



---

# VIZSGAREMEK

---

Kutyakozmetika



2022. ÁPRILIS 19.

Készítette: Barna Marcell Tamás, Dömösi András, Saceanu Krisztián

## Tartalomjegyzék:

Projektről: .....	3
Projekt neve: .....	3
Projekt célja: .....	3
Projekt témája: .....	3
Projekt komponensei: .....	3
Csapatmunkáról: .....	3
Asztali alkalmazás (Desktop Version) .....	4
A desktop alkalmazás témája: .....	4
Az asztali alkalmazás megvalósítása: .....	4
1.Lépés .....	5
2.Lépés .....	6
3.Lépés .....	7
4.Lépés .....	8
5.Lépés .....	9
6.Lépés .....	10
7.Lépés: .....	11
8.Lépés .....	11
9.Lépés .....	12
10.Lépés .....	13
11.Lépés .....	14
12.Lépés .....	15
Mobil alkalmazás .....	16
Mobil alkalmazás célja: .....	16
Mobil alkalmazás megvalósítása: .....	16
1.Lépés : .....	16
2.Lépés .....	17
3.Lépés .....	18
4.Lépés .....	18
5.Lépés .....	20
6.Lépés .....	20
7.Lépés .....	21
Backend a mobil alkalmazáshoz .....	23
Webes alkalmazás: .....	25
Webes alkalmazás témája: .....	25
Program megvalósítása .....	25

1.Lépés .....	25
2.lépés.....	26
3.Lépés .....	27
4.Lépés .....	27
5.Lépés .....	28
Web API a webes alkalmazáshoz .....	29
Web API témája: .....	29
Web Api megvalósítása:.....	29
1.Lépés .....	29
2.Lépés .....	29
3.Lépés .....	30
4.Lépés .....	30
5.Lépés .....	30
6.Lépés .....	31
7.Lépés .....	31
Adatbázis .....	33
Tervezés.....	33
Egyedek:.....	33
Tulajdonságok: .....	33
Egyedek és tulajdonságaik(adattípusok): .....	33
Az adatbázis E-K diagramja.....	35
Az adatbázis Bachmann-ábrája .....	35

# **Projektről:**

## **Projekt neve:**

Doggo Session

## **Projekt célja:**

A szoftver rendszer célja, hogy a felhasználók egyszerűbben tudjanak szolgáltatást rendelni. Célja a projektnek egy újkeletű vállalkozás látványosabbá tétele. Mivel a virtuális világ küszöbén állunk, a projekt megvalósításával egy új szintre emelhető a kutyakozmetika fogalma.

## **Projekt témája:**

Kutyakozmetikához tartozó szolgáltatások egyszerűbbé tétele.

## **Projekt komponensei:**

- Desktop alkalmazás
- Mobil applikáció
- Webes alkalmazás
- Backend
- Adatbázis

## **Csapatmunkáról:**

Munkánk során fontos szerepet játszott a csapatmunka. Minden feladatot egyenlő részre bontottunk. A munkánkat GitHubra töltöttük fel különböző repositorykba. Heti konzultációkat tartottunk, ahol mindig megbeszéltük a program jelenlegi állását, illetve felmerülő problémákat elhárítottuk. A teljes dokumentációt együtt írtuk meg, minden részt közösen foglalmaztunk meg.

## **Asztali alkalmazás (Desktop Version)**

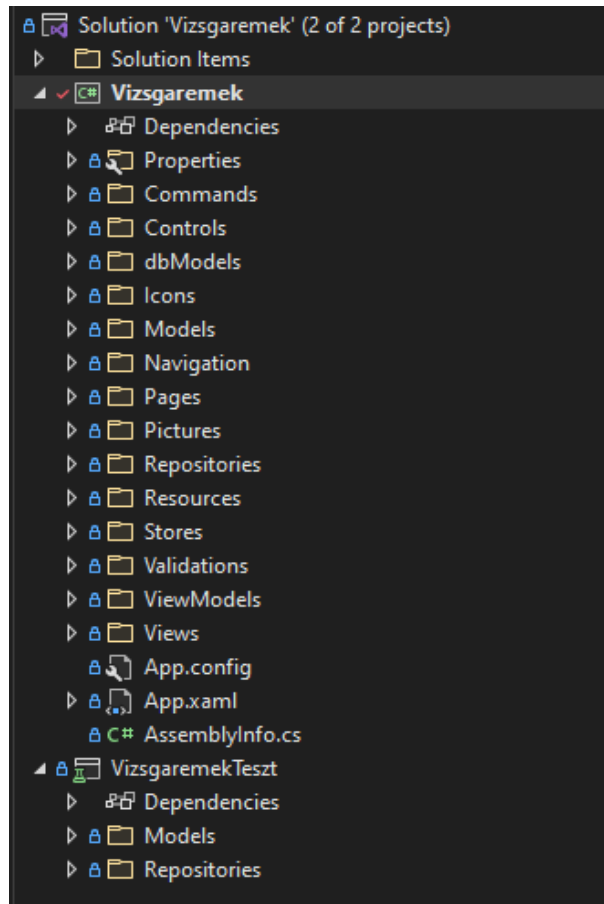
### **A desktop alkalmazás témája:**

Az asztali alkalmazásunk témája, egy kutyakozmetikában alkalmazott adatkezelő program, ahol a kozmetikus nyomon követheti, a vendégek/felhasználók által rögzített adatokat. Ez egy új gondolat, amivel szeretnénk a kutyakozmetikákat kicsit modernebbé tenni. Az esetek 90%-ban még a megszokott módon történik az adatkezelés. Kozmetikánként van egy kis napló ahová felírják az időpontokat, a kutyák illetve a gazdi adatait. Napjainkban az információs technológia fejlődését nézve ez már eléggé elavultnak számít.

### **Az asztali alkalmazás megvalósítása:**

Az asztali alkalmazás C# nyelven íródott és a programot a Microsoft Visual Studio fejlesztői környezet használatával valósítottuk meg. Két verzióját használtuk a Visual Studio-nak a 2019-es és a 2022-eset. A programban mapparendszert alkalmaztunk:

Mivel rengeteg osztályt tartalmaz egy-egy mappa, ezért nélkülük átláthatatlan lett volna, hogy melyik osztály milyen szerepet tölt be a munkánkban.



*1. ábra Mapparendszer*

## 1.Lépés

Elsőként létrehoztuk a „Views” mappát és azon belül a MainWindow-t, ahol Grid-ek használatával felosztottuk részekre az ablakot. Itt egy Label-ben szerepel a programunk neve és annak megjelenésének formázása. Felvettük az első gombunkat, ami a kilépésért felelős. A „Kilépés” gombra kattintva a program bezárul.

```
1 reference
private void Button_Click(object sender, RoutedEventArgs e)
{
    Application.Current.Shutdown(99);
}
```

*2. ábra A „Kilépés” gomb kódolása*

Ezután a „Pages” mappa jött létre, ahol egy WPF ablakban egy üdvözlő szöveg jelenik meg. Ezt a „Navigate” osztályban a „Navigation” metódus használatával átadjuk a MainWindow-nak és a program indulásakor a „WelcomePage” is betöltődik.



3. ábra WelcomePage

## 2.Lépés

A program fejlesztése során bekerült a „ProgramInfo” gombunk, ahol a program aktuális verzióját tekinthetjük meg. Későbbiekben itt feltüntetjük majd a programot fejlesztő céget és fejlesztők GitHub felhasználónevét is. A megvalósításhoz egy WPF ablakot és két új mappát hoztunk létre, a „Models” és a „ViewModels” mappát, ahol egy-egy új osztály segíti a program verziójának megjelenítését.

```
3 references
class ProgramInfoViewModel
{
    private ProgramInfo programInfo;
    1 reference
    public string Version
    {
        get
        {
            return programInfo.Version.ToString();
        }
        set
        {
            Version = value;
        }
    }

    1 reference
    public ProgramInfoViewModel()
    {
        programInfo = new ProgramInfo();
    }
}
```

4. ábra A „ViewModels” mappában szereplő „ProgramInfoViewModel” kód

A kód megírása után ellenőriztük a megjelenést, a program gond nélkül elindult és az adatok helyes megjelentek. Folytattuk a fejlesztést és felvettük a szükséges string típusú változókat, amiket példányosítottunk, majd állítható és módosítható tulajdonságokkal ágyaztuk be a programba. Ezeket a változókat a „ProgramInfo.xaml” oldalon elérési útvonalként adtuk meg, ami ezután megjeleníti a kívánt adatokat.

