Erweitern einer Webanwendung, um automatische Produktinformations-pflegeprozesse auf Basis von Massendaten-Dateien

Prüfungsbewerber

Ralf Alexander Butz

Bergweg 13
73492 Rainau

Identnummer: 501435

E-Mail:
alexander.butz@plan-b-gmbh.com

Telefon: +49 176 23893130

Projektbetreuer

Felix Rohmeier

E-Mail:
felix.rohmeier@plan-b-gmbh.com

Telefon: +49 1515 2602 815

Ausbildungsbetrieb

PlanB. GmbH

Kocherstraße 15
73460 Hüttlingen

Inhalt

1. Einleitung ... 3
1.1 Projektumfeld ... 3
1.2 Projektbegründung ... 4
1.3 Projektabgrenzung ... 4
1.4 Projektziel ... 5
1.5 Ansprechpartner ... 5
2. Projektplanung ... 6
2.1 Projektphasen und Ablaufplan ... 6
2.2 Ressourcenplanung ... 6
2.3 Pflichtenheft ... 8
2.3.1 Prozessschritte ... 8
2.3.2 Ist- / Soll-Analyse ... 9
2.3.3 Musskriterien ... 10
2.3.4 Wunschkriterien ... 11
2.4 Vorbereitungsphase ... 12
2.4.1 Kick-off-Meeting ... 12
2.4.2 Auswahl der Azure Services ... 12
3. Durchführung ... 13
3.1 Implementierungsphase ... 13
3.1.1 Servicebereitstellung ... 13
3.1.2 Implementierung des Backends ... 14
3.1.3 Implementierung des Frontends ... 26
3.2 Logging mit Hilfe von Application Insights ... 32
3.2.1 Application Insights ... 32
3.3 Qualitätssicherung ... 34
3.3.1 Testen ... 34
4. Projektergebnis ... 35
4.1 Zeitlicher Rahmen ... 35
4.2 Soll- /Ist-Vergleich ... 35
4.3 Abnahme ... 36
4.4 Ausblick ... 36
5. Benutzerhandbuch ... 37
5.1 Vorwort ... 37

5.2 Navigation 37
5.3 Übersichtsseite der Produkte 37
5.4 Starten der Produktinformationspflegeprozesse 38
6. Tabellenverzeichnis 40
7. Abbildungsverzeichnis 40
8. Glossar 41
9. Anlagen 45

1. Einleitung

1.1 Projektumfeld

Die PlanB. GmbH ist ein IT-Dienstleister der individuelle Beratungs- und Implementierungsdienstleistungen für unterschiedlichste Infrastrukturlösungen und Anwendungen anbietet.

Als mittelständisches Unternehmen mit über 130 Mitarbeitern, über drei Standorte deutschlandweit verteilt, betreut die PlanB. GmbH Unternehmen weltweit. Als Teil der technologieführenden Anbieter für Produkte von Microsoft sind wir in der Lage unseren Kunden wichtige technologische Neuerungen frühzeitig bereits in der Planungsphase näher zu bringen und diese in produktiven Projekten miteinfließen zu lassen. Wir bieten all unseren Kunden die Möglichkeit einer langfristigen Zusammenarbeit mit einem strategisch orientierten und fokussierten Handlungspartner, daher begleiten wir Lösungen meist für deren gesamten Lebenszyklus.

Abbildung 1: Bereiche der PlanB. GmbH

Das vorliegende Portal dient der Verwaltung des „Mixed Reality (siehe Glossar) Cleaning System“ (im Folgenden „MRCS“ genannt). Innerhalb dieses Systems gibt es eine mobile Anwendung für iOS und Android (im Folgenden „MR-App“ genannt) über die es möglich ist, durch das Kamerabild des Geräts, Bild und Rauminformationen zu erkennen und zu analysieren. Auf erkannten realen Objekten (z.B. Tisch, Stuhl, Boden) können dann virtuelle Objekte (z.B. Säulen, Schilder oder, speziell für den Kinder-Modus, Dracheneier) gelegt und eine Nachbildung vom Raum in der App hinterlegt werden. Somit können Eltern mit Hilfe der Anwendung Aufgaben für die Kinder hinterlegen wie zum Beispiel „Tisch abwischen“ oder „Treppe fegen“. Den Kindern werden diese Aufgaben dann visuell dargestellt, welche dann für Belohnungen ausgeführt werden können.

1.2 Projektbegründung

Der Kunde vertreibt bereits mehrere Produkte innerhalb anderer Systeme. Für das MRCS, das erst in den letzten Jahren entstanden ist, gibt es bisher noch keine Möglichkeit Produkte zu verwalten. Da nun vom Kunden mehrere physische Produkte, wie Wischmopps Reinigungstücher und Eimer, entwickelt wurden, die speziell für das MRCS angepasst sind, sollen diese auch im vorhandenen Portal verwaltet und in der Datenbank gespeichert werden. Die MR-App unterstützt außerdem den Kauf der neuen Produkte. Die genauen geforderten Funktionalitäten sind unter 1.3 Projektabgrenzung ausführlich aufgeführt.

1.3 Projektabgrenzung

Im vorhandenen Portal können bisher nur Benutzer, Informationspunkte und Organisationen verwaltet werden.

Für die Verwaltung der Produkte werden die folgenden Funktionalitäten gefordert:

- Produktinformationen müssen innerhalb des Portals übersichtlich angezeigt werden
- Produktinformationen müssen über eine CSV-Datei (siehe Glossar) dem System hinzugefügt werden können
- Produktinformationen müssen über eine CSV-Datei innerhalb des Systems aktualisiert werden können
- Produktinformationen müssen über eine CSV-Datei aus dem System gelöscht werden können

Die Erweiterung des Managementportals kann wie im Pflichtenheft (siehe Prozessschritte) beschrieben umgesetzt werden. Die Beschreibung der Planung sowie der Durchführung richtet sich an Administratoren mit dem Ziel das nötige Wissen zu vermitteln, die vorliegende Lösung langfristig zu verwalten, zu überwachen und erweitern zu können.

1.4 Projektziel

Für die Umsetzung der geforderten Funktionalitäten, werden Vorgänge die Produktinformationen erstellen, verändern oder löschen sollen in zwei Prozesse aufgeteilt. Über den ersten Prozess sollen Daten entweder erstellt oder aktualisiert werden, abhängig davon, ob das Produkt bereits vorhanden ist oder nicht. Dieser Prozess wird im Folgenden als „Create/Update-Operation“ bezeichnet. Der zweite Prozess, im Folgenden „Delete-Operation“ genannt, soll sämtliche Produkte löschen, mit denen der Vorgang gestartet wird. Gestartet werden beide Prozesse über weitere Seiten im Portal. Für das Anzeigen der Produktinformationen im Portal wird eine Seite geschaffen, welche eine Tabelle mit den wichtigsten Werten der Produkte anzeigt. Innerhalb der Tabelle soll nach vorhandenen Produkten gesucht und nach allen Spalten auf- und absteigend sortiert werden können.

1.5 Ansprechpartner

Der direkte Ansprechpartner auf Seiten der PlanB. GmbH für das Managementportal ist Herr Felix Rohmeier. Er ist Teil des MRCS – Teams und hat bereits den ersten Entwurf des Managementportals betreut und abgenommen. Herr Rohmeier übernimmt die Rolle des „Product Owner“ (siehe Glossar), daher ist er verantwortlich für die korrekte Umsetzung und anschließende Abnahme des Projekts.

2. Projektplanung

2.1 Projektphasen und Ablaufplan

Der Erweiterung des Portals soll innerhalb von 9 Tagen mit einem Aufwand von 8 Stunden pro Tag umgesetzt werden, der letzte Arbeitstag des Projekts beschränkt sich dabei auf 6 Stunden. Die Umsetzung wird vom Product Owner begleitet, welcher den Kunden innerhalb eines Projekts repräsentiert und dementsprechend Zwischenstände in regelmäßigen Abständen abnimmt. Die Projektphasen sind wie folgt in einzelne Tätigkeiten gegliedert und ergeben einen Gesamtaufwand von 70 Stunden.

Phase	Tätigkeit	Aufwand in Stunden
Vorbereitung	Ist-/Soll-Analyse	2
	Pflichtenheft und Projektplan	2
	Auswahl der Azure Services für Verarbeitung	3
Implementierung	Servicebereitstellung	3
	Backend	26
	Frontend	8
	Upgrade .Net Core Version	4
Qualitätssicherung	Tests	4
Abnahme	Besprechung und Abnahme	2
	Erstellung der Dokumentation	16
Summe		70

Tabelle 1: Projektphasen

2.2 Ressourcenplanung

Für die Umsetzung der Erweiterung sowie der Planung dessen werden folgende Ressourcen benötigt. Zu beachten sind die Personalplanung, benötigte Arbeitsmittel und die in Azure (siehe Glossar) benötigten Ressourcen.

Arbeitsmittel:

Arbeitsmittel	Anzahl
Laptop	1
Arbeitsplatz	1
Internetanschluss	1
Azure Account	1
Visual Studio Community	1

Tabelle 2: Arbeitsmittel

Personalkosten

Name, Vorname	Position	Stundenlohn	Zeitaufwand in Stunden	Kosten
Butz, Alexander	Entwickler	47.50€	70	3325€
Rohmeier, Felix	Product Owner	95€	4	380€
Summe				3705€

Tabelle 3: Personalkosten

Azure Ressourcen:

Azure Dienst	Anzahl
App Service Plan	2
Function App	1
App Service	1
Application Insights	1
Storage Account	1
CosmosDB	1

Tabelle 4: Azure Ressourcen

2.3 Pflichtenheft

2.3.1 Prozessschritte

Tätigkeit	Arbeitsschritt	Aufwand in Stunden
1. Vorbereitung	Analyse der vorhandenen Anwendung	2
	Erstellen des Pflichtenhefts und des Projektplans	4
	Auswahl der Azure Services für die Verarbeitung	3
2. Implementierung	**Implementierung des Backends**	
	Servicebereitstellung	3
	Erweitern des vorhandenen Portals um „Products“-Controller	4
	Erstellen einer Function App, Implementieren beider Prozesse als jeweilige Azure Function	6
	Erstellen einer geeigneten Tabelle auf der Daten	0,5
	Implementierung des Frontends	
	Erweitern des vorhandenen Layouts	1
	Erstellen aller benötigter Ansichten	6
3. Qualitätssicherung	Testen des Prozesses „Create/Update-Operation“	1,5
	Testen des Prozesses „Delete-Operation“	1,5
	Prüfen der Integrität der in der Übersicht gezeigten Daten	1
	Ausführen von Black-Box-Test anhand des gesamten Portals	3
4. Dokumentation	Erstellen der Dokumentation	19
	Erstellen eines Layouts	1
	Verfassen der Dokumentation	12
	Erstellung des Benutzerhandbuchs	3
	Finale Abnahme durch Product Owner	2
	Summe	70

Tabelle 5: Prozessschritte

2.3.2 Ist- / Soll-Analyse

Ist-Analyse:

Mit dem derzeitigen Stand des Managementportals ist es möglich Benutzer, Informationspunkte und Organisationen zu verwalten. Produktinformationen werden im Moment nur als Testdaten von Hand eingefügt, wenn diese Daten zum Testen benötigt werden. Dieser Vorgang des händischen Einpflegens von Daten hat sich in der Vergangenheit als sehr zeitintensiv und fehleranfällig erwiesen. Zudem wird das Produktangebot für das MRCS immer umfangreicher was ein Pflegen der Daten von Hand mit der Zeit noch fehleranfälliger macht.

