[Fleet Management App]

Functioneel

Doel van de applicatie

Met de fleet management app (FMA) moet een bedrijf in staat zijn om zijn wagenpark te beheren. Dit wil zeggen dat de applicatie aan volgende functionele vereisten moet voldoen:

- Beheren van voertuigen
- Beheren van tankkaarten
- Beheren van bestuurders

Algemene richtlijnen

Bij het maken van lijstweergaves in de applicatie moet het telkens mogelijk zijn om op alle zichtbare velden in de lijst te filteren. Vanuit de lijst moeten volgende algemene acties beschikbaar zijn:

- Nieuw item toevoegen
- Detailweergave openen
- Bewerk weergave openen
- Verwijderen van een item

Bij het verwijderen van een item moet er altijd bevestiging gevraagd worden aan de gebruiker of ze het item effectief willen verwijderen.

Gegevens die hieronder in het vet zijn aangeduid zijn dingen die verplicht in te vullen zijn bij het aanmaken en/of het editeren.

Beschrijving per onderdeel

Beheren van voertuigen

Bij een voertuig moeten volgende gegevens worden bijgehouden:

- **Merk en model van de wagen**
- **Chassisnummer**
- **Nummerplaat**
- **Brandstoftype** (benzine, diesel, hybride benzine, hybride diesel, elektrisch, …)
- **Type wagen** (personenwagen, bestelwagen, …)
- Kleur
- Aantal deuren
- Persoon die met het voertuig rijdt

Het mag niet mogelijk zijn om een wagen met een zelfde chassisnummer in te geven. Als dit voorkomt dan moet er een melding komen dat deze wagen reeds in het systeem zit. Dit geldt ook voor de nummerplaat, ook hier moet een melding komen dat deze nummerplaat reeds in het systeem aanwezig is.

Bij het toevoegen of het bewerken van een voertuig moet je de persoon die rijdt met het voertuig kunnen selecteren uit een lijst van personen die ook uit het systeem komen.

Beheren van bestuurders

Voor een bestuurder moeten volgende gegevens bijgehouden worden

- **Naam**
- **Voornaam**
- Adres (straat + nummer, Stad, postcode)
- **Geboortedatum**
- **Rijksregisternummer** (met validatie https://nl.wikipedia.org/wiki/Rijksregisternummer)
- **Type rijbewijs**
- Het voertuig waarmee deze bestuurder rijdt
- Tankkaart die de bestuurder in bezit heeft

Het mag niet mogelijk zijn om twee bestuurders met hetzelfde rijksregisternummer toe te voegen. Indien dit toch voorvalt, dan moet er gemeld wordendat deze persoon reeds in het systeem aanwezig is.

Beheren van tankkaarten

Bij een tankkaart moet volgende data bijgehouden worden:

- **Kaartnummer**
- **Geldigheidsdatum**
- Pincode
- Brandstoffen die met de kaart getankt kunnen worden
- Bestuurder die de tankkaart in bezit heeft.

Ook een tankkaart kan slechts eenmaal in het systeem voorkomen. Hierop komt met andere woorden een controle en de nodige meldingen worden aangegeven. Tankkaarten moeten kunnen geblokkeerd worden (bij verlies of diefstal).

Technisch

Technische randvoorwaarden

- De code wordt ontwikkeld met Visual Studio en bevat één solution met verschillende projecten
- Er wordt gebruik gemaakt van C# in een .NET Core omgeving (REST API), gecombineerd met Javascript React voor de user interface
- Als databank wordt er gebruik gemaakt van MySQL
- De technologie om met de databank te communiceren, is ADO.NET
- De code base wordt beheerd in Azure DevOps
- Voor de user interface wordt er gebruik gemaakt van React
- Configuratie info wordt in config bestanden beheerd
- Voor het schrijven van Unit Tests wordt aan serverkant gebruik gemaakt van xUnit

Opmerkingen :

- Voorzie een flexibel ontwerp zodanig dat veranderingen geen al te grote impact hebben.
- Zorg voor een gebruiksvriendelijke interface.
- Er worden geen extra packages gebruikt (indien toch nodig dan moet dit eerst worden besproken en goedgekeurd worden).