## Program adatok

Program verzió: **1.0.0.0**

Program címe: **Vizsgaremek**

Program leírása: **Desktop alkalmazás kutyakozmetikusoknak.**

Fejlesztő cég: **Master Minds**

### *5. ábra A megírt kód eredménye*

### 3.Lépés

A vizsgaremekünk egyik fő komponense az adatbázis helyének kiválasztása. A programunk két fajta adattal dolgozik, amit teszt adatként vittünk fel és amit a helyi adatbázisból lekér a program. A helyi adatbázist a XAMPP felhasználásával oldottuk meg. A XAMPP egy szabad és nyílt forrású platformfüggetlen webservert-szoftvercsomag, amelynek legfőbb alkotóelemei az Apache webservert, a MariaDB (korábban a MySQL) adatbázis-kezelő, valamint a PHP és a Perl programozási nyelvek értelmezői (végrehajtó rendszerei). Két választási lehetőséget tüntettünk fel a „test”(alapértelmezett) és a „localhost”-ot. A választott adatbázisoktól függően töltődnek be az adatok a megfelelő táblába. Itt még a táblázatokkal nem foglalkoztunk, arra majd a fejlesztés későbbi részében kerül sor.



```

public class DatabaseSourceViewModel
{
    private ObservableCollection<string> displayedDatabaseSources;
    private string selectedDatabaseSource;
    private string displayedDatabaseSource;
    DatabaseSources repoDatabaseSources;
    DatabaseSource databaseSource;

    0 references
    public ObservableCollection<string> DisplayedDatabaseSources
    {
        get => displayedDatabaseSources;
    }

    2 references
    public string SelectedDatabaseSource
    {
        get => selectedDatabaseSource;
        set
        {
            selectedDatabaseSource = value;
        }
    }

    2 references
    public DatabaseSourceViewModel()
    {
        repoDatabaseSources = new DatabaseSources();
        displayedDatabaseSources = new ObservableCollection<string>(repoDatabaseSources.GetAllDatabaseSources());
    }

    2 references
    public DbSource DatabaseSource
    {
        get
        {
            if (selectedDatabaseSource == "localhost")
                return DbSource.LOCALHOST;
            return DbSource.NONE;
        }
    }
}

```

6. ábra A „DatabaseSourceViewModel”-ben megírt kód

#### 4.Lépés

Az alkalmazás két nyelven használható. Az alapértelmezett nyelv a magyar azonban ezt a kozmetikus szabadon átállíthatja angolra is. Erre a „Nyelv választás” gombra kattintás után lesz lehetőség. Újabb mappát hoztunk létre „Resources” néven. A mappához két „ResourceDictionary” forrás könyvtárat adtunk hozzá. Itt tároljuk el az angol és a magyar szöveget gombonként.

```

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib">
    <system:String x:Key="strMenuExit">Exit</system:String>
    <system:String x:Key="strMenuDataSource">Database Source selection</system:String>
    <system:String x:Key="strMenuProgInfo">Program information</system:String>
</ResourceDictionary>

```

7. ábra Az angol nyelvű „Dictionary”

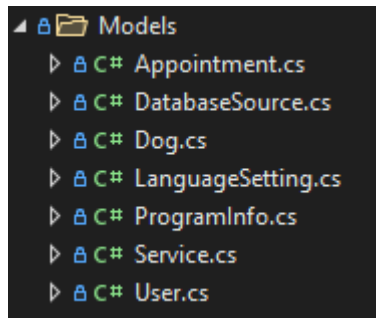
A nyelvválasztás egyik kulcsfontosságú kód részletét a „MainWindow.xaml.cs”-n belül írjuk meg. Ennek a „SetLanguageDictionary” metódus nevet adtuk. Ez a metódus segít abban, hogy a nyelv kiválasztása után a gombok szövege módosuljon.

```
private void SetLanguageDictionary()
{
    switch (Thread.CurrentThread.CurrentCulture.TwoLetterISOLanguageName)
    {
        case "en":
            dict.Source = new Uri("../Resources\\StringResources.xaml", UriKind.Relative);
            break;
        case "fr":
            dict.Source = new Uri("../Resources\\FR\\StringResources.xaml", UriKind.Relative);
            break;
        case "hu":
            dict.Source = new Uri("../Resources\\HU\\StringResources.xaml", UriKind.Relative);
            break;
        default:
            dict.Source = new Uri("../Resources\\StringResources.xaml", UriKind.Relative);
            break;
    }
    int langDictId = -1;
    bool found = false;
    for (int i = 0; i < this.Resources.MergedDictionaries.Count && !found; i++)
    {
        var md = this.Resources.MergedDictionaries[i];
        if (md.Contains(Thread.CurrentThread.CurrentCulture.TwoLetterISOLanguageName))
        {
            langDictId = i;
            found = true;
        }
    }
    if (!found)
    {
        this.Resources.MergedDictionaries.Add(dict);
    }
    else
    {
        this.Resources.MergedDictionaries[langDictId] = dict;
    }
    this.Resources.MergedDictionaries.Add(dict);
}
```

8. ábra A „SetLanguageDictionary” kódja

## 5.Lépés

Munkánk során a csapat összes tagja létrehozott egy-egy osztályt a „Models” rétegben, amelyben a program témájához kapcsolódóan felvettük a szükséges adattagokat és a hozzájuk tartozó konstruktorokat, illetve ezeket állíthatóvá és módosíthatóvá tettük.



9. ábra Models mappa osztályai

## 6.Lépés

Minden főbb osztálynak létrehoztunk egy-egy „UserControl”-t, ami a megjelenítésért felel. Mielőtt megírtuk a hozzá szükséges kódot, figyelembe kellett vennünk azt, hogy az adatbázisban, miként épülnek fel a táblák. Létrehoztunk egy táblázatot, ahova az adatok úgy töltődnek be, hogy minden oszlophoz elérési útvonalat adtunk „Binding” használatával és így minden cellába, a megfelelő adat jelenik meg.

```
<StackPanel HorizontalAlignment="Center" Margin="0 10 0 0">
  <DataGrid x:Name="dgDogs"
    AutoGenerateColumns="False"
    SelectionChanged="dgDog_SelectionChanged"
    ItemsSource="{Binding DisplayedDogs}"
    SelectedValue="{Binding SelectedDog}"
    SelectedItem="{Binding SelectedDog}"
    SelectedIndex="{Binding SelectedDogIndex, Mode=OneWay}"
  >
    <DataGrid.Columns>
      <DataGridTextColumn x:Name="dogID" MinWidth="50" IsReadOnly="True"
        Header="ID" Binding="{Binding DogID}"></DataGridTextColumn>
      <DataGridTextColumn x:Name="dogAggressive" MinWidth="100" IsReadOnly="True"
        Header="Agresszív" Binding="{Binding Aggressive}"></DataGridTextColumn>
      <DataGridTextColumn x:Name="dogAge" MinWidth="100" IsReadOnly="True"
        Header="Kor" Binding="{Binding Age}"></DataGridTextColumn>
      <DataGridTextColumn x:Name="dogBreed" MinWidth="100" IsReadOnly="True"
        Header="Fajta" Binding="{Binding Breed}"></DataGridTextColumn>
      <DataGridTextColumn x:Name="dogOwner" MinWidth="100" IsReadOnly="False"
        Header="Gazda" Binding="{Binding Owner}"></DataGridTextColumn>
    </DataGrid.Columns>
  </DataGrid>
</StackPanel>
```

10. ábra A „DogPage”-ben az adatok megjelenítéséhez megírt kód

## 7.Lépés:

A „Page”-kben szereplő táblázatok adatai, a „Controls” mappában használt „UserControl”-ok segítségével jelennek meg a táblázat alatt, ahol ezek az adatok módosíthatóak illetve törölhetőek. Itt használtuk a validálást, ami azért felelős, hogy a felhasználó egy adott szabály szerint módosíthat az adatokon („A mező nem lehet üres!”).

