Handleiding

# Inleiding

Het WarehouseManagementSimple (WMSimple) project is een simpele WMS-implementatie, ontworpen om inzicht te geven in de features van een typische WMS zonder de complexe configuratie die meestal voordoet bij een complete oplossing.

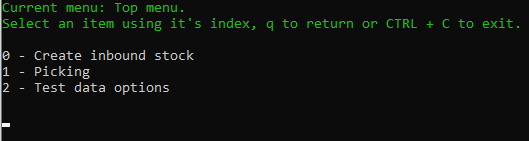
Dit document zal een overzicht bieden van de functionaliteit van het project vanuit een technisch en functioneel oogpunt. Ook zal er een stuk uitleg komen voor het uitbreiden van het project met je eigen code.

# Functionele Features

WMSimple is een C# project met een embedded SQLite database oplossing en een Command-Line interface UI om basis management acties uit te voeren binnen het WMS.

## Command-Line Interface

De command-line interface biedt een aantal sub-menu's waarmee het mogelijk is om een aantal WMS flows uit te voeren. Deze sub-menu's kunnen benaderd worden door het **nummer** op te geven van de optie die je wilt gebruiken. Door **q** in te voeren binnen een sub-menu is het mogelijk om terug te gaan naar de vorige menu. Door **CTRL+C** is het mogelijk om direct de applicatie te stoppen.



Figuur 1 Hoofdmenu van WMSimple

### Create Inbound Stock

Via deze menu optie is het mogelijk om voorraad aan te maken die binnen het warenhuis komt, bijv. dat er een vrachtwagen binnen komt die artikelen aflost. Het artikel, de locatie en het aantal kan vanuit dit menu geselecteerd worden.

A screen shot of a computer

AI-generated content may be incorrect.

Figuur 2 Create inbound stock Menu

Dit sub-menu is bedoeld voor Inbound stock, maar er is verder geen restrictie op de locatie die geselecteerd wordt.

### Picking

A screen shot of a computer

AI-generated content may be incorrect.

Figuur 3 Picking Submenu

Het Picking menu is het mogelijk om op Orders te vervullen. Je begint door eerst een doos te maken via de **Load Box** optie, hierin kun je een bestaande doos kiezen of een nieuwe aanmaken.

A black screen with white text

AI-generated content may be incorrect.

Figuur 4 Load Box Submenu

Vervolgens kun je een Order kiezen om te picken via de **Pick Order** menu optie, hierin selecteer je eerst de Order, en daarna de artikelen die je wilt picken. Artikelen die volledig gepicked zijn zullen **groen** aangegeven worden. Als alles is gepicked, kun je met **q** teruggaan naar het vorige menu.

A screen shot of a computer program

AI-generated content may be incorrect.

Figuur 5 Pick Order Submenu

Nadat alles is gepicked is het tijd om de order naar het uitgaande gebied te brengen, dit kan via de **Put Box** optie. Zodra een locatie is gekozen wordt de geladen doos daar naartoe verplaatst en is hij niet meer ingeladen.

A screen shot of a computer

AI-generated content may be incorrect.

Figuur 6 Put Box Menu

### Test data options

Dit is een klein submenu waarmee het mogelijk is om de testdata te beheren. In de meeste gevallen wil je **optie 1** selecteren, hiermee wordt de bestaande data verwijderd en wordt de testdata opnieuw ingeladen. Het is aangeraden om de **applicatie opnieuw op te starten** na deze optie sinds er een kans is dat een van de submenu nog oude data zal bevatten.

A screenshot of a computer

AI-generated content may be incorrect.

Figuur 7 Test Data options menu

## Workers

WMSimple bevat een concept van Workers, deze voeren in de achtergrond continu taken uit. Binnen een compleet WMS-pakket zouden deze Workers geimplementeerd kunnen worden als aparte processen, maar binnen WMSimple is dit geimplementeerd via aparte threads die hun eigen logica doorlopen.

