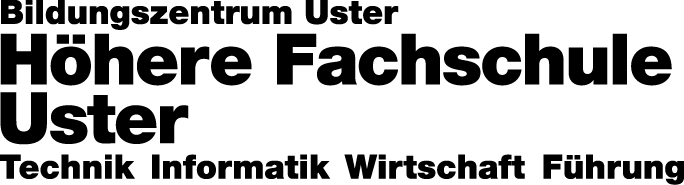
* Ein Bild, das Schrift, Grafiken, Screenshot, Grafikdesign enthält.

Automatisch generierte Beschreibung*

Projektwoche Softwareentwicklung

*Durchführung – 16.06.2025 bis 20.06.2025*

Leadtext für diesen Abend

Lehrperson

Inhalt

[Analyse 2](#_Toc201008342)

[Systemskizze 2](#_Toc201008343)

[Systemkontext 2](#_Toc201008344)

[Use Cases 3](#_Toc201008345)

[Usecase-Diagramm 3](#_Toc201008346)

[User-Stories 3](#_Toc201008347)

[Detaillierte Beschreibung Use Case Shop-API Create, Update, Delete 5](#_Toc201008348)

[Pflichtenheft 6](#_Toc201008349)

[Muss 6](#_Toc201008350)

[Soll 6](#_Toc201008351)

[Einschränkungen 6](#_Toc201008352)

[Datenmodell 7](#_Toc201008353)

[Technologie 8](#_Toc201008354)

[Datenbank 8](#_Toc201008355)

[Schnittstellen 8](#_Toc201008356)

[Service 8](#_Toc201008357)

[Glossar 10](#_Toc201008358)

[Abbildungen 10](#_Toc201008359)

[Literaturverzeichnis 10](#_Toc201008360)