**Kutya adatkezelés**

ID	Agresszív	Kor	Fajta	Gazda
1	igen	5	Németjuhász	Juhász Béla
2	nem	5	Németjuhász	Juhász János
3	igen	5	Németjuhász	Juhász Péter

A kiválasztott kutya adatai:

Azonosító:

Aggresszív:

Kor:

Fajta:

Gazda:

A mező nem lehet üres!

11. ábra A táblázat alatt a kiválasztott sor adatainak betöltése „control”-ba és a validáció által jelzett figyelmeztetés.

## 8.Lépés

A „ViewModel” a nézet absztrakciója, ami publikus tulajdonságokat és metódusokat tartalmaz. Ennek megfelelően írtuk meg a szükséges metódusokat és publikus tulajdonságokat a „ViewModels”-ben szereplő mappában. Létrehoztunk egy „ViewModelBaseClass” osztályt, ami a program futása során végzett módosítások végrehajtásáért felelős.

```

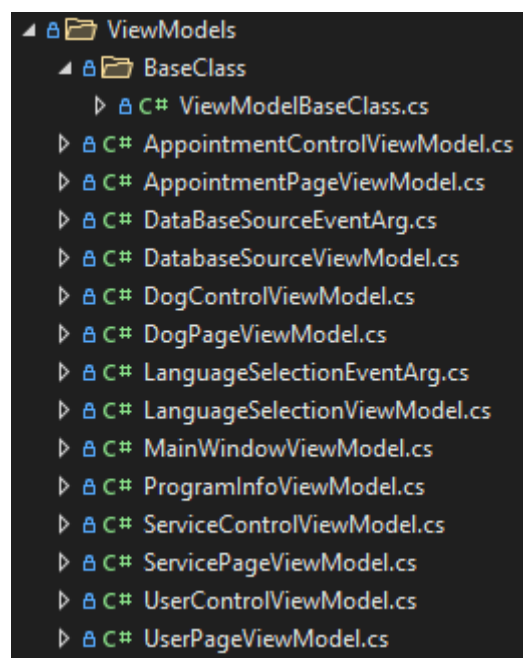
9 references
public class ViewModelBaseClass : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    22 references
    protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }

    0 references
    public virtual void Dispose() { }
}

```

12. ábra A „ViewModelBaseClass” kódja



13. ábra A „ViewModels” mappában szereplő osztályok

## 9.Lépés

A „Repositories” mappában szereplő osztályok felelnek a teszt adatok tárolásáért, illetve itt találhatóak a „localhost”-os lekéréshez használt „API” osztályok. Itt szerepel egy „Interface” mappa, amelyben egy interfész szerepel a következő metódusokkal:

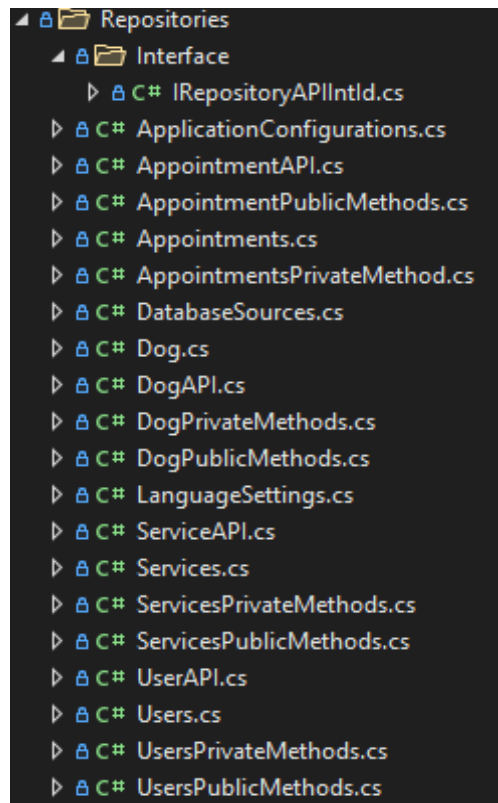
```

16 references
interface IRepositoryAPIIntId<T>
{
    16 references
    public List<T> GetAll();
    8 references
    public void Update(int id, T entity);
    8 references
    public void Delete(int id);
}

```

14. ábra Interfész

A „Repositories” mappa minden osztálya implementálja az imént említett interfészt.



15. ábra A „Repositories” osztályai

## 10.Lépés

A „Commands” mappában létrehoztunk egy „CommandBase” absztrakt osztályt, ami implementálja az „Icommand” beépített interfészt. Ez az osztály felelős a műveletek végrehajtásáért.

```

8 references
public abstract class CommandBase : ICommand
{
    public event EventHandler CanExecuteChanged;

    8 references
    public abstract bool CanExecute(object parameter);

    8 references
    public abstract void Execute(object parameter);

    0 references
    protected void OnCanExecuteChanged()
    {
        if (CanExecuteChanged != null)
            CanExecuteChanged.Invoke(this, new EventArgs());
    }
}

```

16. ábra „CommandBase” absztrakt osztály

Emellett a mappában megtalálható minden műveletre (Update, Delete) egy külön osztály. Az imént említett összes osztály örökli „CommandBase” nevezetű absztrakt class-t. Ahogyan a lenti ábrán látható, a törlés ID szerint történik.

```
2 references
public class DeleteDogCommand : CommandBase
{
    private readonly DogControlViewModel dogControlViewModel;

    private readonly DogStore dogStore;

    1 reference
    public DeleteDogCommand(DogControlViewModel dogControlViewModel, DogStore dogStore)
    {
        this.dogControlViewModel = dogControlViewModel;
        this.dogStore = dogStore;
    }

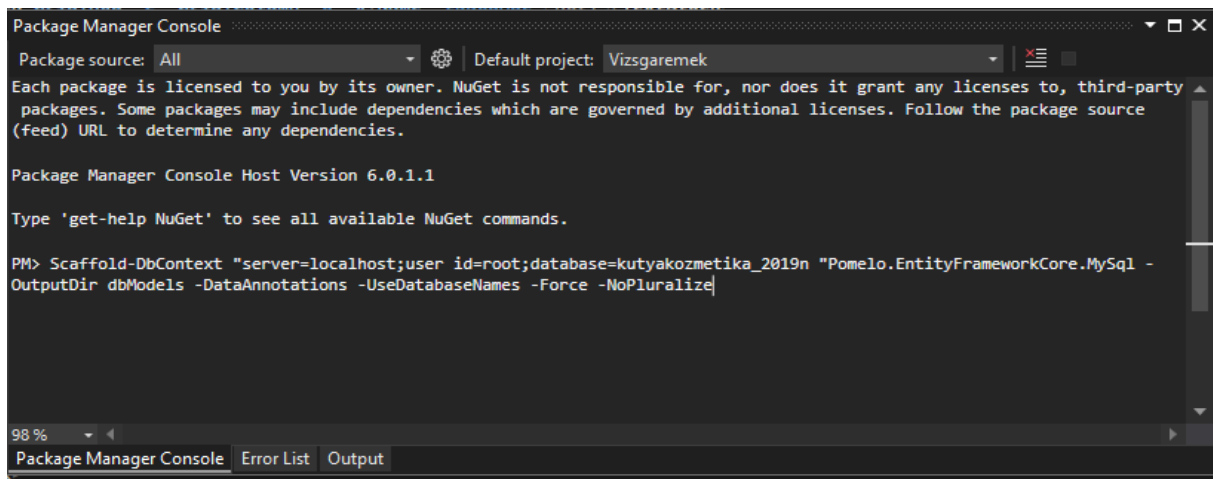
    1 reference
    public override bool CanExecute(object parameter)
    {
        return true;
    }

    1 reference
    public override void Execute(object parameter)
    {
        dogStore.DeleteDog(dogControlViewModel.EditedDog.DogID);
    }
}
```