Sollkriterien:

Es sollen Prozesse geschaffen werden, welche das Pflegen von Produktinformationen mittels eingegebener CSV-Datei ermöglichen. Im produktiven System werden diese CSV-Dateien durch einen Export aus vorhandenen Datenbanken und aus dem SAP-System (siehe Glossar) heraus erzeugt. Zur Umsetzung werden alle Möglichkeiten in zwei Bereiche unterteilt: Das Erstellen und Aktualisieren sowie das Löschen von Produkten. Im Folgenden werden diese beiden Prozesse „Create/Update-Operation“ und „Delete-Operation“ genannt.

Um die vorhandenen Produkte einsehen zu können, soll es im Portal eine Übersichtsseite für Produkte geben. Dem vorhandenen Reitermenü soll ein weiterer Reiter hinzugefügt werden, der genutzt werden kann, um auf die Übersichtsseite der Produkte zu gelangen. Fährt man mit der Maus über den Reiter soll dieser, wie die anderen Reiter, ausfahren und den Text „Products“ anzeigen. Innerhalb dieser Ansicht sollen alle verfügbaren Produkte angezeigt werden. Zur schnellen Prüfung von Daten soll innerhalb dieser Tabelle nach allen vorhandenen Spalten sortiert und gefiltert werden können. Oberhalb der Tabelle sollen zwei Buttons angezeigt werden, welche jeweils auf die Ansichten weiterleiten, die Prozesse starten können. In der Ansicht für das Erstellen bzw. Aktualisieren von Produktinformationen soll die Möglichkeit geschaffen werden, eine CSV-Datei auf dem lokalen Rechner auszuwählen und hochzuladen. Diese CSV-Datei enthält Produkte in einem fest definierten Format. Dadurch soll der erste Produktinformations-pflegeprozess gestartet werden, welche vorhandene Produkte aktualisiert und nicht vorhandene Produkte dem System hinzufügt. Das erfolgreiche Starten des Prozesses soll innerhalb der Anwendung ersichtlich sein. Über den zweiten Button in der Ansicht der Produkte soll die Möglichkeit geschaffen werden Produktinformationen zu löschen. Es soll ebenfalls eine CSV-Datei hochgeladen werden können, welche den zweiten Produktinformationspflegeprozess startet. Dieser Prozess soll Produkte aus dem System entfernen. Außerdem soll auf beiden Ansichten die Möglichkeit gegeben sein eine CSV-Datei herunterzuladen, welche das Format für die eingehenden Daten vorgibt. Diese CSV-Datei enthält nur eine Kopfzeile, um sicher zu stellen, dass eingehende Daten konstant im selben Format eingespeist und verarbeitet werden können.

2.3.3 Musskriterien

Um die Verwaltung von Produktinformationen zu ermöglichen, soll die bestehende Webapplikation erweitert werden. Hierzu soll innerhalb der Applikation eine weitere Ansicht erstellt werden, deren Design sich stark an dem der vorhandenen Ansichten für Benutzer, Informationspunkte und Organisationen orientieren soll. Auf dieser neuen Ansicht soll eine Liste der verfügbaren Produkte angezeigt werden. Diese Produktinformationen sollen bei jedem Aufruf der Seite geladen werden. Außerdem sollen diese Daten über das Hochladen einer CSV-Datei verändert werden. Der Übersichtlichkeit halber soll zwischen zwei klaren Prozessen unterschieden werden. Der erste Prozess, „Create/Update-Operation“ genannt, gleicht eingehende Produkte mit der Datenbank ab, identifiziert werden Produkte über deren eindeutige ID, welche in der CSV-Datei mitgegeben wird. Ist ein Produkt bereits vorhanden, sollen dessen Informationen aktualisiert werden. Ist ein Produkt noch nicht vorhanden soll dieses eingefügt werden. Der zweite Prozess, „Delete-Operation“ genannt, soll jedes Element, das in der CSV-Datei aufgeführt wird, in der Datenbank löschen. Um Fehlern durch Benutzer vorzubeugen, sollen diese beide Prozesse in der Anwendung selbst visuell voneinander getrennt werden. Dies soll über zwei voneinander getrennte Ansichten realisiert werden, in welchen klar ersichtlich ist, welcher Prozess gestartet werden kann. Da das bestehende Portal bereits vor zwei Jahren entwickelt wurde, soll auch die .NET Core Version von 2.0 auf 3.1 aktualisiert werden. Dies bringt ein Aktualisieren der verwendeten externen Code-Pakete mit sich. Sollte dieser Vorgang des Aktualisierens bestehende Funktionalitäten im Portal unbrauchbar machen, sind diese zu ersetzen. Diese Kriterien werden in der folgenden Tabelle übersichtlich zusammengefasst:

Anforderung
Aktualisieren der .NET Core Version
Erweitern des Portals, um Produkte einsehen zu können
Erweitern des Portals, um Produktinformationen mittels Massendaten-Dateien zu pflegen
Implementieren der logischen Prozesse „Create/Update-Operation“ und „Delete-Operation“

Tabelle 6: Musskriterien

2.3.4 Wunschkriterien

Da in der produktiven Umgebung mehrere Hunderttausend Produkte verwaltet werden sollen, ist ein dynamischer Ladevorgang der Produktinformationen wünschenswert, um bei großen Dateien mit vielen Datensätzen nicht lange auf eine Antwort der Anwendung warten zu müssen. Um Fehler frühzeitig erkennen zu können, soll außerdem eine Fehlerüberwachung im System implementiert werden, welche bestimmten Administratoren per Mail informiert sollte einer der Prozesse fehlgeschlagen sein. Um diese Fehler auf lange Zeit beobachten und auswerten zu können, sollen diese getrennt von der Datenbank des Systems gespeichert werden. Eine Auswertung sowie eine Visualisierung sind vorerst nicht vorgesehen. Diese Kriterien werden in der folgenden Tabelle übersichtlich zusammengefasst:

Anforderung
Dynamisches Laden von Produktinformationen, um lange Ladezeiten der Übersichtsseite zu umgehen
Implementieren einer Überwachungslösung mit der Möglichkeit Administratoren über Fehler automatisiert zu benachrichtigen

Tabelle 7: Wunschkriterien

2.4 Vorbereitungsphase

2.4.1 Kick-off-Meeting

In einem Meeting mit dem Product Owner Felix Rohmeier wird sichergestellt, dass alle beteiligten Personen dieselben Ziele verfolgen und dass die Anforderungen klar definiert sind. Es werden mögliche Lösungsansätze für die Verarbeitung der Daten besprochen und Zugangsdaten für die vorhandene Datenbank übergeben. Außerdem wird die aktuelle Version des Portals gemeinsam untersucht, dabei wird festgelegt, worauf beim Design der neuen Ansichten zu achten ist. Dabei wird auch der aktuelle Codestand übergeben. Zudem wurde ein Account im vorhandenen Azure B2C (siehe Glossar) eingerichtet, um das Portal testen zu können.

2.4.2 Auswahl der Azure Services

Hosten des Managementportals

Das Portal wird in einem Azure App Service (siehe Glossar) gehostet, da dies die einfachste Möglichkeit darstellt innerhalb von Azure eine Web-Applikation zu hosten und zu pflegen.

Speichern der Daten

Das bestehende System arbeitet bereits mit einer CosmosDb (siehe Glossar), dabei wird die sog. „Table API“ (siehe Glossar) der Datenbank genutzt. Diese wird um eine Tabelle für die Produktinformationen erweitert. Um sicher zu stellen, dass innerhalb der Prozesse keine Daten verloren gehen, wird jede CSV-Datei, die ins System eingespeist wurde, als erstes in einem Blob Storage (siehe Glossar) als Blob (siehe Glossar) abgelegt. Dieser eignet sich besonders gut für die vorgesehenen Prozesse, da hier automatisiert Azure Functions (siehe Glossar) angesprochen werden können, sobald ein neues Element erstellt wurde.

Verarbeiten der Produktinformationen

Das Abarbeiten der eingehenden Produktinformationen sollte außerhalb vom Portal passieren. Dies soll in einer Ressource, die zwar inaktiv nur sehr geringe Kosten verursacht, aber aktiv eine sehr lange Zeit ohne Unterbrechung arbeiten kann. Aus diesen Gründen wurden zwei Azure Functions ausgewählt, welche unter derselben Azure Function App gehostet werden. Beide können unbegrenzt lange laufen was zwingend notwendig für die Verarbeitung solch großer Datensätze ist.

Überwachung des Systems

Um langfristig einen Überblick zu bekommen, welche Fehler im System auftreten und um Metriken wie die Dauer der einzelnen Prozesse sehr einfach darstellen zu können, wird Azure Application Insights (siehe Glossar) verwendet. Auftretende Fehler werden an diese Ressource gemeldet. Diese kann dann entsprechend konfiguriert werden, um in bestimmten Fällen Benutzer zu informieren.

3. Durchführung

3.1 Implementierungsphase

3.1.1 Servicebereitstellung

Die produktive Version des Managementportals und alle Ressourcen, die dafür benötigt werden, sind bereits vorhanden. Für den ersten Entwurf der Erweiterung sollen allerdings alle Ressourcen, bis auf einige wenige, in einer getrennten Umgebung neu erstellt werden. Erst wenn sich die Erweiterung in einer ausgiebigen Testphase des Kunden bewährt hat, wird diese übernommen. Von der produktiven Version sollen das Azure B2C, für die Identifizierung von Benutzern, und die CosmosDB, für das Speichern der Daten, verwendet werden.

Dementsprechend ist die Erstellung folgender Ressourcen innerhalb der Azure Cloud nötig:

Ressource	Begründung
App Service	Hosten des Portals
Storage Account	Zwischenspeichern von CSV-Dateien
Function App	Auslagern der Logik für beide Produktinformationspflegeprozesse
Application Insights	Überwachung des Systems und Benachrichtigen der Benutzer im Falle eines Fehlers
CosmosDB	Speichern der Produktinformationen

Tabelle 8: benötigte Ressourcen

Zudem müssen einige Ressourcen zusätzlich konfiguriert werden, so sind für das Application Insights sog. „Alert-Rules“ zu erstellen, welche die Häufigkeit von bestimmten Fehlern abfragen und dann die Administratoren informieren. Für einen ersten Entwurf wird lediglich der Product Owner als Empfänger solcher Benachrichtigungen eingetragen, weitere Benutzer können aber ebenfalls hinzugefügt werden. Außerdem müssen die Function App und der App Service, der das Managementportal hostet, konfiguriert werden, sodass sie Fehler an Application Insights melden.

