Ontwerpdocument eTransport



Studentnaam: Owen de Bree

Studentnummer: I461496

Klas: S2-DB01

Datum: 16-04-2021

Inhoudsopgave

[Projectbeschrijving 3](#_Toc68615221)

[Hoofddoel 3](#_Toc68615222)

[Belangrijkste gebruikers 3](#_Toc68615223)

[Vergelijkbare applicaties 3](#_Toc68615224)

[Architectuur 4](#_Toc68615225)

[Klassendiagrammen 5](#_Toc68615226)

[Klassendiagram v2 5](#_Toc68615227)

[Klassendiagram v6 6](#_Toc68615228)

[Database ontwerp 7](#_Toc68615229)

# Projectbeschrijving

Hieronder heb ik kort in de verschillende kopjes toegelicht wat de focus van mijn project is en wat ik wil bereiken met het maken van deze applicatie.

## Hoofddoel

Het hoofddoel van deze applicatie is het toegankelijker maken van speciaal transport voor zowel de particulieren als de bedrijven. Dit ga ik doen door zowel openstaand transport van locatie tot locatie beschikbaar te maken (ridesharing) en het afhuren van privé transport. Ik ga me aan de sensible default houden voor de verschillende keuzes van hoe ik het project ga realiseren. Dit betekent dat ik gebruik ga maken van asp.net core, me ga houden aan de principes van SOLID, gebruik ga maken van UML klassendiagrammen, git gebruik voor versiebeheer en een SQL database gebruik om gegevens op te slaan. Verder ga ik voor mijn front end gebruik maken van het Blazor framework zodat ik de javascript interactie uit mijn project kan halen en kan vervangen met C#.

## Belangrijkste gebruikers

Er zullen hoogst waarschijnlijk 2 rollen beschikbaar worden binnen de applicatie: Gebruikers en de Administrator(s)

De gebruikers hebben de mogelijkheid om zich in te boeken op al bestaande transporten of kunnen een nieuw transport aanvragen.

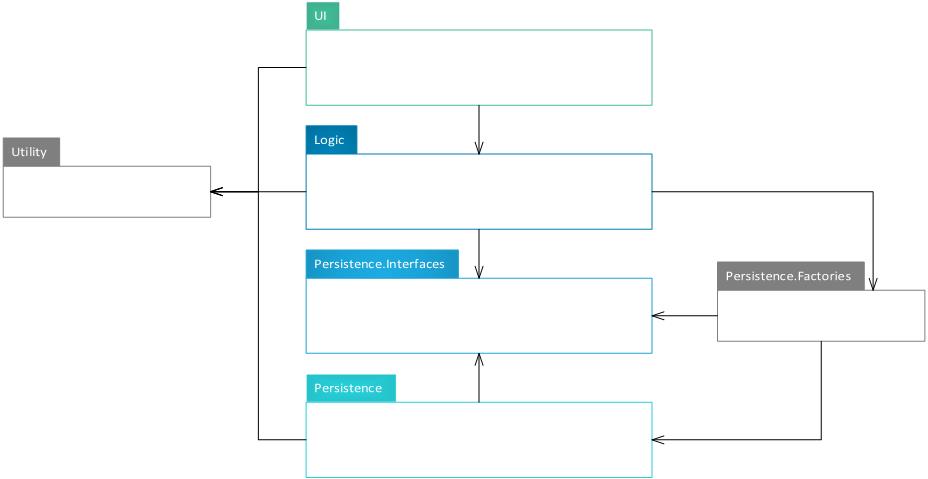
De administrator kan accounts beheren, routes/aanvragen goedkeuren, nieuwe rideshare routes maken, inboekingen op rideshare goedkeuren en nieuwe transporttypes toevoegen aan de applicatie.

## Vergelijkbare applicaties

Er zijn op dit moment geen duidelijk vergelijkbare applicaties voor het huren van speciaal transport. In de meeste gevallen is speciaal transport namelijk alleen te huren door het invullen van een online offerte.

# Architectuur

Hieronder staat de tweede versie van mijn architectuur met de tot nu toe behandelde informatie hierover uit de workshops. Ik heb ervoor gekozen om nog extra afsplitsing te maken voor de logic interfaces, dit is omdat interface segregation nodig is voor het onderscheid tussen de user actor en de admin actor. Om circular dependencies te voorkomen maak ik ook gebruik van factories en dto’s om zo data van en naar de verschillende lagen te krijgen. Om te voorkomen dat ik service models die in de front-end staan moet gaan omzetten naar dto’s heb ik ervoor gekozen om mijn dto’s te verplaatsen naar het Utility project zodat ik geen extra services hoef aan te maken, aangezien het toch alleen maar bakjes zijn. Verder heb ik mijn interfaces naar de UI toe nu samengevoegd met de logica. De front-end maakt nog altijd gebruik van interfaces en heeft geen directe toegang tot de klassen.