17. ábra „DeleteDogCommand” osztály kódja

## 11.Lépés

Ebben a lépésben a backend részét írtuk meg a programnak. A backend a programoknak, weboldalaknak a hátsó, a felhasználó elől rejtett, a tényleges számításokat végző része. Feladata a frontend felől érkező adatok feldolgozása, és az eredményeknek a frontend felé történő visszajuttatása. Ebben az esetben a phpMyAdmin felületen szereplő adatbázisunkat kötöttük össze az asztali alkalmazásunkkal, amit úgy hajtottunk végre, hogy a Package Manager Console-on kiadtuk a scaffold parancsot és ez legenerálta a context osztályt. A parancs kiadása előtt feltelepítettük a Pomelo.EntityFrameworkCore.MySql csomagot, ami szükséges ahhoz, hogy kommunikáljon egymással a két felület.



18. ábra „Scaffold” parancs

## 12.Lépés

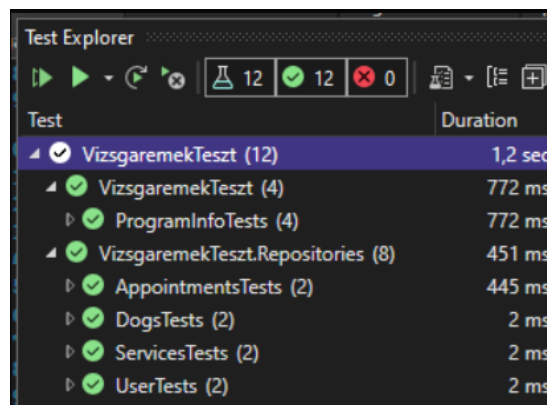
Az utolsó lépésben a teszteket hoztuk létre. A teszteket osztályonként a „GetAll” és az „Update” metódusokra írtuk meg. A „GetAllTest”-ben megvizsgáljuk, hogy helyes számban töltődnek-e be az adatok. A „Users” osztály esetében 3 teszt felhasználót vittünk fel. A teszt csak akkor fut le sikeresen, ha az elvárt értéket háromra állítjuk, más esetben sikertelen.

```
[TestMethod()]
0 references
public void GetAllTest()
{
    ApplicationStore applicationStore = new ApplicationStore();
    applicationStore.DbSource = DbSource.NONE;
    Users users = new Users(applicationStore);

    Assert.IsNotNull(users.AllUsers, "Repositories\\Users.css: A User lista nincs példányosítva!");
    int expected = 3;
    int actaul = users.GetAll().Count;

    Assert.AreEqual(expected, actaul, "Repositories\\User.css: A teszt adatok nem készülnek el megfelelő számban!");
}
```

19. ábra „GetAllTest” teszt kódja



20. ábra Sikeres tesztek



# Mobil alkalmazás

## Mobil alkalmazás célja:

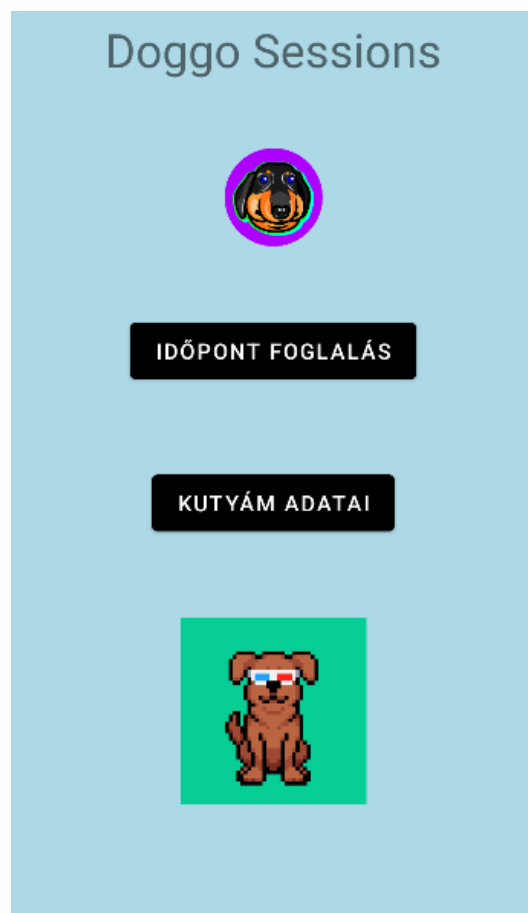
A telefonos alkalmazás célja, hogy a kutyakozmetikushoz a vendégek könnyebben és gyorsabban foglaljanak időpontot. Az applikációban tudnak regisztrálni majd bejelentkezés után tudják megadni a kis kedvenc adatait és időpontot foglalni.

## Mobil alkalmazás megvalósítása:

A telefonos alkalmazást Java nyelven írtuk meg az Android Studio fejlesztői környezet használatával.

### 1.Lépés

A teljesen üres projekt létrehozása után a főoldalon kezdtünk el dolgozni, a UserMenu-n. Először a layout-ot csináltuk meg melyben látható az applikáció neve, logója és két gomb az időpontfoglalás és a kutya adatainak regisztrálásához emellett egy GIF az oldal alján.



21. ábra UserMenu kinézete

Ezután a UserMenu.java fájlban írtuk meg a gombokra a navigálást, Intent használatával.

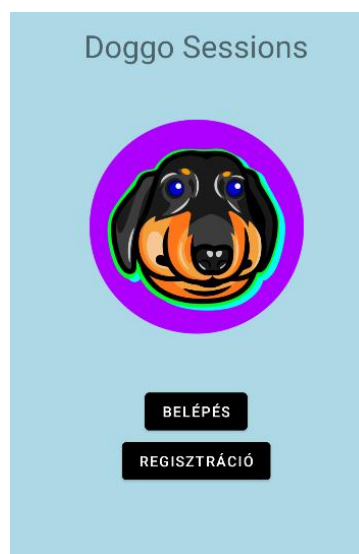
```
AppointmentButton.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getApplicationContext(), Appointment.class);
        intent.putExtra("username", username);
        startActivity(intent);
        finish();
    }
});

dogregisterButton.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v2) {
        Intent intent = new Intent(getApplicationContext(), DogRegistration.class);
        intent.putExtra("username", username);
        startActivity(intent);
        finish();
    }
});
```

22. ábra UserMenu.Java

## 2.Lépés

Második lépésként a RegLoginPage-et csináltuk. Ezen az oldalon található egy "regisztráció" és egy "bejelentkezés" gomb melyek újabb oldalakra navigálják a felhasználót.



23. ábra Reglog kinézete

### 3.Lépés

Következőnek a Regisztráción kezdtünk el dolgozni. Először létre hoztunk egy User-osztályt ahol felvettük a regisztrációhoz szükséges adattagokat majd elkészítettük a konstruktort, getter és settert.