3.1.2 Implementierung des Backends

3.1.2.1 Implementierung der Klasse „Product“

Für die Datenverarbeitung wird die Klasse „Product“ implementiert, diese wird sowohl in den Azure Functions, als auch im Controller des Portals verwendet. Dabei implementiert die Klasse „Product“ das Interface „ITableEntity“ aus dem Azure.Data.Tables SDK (siehe Glossar). Diese SDK bietet Funktionalitäten, die den Zugriff auf Daten einer CosmosDB mit einer sog. „Table-API“ ermöglichen. Durch das Implementieren des Interfaces wird es erst möglich den vollen Funktionsumfang der Datenbank auszuschöpfen, so sind z.B. die Attribute „RowKey“ und „PartitionKey“ für die Identifizierung einzelner Elemente der Tabelle hilfreich.

```
public class Product : ITableEntity
{
    /// <summary>
    /// Modell eines Produkts, wird vom Portal als auch von
    /// Azure Functions verwendet
    /// </summary>
    /// //folgende Properties bilden den Inhalt eines Produkts ab
    4 Verweise
    public string IdentificationNumber { get; set; }
    4 Verweise
    public string ProductId { get; set; }
    4 Verweise
    public string Barcode { get; set; }
    4 Verweise
    public string Country { get; set; }
    4 Verweise
    public string Title { get; set; }
    4 Verweise
    public string Description { get; set; }
    4 Verweise
    public string TitlePictureUrl { get; set; }
    4 Verweise
    public string ProductUrl { get; set; }
    4 Verweise
    public string Category { get; set; }
    4 Verweise
    public string Type { get; set; }
    //folgende Properties werden vom Interface ITableEntity gefordert
    0 Verweise
    public string _rid { get; set; }
    0 Verweise
    public string _self { get; set; }
    0 Verweise
    public string _etag { get; set; }
    0 Verweise
    public string _attachments { get; set; }
    0 Verweise
    public int _ts { get; set; }
    3 Verweise
    public string PartitionKey { get; set; }
    3 Verweise
    public string RowKey { get; set; }
    2 Verweise
    public DateTimeOffset? Timestamp { get; set; }
    0 Verweise
    public ETag ETag { get; set; }
}
```

Abbildung 2: Produkt-Modell in Form der Klasse „Product“

3.1.2.2 Controller des Portals

Der vorliegende Code für das Managementportal, das im ASP.NET Core Framework erstellt wurde, beruht auf dem MVC-Konzept. Dieses Konzept teilt eine Anwendung in drei Bereiche auf: Model, View und Controller. Im Controller wird die Programmlogik abgelegt, Models werden genutzt, um die Datenstruktur der Anwendung darzustellen und über Views wird die grafische Oberfläche für die Interaktion mit dem Benutzer erstellt.

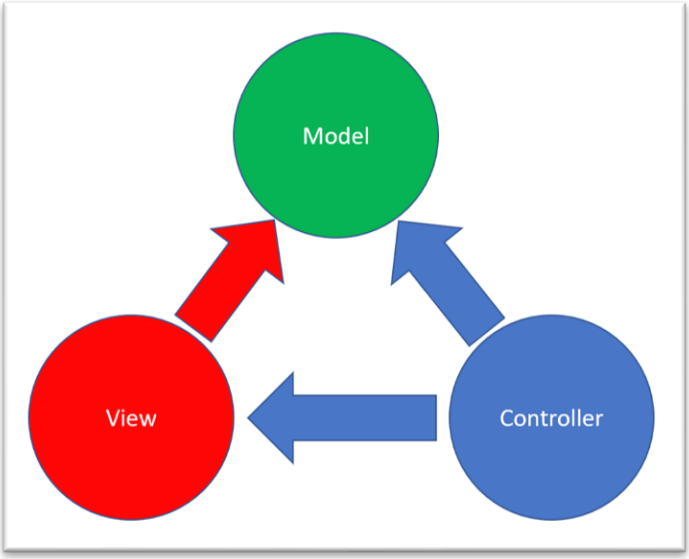

Abbildung 3: MVC-Prinzip

Da bereits, strikt nach dem MVC-Prinzip, für Benutzer, Infopunkte und Organisationen jeweils ein Controller erstellt wurde, wird für die Produkte ebenfalls ein Controller erstellt. Dieser wird die Anwendungslogik für alle Bereiche der Produkte abdecken.

Zwischenspeichern der Daten im Storage Account mit Hilfe der „Create/Update-Operation“

Wird in einer der Ansichten eine CSV-Datei hochgeladen, wird diese an den Controller übergeben. Dieser ignoriert die erste Zeile der Datei, da diese nur den Kopfdatensatz enthält, welche das Format und den jeweiligen Namen der einzelnen Spalten vorgibt. Für jede weitere Zeile wird daraufhin eine Instanz der Klasse „Product“ erstellt und einer Liste „newProducts“ von Produkten hinzugefügt. Um bereits bei der Eingabe der CSV-Datei eine Rückmeldung zu erhalten, ob die eingegebene Datei das richtige Format aufweist, wird vor dem Upload der Datensatz serialisiert. Wird eine Datei mit dem falschen Format eingegeben wird der Benutzer auf die Fehlerseite weitergeleitet.

```
public RedirectToActionResult CreateUpload()
{
    _logger.LogInformation($"Portal|Creation|started batch create/update operation");
    //lege Liste von Produkten an
    List<Product> newProducts = new List<Product>();
    //boolsche Variable um Kopfzeile der CSV zu ignorieren
    bool ignore = true;
    //prüfe ob Datei ausgewählt wurde, falls nicht gebe Fehler-Seite zurück
    if (HttpContext.Request.Form.Files[0] != null)
    {
        var uploadedFile = HttpContext.Request.Form.Files[0];
        //öffne Stream gegen Datei um zu lesen
        using (StreamReader sr = new StreamReader(uploadedFile.OpenReadStream()))
        {
            //solange Ende des Streams noch nicht erreicht wurde
            while (!sr.EndOfStream)
            {
                //erstelle Array von Strings, lese eine Zeile,
                //spalte die Zeile an allen Kommata und füge alle
                //Elemente in Array von Strings hinzu
                string[] dataRow = sr.ReadLine().Split(',');
                if (ignore == false)
                {
                    //füge der Liste von Produkten ein
                    //neues Objekt vom Typ Product hinzu
                    newProducts.Add(new Product
                    {
                        Title = dataRow[0],
                        IdentificationNumber = dataRow[1],
                        RowKey = dataRow[1],
                        PartitionKey = dataRow[1],
                        ProductId = dataRow[2],
                        ProductUrl = dataRow[3],
                        Barcode = dataRow[4],
                        Category = dataRow[5],
                        Country = dataRow[6],
                        Description = dataRow[7],
                        TitlePictureUrl = dataRow[8],
                        Type = dataRow[9]
                    });
                }
                ignore = false;
```

Abbildung 4: Formatieren der eingehenden CSV-Datei für die weitere Verarbeitung

Da es sich um eine CSV-Datei handelt wird die Kopfzeile der Datei, die erste Zeile ignoriert, da diese nur die Kopfzeile der einzelnen Spalten abbildet. Anschließend wird für jede Zeile ein Objekt vom Typ „Product“ erstellt, welches als Datenmodell über das gesamte System hinweg verwendet wird. Außerdem wird gleich zu Beginn der Function eine Nachricht im Application Insights gespeichert, mit einem Format, das es später sehr einfach macht nach bestimmten Ereignissen zu filtern (siehe 3.2 „Logging mit Hilfe von Application Insights“). Die Nachricht, die hierbei erzeugt wird, kann später für die Auswertung des Systems dienen, mit dieser Nachricht hat man z.B. den genauen Zeitpunkt, wann der Prozess gestartet wurde.

Im weiteren Verlauf der Funktion wird die Verbindung zum Blob Storage über einen Client hergestellt. Um Blob-Einträge eindeutig identifizieren zu können, wird eine eindeutige ID erstellt und an den Dateipfad des Blobs innerhalb des Blob Storages angehängt. Um Fehler zu vermeiden, werden Daten, die aus dem „Create/Update-Operation“-Prozess entstanden sind, im Container (siehe Glossar) „products-create“ abgelegt, dies geschieht getrennt von Daten aus dem zweiten Prozess. An dieser Stelle, jeweils für den erfolgreichen Upload zum Blob Storage und für den Fall, dass das Hochladen nicht erfolgreich war, werden ebenfalls Logs (siehe Glossar) in Application Insights geschrieben. Sollte der Upload nicht erfolgreich gewesen sein, wird die Fehlermeldung mit an die Log-Nachricht angefügt, die im Application Insights gespeichert wird.

```
BlobServiceClient blobServiceClient = new BlobServiceClient(blobConnectionString);
string guid = Convert.ToString(Guid.NewGuid());
string containerName = "products-create";
BlobContainerClient containerClient = blobServiceClient.GetBlobContainerClient(containerName);
//initialisiere BlobClient für neuen Blob
BlobClient blobClient = containerClient.GetBlobClient($"{guid}");
//serialisiere Liste von Produkten als JSON um dieses hochladen zu können
var productsAsJson = System.Text.Json.JsonSerializer.Serialize(newProducts);
var memStream = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(productsAsJson));
try
{
    //lade Blob mit neuer GUID hoch
    var result = blobClient.Upload(memStream);
    _logger.LogInformation($"Portal|Creation|successfully completed batch create/update operation");
}
catch(Exception ex)
{
    _logger.LogError($"Portal|Creation|unable to start batch creation, error: {ex.Message}");
}
//leite Controller an die Action "RedirectAfterOperation" weiter, Weiterleitung an Übersichtsseite weiter
return RedirectToAction("RedirectAfterBatchOperation", "Product");
```

Abbildung 5: Hochladen der Produktinformationen als Blob in den Blob Storage

War das Starten des Prozesses erfolgreich, wird an eine Funktion weitergeleitet, welche den Benutzer auf eine View leitet, die dem Benutzer bestätigt, dass der Prozess erfolgreich gestartet wurde.

```
/// <summary>
/// leite weiter an Ansicht zur Bestätigung
/// dass der Prozess gestartet wurde
/// </summary>
/// <returns>Ansicht "AfterBatchView"</returns>
0 Verweise
public IActionResult RedirectAfterOperation()
{
    return View("AfterBatchView");
}
```

Abbildung 6: Weiterleitung an entsprechende View, wenn das Starten des Prozesses erfolgreich war

Zwischenspeichern der Daten im Storage Account: „Delete-Operation“

Die Funktion für das Starten des Prozesses „Delete-Operation“ entspricht der Funktion für das Starten der „Create/Update-Operation“. Einzig die Nachrichten, die an das Application Insights gesendet werden, unterscheiden sich:

```
public RedirectToActionResult DeleteUpload()
{
    _logger.LogInformation($"Portal|Deletion|started batch delete operation");
    //lege Liste von Produkten an
    List<Product> deleteProducts = new List<Product>();
    //boolsche Variable um Kopfzeile der CSV zu ingorieren
    bool ignore = true;
    //prüfe ob Datei ausgewählt wurde, falls nicht gebe Fehler-Seite zurück
    if (HttpContext.Request.Form.Files[0] != null)
    {
        var uploadedFile = HttpContext.Request.Form.Files[0];
        //öffne Stream gegen Datei um zu lesen
        using (StreamReader sr = new StreamReader(uploadedFile.OpenReadStream()))
        {
            //solange Ende des Stream noch nicht erreicht
            while (!sr.EndOfStream)
            {
                //erstelle Array von Strings, lese eine Zeile,
                //spalte die Zeile an allen Kommata und füge alle
                //Elemente in Array von Strings hinzu
                string[] dataRow = sr.ReadLine().Split(',');
                if (ignore == false)
                {
                    //füge der Liste von Produkten ein
                    //neues Objekt vom Typ Product hinzu
                    deleteProducts.Add(new Product
                    {
                        Title = dataRow[0],
                        IdentificationNumber = dataRow[1],
                        RowKey = dataRow[1],
                        PartitionKey = dataRow[1],
                        ProductId = dataRow[2],
                        ProductUrl = dataRow[3],
                        Barcode = dataRow[4],
                        Category = dataRow[5],
                        Country = dataRow[6],
                        Description = dataRow[7],
                        TitlePictureUrl = dataRow[8],
                        Type = dataRow[9]
                    });
                }
                ignore = false;
```

