HOMESTORAGE

Lastenheft

HomeStorage

Version 0.1

Autor des Dokuments	Patschka Sascha-Heinz	Erstellt am	12.04.2007
Dateiname	Lastenheft_HomeStorage_0_2.doc		
Seitenanzahl	6	© 2018 Sascha Patschka	**Vertraulich!**

Historie der Dokumentversionen

Version	Datum	Autor	Änderungsgrund / Bemerkungen
0.1	12.09.2018	Sascha Patschka	Ersterstellung
0.2	12.09.2018	Sascha Patschka	Korrektur – rein EF Core anstatt Umweg über REST API

Inhaltsverzeichnis

1 Einleitung

1.1 Allgemeines

1.1.1 Zweck und Ziel dieses Dokuments

Dieses Lastenheft beschreibt wie und in welchem Umfang die Aplikation HomeStorage erstellt wird, welche Features implementiert werden und wer an diesem Projekt hauptsächlich beteiligt sein wird.

1.1.2 Projektbezug

Der User „MichaHo" alias Michael Hoffmann aus dem Forum vb-paradise.de hat den Wunsch geäußert bereits vorab in die Welt der Programmierung von WPF Applikationen unter Einhaltung des MVVM Patterns eingeführt zu werden, es gibt eine Tutorialreihe in diesem Forum welche allerdings noch länger nicht so weit fortgeschritten sein wird das hier das Thema MVVM angeschnitten werden kann. Das Projekt soll alle Vorteile aber auch Nachteile von MVVM aufzeigen und sohin so viele Funktionen welche für dieses Thema relevant sind beinhalten um einfach zu zeigen das sich MVVM auch lohnen kann da hierdurch auch viel Programmierarbeit geteilt und wiederverwendet werden kann.

1.1.3 Abkürzungen

MVVM – Model – View – ViewModel
WPF – Windows Presentation Foundation
VBP – VB Paradise Forum
DAL – DataAccessLayer = Datenzugriffsschicht
UWP – Universal Windows Plattform

1.1.4 Ablage, Gültigkeit und Bezüge zu anderen Dokumenten

Keine

1.2 Verteiler und Freigabe

1.2.1 Verteiler für dieses Lastenheft

Rolle / Rollen	Name	Telefon	E-Mail	Bemerkungen
Projektleiter	Sascha Patschka		patschka.sascha@live.com	
2. Projektleiter	Michael Hoffmann			

1.3 Reviewvermerke und Meeting-Protokolle

Noch keine

1.3.1 Erstes bis n-tes Review

Noch keines

2 Konzept und Rahmenbedingungen

2.1 Ziele des Anbieters

Ein korrektes (!!) MVVM Beispiel aufzuzeigen und zu zeigen das man mit MVVM alle Teile des Code wiederverwenden und austauschen kann. Die Struktur soll so aufgebaut werden das die Layer so gut wie möglich voneinander getrennt sind und die Application später umgebaut werden kann. Beispielsweise soll man später eine UWP App machen können welche dieselben Daten der Datenbank verwendet. Dies würde erfordern der DataAccessLayer getauscht wird da auch EF Core nicht direkt von einer UWP aus auf eine außenliegende DB zugreifen kann, dies soll mit dem geringsten Aufwand möglich sein weshalb die Layer so weit wie möglich runtergebrochen werden.

Gewisse Hauptfunktionen sollte die Applikation können (Siehe Punkt 3) damit diese ein vollständiges Programm darstellt da ich der Meinung bin das es keinen Sinn macht ein MVVM Beispiel zu erstellen welches zwar korrekt aufgebaut ist jedoch völlig sinnfrei ist. Alle implementierten Funktionen müssen Funktionieren und einen Sinn ergeben und nicht einfach wieder ein MVVM Beispiel abgeben welches im Grunde für nichts da ist und keinen Sinn ergibt. Davon gibt es bereits genug im Netz.

2.2 Ziele und Nutzen des Anwenders

Ein Funktionierendes Beispiel für eine Anwendung zur Verwaltung von Lagerplätzen und als OpenSource Beispiel zum Lernen.

2.3 Benutzer / Zielgruppe

Programmierer

2.4 Systemvoraussetzungen

Es sollte alle gängigen Rechner ab Windows 10 32Bit und 64 Bit unterstützt werden.

2.5 Ressourcen

https://materialdesignicons.com/ - XAML Icons
programmableweb.com/category/barcodes/api – Liste von Barcode APIs

3 Beschreibung der Anforderungen

Schnell und einfach zu bedienen, einfache Ersteinrichtung und der Schwerpunkt auf Usability.

Grundfunktionen:

- Verwaltung von Lagern, Sublagern, Lagerplätze
- Artikelverwaltung verschiedender Typen mit Lagerplatzzurodnung
- Attributverwaltung für Artikel (Ablaufdatum, Abmessung, Farbe usw.)
- Schnittstellen für Artikelverwaltung als Pluginsystem (z.b. Barcodeabruf über API)
- Notificationsystem (unterschreiten von Mindestlagermänge, Ablaufdatum usw)
- Listengenerierung (Einkaufsliste, Obsoletliste)
- Scanfunktion im Pluginsystem (Barcodescanner, NFC Scanner, QR Code Scanner)
- Reportingfunktionen (Übersichten, Einkaufszettel, History)

Unterstützte Endgeräte:

- Desktop WPF

3.1 Anforderung 1

Nr. / ID	HS_01_01	**Nichttechnischer Titel**	Datenpersistierung		
Quelle	Datenbank	**Verweise**	Punkt 3 - Anforderungen	**Priorität**	1

3.1.1 Beschreibung

Es soll EF Core zum Einsatz kommen wobei darauf geachtet werden muss das der Enduser sich für eine DB entscheiden können muss. MySQL, MS Sql, localDB usw.
Der DAL welcher eine Referenz zu EF Core besitzt soll tauschbar sein, darf also EF Core oder dessen Context nicht nach Außen reichen soll aber dennoch über UnitTests testbar sein.
Nur wenn keine EF Core Referenzen nach Außen gereicht werden ist es möglich die Datenzugriffschicht theoretisch jederzeit auszutauschen.

Dies wird im UnitOfWork Stil implementiert wodurch eine nahezu perfekte Abstrahierung erzielt wird.

3.1.2 Wechselwirkungen

Durch den Einsatz einer im UnitOfWork Stil aufgebauten Datenzugriffsschicht wird allerdings ein wenig von der Flexibilität von EF Core verlorengehen, jedoch nur minimal was vernachlässigt werden kann. Wichtig ist nur das Funktionen wie ChangeTracking, EagerLoading und andere Techniken von EF noch korrekt über mehrere Layer hinweg funktionieren sollten.

3.1.3 Risiken

Wenn das ChangeTracking-System von EF genutzt wird sollte ein zukünftiger DAL dies anschließend auch unterstützen da sonst diese die BusinessLogic nicht mehr korrekt arbeiten könnte, was durch den Einsatz von SelfTracking Entitys erreicht werden kann und nur auf Modelebene implementiert werden müsste.

3.1.4 Vergleich mit bestehenden Lösungen

Bis EF 4 gab es Self Tracking Entitys vom MS.

3.1.5 Grobschätzung des Aufwands

Der Aufwand ist auf jeden Fall gerechtfertigt da keine anderen Layer mehr einen Verweis auf EF haben müssen. Erstimplementierungsaufwand ca. 4 Stunden.

4 Freigabe / Genehmigung

Ausstehend!

5 Anhang / Ressourcen

<Ihr Text>