Ezután elkészítettük a RegisterUser osztályt. Ebben egy PutDataToDB metódust írtunk melyben két tömböt hoztunk létre, az egyik az adatbázis felhasználó tábla oszlopai neveit kapta a másik pedig a User osztály-ból a getter segítségével kapja meg. A PutData beépített osztály segítségével pedig megadtuk az útvonalat a php-fájlhoz amely segítségével POST-olni tudjuk az adatokat az adatbázisba.

```
String[] field = new String[6];
field[0] = "nev";
field[1] = "cim";
field[2] = "email";
field[3] = "telefonszam";
field[4] = "felhasznalonev";
field[5] = "jelszo";
//Creating array for data
String[] data = new String[6];
data[0] = user.getFullname();
data[1] = user.getAddress();
data[2] = user.getEmail();
data[3] = user.getPhoneNumber();
data[4] = user.getUsername();
data[5] = user.getPassword();
```

24. ábra RegisterUser.Java

```
PutData putData =
new PutData( url: "http://192.168.100.56/kutyakozmetikaphp/signup.php"
, method: "POST", field, data);
```

25. ábra PutData beépített osztály

### 4.Lépés

A harmadik lépés után a Registration osztályt írtuk meg. Itt példányosítottuk a RegisterUser osztályt és a regisztrációs gomb OnClick eseménynél a User osztályt is. Majd a user objektumhoz kiolvassuk a felhasználó által megadott adatokat és a RegisterUser osztályban létrehozott putDataToDB metódussal feltöltjük az adatokat az adatbázisba.

```

registerUser = new RegisterUser();
registerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        User user = new User(String.valueOf(textInputEditTextFullname.getText())
            ,String.valueOf(textInputLayoutAddress.getText())
            ,String.valueOf(textInputEditTextEmail.getText())
            ,String.valueOf(textInputLayoutPhoneNumber.getText())
            ,String.valueOf(textInputEditTextUsername.getText())
            ,String.valueOf(textInputEditTextPassword.getText()));

        registerUser.putDataToDB(user);
        Intent intent = new Intent(getApplicationContext(), Login.class);
        startActivity(intent);
    }
});

```

26. ábra Registration.java

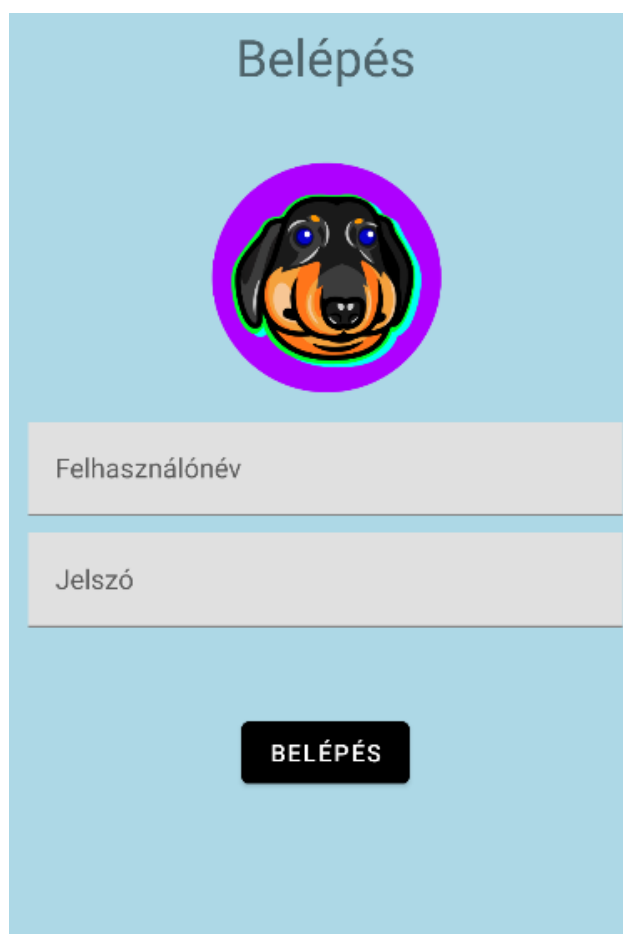
The image shows a registration form titled "Regisztráció" (Registration) on a light blue background. The form consists of six stacked text input fields with light gray borders and labels in Hungarian: "Teljes név" (Full name), "Lakcím" (Address), "E-mail cím" (Email address), "Telefonszám" (Phone number), "Felhasználónév" (Username), and "Jelszó" (Password). Below these fields is a black button with the text "REGISZTRÁCIÓ" in white capital letters.

27. ábra Regisztrációs felület kinézete

## 5.Lépés

Következőnek a bejelentkezésen dolgoztunk. A bejelentkezés gombra kattintva ki olvassuk a felhasználó által megadott felhasználónevet és jelszót majd ezeket az adatokat PHP-ban ellenőrizzük, hogy létezik-e ilyen felhasználónév – jelszó páros, és ha létezik akkor a fő oldalra navigál az applikáció.

A bejelentkezés gombra kattintva a felhasználónevet tároljuk és mindig tovább adjuk a következő oldalnak az "intent.putExtra"-val, ugyanis ezt a felhasználónevet felhasználjuk az időpontfoglaláshoz.



The image shows a login screen titled "Belépés" (Login) on a light blue background. At the top center is a circular avatar of a black and tan dog with glowing blue eyes. Below the avatar are two light gray input fields: the first is labeled "Felhasználónév" (Username) and the second is labeled "Jelszó" (Password). At the bottom center is a black button with the white text "BELÉPÉS" (Login).

28. ábra Belépés felület kinézete

## 6.Lépés

Hatodik lépésként a kutya adatainak regisztrálásán dolgoztunk. A kis kedvenc adatait bejelentkezés után tudják regisztrálni a felhasználók. Ezt a felhasználók regisztrálásához hasonló módon tettük meg. A különbség egy ImageButton ami arra szolgál, hogy a felhasználó vissza tudjon jutni a fő oldalra, ha esetleg félre kattintott volna.

Kutyám adatai

Agresszív-e

Életkor

Kutya fajtája

Tulaj neve

REGISZTRÁCIÓ


←

29. ábra Kutya regisztrációs felület kinézete

## 7.Lépés

Utolsó lépésként az időpontfoglalást készítettük el. Az időpont foglalásnál Spinner-ek által tudjuk kiválasztani a kívánt napot és órát amikor el szeretnénk vinni. Végül a kívánt szolgáltatást tudjuk kiválasztani. A szolgáltatások nevét az adatbázisból kérdezzük le JSON formában majd Stringként megjelenítjük a Spinnerbe. Majd miután kiválasztottunk minden adatot az "Időpont foglalás" gombra kattintva le is foglaltuk az időpontot.

# Időpont foglalás




Hétfő ▼

10:00 ▼

Szőrzetkezelés ▼

**IDŐPONT FOGLALÁS**



30. ábra Időpontfoglalás felület kinézete

## Backend a mobil alkalmazáshoz

A telefonos alkalmazáshoz a backendet PHP-ban írtuk meg, ez a backend arra szolgál, hogy az alkalmazás tudjon kommunikálni az adatbázisunkkal. Ezeket a PHP-fileokat a xamp/htdocs mappába kellett létrehozni, hogy el tudjuk érni őket az android studioból.

Először a DataBaseConfig.php-t írtuk meg ahol megadtuk melyik szerveren és adatbázisban dolgozunk :

```
public $servername;  
public $username;  
public $password;  
public $databasename;  
  
public function __construct(){  
    $this->servername = 'localhost';  
    $this->username = 'root';  
    $this->password = '';  
    $this->databasename = 'kutyakozmetika_2019n';  
}
```

31. ábra DataBaseConfig.php

Következőnek a DataBase.php-t csináltuk meg. Itt több függvényt írtunk meg az adatbázishoz való kapcsolódásra, regisztrációra, bejelentkezésre, időpont foglalásra és a szolgáltatások fetchelésére.