Abbildung 7: Funktion der „Delete-Operation“

Wie für die Log-Nachrichten der Funktion „CreateUpload“ haben die der „DeleteUpload“ ebenfalls ein striktes Format, dieses wird genauer unter Punkt 3.2 „Logging mit Hilfe von Application Insights“, erklärt.

```
string containerName = "products-delete";
BlobContainerClient containerClient = blobServiceClient.GetBlobContainerClient(containerName);
//initialisiere BlobClient für neuen Blob
BlobClient blobClient = containerClient.GetBlobClient(guid);
//serialisiere Liste von Produkten als JSON um dieses als ByteStream hochladen zu können
var productsAsJson = System.Text.Json.JsonSerializer.Serialize(deleteProducts);
var memStream = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(productsAsJson));
try
{
    //erstelle Blob unter neuer GUID
    var result = blobClient.Upload(memStream);
    _logger.LogInformation($"Portal|Deletion|successfully started batch deletion operation");
}
catch (Exception ex)
{
    _logger.LogError($"Portal|Deletion|unable to start batch deletion operation, error message: {ex.Message}");
}
return RedirectToAction("RedirectAfterBatchOperation", "Product");
```

Abbildung 8: Loggen von Fehlern beim Erstellen des Blobs von Produktinformationen

Produktinformationen lesen

Um dem Benutzer die Produkte in der Datenbank anzuzeigen, wurde folgende Methode im Controller implementiert. Diese wird bei jedem Aufrufen der Übersichtsseite der Produkte ausgelöst. Dabei wird eine Abfrage mit einem leeren Filter ausgeführt, wodurch sämtliche Einträge der Tabelle „Products“ zurückgegeben werden. Mit Hilfe einer weiteren Methode „MapEntityToProduct“ werden die Informationen in Objekte konvertiert, welche dann einer Liste hinzugefügt werden. Diese Liste wird als JSON (siehe Glossar) formatiert und an die View weitergegeben.

```
/// <summary>
/// Lese alle Produkte von der Datenbank und gebe diese als String formatiert zurück
/// </summary>
/// <returns>String einer Liste von Produkten</returns>
[HttpGet]
0 Verweise
public string GetProducts()
{
    _logger.LogInformation($"Portal|GetProducts|Starting fetching of products");
    string EndResult = "";
    Pageable<TableEntity> QueryResult;
    try
    {
        //führe eine Abfrage ohne jeden Filter aus um alle Elemente zu erhalten
        QueryResult = tableClient.Query<TableEntity>();
        //erstelle Liste von Produkten
        List<Product> ProductsResult = new List<Product>();
        //für jedes Element in der Antwort der Abfrage
        foreach (TableEntity entity in QueryResult)
        {
            //gebe eine Entity an die Methode MapEntityToProduct
            //und füge die Antwort der Liste hinzu
            ProductsResult.Add(MapEntityToProduct(entity));
            _logger.LogInformation($"Portal|GetProducts|Fetched {ProductsResult.Count} products successfully");
        }
        //Serialisiere die Liste an Objekten zu einem String
        EndResult = JsonConvert.SerializeObject(ProductsResult);
    }
    catch (Exception ex)
    {
        _logger.LogError($"Portal|GetProducts|unable to fetch products from cosmosDb, error message: {ex.Message}");
    }
    //gebe alle Produkte als String einer Liste von Produkten
    return EndResult;
```

Abbildung 9: Auslesen der Produktinformationen zum Darstellen dieser im Portal

Die Methode „MapEntityToProduct“ nimmt als Parameter ein Element vom Typ „TableEntity“ entgegen, welches aus der Bibliothek Azure.Data.Tables kommt. Dieses Objekt erlaubt es innerhalb einer Zeile, die aus einer Tabelle ausgelesen wurde, nach einer Zeichenfolge zu suchen. Da jede „TableEntity“ ebenfalls die Kopfzeile enthält können die entsprechenden Werte durch die Methode „GetString()“ wie folgt ausgelesen werden.

```
/// <summary>
/// Konvertiere eine TableEntity in ein Product
/// </summary>
/// <param name="entity"></param>
/// <returns>formatierte TableEntity als Product</returns>
1 Verweis
public Product MapEntityToProduct(TableEntity entity)
{
    //gebe ein neues Product zurück
    return new Product()
    {
        //erstelle neues Product mit den Werten der TableEntity
        //nutze Funktion der TableEntity GetString um die Properties
        //Szu identifizieren und deren Werte zu erhalten
        Barcode = entity.GetString("Barcode"),
        Category = entity.GetString("Category"),
        Country = entity.GetString("Country"),
        Description = entity.GetString("Description"),
        ProductId = entity.GetString("ProductId"),
        ProductUrl = entity.GetString("ProductUrl"),
        Timestamp = entity.GetDateTimeOffset("Timestamp"),
        Title = entity.GetString("Title"),
        TitlePictureUrl = entity.GetString("TitlePictureUrl"),
        Type = entity.GetString("Type"),
        IdentificationNumber = entity.GetString("IdentificationNumber"),
        PartitionKey = entity.GetString("IdentificationNumber"),
        RowKey = entity.GetString("IdentificationNumber")
    };
}
```

Abbildung 10: Funktion MapEntityToProduct

Implementierung des Backends: Function App

Die Function App besteht aus zwei Funktionen, welche beide durch einen BlobTrigger (siehe Glossar) ausgelöst werden. Die Funktionalität eines BlobTrigger wird im Folgenden anhand der Function „CreateProductsBlobTrigger“ erklärt.

Function „CreateProductsBlobTrigger“

Der Funktion „Run“, welche die Main-Methode einer Azure Function deklariert, wird ein Objekt vom Typ BlobTrigger übergeben, welches mit bestimmten Parametern initialisiert wird. Über den Pfad „products-create/{name}“ wird der Geltungsbereich des Triggers eingeschränkt. Hierbei besteht „name“ aus der einzigartigen Kennziffer für den anzulegenden Blob. Mit dem verwendeten Pfad wird die Azure Function also nur ausgelöst, wenn ein Element unter dem Pfad „/products-create“ erstellt wird. Der Name des neuen Elements kann dann als Variable „name“ innerhalb der Function verwendet werden. Über den Übergabeparameter „Connection“ wird ein Connection String (siehe Glossar) referenziert, welcher zuvor innerhalb der Azure Ressource konfiguriert werden muss. Über dieses Verfahren wird sichergestellt, dass Verbindungszeichenfolgen nicht als Klartext im Code beziehungsweise dem verwendeten Git-Repository (siehe Glossar) liegen. Über den Parameter „myBlob“ kann auf die Daten des erstellten Blobs zugegriffen werden. Der eingehende Stream wird komplett gelesen und dann in eine Liste des Datenmodells „Product“ serialisiert. Um zu prüfen, ob ein Produkt bereits vorhanden ist, wird in der Datenbank nach dem Produkt gefiltert woraufhin versucht wird das Produkt anhand der erhaltenen Antwort zu aktualisieren. Ist das Produkt noch nicht vorhanden liefert die Abfrage null zurück. Innerhalb der If-Abfrage wird das Produkt dann neu angelegt. Sollte beim Anlegen eines Produkts eine Ausnahme auftreten wird diese abgefangen und wie unter Punkt 32 „Logging mit Hilfe von Application Insights“ beschrieben, weiterverarbeitet.

```
public static class CreateProductsBlobTrigger
{
    /// <summary>
    /// Wird durch neuen Blob ausgelöst der unter dem
    /// Dateipfad /products-create/{neuesElement} angelegt wird
    /// Formatiert Inhalt des Blobs in Liste von Produkten
    /// Gleicht eingehende Produkte mit Datenbank ab
    /// Produkt vorhanden -> Produkt wird aktualisiert
    /// Produkt nicht vorhanden -> Produkt wird erstellt
    /// </summary>
    /// <param name="myBlob">Inhalt des Blobs als Stream</param>
    /// <param name="name">Name des hinzugefügten Blobs</param>
    /// <param name="log">Application Insights Instanz</param>
    [FunctionName("CreateUpdateProductsBlobTrigger")]
    0 Verweise
    public static void Run([BlobTrigger("products-create/{name}", Connection = "blobConnection")]
    Stream myBlob, string name, ILogger log)
    {
        //Lese Connection String für CosmosDb aus Umgebungsvariablen aus
        string cosmosDbConnectionString = Environment.GetEnvironmentVariable("cosmosDbConnectionString");
        string inputData = "";
        log.LogInformation($"CreateProductsBlobTrigger| received a request! Starting create/update operation");
        //lese eingehenden Stream bis ans Ende
        using (StreamReader sr = new StreamReader(myBlob, Encoding.UTF8))
        {
            inputData = sr.ReadToEnd();
        }
        TableEntity queryEntity;
        //Deserialisiere den String einer Liste mit Produkten in ein Liste vom Typ Product
        List<Product> incomingProducts = JsonSerializer.Deserialize<List<Product>>(inputData);
        //instanziere tableClient für Zugriff auf Datenbank, adressiere Tabelle "Products"
        var tableClient = new TableClient(cosmosDbConnectionString, "Products");
        //für jedes Produkt der eingehenden Liste von Produkten
        foreach (Product product in incomingProducts)
        {
            //try-catch um Fehler abzufangen und in Application Insights zu speichern
            try
            {
                //frage Datenbank ab, filter nach ID des Produkts
                //ist das Produkt nicht vorhanden liefert der Client null zurück
                queryEntity = tableClient.Query<TableEntity>(filter:
                TableClient.CreateQueryFilter($"PartitionKey eq {product.IdentificationNumber}")).FirstOrDefault();
                //ist das Produkt nicht vorhanden liefer der Client null zurück
                if (queryEntity == null)
                {
                    //Produkt ist nicht vorhanden und muss neu erstellt werden
                    tableClient.AddEntity(product);
                }
                else
                {
                    //Produkt ist vorhanden und kann aktualisiert werden
                    var response = tableClient.UpdateEntity(product, queryEntity.ETag, TableUpdateMode.Replace);
                }
            }
            catch (Exception ex)
            {
                //Exception ausgelöst, logge Error in Application Insights mit ID des Produkts
                log.LogError($"Func|CreateProductsBlobTrigger|unable to create/update product, " +
                    $"error: {ex.Message} for product with id: {product.IdentificationNumber}");
            }
        }
        log.LogInformation($"Func|CreateProductsBlobTrigger| finished create/update operation!");
    }
}
```