Communicatie tussen de workers gebeurt via een MessageBus, verder is directe communicatie niet mogelijk. De MessageBus is in dit geval een simpele zelfgemaakte implementatie, maar een meer completere implementatie zoals RabbitMQ zou ook een optie kunnen zijn.

De volgende Workers zijn er standaard beschikbaar:

* InboundStockWorker
  + Een worker die samenwerkt met het Inbound Stock menu, op basis van de ontvangen berichten in de MessageBus maakt het nieuwe Stock aan.
* GenericMoveWorker
  + Een worker die op basis van meegegeven condities Storage Units verplaatst van Locatie A naar Locatie B.
  + De standaard Setup is om voorraad van INBOUND01 naar een picking area te brengen.

# Data Objecten

Binnen WMSimple zijn er een aantal objecten die onderdelen van het warenhuis representeren, deze zijn als volgt:

## StUnit (Storage Unit)

Een algemeen concept voor objecten binnen een warenhuis die artikelen kunnen bevatten, bijv. dozen, pallets en totes. De artikelen binnen een Storage Unit kunnen bestemd zijn voor een Order, of als voorraad dienen.

* *int* **StUnitID**
  + Primary Key voor StUnit, begint bij 1 en wordt automatisch opgehoogd.
* *string* **StUnitType**
  + Type van de storage unit, bijv. een kleine doos, een pallet etc.
* *int* **FK\_LocationID**
  + Huidige locatie van de StUnit. Een 0 of minder betekent dat hij niet op een locatie staat.
  + Foreign Key naar Location.LocationID

## Location

Een locatie binnen het warenhuis, dit kan of een locatie zijn zonder coördinaten of met coordinaten zoals een rack.

* *string* **Mha**
  + Primary Key, the top-level identificatie voor een locatie.
  + Volledige benaming is “Material Handling Area”.
* *string* **Rack**
  + Primary Key, rack Identifier.
* *string* **HorCoor**
  + Primary Key, horizontale coördinaat.
* *string* **VerCoor**
  + Primary Key, verticale coördinaat.
* *int* **LocationID**
  + Unique Key, begint bij 1 en wordt automatisch opgehoogd.

## Article

Een artikel definitie wat binnen het warenhuis bevindt, bijv. etenswaren, elektronica, boeken etc. De testdata van het project is voornamelijk gefocust op elektronica.

* *string* **Partno**
  + Primary Key, artikel identifier.
* *string* R**evision**
  + Primary Key, artikel revisie.
  + Bepaalde artikelen kunnen verschillende versies hebben. Bijv. een fles Cola van 0,5L en eentje van 1,5L
* *int* **ArticleID**
  + Unique Key, begint bij 1 en wordt automatisch opgehoogd.
* *decimal* **DimensionsLength**
  + Artikel lengte in meters.
* *decimal* **DimensionsWidth**
  + Artikel breedte in meters.
* *decimal* **DimensionsHeight**
  + Artikel hoogte in meters.

## Order

Een Order is een uitgaande opdracht om bepaalde artikelen naar een klant to vervoeren. Op basis van de Order kunnen Pick opdrachten uitgevoerd worden die ervoor zorgen dat de benodigde artikelen verzameld worden in een doos en deze klaargezet wordt voor verzending.

De inhoud van een Order wordt in de OrderContent object verwerkt.

* *int* **OrderID**
  + Primary Key voor Order, begint bij 1 en wordt automatisch opgehoogd.
* *string* **CustomerName**
  + Naam van de klant waar het Order naar verstuurd moet worden.
* *string* **CustomerAddress**
  + Adres van de klant waar het Order naar vestuurd moet worden.

## OrderContent (Inhoud van de Order)

De OrderContent object bevat de inhoud van een Order. Het bestaat uit een referentie naar de Order waar hij bij hoort, een artikel referentie, en het aantal van het artikel.