# Analyse

## Systemskizze

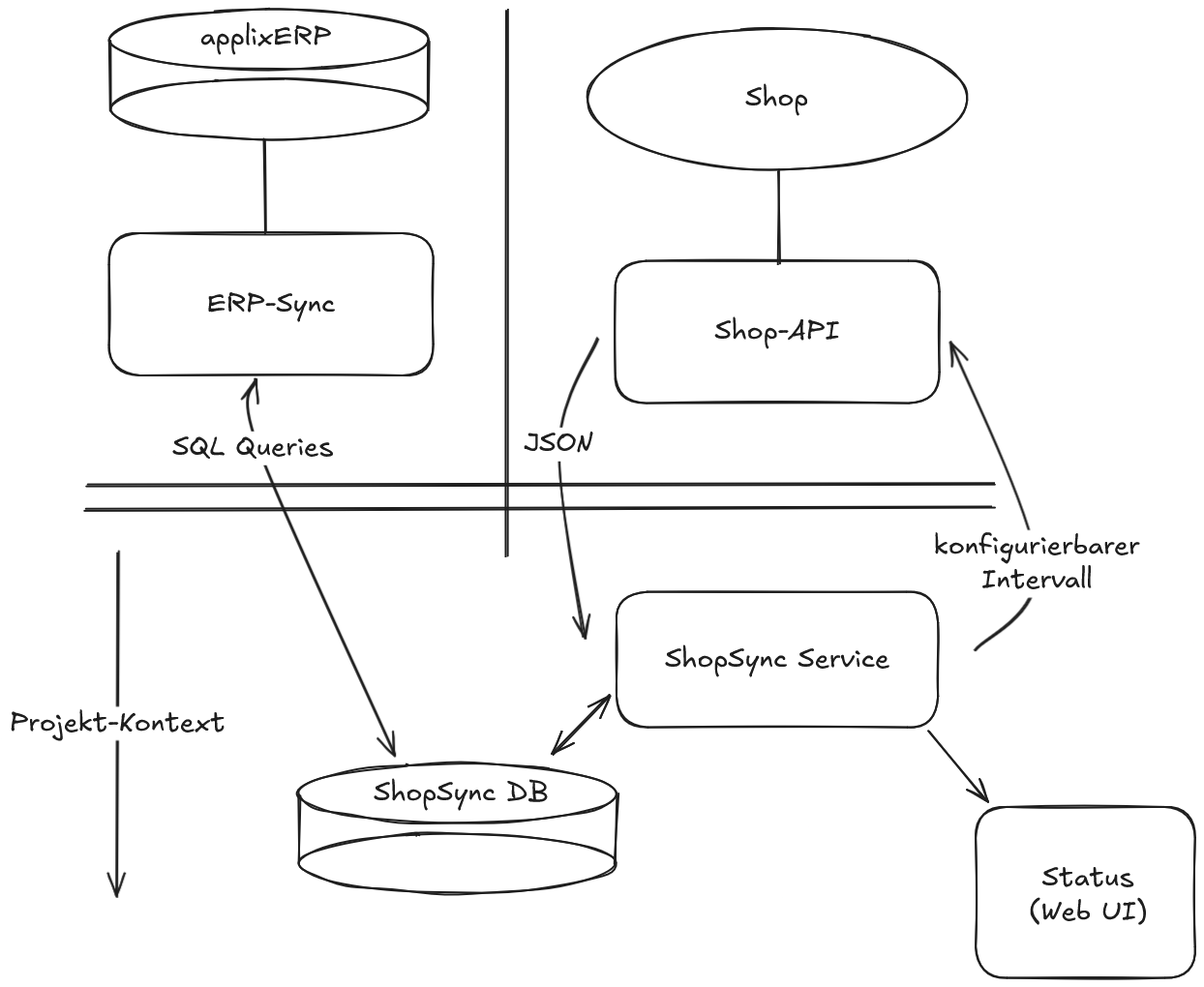


Abbildung : Systemskizze

## Systemkontext

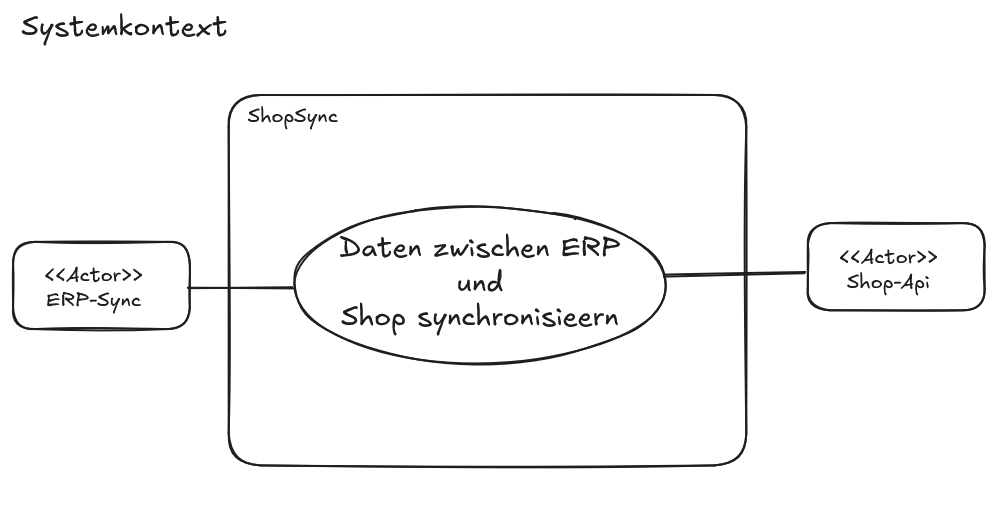


Abbildung : Systemkontext

## Use Cases

### Usecase-Diagramm

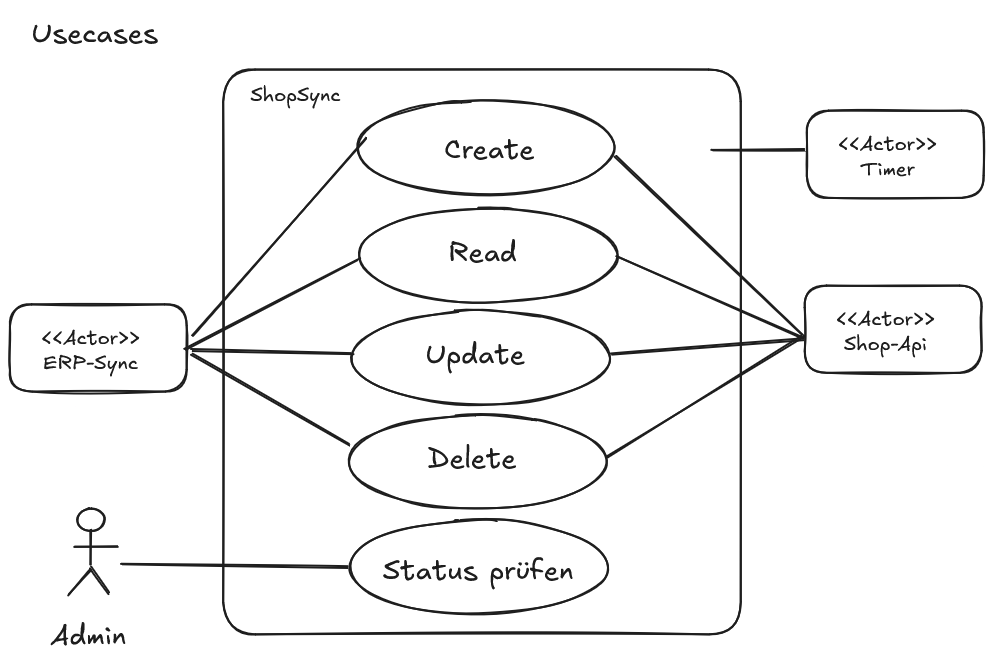


Abbildung : Usecase Diagramm

### User-Stories

#### ERP-Sync: Create, Update, Delete

##### User Story

Als Aktor ERP-Sync möchte ich Shop-Daten in der ShopSync DB

* erstellen,
* updaten
* löschen,

damit sie im Shop aktualisiert werden können.

##### Erläuterung

Die Komponente ERP-Sync des ERPs (ausserhalb des Kontexts des Auftrags) aktualisiert die Shop-Daten direkt in der Datenbank von ShopSync. Damit ShopSync die veränderten Datensätze erkennt, markiert ERP-Sync diese mit einem Flag. Gelöschte Datensätze müssen mit einem separaten Flag versehen werden. Damit ShopSync später weiss, welchem Shop die Daten zugeordnet werden müssten, wird dem Datensatz eine Shop-ID hinzugefügt.

#### ERP-Sync: Read

##### User Story

Als ERP-Sync möchte ich von der ShopSync DB Datensätze mit aus der Shop-API aktualisierte Daten lesen, um das ERP mit dem Shop synchron zu halten.