A bejelentkezésnél itt ellenőrizzük, hogy a beolvasott felhasználónév és jelszó páros létezik-e.

```
function login($table, $felhasznalonev, $jelszo)  
{  
    $felhasznalonev = $this->prepareData($felhasznalonev);  
    $jelszo = $this->prepareData($jelszo);  
    $this->sql = "select * from " . $table  
    . " where felhasznalonev = '" . $felhasznalonev . "'";  
    $result = mysqli_query($this->connect, $this->sql);  
    $row = mysqli_fetch_assoc($result);  
    if (mysqli_num_rows($result) != 0) {  
        $dbusername = $row['felhasznalonev'];  
        $dbpassword = $row['jelszo'];  
        if ($dbusername == $felhasznalonev  
            && password_verify($jelszo, $dbpassword)) {  
            $login = true;  
        } else $login = false;  
    } else $login = false;  
  
    return $login;  
}
```

32. ábra Database.php-ban a login függvény



A függvények mellé további PHP fileokat hoztunk létre mint például login.php vagy szolgáltatásFetch.php.

Ezek a PHP fileok arra szolgálnak, hogy a felhasználó értesüljön a felmerülő hibákról. Például a bejelentkezésnél ki írja üzenetként, hogy belépés sikeres, ha nem sikeres akkor pedig hiba üzenetet ad vissza.

```
<?php
require "DataBase.php";
$db = new DataBase();
if (isset($_POST['felhasznalonev'])
    && isset($_POST['jelszo'])) {
    if ($db->dbConnect()) {
        if ($db->login("felhasznalo"
            , $_POST['felhasznalonev']
            , $_POST['jelszo'])) {
            echo "Login Success";
        } else echo "Username or Password wrong";
    } else echo "Error: Database connection";
} else echo "All fields are required";
?>
```

33. ábra login.php

# Webes alkalmazás:

## Webes alkalmazás témája:

A webes alkalmazást egy kutyakozmetikának készítettük. A weblap kizárólag tájékoztatás szempontjából készült. A weblapon megtekinthető a kutyakozmetika neve, rövid leírás róla. Ezek mellett még megtalálható az oldalon a szolgáltatások neve és ára, valamint a kozmetikus címe és elérhetősége.

## Program megvalósítása

### 1.Lépés

Létrehoztuk a projektet Vue.js (egy JavaScript könyvtár) keretrendszer és Vue Routert használva.

A projekt létrejötte után elkezdtünk dolgozni a főoldalon a Home.vue-ban.

A Home.vue template része tartalmaz egy rövid leírást a kozmetikusról és egy Slideshow-t, ami a script részben megírt tömbből olvassa ki az adatokat és jeleníti meg. Az oldal kinézetéért a style-ban megírt kódok felelnek.

```
<div>
  <transition-group name="fade" tag="div">
    <div v-for="i in [currentIndex]" :key="i">
      
    </div>
  </transition-group>
</div>
```

34. ábra Slideshow template-be megírt része

```
<script>
export default {
  name: "slider",
  data() {
    return {
      images: [
        "https://oroscafe.hu/wp-content/uploads/2021/03/210318_Kutyakozmetika.jpg",
        "https://www.pet4you.hu/img/big/820.jpg",
        "https://www.pet4you.hu/img/big/452_1.jpg",
        "https://adatlistazo.hu/wp-content/uploads/2022/03/miert-fontos-a-kutyakozmetika-1.jpg"
      ],
      timer: null,
      currentIndex: 0
    };
  },
  mounted: function() {
    this.startSlide();
  },
  methods: {
    startSlide: function() {
      this.timer = setInterval(this.next, 2000);
    },
    next: function() {
      this.currentIndex += 1;
    },
    prev: function() {
      this.currentIndex -= 1;
    }
  },
  computed: {
    currentImg: function() {
      return this.images[Math.abs(this.currentIndex) % this.images.length];
    }
  }
};
</script>
```

35. ábra Slideshow script-ben megírt rész

## 2.lépés

A főoldal után a szolgáltatások oldallal foglalkoztunk, melynek a kódját a Szolgáltatások.vue-ban írtuk meg. Ezen az oldalon kártyákban megjelenik a szolgáltatások neve és időtartama.

```
<template>
<div class="container">
  <div class="row">
    <div class="card col-lg-12 col-md-6 col-sm-12 p-1" v-for="szolgaltatas in szolgaltatasok" :key="szolgaltatas.szolgaltatasNev">
      <div class="card-header">
        <h4>{{szolgaltatas.szolgaltatasID}} </h4>
        <h1>{{szolgaltatas.szolgaltatasNev}}</h1>
      </div>
      <div class="card-body">
        
      </div>
      <div class="card-footer">
        <h3>Időtartam: {{szolgaltatas.idotartam}} perc</h3>
      </div>
    </div>
  </div>
</div>
</template>
```

36. ábra Kártyák kódja a template-ben

A kártyákat bootstrap segítségével formáztuk és tettük reszponzívvá. A kártyákon megjelenő adatokat az adatbázisból olvassa ki a program egy axios lekérés segítségével. Ehhez importálnunk kellett az axios és bootstrap könyvtárakat.

```
<script>
import axios from "axios";

export default {
  data() {
    return {
      szolgaltatasok: [],
    };
  },
  mounted() {
    axios.get("http://localhost:5000/api/szolgaltatas")
      .then(response => {
        this.szolgaltatasok = response.data
      })
  }
};
</script>
```

37. ábra axios lekérés a script-ben

### 3.Lépés

A szolgáltatások után az árlista oldallal foglalkoztunk, ahol a szolgáltatások neve és ára jelenik meg egy táblázatban. A szolgáltatásokat ezen az oldalon is egy axios lekérés segítségével jelenítjük meg a táblázatban (lásd 37. ábra).

```
<template>

<table>
<tr>
  <th class="nev">Szolgáltatások</th>
  <th class="ar">Ár</th>
</tr>
</table>
<table v-for="szolgaltatas in szolgaltatasok" :key="szolgaltatas.szolgaltatasNev">

  <td class="nev">{{szolgaltatas.szolgaltatasNev}}</td>
  <td class="ar"> 4500 Ft</td>
</table>

</template>
```

38. ábra Táblázat kódja a template-ben

### 4.Lépés

Az árak megjelenítése után következett a kapcsolat oldal, ahol megjelenik a kozmetikus elérhetősége és címe. A címe alatt megjelenik egy térkép, ami úgy van beállítva, hogy a kozmetikus címét mutassa alapértelmezetten.

```
<iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d2758.8453776037513!2d20.141222215547554!3d46.253301579118244!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x4744887400ddad333A0x898a2ad0addb8651!2sSzegedi%20SZC%20Vasv%C3%A1ri%20P%C3%A1l%20Gazdas%C3%A1gi%20%C3%A9s%20Informatikai%20Technikum!5e0!3m2!1shu!2shu!4v1650032860192!5m2!1shu!2shu" width="600" height="450" style="border:0;" allowfullscreen="" loading="lazy" referrerpolicy="no-referrer-when-downgrade"></iframe>
```

39. ábra Térkép kódja a template-ben

## 5.Lépés

Az App.vue-ban és index.js-ben beállítottuk, hogy a különböző oldalak között lehessen váltani.

```
<template>
  <div id="nav">
    <router-link to="/">Főoldal</router-link> |
    <router-link to="/szolgaltatas">Szolgáltatások</router-link> |
    <router-link to="/arlista">Arlista</router-link> |
    <router-link to="/kapcsolat">Kapcsolat</router-link>
    
  </div>
  <router-view/>
</template>
```

40. ábra App.vue-ban megírt kódrészlet

```
import { createRouter, createWebHistory } from 'vue-router'
import Home from '../views/Home.vue'
import Szolgaltatasok from '../views/Szolgalaltatasok.vue'
import Arlista from '../views/Arlista.vue'
import Kapcsolat from '../views/Kapcsolat.vue'
import 'bootstrap/dist/css/bootstrap.min.css'

const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/szolgaltatas',
    name: 'Szolgaltatasok',
    component: Szolgaltatasok
  },
  {
    path: '/arlista',
    name: 'Arlista',
    component: Arlista
  },
  {
    path: '/kapcsolat',
    name: 'Kapcsolat',
    component: Kapcsolat
  }
]
```