* *int* **FK\_OrderID**
  + Primary Key
  + Foreign Key naar Order.OrderID
  + Order.OrderID moet bestaan, het kan geen lege referentie zijn.
* *int* **FK\_ArticleID**
  + Primary Key
  + Foreign Key naar Article.ArticleID
  + Article.ArticleID moet bestaan, het kan geen lege referentie zijn.
* *int* **Quantity**
  + Het aantal benodigde artikelen voor de Order.

## StUnitContent (Inhoud van een StUnit)

De StUnitContent bevat de inhoud van een storage unit, het bevat een referentie naar de StUnit ID, de Artikel en het aantal van dit Artikel. De inhoud van een StUnit kan gekoppeld worden aan een order doormiddel van de OrderID referentie. Hiermee kan er onderscheid gemaakt worden tussen artikelen zonder bestemming en artikelen met.

* *int* ***FK\_StUnitID***
  + Primary Key
  + Foreign Key naar StUnit.StUnitID
  + StUnit.StUnitID moet bestaan, het kan geen lege referentie zijn.
* *int* **FK\_ArticleID**
  + Primary Key
  + Foreign Key naar Article.ArticleID
  + Article.ArticleID moet bestaan, het kan geen lege referentie zijn.
* *int* **FK\_OrderID**
  + Primary Key
  + Foreign Key naar Order.OrderID
  + Een waarde van 0 of lager geeft aan dat de inhoud niet aan een Order is verbonden.
* *int* **Quantity**
  + Het aantal artikelen binnen de StUnit.

# Architectuur

Het project, wat uitgewerkt is in een C# solutie, bestaat uit verschillende projecten/modules. Dit is gedaan om de Domein Objecten en de Database connectie apart te houden van de Front-end CLI. Dit maakt het ook mogelijk om bepaalde onderdelen in UnitTests te verwerken zonder afhankelijk te zijn van het CLI-project.

De volgende projecten/modules bestaan er:

* **Core** – Dit project bevat puur alleen de Domein Objecten, zonder extra logica eraan verbonden.
* **DB** – Dit project verzorgt de verbinding met de Database.
  + Het biedt verschillende classes aan die de CRUD-acties van de Domein Objecten bieden als functies. Hiermee kunnen de basis database acties uitgevoerd worden zonder dat er nieuwe query’s geschreven hoeven worden.
  + Extra functionaliteit die niet CRUD-functies zijn, kunnen uitgewerkt worden in de *Services* namespace/folder van dit project.
  + De achterliggende database technologie, SQLite, is qua query gebruik heel vergelijkbaar met andere SQL-databases. Voor meer informatie over SQLite query opties, zie hier: <https://www.sqlite.org/lang.html>
* **MessageBus** – Dit project bevat de MessageBus implementatie die wordt gebruikt voor de communicatie tussen workers.
  + Binnen dit project zijn ook de definities beschikbaar van een aantal workers, zoals de *GenericMoveWorker* en de *ExampleWorker*.
* **InboundStockWorker** – Dit project bevat de InboundStockWorker, die verantwoordelijk is voor het aanmaken van inkomende vracht.
* **CLI** – Bevat de front-end code, in de vorm van een Command Line Interface.
  + Wanneer je het project wilt draaien, zal dit project het opstart project moeten zijn.
* **UnitTests** - Bevat de UnitTests van het project, gebruikt XUnit als test platform.

# Customization

Om een duidelijk start punt aan te geven zijn hier de aanbevolen manieren om een eigen worker of menu toe te voegen aan de applicatie.

## Eigen Worker

Voor het maken van een worker gaat de voorkeur uit om eerst een aparte class library te maken, zoals dit is gedaan voor *InboundStockWorker*. Maak een class aan voor de worker die de *Worker* class inherit, en maak de benodigde functies aan.

A screenshot of a computer program

AI-generated content may be incorrect.

Figuur 8 ExampleWriteWorker

A computer screen shot of text

AI-generated content may be incorrect.