Abbildung 11: Azure Function der Create/Update-Operation

Function „DeleteProductsBlobTrigger“

Da über diesen Prozess alle Produkte, die in das System eingespeist werden, direkt gelöscht werden sollen, wird der eingehende Stream des Blobs für eine Liste des Modells „Product“ serialisiert. Die Liste wird anschließend abgearbeitet und jedes Produkt wird durch dessen Identifikationsnummer eindeutig adressiert und innerhalb der Datenbank gelöscht. Ein entsprechendes Abfangen von Fehlern speichert diese innerhalb Application Insights, um Administratoren in Echtzeit informieren zu können (siehe Punkt 3.2.1.3 „Verwendung von Application Insights am Beispiel der Azure Function App“).

```
public static class DeleteProductsBlobTrigger
{
    /// <summary>
    /// Wird durch neuen Blob ausgelöst der unter dem
    /// Dateipfad /products-delete/{neuesElement} angelegt wird
    /// Formatiert Inhalt des Blobs in Liste von Produkten
    /// Löscht eingehende Produkte in der Datenbank
    /// </summary>
    /// <param name="myBlob">Inhalt des Blobs als Stream</param>
    /// <param name="name">Name des hinzugefügten Blobs</param>
    /// <param name="log">Application Insights Instanz</param>
    [FunctionName("DeleteProductsBlobTrigger")]
    0 Verweise
    public static void Run([BlobTrigger("products-delete/{name}", Connection = "blobConnection")]
    Stream myBlob, string name, ILogger log)
    {
        //Lese Connection String für CosmosDb aus Umgebungsvariablen aus
        string cosmosDbConnectionString = Environment.GetEnvironmentVariable("cosmosDbConnectionString");
        string inputData = "";
        log.LogInformation($"DeleteProductsBlobTrigger| received a request! Starting delete operation");
        //lese eingehenden Stream bis ans Ende
        using (StreamReader sr = new StreamReader(myBlob, Encoding.UTF8))
        {
            inputData = sr.ReadToEnd();
        }
        //Deserialisiere den String einer Liste mit Produkten in eine Liste vom Typ Product
        List<Product> incomingProducts = JsonSerializer.Deserialize<List<Product>>(inputData);
        //instanziere tableClient für Zugriff auf Datenbank, adressiere Tabelle "Products"
        var tableClient = new TableClient(cosmosDbConnectionString, "Products");
        //für jedes Produkt der eingehenden Liste von Produkten
        foreach (Product product in incomingProducts)
        {
            try //try-catch um auftretende Fehler abzufangen
            {
                //lösche Produkt in Tabelle, identifiziert durch ID des Produkts
                tableClient.DeleteEntity(product.PartitionKey, product.RowKey);
            }
            catch (Exception ex)
            {
                //Exception ausgelöst, logge Error in Application Insights mit ID des Produkts
                log.LogError($"DeleteProductsBlobTrigger|unable to delete product, " +
                    $"error: {ex.Message} for product with id: {product.IdentificationNumber}");
            }
        }
        log.LogInformation($"DeleteProductsBlobTrigger| finished delete operation!");
    }
}
```

Abbildung 12: Azure Function der Delete-Operation

3.1.3 Implementierung des Frontends

Um eine lösungsorientierte Implementierung zu garantieren wurde, soweit möglich, vorhandene CSS-Klassen (siehe Glossar) wiederverwendet und neue CSS-Klassen auf Basis der vorhandenen Styling-Elemente erstellt. Insgesamt wurden dem Portal drei weitere CSHTML-Seiten hinzugefügt, die im Folgenden genauer beschrieben werden. CSHTML-Dateien sind HTML-Seiten (siehe Glossar), die zwar den Aufbau einer HTML-Seite aufweisen, gleichzeitig aber auch C#-Code enthalten können.

Layout

Jede Ansicht erhält, analog zu den vorhandenen Ansichten, das Menü über die vorhandene Layoutseite. Dieser wurde lediglich ein weiterer Reiter hinzugefügt, welcher auf den neuen Controller „Product“ und dessen Index-Methode verweist (siehe Markierung). Die Index-Methode stellt unter dem MVC-Konzept den Einstiegspunkt des Controllers dar, die Index-Seite ist also die erste Seite, die ein Controller aufruft.

```
//Implementation Navigation
<div class="mySidenav" onmouseover="moveBody()" onmouseout="expandBody()">
    <a id="Benutzer" asp-area="" asp-controller="Home" asp-action="Index">
        <i class="glyphicon glyphicon-user"></i>
    </a>
    <a id="Points" asp-area="" asp-controller="InfoPunkt" asp-action="Index">
        <i class="glyphicon glyphicon-record"></i>
    </a>
    <a id="Organisations" asp-area="" asp-controller="Organisations" asp-action="Index">
        <i class="glyphicon glyphicon-list-alt"></i>
    </a>
    <a id="Product" asp-area="" asp-controller="Product" asp-action="Index">
        <i class="glyphicon glyphicon-play-circle"></i>
    </a>
    <a id="Logout" asp-area="AzureAD" asp-controller="Account" asp-action="SignOut">
        <i class="glyphicon glyphicon-log-out"></i>
    </a>
</div>
```

Abbildung 13: Erweiterung Frontend Reitermenü

Products-Index-Ansicht

Die Index-Seite der Produkte besteht aus einer Tabelle, die die wichtigsten Informationen des Modells „Product“ abdeckt. Außerdem gibt es zwei Buttons, welche genutzt werden, um auf die Ansichten für das Starten des jeweiligen Prozesses zu kommen. Um der Tabelle, auf möglichst einfachem Wege, die Funktionalität einer Suchleiste sowie dem Filtern nach vorhandenen Spalten zu ermöglichen wurde das Plug-In „DataTables“ verwendet.

```
@*Tabelle für das Anzeigen der Produkte*@
<div class="panel panel-primary" id="TableStyle">
    <div class="panel-body">
        <div style="margin-bottom: 10px">
            <input type="button" value="Create/Update-Operation" class="ActionButtons" onclick="CreateBatch()" />
            <input type="button" value="Delete-Operation" class="ActionButtons" onclick="DeleteBatch()" />
        </div>
        <table id="table_products_id" class="table table-striped table-bordered table-hover responsive" width="100%">
            <thead class="thin-border-bottom">
                <tr>
                    <th>ID</th>
                    <th>ProductId</th>
                    <th>Country</th>
                    <th>Title</th>
                    <th>Description</th>
                    <th>ProductUrl</th>
                    <th>Category</th>
                    <th>Type</th>
                </tr>
            </thead>
        </table>
    </div>
</div>
```

Abbildung 14: Erweiterung Frontend Übersichtsseite Produkte

Gefüllt wird diese Tabelle bei jedem Laden der Index-Seite. Dabei wird ein GET-Request (siehe Glossar„CRUD-Operationen“) an den Controller „Product“ gesendet welcher die Methode „GetProducts()“ abruft. Wie diese Methode arbeitet, wird genauer unter Punkt 3.1.2.2 „Controller des Portals“ beschrieben.

```
@section scripts{
    <script type="text/javascript" src="~/datatables.net/js/jquery.dataTables.min.js"></script>
    <script>
        $(document).ready(function () {
            $.ajax({
                type: "GET",
                url: "Product/GetProducts",
                contentType: "application/json;charset=utf-8",
                dataType: "json",
                success: function (result) {
                    var JsonResult = result;
                    var UserTable = $("#table_id").dataTable({
                        "processing": false,
                        "ordering": true,
                        "paging": true,
                        "data": JsonResult,
                        "columns": [
                            { "data": "IdentificationNumber" },
                            { "data": "ProductId" },
                            { "data": "Country" },
                            { "data": "Title" },
                            { "data": "Description" },
                            { "data": "ProductUrl" },
                            { "data": "Category" },
                            { "data": "Type" }
                        ],
                    });
                },
                error: function (response) {
                    alert('error');
                }
```

Abbildung 15: Laden von Produktinformationen auf Übersichtsseite

Create/Update-Ansicht / Delete-Ansicht

Für die Ansichten beider Prozesse werden lediglich drei Buttons erstellt. Über den ersten Button kann eine CSV-Datei mit dem vorgegebenen Format heruntergeladen werden. Der zweite Button bietet die Möglichkeit eine Datei auf dem lokalen Rechner auszuwählen und hochzuladen. Mit Hilfe des dritten Buttons wird die CSV-Datei an den Controller übergeben. Im Falle der Create/Update-Ansicht wird die CSV-Datei an die Methode „CreateUpload“ übergeben.

```
<h2 style="text-decoration: underline #FF6600">Create/Update-Operation</h2>
@using (Html.BeginForm("CreateUpload", "Product", FormMethod.Post, new { enctype = "multipart/form-data" }))
{
    <br />
    <br />
    <div style="margin-top: 200px; margin-right: 20px;">
        <div style="margin-right: 50px;" class="panel panel-primary batch-sable-style">
            <div class="panel-body">
                <div>
                    <span>
                        Please make sure that you have chosen a .csv with the right formatting.
                        <br />
                        Also note that any product that is inserted into the system in a .csv will be processed without being checked.
                        So be sure if you want to create/update or delete products and check if you are on the right page to do so.
                    </span>
                    <br />
                    <span style="font-weight: bold;">
                        This page is used only to create or update existing products!
                    </span>
                </div>
            </div>
        </div>
    </div>

    <div style="margin-block: 20px;">
        @Html.TextBox("file", "", new { type = "file" })<br />
        <input type="submit" value="Start Create/Update-Operation" data-buttonText="some text look" />

        @ViewBag.Message
    </div>

    <div style="margin-block: 20px;">
        <a style="color: black" type="submit" href="@Url.Action("DownloadTemplatCSV","Product")">
            <button type="button">Download template file</button>
        </a>
    </div>
}
<br />
```

Abbildung 16: Create/Update-Ansicht

Die Ansicht zum Starten des „Delete-Operation“-Prozesses entspricht der Create/Update-Ansicht, mit dem Unterschied, dass darauf hingewiesen wird, dass man sich auf der Ansicht für den „Delete-Operation“-Prozess befindet. Außerdem übergibt der Button die CSV-Datei an die Methode „DeleteUpload“ des Controllers für die Produkte.

```
<h2 style="text-decoration: underline #FF6600">Delete-Operation</h2>
@using (Html.BeginForm("DeleteUpload", "Product", FormMethod.Post, new { enctype = "multipart/form-data" }))
{
    <br />
    <br />
    <div style="margin-top: 200px; margin-right: 20px;">
        <div style="margin-right: 50px;" class="panel panel-primary batch-sable-style">
            <div class="panel-body">
                <div>
                    <span>
                        Please make sure that you have chosen a .csv with the right formatting.
                        <br />
                        Also note that any product that is inserted into the system in a .csv will be processed without being checked.
                        So be sure if you want to create/update or delete products and check if you are on the right page to do so.
                    </span>
                    <br />
                    <span style="font-weight: bold;">
                        This page is used only to delete existing products!
                    </span>
                </div>
            </div>
        </div>
    </div>

    <div style="margin-block: 20px;">
        @Html.TextBox("file", "", new { type = "file" })<br />
        <input type="submit" value="Upload product information" data-buttonText="some text look" />

        @ViewBag.Message
    </div>

    <div style="margin-block: 20px;">
        <a style="color: black" type="submit" href="@Url.Action("DownloadTemplatCSV","Product")">
            <button type="button">Download template file</button>
        </a>
    </div>
```

Abbildung 17: Delete-Ansicht

AfterOperation-Ansicht

Diese Seite zeigt lediglich eine Nachricht an, die den Benutzer darüber informieren soll, dass er den Prozess erfolgreich gestartet hat. Dabei wird erwähnt, dass ein solcher Prozess, je nach Menge der Daten, die in der CSV-Datei enthalten sind, mehrere Stunden andauern kann. Über einen Button wird eine JavaScript-Funktion (siehe Glossar) aufgerufen, welche den Nutzer auf die Adresse des Index für die Produkte weiterleitet.

```
<div class="BatchTableStyle">
    <div class="panel panel-primary batch-sable-style">
        <div class="panel-body">
            <div>
                <h2>Status</h2>
            </div>
            <div>
                <span id="span-afterbatch">
                    You'r Operation is running! Depending on the payload it can take up to several hours to complete.
                    <br />
                    Use the following button to view all Products:
                </span>
                <input type="button" value="Products" class="ActionButtons" onclick="RedirectIndex()" />
            </div>
        </div>
    </div>
</div>
@section scripts{
    <script type="text/javascript" src="~/datatables.net/js/jquery.dataTables.min.js"></script>
    <script>
        function RedirectIndex() {
            window.location.href = '@Url.Action(nameof(ProductController.Index), "Product")';
        }
    </script>
}
```

Abbildung 18: AfterOperation-Ansicht

3.2 Logging mit Hilfe von Application Insights

3.2.1 Application Insights

Unter Application Insights versteht man einen erweiterbaren Dienst der Azure Cloud zur Überwachung von Anwendungen. Mit verschiedenen Überwachungs- und Analysetools wird es Entwicklern ermöglicht Metriken und Fehler von einer Vielzahl an Azure Ressourcen zu verwalten und übersichtlich in Dashboards darzustellen. Mit seinen zusätzlichen Diagnosetools soll es Nutzern der Azure Cloud erleichtert werden auftretende Fehler in ihren Azure Ressourcen zu erkennen und zu beheben.

