

User Story 1: Synchronisation einer AAS-Variable

Als Endnutzer (Systemingenieur bei Bosch)

möchte ich eine Variable aus einem Quell-AAS-System in ein Ziel-AAS-System synchronisieren lassen,

damit im Zielsystem stets die aktuellen Daten verfügbar sind und der Zugriff kontrolliert erfolgt.

Akzeptanzkriterien:

- Konfiguration über Konfigurator-UI anlegen.
- Regel wird im Core-Management gespeichert.
- Core-Sync-Node pollt und synchronisiert geänderte Daten.
- Zugriffskontrolle über EDC wird sichergestellt.
- Monitoring zeigt Status und Fehler an.
- Generell werden auch andere REST-Schnittstellen zum Sync unterstützt

User Story 2: Bearbeiten einer bestehenden Synchronisationsregel

Als Systemingenieur

möchte ich eine bestehende Synchronisationsregel bearbeiten,

damit ich Konfigurationen flexibel anpassen kann.

Akzeptanzkriterien:

- Bestehende Regeln sichtbar und editierbar.
- Aktualisierte Regeln werden gespeichert und delegiert.
- Monitoring zeigt aktualisierte Zustände an.

User Story 3: Fehlerhafte Synchronisation erkennen und analysieren

Als Operator

möchte ich Fehler im Synchronisationsprozess analysieren,

damit ich Probleme schnell erkennen und beheben kann.

Akzeptanzkriterien:

- Fehler werden geloggt und angezeigt.
- Verlinkung zur betroffenen Sync-Regel möglich.
- Inputs werden geloggt und Fehler angezeigt

User Story 4: Monitoring-UI zur Übersicht der Sync-Flows und Systemnetzwerke

Als Administrator

möchte ich eine grafische Darstellung aller aktiven Synchronisationsflüsse und deren Quell- und Zielsysteme sehen,
damit ich die Datenflüsse und Systemverbindungen schnell und übersichtlich nachvollziehen kann.

Akzeptanzkriterien:

- Grafische Darstellung aller aktiven Synchronisationsflüsse.
- Status farblich hervorgehoben (aktiv, fehlerhaft, inaktiv).
- Detailansicht für jeden Flow (inkl. Quell- und Zielsysteme).
- Darstellung der Verbindungen zwischen Quell- und Zielsystemen (Systemnetzwerkgraph).
- Übersichtliche Navigation zwischen Flows und Systemknoten.

User Story 5: Verwaltung von Zugriffspolicies im EDC

Als Security-Manager

möchte ich Zugriffspolicies im EDC verwalten,
damit der Datenzugriff sicher geregelt ist.

Akzeptanzkriterien:

- Neue Policies erstellen und bestehenden Partnern zuweisen.
- Policies enthalten zeitliche oder funktionale Zugriffsbeschränkungen.
- Änderungen im Monitoring angezeigt.
- Unterstützt verschiedenen EDC Zugangsvariant mit Sicherheitskonzept

User Story 6: Dry Run einer Sync-Config

Als Nutzer

möchte ich eine Sync-Konfiguration in einem Dry Run testen,
damit ich die Konfiguration validieren kann.

Akzeptanzkriterien:

- Die ausgeführten Aktionen werden geloggt und angezeigt.
- Die Aktionen werden nicht auf das Zielsystem geschrieben.
- Status der Transformationskonfiguration wird angezeigt.
- parameterisiert, im Monitoring
- Unterstützt Submodeltemplates zum generischen Laden der Datentypen

User Story 7: EDC-Verwaltung

Als Nutzer

möchte ich EDC-Instanzen erstellen, bearbeiten und löschen,
damit Zielsysteme für die Sync-Konfiguration vorkonfigurierbar sind.

Akzeptanzkriterien:

- In der Konfigurationsseite auffindbar.
- Unterstützung/Bereitstellung eines eigenen EDCs.
- Auslesen bestehender Netzwerke aus einem Identity Provider.

User Story 8: AAS-Verwaltung

Als Nutzer

möchte ich AAS (BaSyx) erstellen, bearbeiten und löschen,
damit Quellsysteme für die Sync-Konfiguration vorkonfigurierbar sind.

Akzeptanzkriterien:

- In der Konfigurationsseite auffindbar.
- Auslesen verfügbarer Schnittstellenpfade und Datentypen.

User Story 9: IDE-Unterstützung

Als Nutzer

möchte ich Skripte in der Webapplikation mit IDE-Features wie Auto-Complete erstellen,
damit ich diese für meine Transformation nutzen kann.

Akzeptanzkriterien:

- Grundlegende Validierung der Korrektheit.
- Unterstützung von Datentypen und APIs.
- testbarkeit und replayfeature demoumgebung mit sample requests Fehlerzustand aus dem monitoring runterladen, einspielen und zu reproduzieren sind gegeben

User Story 10: Logs für das Monitoring

Als Nutzer

möchte ich Zugriff auf alle System-Log-Informationen haben,
damit ich die Aktionen auswerten und Troubleshooting betreiben kann.

Akzeptanzkriterien:

- Alle relevanten Logs der Sync-Nodes und des Managements.
- Fehlerfall ist nachvollziehbar

User Story 11: Systemauslastung

Als Nutzer

möchte ich Metriken der Sync-Dienste auswerten können,
damit ich einen Überblick über die Auslastung habe.

Akzeptanzkriterien:

- Requestanzahl, Systemauslastung, Last pro Skript, Ausführungszeiten, Quell- und Zielsystem sollen auffindbar sein.

User Story 12: Überwachung einzelner Sync-Konfigurationen

Als Nutzer

möchte ich einzelne Sync-Konfigurationen überwachen,
damit ich die korrekte Ausführung feststellen kann.