##### Erläuterung

ERP-Sync liest die aktualisierten Datensätze direkt aus der ShopSync Datenbank. Welche Datensätze verändert wurden, kann ERP-Sync an einem Flag in der DB-Tabelle erkennen.

#### Shop-API: Create, Update, Delete

##### User Story

Als Aktor Shop-API möchte ich dem ShopSync neu erstellte, aktualisierte und gelöschte Datensätze bekannt geben, damit diese vom ERP-Sync konsumiert und im ERP aktualisiert werden können.

##### Erläuterung

ShopSync bezieht periodisch die aktuellen Daten über die REST-Schnittstelle der Shop-API. Die Daten werden mit den vorhandenen Daten in der ShopSync-DB abgeglichen. Veränderte Daten werden in der DB aktualisiert und entsprechend (insert, update) geflaggt. Gelöschte Datensätze werden mit einem separaten Flag markiert. Die Datensätze werden mit einer ID des Shops versehen, damit sie in der Datenbank eindeutig einem Shop zugeordnet werden können.

#### Shop-API: Read

##### User Story

Als Shop-API möchte ich aktualisierte Datensätze von ShopSync erhalten, um damit die Daten im Shop zu aktualisieren.

##### Erläuterung

ShopSync prüft periodisch seine Datenbank auf von ERP-Sync als verändert markierte Datensätze und aktualisiert diese über die REST-Schnittstelle (POST, PUT, DELETE) der Shop-API im Shop.

