBuilder.Entity<User>().HasData(new User
19             {
20                 Role = "Admin",
21                 Password = PasswordHasher.Hash("Admin"),
22                 FirstName = rsa.Encrypt("Admin"),
23                 LastName = rsa.Encrypt("Admin"),
24                 UserName = "Admin",
25                 UserId = Guid.NewGuid(),
26             });
27         }
28     }
29 }
```

0.2.28 PasswordHasher.cs

```
1  using System;
2  using System.Security.Cryptography;
3
4  namespace medic_api.Helpers
5  {
6      public static class PasswordHasher
7      {
8          private const int SaltSize = 16;
9          private const int HashSize = 20;
10         private const int Iterations = 10;
11         public static string Hash(string password)
12         {
```

```
13              if (string.IsNullOrEmpty(password)) return "";
14              byte[] salt;
15              new RNGCryptoServiceProvider().GetBytes(salt = new byte[
                    SaltSize]);
16
17              var pbkdf2 = new Rfc2898DeriveBytes(new String(password),
                    salt, Iterations);
18              var hash = pbkdf2.GetBytes(HashSize);
19
20              var hashBytes = new byte[SaltSize + HashSize];
21              Array.Copy(salt, 0, hashBytes, 0, SaltSize);
22              Array.Copy(hash, 0, hashBytes, SaltSize, HashSize);
23
24              var base64Hash = Convert.ToBase64String(hashBytes);
25
26              return string.Format(base64Hash);
27          }
28
29          public static bool Verify(string password, string hash)
30          {
31              if (string.IsNullOrEmpty(password) || string.IsNullOrEmpty(
                    hash)) return false;
32              var stringHash = Convert.FromBase64String(new String(hash));
33              var salt = new byte[SaltSize];
34              Array.Copy(stringHash, 0, salt, 0, SaltSize);
35
36              var pbkdf2 = new Rfc2898DeriveBytes(password, salt,
                    Iterations);
37              var hashed = pbkdf2.GetBytes(HashSize);
38
39              for (var i = 0; i < HashSize; i++)
40              {
41                  if (stringHash[i + SaltSize] != hashed[i])
42                  {
43                      return false;
44                  }
45              }
46              return true;
47          }
48      }
49  }
```

0.2.29 RSA.cs

```
1  using System;
2  using System.IO;
3  using System.Security.Cryptography;
4  using System.Text;
5  using System.Xml;
6  using System.Xml.Serialization;
7
8  namespace medic_api.Helpers
9  {
```

```
10      public class RSA
11      {
12          private static RSACryptoServiceProvider _cryptoServiceProvider =
                new RSACryptoServiceProvider(2048);
13          private readonly RSAParameters _privateKey;
14          private readonly RSAParameters _publicKey;
15
16          public RSA()
17          {
18              _privateKey = _fileToKey("privateKey.xml");
19              _publicKey = _fileToKey("publicKey.xml");
20          }
21
22          private RSAParameters _fileToKey(string filename)
23          {
24              try
25              {
26                  XmlDocument xmlDocument = new XmlDocument();
27                  xmlDocument.Load(filename);
28                  XmlSerializer xmlSerializer = new XmlSerializer(typeof(
                        RSAParameters));
29                  XmlReader xmlReader = new XmlNodeReader(xmlDocument.
                        DocumentElement);
30                  var key = (RSAParameters) xmlSerializer.Deserialize(
                        xmlReader);
31                  return key;
32              }
33              catch
34              {
35                  var key =_cryptoServiceProvider.ExportParameters(true);
36                  _keyToFile(_keyToString(key), filename);
37                  return key;
38              }
39          }
40
41          private void _keyToFile(string key, string filename)
42          {
43              XmlDocument xmlDocument = new XmlDocument();
44              xmlDocument.LoadXml(key);
45              xmlDocument.Save(filename);
46          }
47
48          private string _keyToString(RSAParameters key)
49          {
50              var stringWriter = new StringWriter();
51              var xmlSerializer = new XmlSerializer(typeof(RSAParameters))
                    ;
52              xmlSerializer.Serialize(stringWriter, key);
53              return stringWriter.ToString();
54          }
55
56          public string Encrypt(string plainText)
57          {
58              _cryptoServiceProvider = new RSACryptoServiceProvider();
59              _cryptoServiceProvider.ImportParameters(_publicKey);
```

```
60              var data = Encoding.UTF8.GetBytes(plainText);
61              var cypher = _cryptoServiceProvider.Encrypt(data, true);
62              return Convert.ToBase64String(cypher);
63          }
64
65          public string Decrypt(string cypher)
66          {
67              var dataBytes = Convert.FromBase64String(cypher);
68              _cryptoServiceProvider.ImportParameters(_privateKey);
69              var plainText = _cryptoServiceProvider.Decrypt(dataBytes,
                    true);
70              return Encoding.UTF8.GetString(plainText);
71          }
72      }
73  }
```

0.2.30 UserModelToResponse.cs

```
1  using medic_api.Controllers.Users;
2  using medic_api.DAL.Models;
3
4  namespace medic_api.Helpers
5  {
6      public static class UserModelToResponse
7      {
8          public static UserResponse UserToResponse(User user)
9          {
10             var dataEncryptor = new RSA();
11             return new UserResponse()
12             {
13                 Role = user.Role,
14                 FirstName = dataEncryptor.Decrypt(user.FirstName),
15                 LastName = dataEncryptor.Decrypt(user.LastName),
16                 UserId = user.UserId.ToString(),
17                 UserName = user.UserName,
18             };
19         }
20     }
21 }
```

0.2.31 Program.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Hosting;
6  using Microsoft.Extensions.Configuration;
7  using Microsoft.Extensions.Hosting;
8  using Microsoft.Extensions.Logging;
9
```

```
10 namespace medic_api
11 {
12     public class Program
13     {
14         public static void Main(string[] args)
15         {
16             CreateHostBuilder(args).Build().Run();
17         }
18 
19         public static IHostBuilder CreateHostBuilder(string[] args) =>
20             Host.CreateDefaultBuilder(args)
21                 .ConfigureWebHostDefaults(webBuilder => { webBuilder.
                       UseStartup<Startup>(); });
22     }
23 }
```