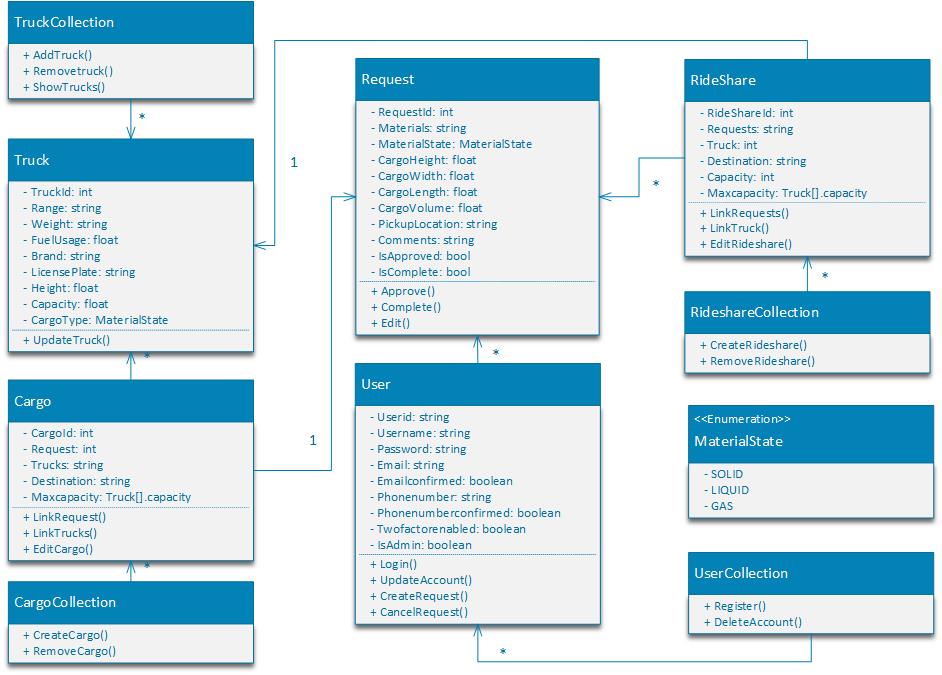


Aangezien ik gebruik maak van Blazor als front end framework en entity framework voor mijn data laag. Heb ik een extra laag toegevoegd aan mijn project waar alle frameworks mogelijk bij kunnen komen vanuit de verschillende lagen.

# Klassendiagrammen

## Klassendiagram v2

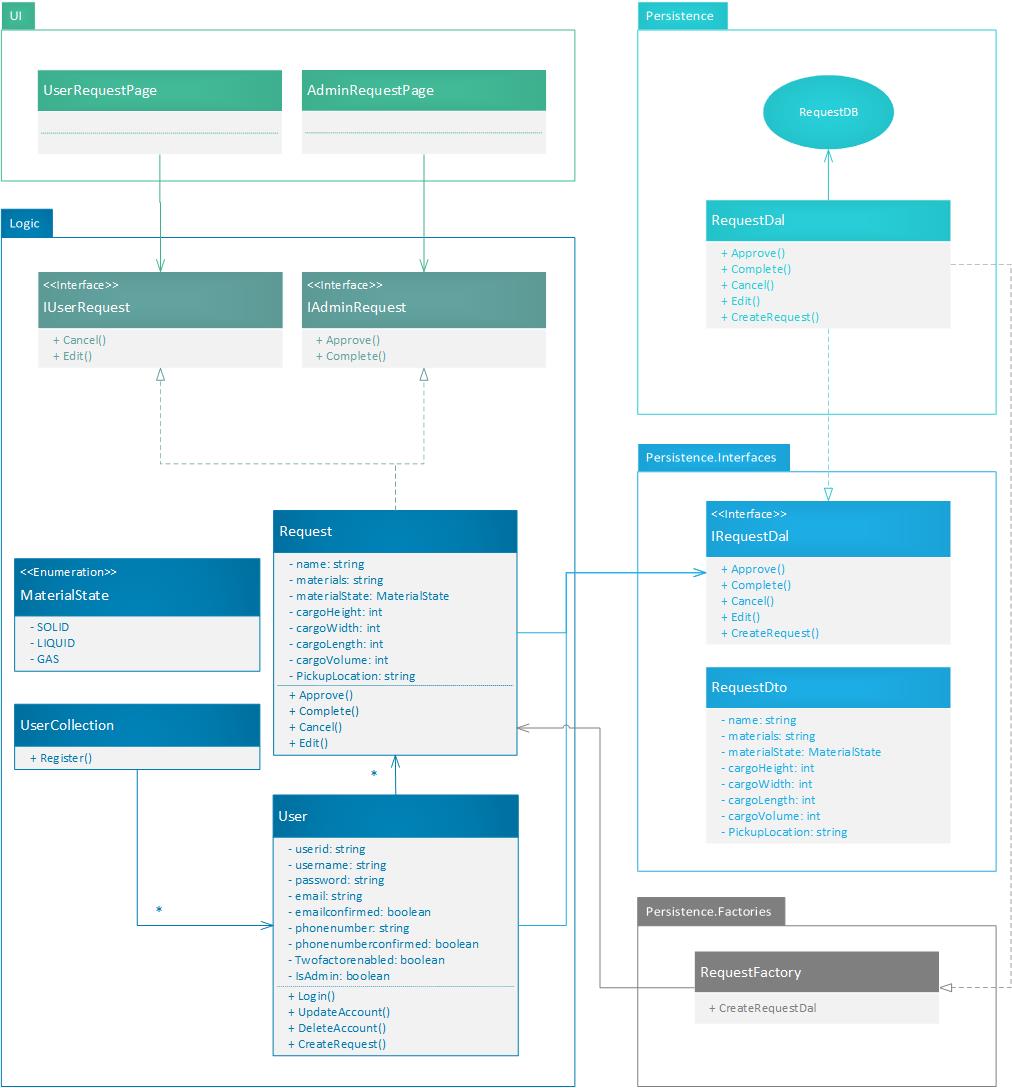
Hieronder staat mijn laatste versie van mijn klassendiagram versie 2. Hier is te zien hoe de requirements en use-cases terug komen in de logica laag binnen mijn project.