#### Admin: Status prüfen

##### User Story

Als Administrator von ShopSync möchte ich auf einem Dashboard sehen können, ob die DB und die Shops erreichbar sind (Health Check). Ich möchte zudem über fehlgeschlagene Synchronisationen informiert werden (Error Log). Dies, um mögliche Fehler zeitnah erkennen und beheben zu können.

##### Erläuterung

Auf einem Dashboard (Website) ist ersichtlich, ob die Datenbank von ShopSync und die angehängten Webshops erreichbar sind. In einem weiteren Bereich werden fehlgeschlagene Sync-Jobs mit der entsprechenden Fehlerursache aus den Logs angezeigt.

### Detaillierte Beschreibung Use Case Shop-API Create, Update, Delete

* **Motivation:** Die aktuellen Daten aus dem Shop sollen in der ShopSync DB bereitstehen, damit ERP-Sync diese für die Aktualisierung des ERPs verwenden kann
* **Beteiligtre Aktoren:** Shop-API, ShopSync, Timer
* **Input:** JSON-Respose Daten Shop-API, Daten ShopSync DB
* **Output:** -
* **Vorbedingungen:** Die Shop-API URL und die API-Credentials sind in ShopSync eingetragen. Der Netzwerk-Zugriff von ShopSync auf die Shop-API funktioniert.
* **Nachbedingungen:** Geänderte Datensätze sind in der ShopSync-DB mit U geflaggt. Neue Datensätze sind in der DB eingefügt und mit I geflaggt. Gelöschte Datensätze sind mit D geflaggt.
* **Fehlerfall:** Gleichzeitiges Update im ERP und Shop (gleichzeitig = zwischen zwei Synchronisationen)
* **Nachbedingungen im Fehlerfall:** In Absprache mit Kunden noch zu definieren.
* **Ablauf:**
* Der Timer löst nach einer einstellbaren Periode die Synchronisation aus
* ShopSync bezieht alle Datensätze von der REST-Schnittstelle der Shop-API.
* ShopSync lädt die Datensätze aus seiner eigenen Datenbank
* ShopSync vergleicht die Datensätze aus seiner DB mit den Daten von der Shop-API, die entsprechende Aktion wird auf der DB ausgeführt.
  + Datensatz existiert im JSON aber nicht in der DB => **Insert**
  + Datensatz existiert in der DB aber nicht im JSON => **Delete**
  + Datensatz existiert in der DB und dem JSON UND last\_update der beiden Datensätze differiert UND DB-Datensatz wurde vom ERP-Sync nicht als geändert geflaggt => **Update**
  + Datensatz existiert in der DB und dem JSON UND last\_update der beiden Datensätze differiert UND DB-Datensatz wurde vom ERP-Sync als geändert geflaggt => **Fehler, Konflikt wird aufgelöst.**

## Pflichtenheft

Aus den Usecases und den Rahmenbedingungen leitet sich das Pflichtenheft ab.

### Muss

#### ERP Sync Read

Aktuelle Shop Daten in ShopSync DB mit Status-Flags bereitstellen.

#### ERP Sync Create Update Delete

Datenmodell und zu setzende Flags für Entwicklung ERP-Sync definieren und dokumentieren.

#### Shop API Read

Eingabe

* JSON aus REST-API Call zur Shop-API

Verarbeitung

* Abgleich mit Daten aus ShopSync Datenbank

Ausgabe

* Aktuelle Shop Datensätze in ShopSync DB mit Flags für Changetracking

#### Mandantenfähigkeit

Shop-Id in Kategorien Tabellen im Datenmodell.

### Soll

#### Admin Dashboard

Web-UI mit Status Dashboard und Error-Log.

### Einschränkungen

Im Rahmen der Projektarbeit soll die Umsetzung vorerst nur anhand des Beispiels eines Produkts umgesetzt werden. (Proof of Concept).

Die Muss Anforderungen gemäss Aufgabenstellung können wegen des grossen Aufwands der Implementierung des Datenmodells in der gegebenen Zeit nicht umgesetzt werden.