3.2.1.1 Implementieren von Azure Application Insights

Azure Application Insights ist als Codepaket für eine Vielzahl von Plattformen verfügbar, darunter .NET, Node.js, Java und Python. Um den vollen Funktionsumfang nutzen zu können muss ein SDK der Anwendung hinzugefügt werden, welches die Kommunikation mit Application Insights ermöglicht. Über eine Konfiguration kann definiert werden, welche Tiefe der Fehlernachrichten an Application Insights gemeldet werden soll. Dabei können ebenfalls benutzerdefinierte Nachrichten erstellt werden, mit Hilfe derer man Nutzer über bestimmte Ereignisse der Anwendung informieren kann, dieser Vorgang wird unter 3.2.1.2 Protokollwarnungen genauer erklärt. Des Weiteren kann das SDK genutzt werden um Telemetriedaten, wie z.B. Anforderungsraten, Antwortzeiten, Fehlerraten, Ausnahmen und Seitenansichten, zu speichern.

3.2.1.2 Protokollwarnungen

Über Protokollwarnungen können Entwickler sämtliche Ereignisse abfragen, welche das Instrumentierungspaket an Application Insights gemeldet hat. Tritt ein bestimmtes Ereignis in einer bestimmten Häufigkeit auf, wird eine Protokollwarnung ausgelöst, welche an eine oder mehrere Aktionsgruppen gebunden ist. Über Aktionsgruppen kann dann definiert werden wie mit einer Protokollwarnung umgegangen werden soll. Innerhalb dieser Gruppen wird zwischen Benachrichtigungen und Aktionen unterschieden. Über Benachrichtigungen können Nutzer per E-Mail, SMS-Nachricht, Push-Benachrichtigung oder Sprachnachricht mit einer benutzerdefinierten Nachricht informiert werden, welche Details der ausgelösten Protokollwarnung enthalten kann. Unter Aktionen kann eine Vielzahl von Azure Ressourcen angesprochen werden, um Benachrichtigungen selbst zu senden oder mit entsprechenden Maßnahmen auf Fehler reagieren zu können.

3.2.1.3 Verwendung von Application Insights am Beispiel der Azure Function App

Innerhalb der Function App wird der Hauptteil der Prozesslogik abgearbeitet, daher eignen sich die Start- und Endzeiten der Ausführungen dieser Azure Function um dem Systemadministrator einen Überblick zu verschaffen wie lange seine Prozesse laufen.

Im Folgenden ist der Inhalt einer solchen Log-Nachricht zu sehen. Diese wurde im Azure Portal (siehe Glossar) über den Dienst Application Insights ausgelesen. Anhand der Zeile „Message“ (Markierung 1) kann man erkennen, dass diese Nachricht, die durch die Ausführung der Azure Function „DeleteProductsBlobTrigger“ darstellt, ebenfalls erkennbar ist, durch welchen neuen Blob die Function gestartet wurde und unter welchem Container diese abgelegt ist. An der Zeile „LogLevel“ (siehe Glossar) (Markierung 2) wird deutlich, unter welcher Tiefe des Loggings der Anwendung die Nachricht ausgelöst wurde. Diese Nachricht wurde also nur an das Application Insights weitergegeben, weil sie mit einem Log-Level auf der Ebene „Information“ (Stufe drei von sieben, siehe Anlage „Azure Functions Protokolliergrade und Kategorien“) erstellt wurde und die Anwendung darauf konfiguriert ist Log- Nachrichten diese Stufe und höher zu verarbeiten.

timestamp [UTC]	message	severityLevel	itemType	customDimensions
11/24/2021, 6:02:28.080 P...	Executing 'BatchDeleteProductsBlobTrigger' (Reason='New blo...	1	trace	{"prop__{OriginalFormat}":"Executing '{f

timestamp [UTC]	2021-11-24T18:02:28.080893Z
1 message	Executing 'BatchDeleteProductsBlobTrigger' (Reason='New blob detected: products-delete/fbdae527-fbef-4368-9add-8182c8f8e303' Id=2c36be44-b15
severityLevel	1
itemType	trace
customDimensions	{"prop__{OriginalFormat}":"Executing '{functionName}' (Reason='{reason}', Id={invocationId})","HostInstanceId":"f0341eef-7bee-428f-89ec-01c82ed9f33

Category	Function.BatchDeleteProductsBlobTrigger
EventId	1
EventName	FunctionStarted
HostInstanceId	f0341eef-7bee-428f-89ec-01c82ed9f33f
InvocationId	2c36be44-b151-4bb8-985b-67b04d86d535
2 LogLevel	Information
ProcessId	7876
prop__functionName	BatchDeleteProductsBlobTrigger
prop__invocationId	2c36be44-b151-4bb8-985b-67b04d86d535
prop__reason	New blob detected: products-delete/fbdae527-fbef-4368-9add-8182c8f8e303
prop__{OriginalFormat}	Executing '{functionName}' (Reason='{reason}', Id={invocationId})

operation_Name	BatchDeleteProductsBlobTrigger

Abbildung 19: Log-Nachricht ausgelöst von einer Azure Function, ausgelesen im Application Insights

Reagieren auf Warnungen

Über sog. Warnungsregeln können benutzerdefinierten Abfragen dann automatisiert ausgeführt werden. Warnungsregeln können eine oder mehrere Aktionsgruppen zugewiesen werden, welche den Inhalt der Log-Nachricht weiterverarbeiten können. Diese können entweder Benutzer per E-Mail, SMS-Nachricht, Push-Benachrichtigung oder Anruf mit einer benutzerdefinierten Nachricht benachrichtigen oder den Inhalt der Log-Nachricht an eine Vielzahl von Azure Services weiterleiten um auf die eingetretenen Ereignisse automatisiert reagieren zu können.

3.3 Qualitätssicherung

3.3.1 Testen

Um die Funktionalität der einzelnen Prozesse garantieren zu können, wurden diese nach der jeweiligen Implementierung im Einzelnen getestet. Mit Hilfe eines Black-Box-Test (siehe Glossar) wurde die Funktionalität des der Erweiterung bestätigt, indem lediglich das Verhalten der Anwendung analysiert wurde. Die vollständige Funktionalität und die entsprechend den vordefinierten Kriterien konforme Umsetzung des Projekts wurde bei der Abnahme vom Product Owner abgenommen und bestätigt.

4. Projektergebnis

4.1 Zeitlicher Rahmen

Die Muss-Kriterien des Projekts konnten im geplanten zeitlichen Rahmen und Ablauf durchgeführt werden. Die Wunsch-Kriterien des Kunden konnten aus zeitlichen Gründen nur teilweise erfüllt werden.

4.2 Soll- /Ist-Vergleich

Die Soll-Kriterien des Kunden konnten vollständig umgesetzt werden. Das bereits verwendete Portal wurde um eine Produktansicht und mehrere Ansichten zum Starten der Produktinformations-Pflegeprozesse erweitert.

In der folgenden Tabelle werden die Soll-Kriterien des Kunden aufgelistet, mit einem Vermerk, ob diese erfüllt wurden oder nicht.

Anforderung	Vollständig umgesetzt
Aktualisieren der .NET Core Version	Ja
Erweitern des Portals, um Produkte einsehen zu können	Ja
Erweitern des Portals, um Produktinformationen mittels Massendaten-Dateien zu pflegen	Ja
Implementieren der logischen Prozesse „Create/Update-Operation“ und „Delete-Operation“	Ja

Tabelle 9: Vergleich Musskriterien/ Projektergebnis

Da nach dem Implementieren der Muss-Kriterien noch einige Stunden für weitere Implementierungen zur Verfügung standen, konnten die Wunsch-Kriterien des Kunden teilweise umgesetzt werden. In der folgenden Tabelle werden die Wunsch-Kriterien des Kunden aufgelistet, mit einem Vermerk, ob diese erfüllt wurden oder nicht.

Anforderung	Vollständig umgesetzt
Dynamisches Laden von Produktinformationen, um lange Ladezeiten der Übersichtsseite zu umgehen	Nein
Implementieren einer Überwachungslösung mit der Möglichkeit Administratoren über Fehler automatisiert zu benachrichtigen	Ja

Tabelle 10: Vergleich Wunschkriterien/ Projektergebnis

Das Implementieren einer Überwachungslösung, mit dem Ziel Administratoren über auftretende Fehler automatisiert zu informieren konnte, wie unter Punkt 3.2 „Logging mit Hilfe von Application Insights“ beschrieben, umgesetzt werden. Das dynamische Laden der Produktinformationen konnte aus zeitlichen Gründen nicht umgesetzt werden.

4.3 Abnahme

Das Projekt wurde vom Product Owner Felix Rohmeier durch einen Vergleich der Soll-/und Wunschkriterien mit dem Projektergebnis, abgenommen.

4.4 Ausblick

Die Erweiterung des Portals wird vom Kunden einem ausführlichen Belastungstest mit mehreren Datei-Exporten von SAP unterzogen. Sobald sich das System bewährt hat, wird die Erweiterung des Portals sowie die benötigten zusätzlichen Azure Ressourcen in der produktiven Umgebung des Kunden eingeführt.

5. Benutzerhandbuch

5.1 Vorwort

Das folgende Benutzerhandbuch richtet sich an die Administratoren, die die aktuelle Version des Managementportals verwalten, daher werden ausschließlich die Funktionen erklärt, die mit der Erweiterung hinzugekommen sind. Es wird vorausgesetzt, dass bekannt ist wie Administratoren sich im Portal An- und Abmelden können.

5.2 Navigation

Die erste Seite nach dem Login zeigt die Übersichtsseite der Benutzer. Um die Erweiterung der Produkte nutzen zu können, muss innerhalb des Reitermenüs an der linken Seite des Bildschirms das vierte Element von oben ausgerufen werden. Fährt der Benutzer mit der Maus über die einzelnen Elemente des Reitermenüs öffnen sich diese mit dem Titel der jeweiligen Seite.