Werkwijze

Dit project wordt uitgevoerd door een team van programmeurs (in dit geval 3 tot 4 personen) waarbij samenwerken een belangrijke rol speelt. Naast het schrijven van code zijn er nog een heleboel andere taken (misschien niet de leukste) die moeten worden uitgevoerd. Het uitvoeren van een project is niet enkel de applicatie opleveren op een voorgestelde datum. In de eerste plaats moeten we er zeker van zijn dat hetgeen we opleveren wel degelijk is wat de klant wenst. Daarom zullen we op een 'agile'-achtige manier te werk gaan. Dat wil zeggen dat we een aantal tussentijdse opleveringen gaan doen waarbij we afchecken bij de klant of dit is volgens de verwachtingen. Daarnaast moeten we er ook rekening mee houden dat deze applicatie zal moeten worden onderhouden (misschien niet door ons) en dat er dus voldoende documentatie beschikbaar is zodat een andere programmeur dit ook zou kunnen uitvoeren.

Om dit vlot te laten verlopen, definiëren we een aantal rollen en verantwoordelijkheden. Deze rollen zijn:

Organisatie-rol :

- Inplannen meetings
- Rapporteert over de taakverdeling
- Zorgt voor de opvolging van het project (ook rapportage)
- Zorgt ervoor dat er voldoende documentatie is

Business-rol :

- Klaart elke onduidelijkheid over de business uit aan het team
- Controleert op inhoudelijke correctheid
- Stelt (inhoudelijke) scenario's op die moeten worden getest
- Zorgt ervoor dat de business duidelijk wordt geïnformeerd over de vooruitgang

Testing-rol :

- Zorgt dat er voor elke klasse de nodige unit tests zijn
- Zorgt ervoor dat alle unit tests steeds op groen staan (ook na aanpassingen)
- Geeft overzicht van wat er getest is
- Zorgt voor integratie tests
- Zorgt ervoor dat volledige applicatie is getest

Architectuur-rol :

- Is verantwoordelijk voor het ontwerp
- Maakt beslissingen over architectuur
- Documenteert en motiveert beslissingen
- Is verantwoordelijk voor de code base (Git)

Opmerking : als er bij verantwoordelijkheden staat "zorgt ervoor dat ...", dan wil dat niet per se zeggen dat de persoon met deze rol ook deze taak moet uitvoeren. Deze persoon moet er enkel voor zorgen dat de taak wordt uitgevoerd. Bij de organisatie-rol staat er bijvoorbeeld dat deze persoon ervoor zorgt dat er voldoende documentatie is. Dit wil niet zeggen dat deze persoon alle documentatie moet schrijven. Dit betekent wel dat deze persoon erop moet toezien dat elke klasse gedocumenteerd is op een eenduidige manier, dat overzichtschemas van de klassen (rol architectuur) en testen (rol testing) op één plaats beschikbaar zijn en dat de documentatie inhoudelijk voldoet.

Git/Azure DevOps

De code moet beheerd worden in Azure DevOps (git), en dit is een taak van elk lid van het team. Dit wil zeggen dat iedereen een basiskennis moet hebben aangaande het gebruik van Git/Azure DevOps. Onderstaande links wijzen naar een aantal tutorials die daarbij kunnen helpen.

Opmerking: wanneer er aan verschillende taken tegelijkertijd wordt gewerkt door verschillende teamleden is het belangrijk dat we elkaar niet 'storen', branches/pull requests zouden daarbij zeker kunnen helpen.

Referenties

- Entity Framework Core: https://docs.microsoft.com/en-us/ef/core/
- Dot net Core: https://docs.microsoft.com/en-us/dotnet/core/
- Design patterns: https://www.dofactory.com/net/design-patterns
- Repository Pattern: https://deviq.com/repository-pattern/
- https://www.youtube.com/playlist?list=PLB5jA40tNf3v1wdyYfxQXgdjPgQvP7Xzg
- https://www.youtube.com/watch?v=Xb7r3KETy6g
- https://www.youtube.com/watch?v=nEzeMdzA_Wk