Diese wären:

* ***Upload von Kategorien und Produkten***
* ***Synchronisation von Kunden***
* ***Download von Bestellungen***

## Datenmodell

Im Rahmen der Projektwoche soll nur ein Prototyp erstellt werden. Wir beschränken uns auf die Abbildung der Entität **Product**. Bei den Attributen werden exemplarisch einige dargestellt. Es werden aus Zeitgründen nicht alle implementiert. (Proof of Concept)

Die Mandantenfähigkeit wird mittels Fremdschlüsselattribut shop\_id sichergestellt. Für das Changetracking werden die Attribute (erp|shop)\_changed und (erp|shop)\_deleted verwendet.

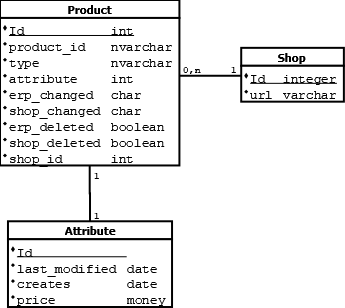


Abbildung : Vereinfachtes Datenmodell (ERD) für Prototyp

## Technologie

### Datenbank

Das Datenbank-Produkt wird vom Kunden vorgegeben und ist **Microsoft SQL-Server.** Die Datenbank für das Projekt wird vom Kunden gestellt**.**

### Schnittstellen

Die Schnittstellen werden durch die bestehenden Komponenten definiert. Die Kommunikation mit der Shop-API erfolgt mittels **REST**-Schnittstelle.

Die Kommunikation der Komponente ERP-Sync (nicht Kontext des Projekts) mit dem ShopSync erfolgt direkt mittels SQL Queries.

### Service

Die Technologie für die Implementierung der Service Komponente ist nicht vorgegeben. Hier soll mittels **Variantenvergleich** eine Entscheidung getroffen werden.

Der Service muss einige Muss-Kriterien erfüllen: Anbindung an MS SQL-Server, die Konfiguration mittels Datei o.Ä. und das Abfragen und verarbeiten von REST-Calls beherrschen praktisch alle modernen Sprachen.

Für die Implementierung des Admin-Dashboards wollen wir der Einfachheit halber auf ein Rendering der Seite im Backend setzen.

Bewertungsmasstab: **1** am Schlechtesten – **5** am Bestem

Kriterien:

1. Anbindung MS SQL Server
2. Einfachheit REST-Calls
3. Konfiguration via Datei o.Ä.
4. Templating für Server-Side HTML-Rendering
5. Know-How der Gruppenmitglieder

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **Total** | **Rang** | **Programmiersprache (Framework)** |
| 5 | 5 | 4 | 3 | 4 | **21** | **1** | C# (ASP.Net / Blazor für Dashboard) |
| 3 | 5 | 4 | 5 | 3 | **20** | **2** | Python (Flask) |
| 4 | 5 | 4 | 4 | 2 | **19** | **3** | Java (Springboot) |
| 3 | 5 | 4 | 4 | 2 | **18** | **4** | Go |
| 3 | 3 | 4 | 5 | 2 | **17** | **5** | PHP |

Tabelle : Entscheidungsmatrix Service

Als Sieger hat sich **C#** herausgestellt. Mit Entity Framework ist die Anbindung an einen SQL Server sehr komfortabel. Das Know-How in der Gruppe ist am Grössten. Blazor schein für HTML-Templating nicht gleich einfach zu sein, wie etwa die Jinja2 Templates von Flask. Die Vorteile können dieses Manko ausgleichen.

# Glossar

|  |  |
| --- | --- |
| Begriff | Erklärung |

# Abbildungen

[Systemskizze 2](#_Toc201008019)

[Systemkontext 2](#_Toc201008020)

[Usecase Diagramm 3](#_Toc201008021)

[Vereinfachtes Datenmodell (ERD) für Prototyp 7](#_Toc201008022)

# Literaturverzeichnis

Zürich, E. (2017). *Kompetenzorientierte Lernziele formulieren*. Von https://ethz.ch/content/dam/ethz/main/eth-zurich/education/lehrentwicklung/files\_DE/Vorlage\_LernzieleFormulierenDe.pdf abgerufen