Zawartość

1.Wstęp

Praca z danymi będzie zachodziła poprzez kod aplikacji. Wszelkie ograniczenia i sprawdzanie poprawności danych również będzie wprowadzane i sprawdzane po stronie kodu. Firma Microsoft oraz EntityFramework udostępnia programistom zaawansowane API umożliwiające pracę z bazą danych.
Kod przekazujący dane do bazy jest jednocześnie kodem tworzącym ją.

2. Referencje do bazy danych

```
public class ApplicationDbContext : IdentityDbContext<User>
{
    Odwołania: 0
    public ApplicationDbContext(DbContextOptions options) : base(options)
    {

    }

    Odwołania: 0
    public DbSet<Deposit> Deposit { get; set; }
    Odwołania: 0
    public DbSet<Truck> Trucks { get; set; }
    Odwołania: 0
    public DbSet<Orders> Orders { get; set; }
    Odwołania: 0
    public DbSet<Route> Routes { get; set; }
}
```

Tabelki takie jak IdentityUsers,Roles i Permissons są domyślnie wbudowane w klasę IdentityDbContext, która jest tworzona na podstawie naszej klasy User.

3.Tworzenie nowego użytkownika w systemie

3.1 Metoda rejestrująca nowego użytkownika

Kiedy użytkownik kliknie przycisk rejestracja, na początku jego dane zostaną sprawdzone pod względem poprawności. Potem zostaną przekazane na serwer gdzie zostaną wczytane do bazy danych użytkownika. Uprawnienia początkowego użytkownika będą znikome. Będą one zmienione później przez administratora

```
[HttpPost]
Odwołania: 0
public async Task<IActionResult> Post([FromBody]UserViewModel model)
{
    // Dane od klienta niewłaściwe,zwraca Http 500 ( serwer error)
    if (model == null) return new StatusCodeResult(500);

    User user = await UserManager.FindByNameAsync(model.UserName);
    if (user != null) return BadRequest("Nazwa użytkownika jest już zajęta!");

    var now = DateTime.Now;

    user = new User()
    {

        SecurityStamp = Guid.NewGuid().ToString(),
        UserName = model.UserName,
        Email = model.Email

    };

    user.EmailConfirmed = true;
    user.LockoutEnabled = false;

    string user_role = "User";

    await UserManager.CreateAsync(user, model.Password);

    if (await RoleManager.RoleExistsAsync(user_role))
        await UserManager.AddToRoleAsync(user, user_role);

    DbContext.SaveChanges();

    return Json(user.Adapt<UserViewModel>(), JsonSettings);
}
```

3.2 Metoda zwarający Kod JSON z bazy danych użytkowników


```
[
  - {
        id: "03cee67c-f2c1-44ed-8496-318532051503",
        userName: "Kasprzycki",
        normalizedUserName: "KASPRZYCKI",
        email: "Piotr.kasprzycki@onet.pl",
        normalizedEmail: "PIOTR.KASPRZYCKI@ONET.PL",
        emailConfirmed: true,
        passwordHash: "AF4V9pS2K7xHThltrtPaxKKNx0vo53UG49YoFgHfO4LyrU6wakg6q7rpAvjXJYLx0w==",
        securityStamp: "7XZTG4HARCFS2UX6V4KB2KBSLSHVYSAS",
        concurrencyStamp: "880003a1-5eaf-4483-a0fa-4a92234fa6cc",
        phoneNumber: null,
        phoneNumberConfirmed: false,
        twoFactorEnabled: false,
        lockoutEnd: null,
        lockoutEnabled: true,
        accessFailedCount: 0
    },
  - {
        id: "06990d21-e838-4093-84c8-e521d9e3b796",
        userName: "Koniakowski",
        normalizedUserName: "KONIAKOWSKI",
        email: "Koniakowski@onet.pl",
        normalizedEmail: "KONIAKOWSKI@ONET.PL",
        emailConfirmed: true,
        passwordHash: "AIAxxLQS0oNLrgIRu4n+YwjEbjNTyTjz1sz45ATnGVyKQcWicpRGxEeooSzqhA1GoA==",
        securityStamp: "JMJDBHPKLMYPH6QDEAFXTZBXTFTLVURF",
        concurrencyStamp: "6449e405-9791-4d02-a0e3-24119d540d17",
        phoneNumber: null,
        phoneNumberConfirmed: false,
        twoFactorEnabled: false,
        lockoutEnd: null,
        lockoutEnabled: true,
        accessFailedCount: 0
    },
```


Jest to przykładowy wygląd danych jakie serwer będzie wysłał na stronę w celu wyświetlenia użytkownika. W przyszłości mamy na celu stworzenie specjalnej klasy ViewModel, która będzie posiadała najważniejsze informacje o użytkowniku

3.3 Zmiana uprawnień użytkownika

```
[Authorize(Roles="Admin")]
[HttpPut]
Odwołania: 0
public async Task<IActionResult> Put(string username, string roleName)
{

    var user = DbContext.Users.FirstOrDefault(model => model.UserName == username);


    await UserManager.AddToRoleAsync(user, roleName);

    return new JsonResult(roleName);

}
```

Na serwer zostają przekazane informacje o tym jaki użytkownik ma być zmodyfikowany oraz jakie uprawnienia są mu nadawane.

3.4 Dodanie ciężarówki do bazy danych

```
[HttpPost]
Odwołania: 0
public async Task<IActionResult> Post([FromBody] Truck model)
{

    if (model.Model != null)
    {
        model.Id = Guid.NewGuid().ToString();
        DbContext.Trucks.Add(model);
        await DbContext.SaveChangesAsync();
    }

    return new JsonResult("OK");
}
```

Metoda pobiera dane od użytkownika i zapisuje je do bazy danych.

3.5 Metoda zwracająca dane z bazy danych ciężarówek

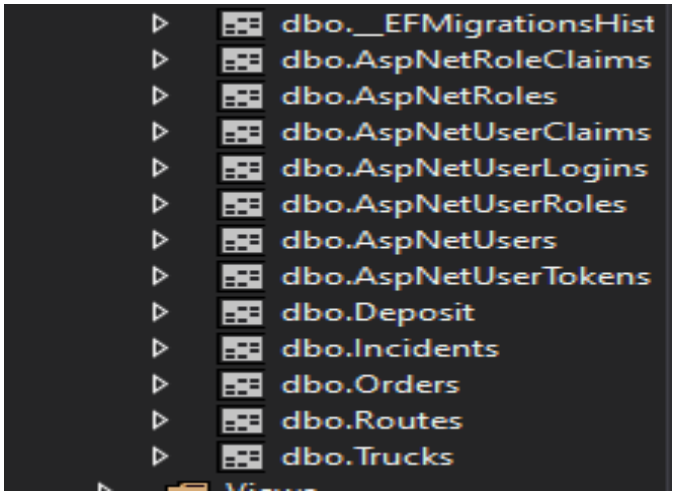
```
public async Task<IActionResult> Index()
{
    var list = await DbContext.Trucks.ToListAsync();

    return new JsonResult(list);

}
```

Metoda wyciąga z bazy danych informacje o ciężarówkach i zwraca je w postaci kodu JSON

3.6 Kod JSON zwracany przez metodę


```
[
  - {
        id: "9cb06737-f145-4893-8124-d03ac5c98943",
        brand: "MAN",
        model: "TRUCK",
        vin: "VINPRZYKLADOWY",
        registrationNumber: "WWLSAMPLE"
    },
  - {
        id: "b760fdda-fb60-4d22-bfd2-ae61bbff8cbe",
        brand: "VOLVO",
        model: "TRUCK",
        vin: "VINPRZYKLADOWY",
        registrationNumber: "WWLSAMPLE"
    }
]
```

3.7 Ostateczny wygląd bazy danych



Tak wygląda gotowa baza danych. Wszystkie tabelki które, mają w sobie nazwę ASPNET są to tableki domyślne wygenerowane przez ASP .NET CORE. Tableka MigrationsHist zawiera każdą zmianę struktury bazy danych jaka kiedykolwiek została przeprowadzona.