41. ábra Az index.js-ben ez a rész felel az oldalak közötti navigálásért

# Web API a webes alkalmazáshoz

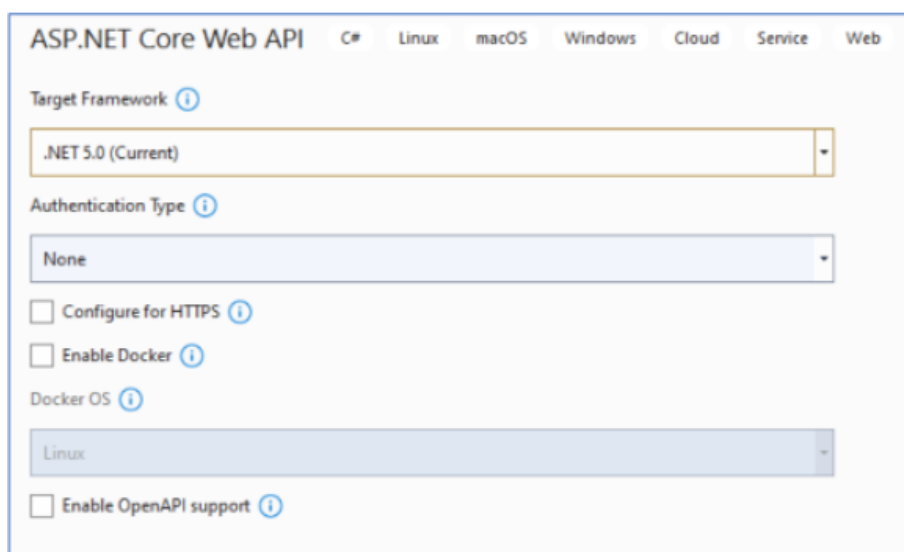
## Web API témája:

A Web API biztosítja a kapcsolatot az adatbázis és webes alkalmazás között.

## Web API megvalósítása:

### 1.Lépés

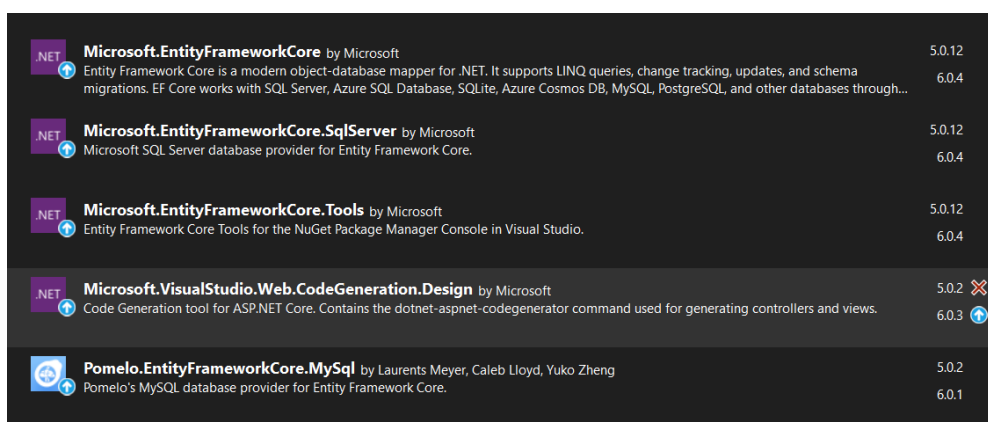
Elkészítettünk egy ASP.NET Core Web API projektet .NET 5.0-ás Frameworkot használva.



42. ábra Projekt létrehozása

### 2.Lépés

Telepítettük a szükséges Nuget csomagokat



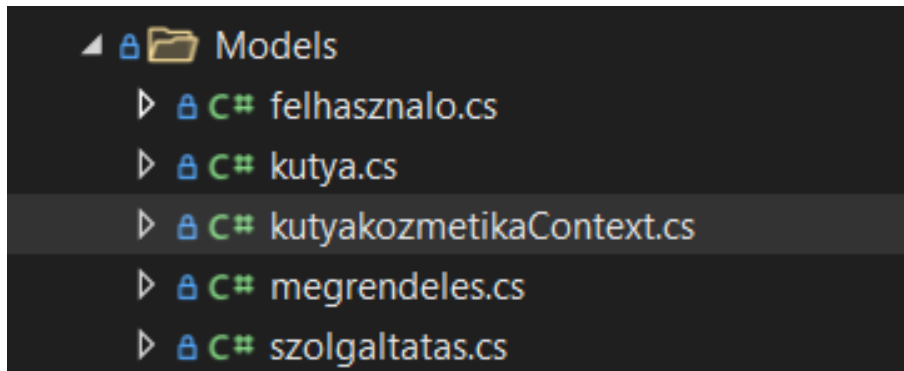
43. ábra Nuget csomagok

### 3.Lépés

Kiadtuk a Scaffold-DbContext parancsot a Package Manager Consolon:

```
Scaffold-DbContext "server=localhost;user id=root;database=kutyakozmetika_2019n"  
Pomelo.EntityFrameworkCore.MySql -OutputDir Models -DataAnnotations -  
UseDatabaseNames -Force -NoPluralize
```

Ez a parancs olvassa be az adatbázisból a táblákat és hozza létre a Context osztályt.



44. ábra beolvasott táblák és Context osztály

### 4.Lépés

Az appsettings.json-ben ezután megírtuk a szükséges kapcsolatot a Web API és az adatbázis között.

```
"ConnectionStrings": {  
  :  
  "KutyakozmetikaDB": "server=localhost;user id=root;database=kutyakozmetika_2019n"  
}
```

45. ábra kapcsolat a Web API és adatbázis között

### 5.Lépés

A Startup.cs-ben szintén beállítottuk az adatbázis és Web API közti kommunikációhoz szükséges metódust. Illetve felülírtuk a CORS policyt, hogy a webes alkalmazás megtudja jeleníteni az adatokat.

```

services.AddDbContext<kutyakozmetikaContext>(
    options =>
        // Kiolvassa az appsettings.json fájlból a kapcsolati értéket
        options.UseMySQL(Configuration.GetConnectionString("KutyakozmetikaDB"),
            ServerVersion.Parse("10.4.6-mariadb"));

```

46. ábra kommunikációhoz szükséges metódus

```

#if DEBUG
services.AddCors(options => options.AddDefaultPolicy(c =>
{
    c.AllowAnyMethod()
    .AllowAnyOrigin()
    .AllowAnyHeader();
}));
#else
services.AddCors(options => options.AddDefaultPolicy(c =>
{
    c.AllowAnyMethod()
    .AllowAnyOrigin()
    .AllowAnyHeader();
}));
#endif

```

47. ábra CORS policy felülírása

## 6.Lépés

Létrehoztunk egy üres Controllert és megírtuk bele a szükséges lekérést.

```

[Route("api/[controller]")]
[ApiController]
1 reference
public class SzolgaltatasController : ControllerBase
{
    private kutyakozmetikaContext context;

    0 references
    public SzolgaltatasController(kutyakozmetikaContext kutyakozmetikaContext)
    {
        context = kutyakozmetikaContext;
    }

    [HttpGet]
    0 references
    public async Task<IActionResult> GetSzolgaltatasok()
    {
        return Ok(await context.szolgaltatas.ToListAsync());
    }
}

```

48. ábra Get lekérés

## 7.Lépés

A Web API megírása után a lekérést Google Chrome-ban a Rested bővítményen keresztül teszteltük, ahol sikeresen megjelent minden adat.