Akzeptanzkriterien:

- Statusanzeige und Benachrichtigung bei Fehlern oder Abweichungen.

User Story 13: Verwendung von Skripten

Als Nutzer

möchte ich Skripte für die Transformation verwenden,
damit ich Daten in der Transformation manipulieren kann.

Akzeptanzkriterien:

- TypeScript-Unterstützung.
- (Optional) Paket-Management-Funktionen zum Nachladen externer Bibliotheken.
- Skripte sollten wiederverwendbar sein.
- Sicherheitskonzepte müssen Code-Injection über Variablen der Quellsysteme verhindern.

User Story 14: Polling & Synchronisationslogik - Bedingungs und Zeitbasiert

Als Nutzer

möchte ich möchte ich festlegen, wann eine Synchronisation ausgeführt wird. Dazu möchte ich eine Polling-Rate konfigurieren und logische Bedingungen festhalten. Beim erfüllen der Bedingungen soll eine Transformation/sync getriggert werden. Ohne logische Bedingungen soll die Synchronisation nur zeitbasiert stattfinden.

damit die Daten nur wenn nötig synchronisiert werden

Akzeptanzkriterien:

- Verknüpfen logischer Bedingungen mit Zeitintervallen(Wenn die Bedingung in dem Intervall auftritt findet sync statt, sonst nicht)
- Die Bedingungen sollen über die Benutzeroberfläche graphbasiert (oder Skriptbasiert) erzeugt und mit logischen Operatoren verknüpft werden können.
- Man kann Skripte zu den Regeln hinzufügen, Skript basierte Logik mit booleschen Wert als Teil davon

Gedanken zur Umsetzung: Sync-Node: Snapshots der Inputvariablen, werden nach konfiguriertem Zeitabstand erstellt (in-memory) und mit dem vorher persistierten Snapshot verglichen auf Basis der Bedingungen des Nutzers

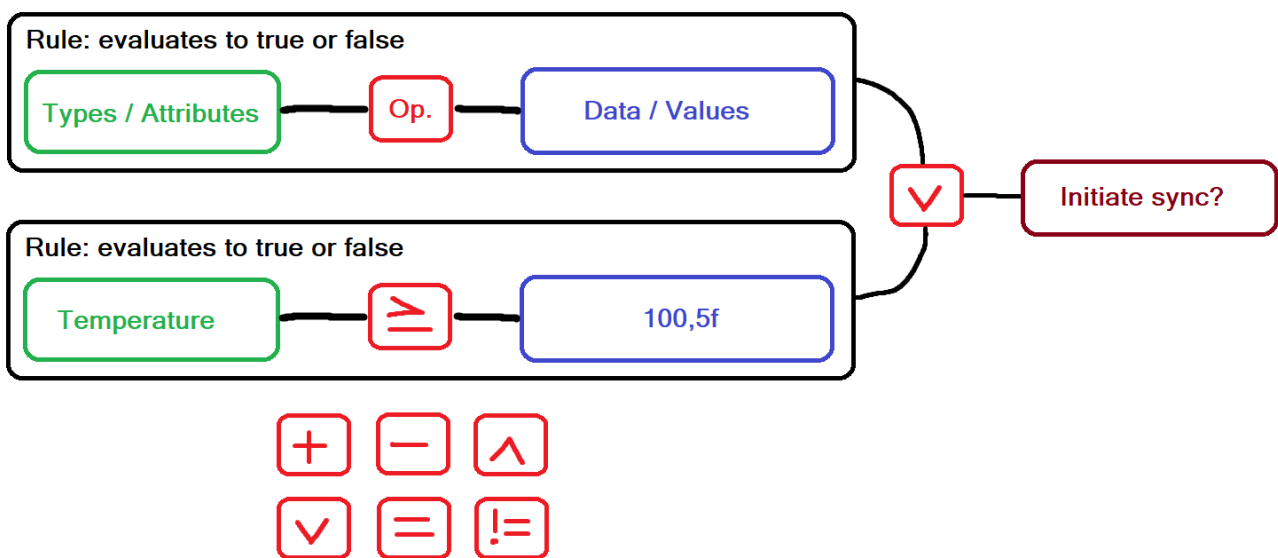


Abbildung 1: ui-Mockup

User Story 15: Initialkonfiguration des Core-Service

Als Nutzer

möchte ich die Konfiguration des Core Service via .config Dateien (eventuell YAML) einspielen. Einspielen über UI via im Configurator auf Applikationskonfigurations-Seite.

damit die Anwendungskonfiguration übernommen werden kann

Akzeptanzkriterien:

- Steuerbar ob Änderungen in der Konfig persistiert werden via Startoption

Gedanken zur Umsetzung:

Sync-Node: Snapshots der Inputvariablen, werden nach konfiguriertem Zeitabstand erstellt (in-memory) und mit dem vorher persistierten Snapshot verglichen auf Basis der Bedingungen des Nutzers

Initialisierungsservice soll initiale Konfigdatei lesen, und Anwendung entsprechend mit Daten

befüllen(Datenbank: Quellsysteme speichern, Synckonfigurationen speichern, Zielsysteme speichern) Syncaufträge an Syncnodes verteilen

User Story 16: Skalierung

Als Nutzer

möchte ich, dass die Anwendung horizontal skalierbar ist

damit die die Anwendungen mit einer höheren Auslastung umgehen kann

Akzeptanzkriterien:

- Die Anwendung ist als Standalone, Docker-Container und in kubernetes deploybar
- Das Core-Managment delegiert aufträge an die Sync-Nodes

Gedanken zur Umsetzung: Deployment auf kubernetes mit helmcharts, skalierung der Syncnodes durch das Managment mithilfe der Kubernetes-API, Sync-Aufträge werden vom Management an die Sync-Nodes delegiert