Figuur 9 ExampleReadWorker

Een goed startpunt zijn de *ExampleWriteWorker.cs* en *ExampleReadWorker.cs* classes van het MessageBus project. Deze hebben voorbeelden hoe je data van de MessageBus kan inlezen, en kan uitschrijven. De data die op de MessageBus geschreven wordt is altijd in string formaat, dus om meer geadvanceerde datastructuren te gebruiken is het best pratice om een datastructuur te serializen naar JSON. Dit is vrij makkelijk te implementeren door een aparte class te maken met de datastructuur die nodig is, en deze vervolgens de *MessageBug.Transaction<T>* class te laten inheriten. Zie het volgende voorbeeld van de *InboundStockTransaction*:

A screen shot of a computer program

AI-generated content may be incorrect.

Figuur 10 InboundStockTransaction class

Hiermee krijgt de class *FromJson()* en *ToJson()* functies die serialization makkelijk maken.

Eenmaal dat de Worker klaar voor gebruik is, moet hij nog geregistreerd worden binnen het CLI project, dit wordt gedaan binnen de *WorkerSetup* class in de *CreateWorkers()* functie:

A computer screen shot of text

AI-generated content may be incorrect.

Figuur 11 CreateWorkers functie van WorkerSetup

## Eigen Menu

Het maken van een eigen Menu gebeurt binnen het CLI project, maak een nieuwe class die het MenuComponent class inherit. Hiervoor moet je 2 functies specifiek implementeren: *ShowOptions()* en *UseInput()*. *ShowOptions()* is bedoeld om de tekst op het scherm te tonen via *Console.WriteLine()*. *UseInput()* wordt gebruikt om de user input te verwerken. *UseInput()* zal altijd “q” input interpreteren als “go back” dus het is niet nodig om deze zelf te implementeren.

Verder is er niet veel nodig om je eigen menu samen te stellen, het is mogelijk om submenu’s aan te maken en te openen door deze in de *UseInput()* functie te returnen. Voor een simpel voorbeeld, zie het *TestDataMenu.cs* class:

A screenshot of a computer program

AI-generated content may be incorrect.

Figuur 12 TestDataMenu

Zodra je Menu is afgerond kan deze worden geregistreerd binnen de *Menu* class in de *CreateMenu()* functie. Zodra hij hierin is geregistreerd, dan zal hij verschijnnen binnen het MainMenu.

A screenshot of a computer program

AI-generated content may be incorrect.

Figuur 13 CreateMenu

## Database gebruik

De standaard database objecten (*DBArticle, DBOrder* etc.) bestaan alleen uit CRUD-acties. In het algemeen volgen alle CRUD-operaties de volgende interface:

A screenshot of a computer code

AI-generated content may be incorrect.

Als het **ref** keyword aanwezig is, dan betekent dit dat er ook data geschreven wordt naar het input object. Een simpel voorbeeld is de *Get()*, in het input object geef je de Primary Keys mee, deze worden gebruikt om het record te vinden. Als deze gevonden is dan wordt er **true** gereturrned en zal het **obj** ingevuld worden met de overige velden.

Het kan zich voorkomen dat er een meer geavanceerde Database acties uitgevoerd moet worden. Hiervoor wordt het aangeraden om deze in de Services namespace van het DB-project toe te voegen. Een goed voorbeeld hierin is de *StUnitContentExtended* class. Doormiddel van de basis CRUD-operaties en LINQ-operaties worden er meer geadvanceerde queries uitgevoerd.

Mocht het nodig zijn, dan is het ook mogelijk om daadwerkelijk een eigen SQL-query te schrijven. Maar het is aangeraden om dit alleen te doen wanneer het onlogisch of niet mogelijk is om het via LINQ en C# logica af te handelen.

A screenshot of a computer code

AI-generated content may be incorrect.

Figuur 14 StUnitContentExtend