GET  [Send request](#)

[Headers >](#)  
[Basic auth >](#)

---

Response (1.698s) - http://localhost:5000/api/szolgalatas

**200** OK

[Headers >](#)

```
[
  {
    "szolgalatasID": 1,
    "szolgalatasNev": "Szőrzetkezelés",
    "idotartam": 60
  },
  {
    "szolgalatasID": 2,
    "szolgalatasNev": "Fürdetés",
    "idotartam": 40
  },
  {
    "szolgalatasID": 3,
    "szolgalatasNev": "Körömvágás",
    "idotartam": 30
  }
]
```

49. ábra Adatok sikeres lekérése

# Adatbázis

## Tervezés

Egyedek:

- felhasználó
- kutya
- megrendelés
- szolgáltatás

Tulajdonságok:

felhasználó (**felhasználóID**, nev, cím, email, telefonszám, felhasználónev, jelszó)

kutya (**kutyaID**, agresszívE, életkor, fajta, tulajNév)

megrendelés (**megrendelésID**, foglalásNapja, foglalásOraja, felhasználónev, szolgáltatásNév)

szolgáltatás (**szolgáltatásID**, szolgáltatásNév, időtartam)

## Egyedek és tulajdonságaik(adattípusok):

*felhasználó*

---

<b>felhasználóID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
nev	szöveg (VARCHAR(100)) – a felhasználó neve
cím	szöveg (VARCHAR(100)) – a felhasználó címe
email	szöveg (VARCHAR(100)) – a felhasználó e-mail címe
telefonszám	szöveg (VARCHAR(100)) – a felhasználó telefonszáma
felhasználónev	szöveg (VARCHAR(100)) – a felhasználó felhasználóneve
jelszó	szöveg (VARCHAR(100)) – a felhasználó jelszava

### *kutya*

---

<b>kutyaID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
agresszivE	szöveg (VARCHAR(255)) – a kutya agresszív-e
eletkor	szöveg (VARCHAR(255)) – a kutya életkora
fajta	szöveg (VARCHAR(255)) – a kutya fajtája
tulajNev	szöveg (VARCHAR(255)) – a kutya tulajának a neve

### *megrendeles*

---

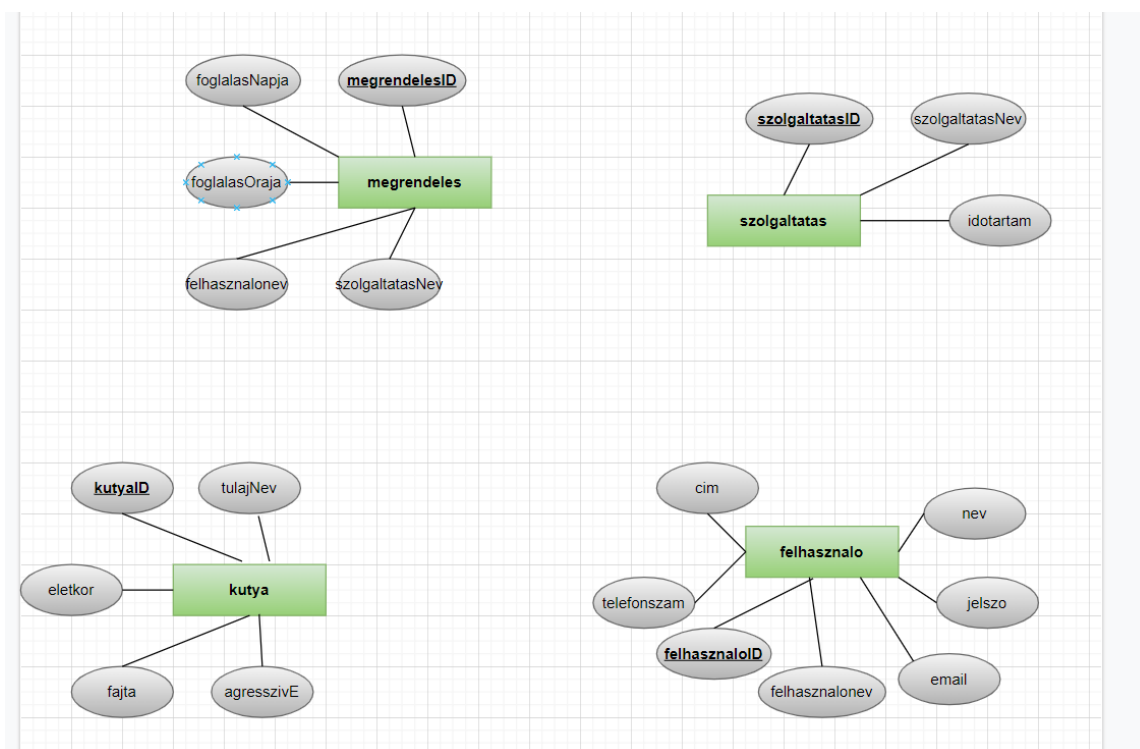
<b>megrendelesID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
foglalasNapja	szöveg (VARCHAR(15)) – a foglalás napja
foglalasOraja	szöveg (VARCHAR(15)) – a foglalás órája
felhasznalonev	szöveg (VARCHAR(255)) – a felhasználó felhasználóneve
szolgaltatasNev	szöveg (VARCHAR(255)) – a szolgáltatás neve

### *szolgaltatas*

---

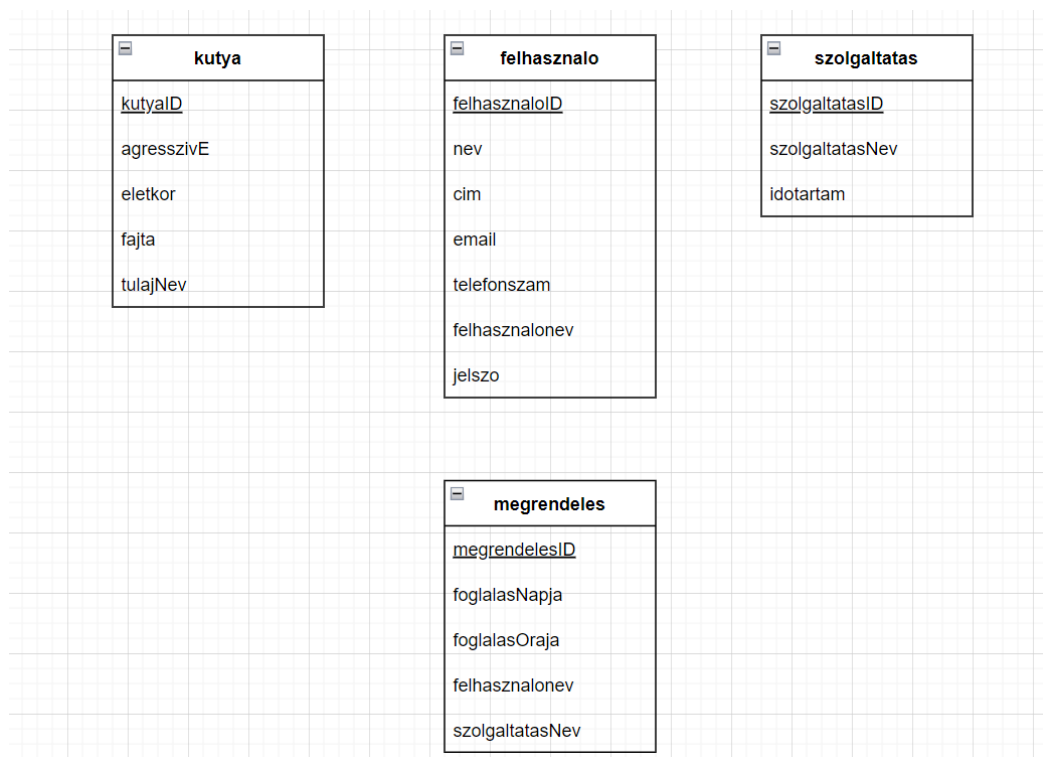
<b>szolgaltatasID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
szolgaltatasNev	szöveg (VARCHAR(100)) – a szolgáltatás neve
idotartam	szám (INT) – a szolgáltatás időtartama

## Az adatbázis E-K diagramja



50. ábra E-K diagram

## Az adatbázis Bachmann-ábrája



S

51. ábra Bachmann-ábra