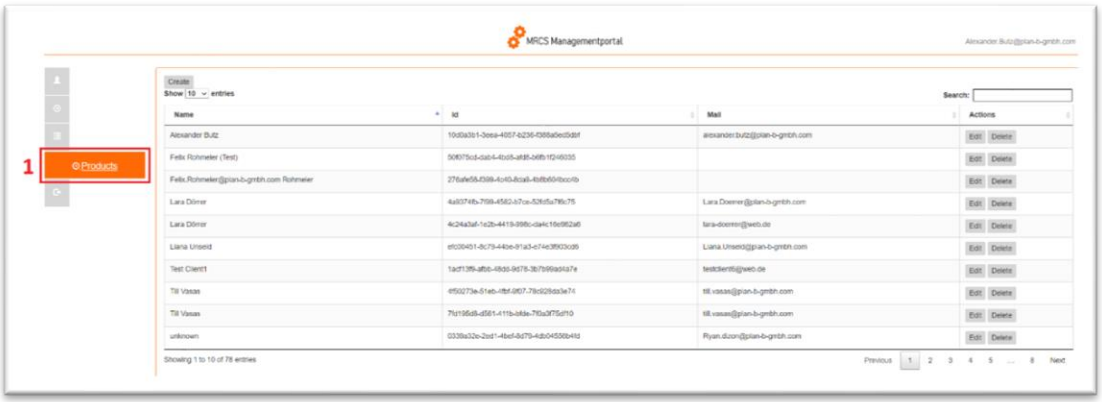

Abbildung 20: Startseite der Anwendung - Übersichtsseite der Benutzer

5.3 Übersichtsseite der Produkte

Auf der Übersichtsseite der Produkte werden die Produktinformationen in einer Tabelle dargestellt. Über der Tabelle befinden sich zwei Buttons (Markierung 1), welche auf die Ansichten zum Starten der jeweiligen Prozesse weiterleiten. Klickt man die Kopfzeile, der jeweiligen Tabellenspalte, an wird die Tabelle nach dieser Spalte erst absteigend, durch einen weiteren Klick aufsteigend, sortiert. Wird bereits nach einer Spalte sortiert wird das in der Kopfzeile mit Hilfe eines Symbols (Markierung 3) deutlich gemacht, ansonsten wird in der Kopfzeile ein Symbol angezeigt, dass deutlich machen soll, dass nach dieser Spalte sortiert werden kann (Markierung 2). Über die Suchleiste am rechten Bildschirmrand kann nach Inhalten aller Spalten gesucht werden, die Ergebnisse werden dynamisch gefiltert und nach jeder Eingabe erneut angezeigt.

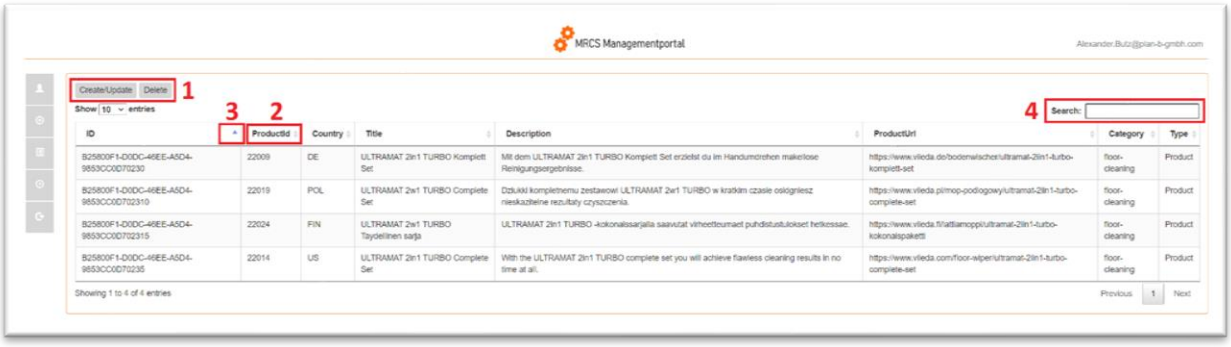

Abbildung 21: Übersichtsseite der Produkte

5.4 Starten der Produktinformationspflegeprozesse

Ansicht „Create/Update-Operation“

Die Überschrift als auch die Beschreibung (Markierung 1) der Seite sollen darauf hinweisen, welchen Produktinformationspflegeprozess der Benutzer starten kann. Über den dritten Button (Markierung 4) kann eine Vorlage in Form einer CSV-Datei heruntergeladen werden, welche die Kopfzeile, der erwarteten CSV-Datei enthält. Über den ersten Button (Markierung 2) kann eine CSV-Datei auf dem lokalen Rechner ausgewählt und hochgeladen werden. Wurde diese erfolgreich hochgeladen wird ihr Name rechts von Button 1 (Markierung 2) angezeigt. Über den zweiten Button (Markierung 3) wird der Informationspflegeprozess „Create/Update-Operation“ gestartet.

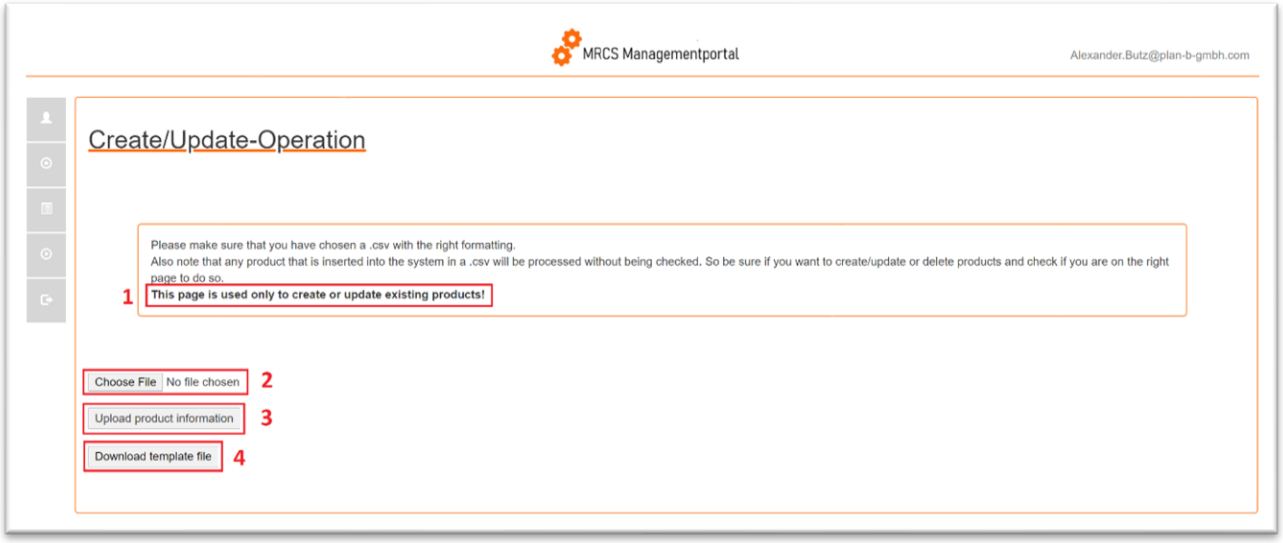

Abbildung 22: Ansicht der „Create/Update-Operation“

Ansicht „Delete-Operation“

Die Ansicht der „Delete-Operation“ ähnelt optisch stark der Ansicht der „Update/Create-Operation“, einzig der Hinweis (Markierung 1) sowie die Überschrift der Seite unterscheiden sich. Die Funktionen der Buttons sind exakt dieselben, mit dem Unterschied, dass der zweite Button (Markierung 3) den Prozess der „Delete-Operation“ startet.

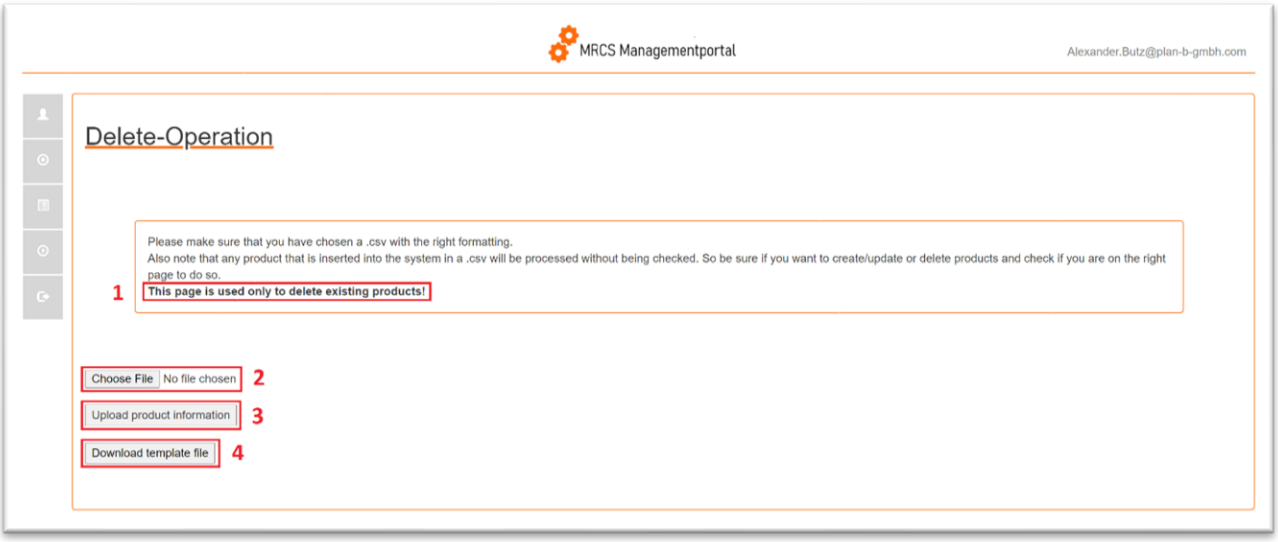

Abbildung 23: Ansicht der „Delete-Operation“

Nachdem einer der beiden Prozesse erfolgreich gestartet wurde, wird der Benutzer auf die folgende Seite weitergeleitet. Diese soll den Benutzer darüber informieren, dass der jeweilige Prozess, abhängig vom Umfang der eingegebenen CSV-Datei, mehrere Stunden andauern kann. Über den Button „Products“ (Markierung 1) bzw. über den vierten Punkt im seitlichen Reitermenü (Markierung 2) kann der Benutzer wieder auf die Übersichtsseite der Produkte gelangen.

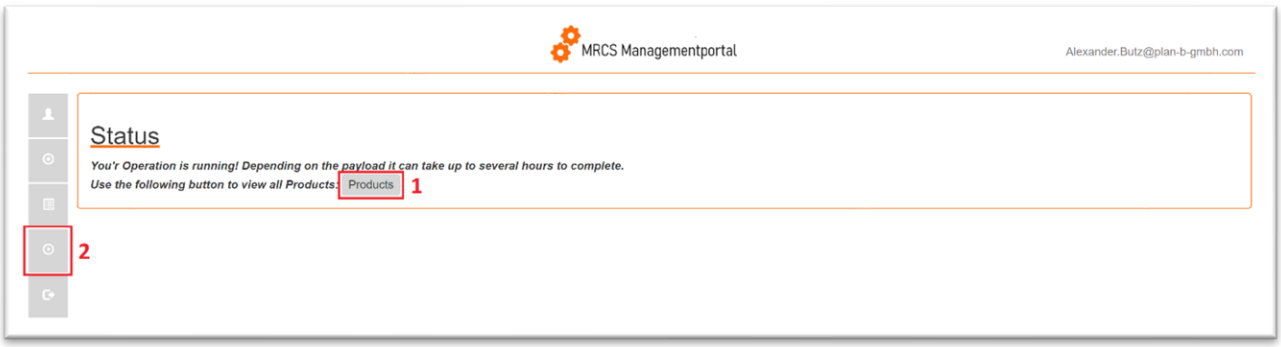

Abbildung 24: AfterOperation-Ansicht, Bestätigt dass ein Produktinformationspflegeprozess erfolgreich gestartet wurde

6. Tabellenverzeichnis

Tabelle 1: Projektphasen 6
Tabelle 2: Arbeitsmittel 6
Tabelle 3: Personalkosten 7
Tabelle 4: Azure Ressourcen 7
Tabelle 5: Prozessschritte 8
Tabelle 6: Musskriterien 10
Tabelle 7: Wunschkriterien 11
Tabelle 8: benötigte Ressourcen 13
Tabelle 9: Vergleich Musskriterien/ Projektergebnis 35
Tabelle 10: Vergleich Wunschkriterien/ Projektergebnis 35