Alle collections zijn voor het toevoegen van nieuwe objecten van de bijbehorende klassen. Elke user heeft de mogelijkheid om vrachtoffertes aan te vragen en deel te nemen aan rideshares. Ik heb aleen een admin als actor nodig en niet als entiteit aangezien ik geen extra informatie over admins opsla, vandaar dat er alleen een boolean met isAdmin in het diagram staat.

## Klassendiagram v6

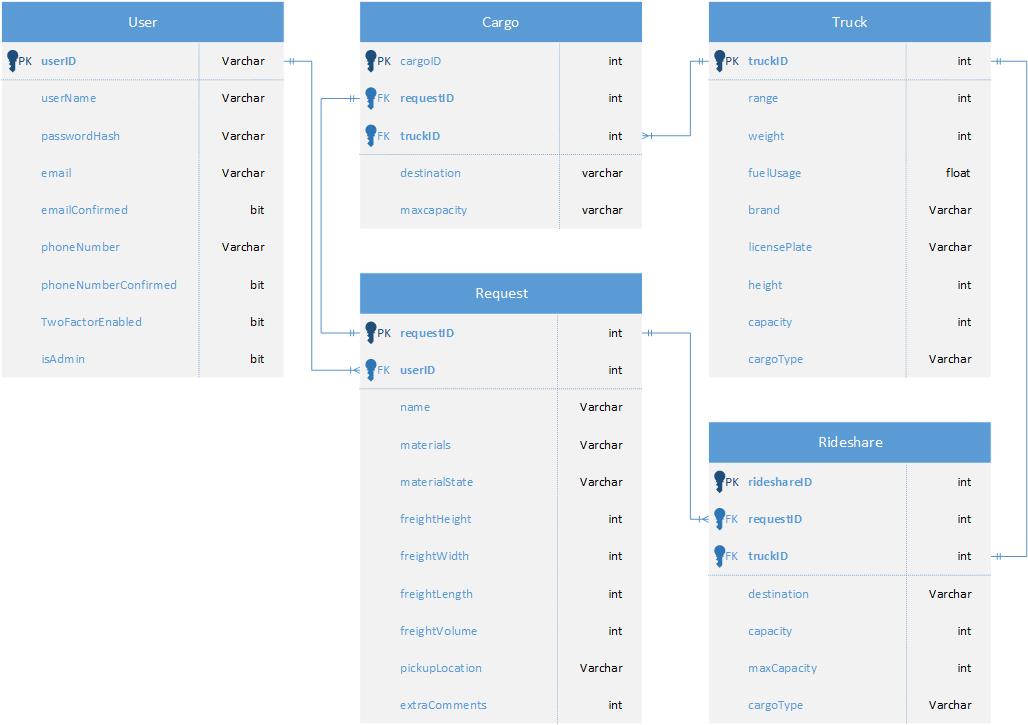
Hieronder staat het tweede klassendiagram dat de communicatie tussen de onderlinge lagen weergeeft voor 1 entiteit. De structuur van dit klassendiagram is dus ook aan te houden voor de andere entiteiten binnen mijn project.



Ik heb ervoor gekozen om de communicatie tussen zowel de logica en persistentie laag als de ui en logica laag weer te geven, omdat er door interface segregation een duidelijk verschil is tussen de communicatie van de onderlinge lagen. Doordat ik mijn interfaces naar de UI in de logica laag heb gezet heb ik hier geen factories meer nodig om instanties te maken.

# Database ontwerp

In het onderstaande ontwerp zijn mijn database relaties te vinden met alle bijbehorende typen.



Ik heb ervoor gekozen om de admin als boolean toe te voegen aan de user table aangezien er geen andere entiteiten nodig zijn waar ik een rollen systeem voor zou moeten maken.