# WEBAPI PROJEKT

## JUNIOR SOFTWARE ENGINEER – BACKEND HOME ASSIGNMENT

## TAXIK UTAZÁSTERVEZŐ ALKALMAZÁSA

*KÉSZÍTETTE:* Papp Márton
*E-MAIL: papp.marton20030929@gmail.com*

# Tartalom

# A feladat

Create a backend application that provides a REST API for the following tasks. It should satisfy quality requirements you would set: understandable, maintainable, portable, and efficient. Please attach a documentation to your solution.

This backend application would be integrated to the administrative system of a passenger transportation (taxi) company. The used technology stack, data storage method, the particular endpoint definitions, etc. are up to the author.

## *1. feladatrész*

Adding a new vehicle to the fleet. A vehicle may have following properties:

- Passenger capacity
- Range (how many kilometres can it travel with a full tank/charge)
- Fuel (gasoline; mild hybrid; pure electric)

## *2. feladatrész*

Provide a suggestion for assigning vehicles to a trip. Here we get the number of passengers and the distance as parameters. Based on this the application should list all the combinations based on the available vehicles. For each such suggestion the application should compute the assumed profit and show it to help the dispatcher assigning vehicles to the trip.

Before the route, all the vehicles are fully fuelled/charged. We cannot stop for tanking/charging during a trip. After the trip the vehicles are always going to be refuelled for the maximum. Approaching the gas station is considered immediate.

What does the profit consist of?

- Travel fee: Every customer pays €2 for every kilometre travelled – plus another €2 for every half hour
- started
- Refuelling: After every trip, we fill the tank. This costs €2 per kilometre travelled for gasoline and €1 for electric drive

Note: The part of the trips under 50 km mostly takes place in the city, here every kilometre travelled should be calculated as taking 2 minutes, while from 50 km onwards, a kilometre travelled takes 1 minute. Similarly, for city trips (which we can consider as the part of the trips under 50 km) for hybrid cars, the remaining range should only decrease by 1 km after every second kilometre travelled.

# Elemzés

## *A koncepció megvalósítása*

- A WebAPI-t C# nyelven az ASP.NET segítségével valósítottam meg. Az atabázishoz SQL Servert használtam, melyhez Entity Framework segítségével kapcsolódik az alkalmazás.
- Az alkalmazás három projektből áll
  - *API*: kéréseket szolgálja ki
  - *DataAccess*: biztosítja az adatok tárolását és kezelését
  - *Shared*: az adatátviteli *DTO* objektumokat tartalmazza
- Az API-nak két aktív végpontja van, ezek kezelik a kéréseket. Az egyik segítségével lehet új járművet hozzáadni a flottához, a másik pedig az utazások tervezésében segít.
- Az API a DataAccess projektben lévő *service* segítségével éri el az adatbázist
- Mind az API kérésekben, mind a válaszokban *DTO* objektumokat használok, hogy könnyen integrálható legyen más *front-end* rendszerekkel

# Tervezés

## *Programszerkezet*

- A három projekt alapján `Taxio.DataAccess`, `Taxio.Shared` és `Taxio.API` névtereket hozunk létre.
- Az adatbázis-kontextus eléréséhez egy `Services` névteret is bevezetünk.

## *Szolgáltatások*

- A `Taxio.DataAccess.Services` névtér elemei végzik az adatbázis-kontextussal a kommunikációt.
- A névtérnek 2 eleme van, ebből 1 interfész és a hozzá tartozó 1 megvalósítás. Az `IVehicleService` és a hozzá tartozó `VehicleService` végzi a járművekkel kapcsolatos kéréseket.
- A többi szolgáltatás a `DataAccess` projekt kiegészítő `AddDataAccess` metódusában van regisztrálva. Ezen metódust szükséges meghívni az API projektben, hogy tudjunk kapcsolódni az adatbázishoz.

## *Adatbázismodell*

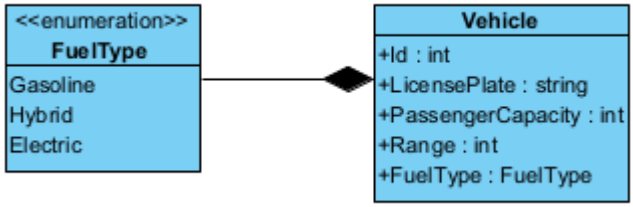
- A `Taxio.DataAccess.Models` névtér az adatbázisban használt táblák felépítését írják le. Ebben a névtérben két elem található:
  - `Vehicle` osztály
  - `FuelType` felsorolási típus (enum)
- Ezen kívül megtalálható egy `TaxioDbContext` osztály, ami az egész adatbázis felépítését foglalja magában. Ezekkel tudjuk létrehozni az adatbázist (az *Entity Framework* segítségével) code-first megközelítéssel.

- Minden objektumban ahol szükséges, egyes mezőkhöz tartozik kiegészítő információ. (Pl.: *Required, Key, ...*) Ezek segítik az adatbázis generálását és később a modellek helyességének ellenőrzését.
- Az adatbázist feltöltjük (ha üres az adatbázis) „dummy” adatokkal, amikkel tudjuk tesztelni az alkalmazás működését. Ezt a statikus `DbInitializer` osztály végzi.

## `DTO`-*k*

- A `Taxio.Shared.DTO` névtérben helyezkednek el a nézetekben és az adatátvitelhez használt modellek. Ezek egyszerű adatokat tárolnak attól függően, hogy mi szükséges egy végpont meghívásához, illetve eredményként is ilyen objektumokat kapunk.

## `Taxio.DataAccess.Model` *osztálydiagrammja*




## *API Vezérlő*

- A projekt egy API vezérlőt tartalmaz: `VehicleController`. Ez felelős az összes beérkező kérés kiszolgálásáért
- A vezérlőben az API-beli elérési útja szerint minden *resource*-hoz egy metódus van létrehozva. Az alap elérési útvonal: `/vehicles`
- Ezen metódusoknál jelölve van a kérés típusa is a nagyobb átláthatóság érdekében
- A vezérlőhöz *dependency injection* segítségével csatlakoznak a felhasznált szolgáltatások.

*Szolgáltatások, API vezérlők és a DTO-k osztálydiagrammja*