7. Abbildungsverzeichnis

Abbildung 1: Bereiche der PlanB. GmbH 3
Abbildung 2: Produkt-Modell in Form der Klasse „Product“ 14
Abbildung 3: MVC-Prinzip 15
Abbildung 4: Formatieren der eingehenden CSV-Datei für die weitere Verarbeitung 16
Abbildung 5: Hochladen der Produktinformationen als Blob in den Blob Storage 17
Abbildung 6: Weiterleitung an entsprechende View, wenn das Starten des Prozesses erfolgreich war 18
Abbildung 7: Funktion der „Delete-Operation“ 19
Abbildung 8: Loggen von Fehlern beim Erstellen des Blobs von Produktinformationen 20
Abbildung 9: Auslesen der Produktinformationen zum Darstellen dieser im Portal 21
Abbildung 10: Funktion MapEntityToProduct 22
Abbildung 11: Azure Function der Create/Update-Operation 24
Abbildung 12: Azure Function der Delete-Operation 25
Abbildung 13: Erweiterung Frontend Reitermenü 26
Abbildung 14: Erweiterung Frontend Übersichtsseite Produkte 27
Abbildung 15: Laden von Produktinformationen auf Übersichtsseite 28
Abbildung 16: Create/Update-Ansicht 29
Abbildung 17: Delete-Ansicht 30
Abbildung 18: AfterOperation-Ansicht 31
Abbildung 19: Log-Nachricht ausgelöst von einer Azure Function, ausgelesen im Application Insights 33
Abbildung 20: Startseite der Anwendung - Übersichtsseite der Benutzer 37
Abbildung 21: Übersichtsseite der Produkte 38
Abbildung 22: Ansicht der „Create/Update-Operation“ 38
Abbildung 23: Ansicht der „Delete-Operation“ 39
Abbildung 24: AfterOperation-Ansicht, Bestätigt dass ein Produktinformationspflegeprozess erfolgreich gestartet wurde 39

8. Glossar

Application Insights

Dienst zur Überwachung von aktiven Anwendungen, native Diagnosetools bieten Unterstützung zur Fehleridentifikation und -behebung. Stellt standardmäßig Metriken verbundener Ressourcen in einer Übersichtsseite bereit. Ermöglicht außerdem das Speichern und Auswerten von Log-Nachrichten. Stellt ebenfalls Funktionalitäten bereit, die Nutzer informieren, wenn bestimmte Ereignisse in zuvor definierten Intervallen auftreten.

App Service

Ermöglicht das Hosten von Applikationen ohne Infrastruktur verwalten zu müssen. Bietet Möglichkeiten für automatische Skalierung und Hochverfügbarkeit.

Azure

Die Microsoft-Cloud. Stellt viele verschiedene Dienste weltweit bereit, um unterschiedlichste Lösungen realisieren zu können.

Azure B2C

Azure „Business to Customer“ Ermöglicht Benutzeridentifizierung für Unternehmen. Kunden können ihre bevorzugten Accounts für soziale Netzwerke, Unternehmen oder lokale Konten nutzen, um Zugriff auf Anwendungen zu erhalten.

Azure Function

Code-Funktion die von verschiedenen Triggern aufgerufen werden kann. Kann konfiguriert werden um endlos Aufgaben auszuführen.

Azure Function App

Serverlose Lösung (serverlos bedeutet, dass die Infrastruktur von Microsoft verwaltet wird), um Code in unterschiedlichen Sprachen ausführen zu lassen ohne Konfigurationen auf Hardwareebene ausführen zu müssen. Unterstützt werden nativ C#, PowerShell, Python und JavaScript/ TypeScript. Gängige Anwendungsfälle sind beispielsweise: Web-APIs und das Ausführen von geplanten Aufgaben. Eine Function App besteht aus mindestens einer Azure Functions „Function“.

Azure Function Trigger

Wird verwendet, um zu definieren wann eine Azure Function ausgelöst wird. Mögliche Auslöser sind: eingehender http-Request, zeitlich definiert (z.B. jeden Sonntag 8:00Uhr oder jeden zweiten Tag) oder ein neues Element in einem Blob Storage.

Azure Portal

Webbasierte zentrale Verwaltungsplattform für sämtliche Azure Ressourcen und Dienste.

Azure Storage Account

Verwaltet Speicher in Form von Blobs, Dateifreigaben, Warteschlangen, Tabellen und Datenträger. Meist genutzt als Blob Storage.

Black-Box-Test

Bei einem Black-Box-Test wird nur das Verhalten der Anwendung beachtet. Dabei wird aus der Sicht eines Anwenders getestet, wobei Tester nur die Benutzeroberfläche auf Fehler untersuchen sollen.

Blob

„Binary Large Objects“ sind große binäre Datenobjekte wie z.B. Bild- oder Audiodateien.

Blob Storage

Objektspeicherlösung die für große Mengen aller möglichen Dateien optimiert ist.

Connection String

Auch „Verbindungszeichenfolge“ genannt, enthält alle benötigten Informationen (wie z.B. Adresse, Zugangsweg und Zugangsdaten) für einen Dienst oder eine einzelne Ressource, um eine Verbindung damit aufbauen zu können.

Container

Container dienen zum Separieren von Blobs innerhalb eines Azure Storages.

CosmosDB

NoSQL Datenbank von Microsoft. Kann, im Gegensatz zu herkömmlichen Datenbanken, mit folgenden APIs verwendet werden: Core (SQL) API, MongoDB API, Cassandra API, Gremlin API, Table API.

CRUD-Operationen

Steht für „Create, Read, Update, Delete“. Stellt die wichtigsten vier Datenoperationen (erstellen, lesen, aktualisieren, löschen) für Anwendungen dar.

CSS

„Cascading Style Sheets“ - Gestaltungs- und Formatierungssprache. Wird verwendet, um das Aussehen von html-Dateien zu verändern.

CSV-Datei

„Comma seperated values“ – Dateityp für einfach strukturierte Daten. Üblicherweise werden einzelne Datensätze durch Kommata getrennt.

Git

Versionsverwaltungstool für Dateien. Erlaubt das Versionieren von Codeständen um den Verlauf eines Projekts zu dokumentieren und im Ernstfall auf alten Code zurückgreifen zu können. Vereinfacht außerdem das gemeinsame Arbeiten innerhalb eines Projekts.

Git-Repository

Virtueller Speicher für ein Projekt, wird durch Git bereitgestellt.

HTML

„Hypertext Markup Language“ – Wird genutzt um das Format einer Webseite zu erstellen.

JavaScript

Programmiersprache die innerhalb von Webseiten verwendet werden kann um Funktionalität hinzuzufügen.

JSON

„JavaScript Object Notation“ – standardisiertes Datenformat für die Codierung von Daten.

Logging

Automatisiertes und händisches Protokollieren von Statusinformationen und Ereignissen von Anwendungen und Systemen. Wird zur Fehler- und Angriffsanalyse genutzt.

Log-Level

Über Log-Level werden verschiedene Ebenen definiert auf denen Ausnahmen ausgelöst werden können. Je nach „Schwere“ der Ausnahme werden diese auf die Ebenen verteilt.

Logs

Datensatz an Statusinformationen die automatisiert gespeichert werden. Dient der genauen Nachverfolgung von auftretenden Fehlern und Angriffen.

Mixed Reality

Verbindet digitale und virtuelle Welt. Mixed Reality-Lösungen können beispielsweise aus einer Brille bestehen, die dem Benutzer virtuelle Objekte auf die Linse der Brille projiziert, dabei aber die tatsächliche Umgebung nicht ausblendet.

MVC

„Model-View-Controller“ – Konzept der Softwarearchitektur das Logik, Optische Elemente und Datenmodelle voneinander trennt.

NoSQL Datenbank

NoSQL bezeichnet Datenbanken die einen nicht relational aufgebaut sind. Diese benötigen keine festgelegten Tabellenschemata.

Product Owner

Ist für den Verlauf des Projekts verantwortlich. Vertritt die Vision bzw. die Ziele des Projekts und ist verantwortlich für deren Umsetzung. Dienst als Ansprechpartner für Entwickler. Wird oftmals vom Kunden gestellt.

SAP

„Systeme Anwendungen und Produkte“, bezeichnet eine weit verbreitete Software, die in der Datenverarbeitung genutzt wird, um Informationen in und unter verschiedenen Unternehmen auszutauschen

SDK

„Software Development Kit“ – stellt hilfreiche Ressourcen in Form von Code bereit zur erleichterten Verwendung bestimmter Technologien.

TablesAPI (CosmosDB)

Framework das Funktionalitäten bereitstellt, um Daten in einer CosmosDb zu verwalten.

9. Anlagen

Azure Functions Protokolliergrade und Kategorien

LogLevel	Code	BESCHREIBUNG
Trace	0	Protokolle, die die ausführlichsten Meldungen enthalten. Diese Meldungen enthalten möglicherweise sensible Anwendungsdaten. Sie sind standardmäßig deaktiviert und sollten nie in einer Produktionsumgebung aktiviert werden.
Debuggen	1	Protokolle, die für interaktive Untersuchungen während der Entwicklung verwendet werden. Diese Protokolle sollten hauptsächlich für das Debuggen hilfreiche Informationen enthalten. Sie besitzen keinen langfristigen Nutzen.
Information	2	Protokolle, die den allgemeinen Ablauf der Anwendung nachverfolgen. Diese Protokolle sollten einen langfristigen Nutzen besitzen.
Warnung	3	Dies sind Protokolle, die ein ungewöhnliches oder unerwartetes Ereignis im Anwendungsfluss hervorheben, jedoch nicht bewirken, dass die Anwendung beendet wird.
Fehler	4	Dies sind Protokolle, die hervorheben, wenn der aktuelle Ausführungsfluss aufgrund eines Fehlers beendet wird. Diese Fehler sollten auf einen Fehler im Zusammenhang mit der aktuellen Aktivität und nicht auf einen anwendungsweiten Fehler hinweisen.
Kritisch	5	Protokolle, die einen nicht behebbaren Anwendungs- oder Systemabsturz beschreiben oder einen schwerwiegenden Fehler, der unmittelbare Aufmerksamkeit erfordert.
Keine	6	Hiermit wird die Protokollierung für die angegebene Kategorie deaktiviert.

IHK Industrie- und Handelskammer Ostwürttemberg

Diese Erklärung ist beim Einreichen der Dokumentation jedem Exemplar anzuhängen!

IV. 3 Persönliche Erklärung

zur Projektarbeit und Dokumentation im Rahmen der Abschlussprüfung in den IT-Berufen

Ich versichere durch meine Unterschrift, dass ich die Durchführung der betrieblichen Projektarbeit als auch die dazugehörige Dokumentation selbständig in der vorgegebenen Zeit erarbeitet habe. Alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, wurden von mir als solche kenntlich gemacht.

Ebenso bestätige ich, dass ich bei der Erstellung der Dokumentation meiner Projektarbeit weder teilweise noch vollständig Passagen aus Projektarbeiten übernommen habe, die bei der prüfenden oder einer anderen Kammer eingereicht wurden.

Ich bestätige, dass die Dokumentation keine Betriebsgeheimnisse bzw. schutzwürdige Betriebs- oder Kundendaten enthält, das Urheberrecht beachtet wurde und es keine datenschutzrechtlichen Bedenken gibt

Ort, Datum: Hüttlingen, 01.12.2021

Unterschrift des Prüfungsteilnehmers: Butz Alexander

Ich habe die obige Erklärung zur Kenntnis genommen und bestätige, dass die betriebliche Projektarbeit einschließlich der Dokumentation in der vorgegebenen Zeit in unserem Betrieb durch den Prüfungsteilnehmer angefertigt wurde.

Ort, Datum: Hüttlingen, 01.12.2021

Unterschrift des Ausbilders: