



MISTA GMBH
BERATUNG • IT • SOFTWARE

Automatische Dokumentationserstellung

für Firma Schönenberger

Konzept und technische Architektur

Sphinx-basierte Zwei-Repo-Architektur mit JSON-gesteuertem Build

Erstellt von: Mista GmbH

Datum: April 2026

Version 1.0 – Entwurf

Inhaltsverzeichnis

1. Vorteile einer automatischen Erstellung	3
1.1 Zeitersparnis und Effizienz	3
1.2 Konsistenz und Qualität	3
1.3 Mehrsprachigkeit ohne Mehraufwand	3
1.4 Wartbarkeit und Skalierung	4
1.5 Vergleich: Manuell vs. Automatisiert	4
2. Annahmen und Voraussetzungen	5
2.1 Ausgabeformat und Umfang	5
2.2 Mehrsprachigkeit	5
2.3 Datenquelle und Steuerung	5
2.4 Bildverwaltung	5
2.5 Firmenglossar	6
2.6 Versionierung und Zusammenarbeit	6
3. Technische Umsetzung	7
3.1 Zwei-Repo-Architektur	7
Repo 1: transport-docs-base (Allgemeine Ressourcen)	7
Repo 2: projekt-kunde-xyz (Projektspezifisch)	7
3.2 Technologie	7
3.3 Build-Prozess	9
Phase 1: Konfiguration einlesen	9
Phase 2: Abhängigkeiten auflösen	9
Phase 3: Sphinx-Quellen generieren	9
Phase 4: PDF erzeugen (pro Sprache)	9
3.4 Modul-Architektur (meta.yaml)	9
3.5 Mehrsprachigkeitskonzept	10
Ebene 1: Textbausteine	10
Ebene 2: Bilder	10
Ebene 3: Glossar	10
3.6 Verzeichnisstruktur	10
3.6.1 Allgemein	10
3.6.2 Projektspezifische Angaben	13
3.7 Empfohlene nächste Schritte	Fehler! Textmarke nicht definiert.

1. Vorteile einer automatischen Erstellung

Die Erstellung technischer Dokumentation für Transport- und Hängefördersysteme ist ein komplexer, wiederkehrender Prozess. Jede Anlage besteht aus einer individuellen Kombination von Maschinentypen, die jeweils eigene Kapitel in der Dokumentation erfordern – von Sicherheitshinweisen über Risikomatrizen bis hin zu Wartungsplänen. Die manuelle Zusammenstellung dieser Dokumente ist fehleranfällig, zeitaufwändig und schwer konsistent zu halten.

1.1 Zeitersparnis und Effizienz

Eine automatisierte Dokumentationspipeline reduziert den Aufwand pro Anlagendokumentation drastisch:

- Einmalige Pflege von Textbausteinen statt wiederholtes Copy-Paste aus alten Projekten
- Automatische Zusammenstellung basierend auf dem ERP-Export (JSON) – kein manuelles Kapitel-Zusammensuchen
- Parallele Erzeugung aller Sprachversionen in einem Build-Lauf
- Sofortige Aktualisierung aller zukünftigen Projekte bei Änderung eines Textbausteins

1.2 Konsistenz und Qualität

Das größte Risiko manueller Dokumentation ist Inkonsistenz: unterschiedliche Formulierungen für dieselbe Maschine, veraltete Sicherheitshinweise in einzelnen Projekten, fehlende Kapitel.

- Zentrale Textbausteine garantieren einheitliche Formulierungen über alle Projekte
- Firmenspezifisches Glossar mit automatischer Verlinkung im gesamten Dokument
- Versionierte Quellen in Git – jede Änderung ist nachvollziehbar und rückverfolgbar
- Validierung beim Build: fehlende Module, Bilder oder Übersetzungen werden sofort erkannt

1.3 Mehrsprachigkeit ohne Mehraufwand

Mit dem vorgeschlagenen System wird Mehrsprachigkeit systemisch gelöst, nicht als nachträgliche Übersetzungsaufgabe:

- Jeder Textbaustein existiert als sprachspezifische Datei (z. B. `beschreibung.de.rst`, `beschreibung.en.rst`)
- **Sprachspezifische** Bilder (HMI-Screenshots, Risikographen) werden automatisch nach Build-Sprache aufgelöst
- **Normen**verweise können regionsspezifisch gesteuert werden (CE vs. UKCA)
- Neue Sprachen lassen sich sukzessive ergänzen, ohne bestehende Projekte zu beeinträchtigen

1.4 Wartbarkeit und Skalierung

Das System ist darauf ausgelegt, von einem einzelnen technischen Redakteur verwaltet zu werden, aber auch für wachsende Teams zu funktionieren:

- Klare Trennung: allgemeine Ressourcen (Base-Repo) vs. projektspezifische Anpassungen (Projekt-Repo)
- Override-Mechanismus: Projekte können einzelne Bausteine überschreiben, ohne das Base-Repo zu verändern
- Git-basierte Versionierung mit Semver – Projektrepos können eine Mindestversion des Base-Repos fordern (Bsp. *2.3.10* aus *Major.Minor.Patch*)
- Neue Maschinentypen werden als Module hinzugefügt – sofort für alle zukünftigen Projekte verfügbar

1.5 Vergleich: Manuell vs. Automatisiert

Kriterium	Manuell	Automatisiert
Zeitaufwand pro Projekt	2–4 Wochen	1–2 Tage (nach Ersteinrichtung)
Konsistenz	Abhängig vom Bearbeiter	Garantiert durch zentrale Quellen
Mehrsprachigkeit	Separater Übersetzungsprozess	Integriert im Build-Prozess
Fehlerrisiko	Hoch (fehlende Kapitel, Versionen)	Niedrig (Build-Validierung)
Nachvollziehbarkeit	Schwierig (Dateiversionen)	Vollständig (Git-Historie)
Aktualisierung bei Änderungen	Alle Projekte manuell anpassen	Ein Commit, alle Projekte aktuell

2. Annahmen und Voraussetzungen

Die folgenden Annahmen und Rahmenbedingungen bilden die Grundlage für die technische Architektur. Sie basieren auf der Analyse der bestehenden Dokumentationsprozesse und der geplanten Systemlandschaft.

2.1 Ausgabeformat und Umfang

- Endformat: **PDF** via **LaTeX** (hohe typographische Qualität, professionelles Layout)
- Erwarteter Umfang: ca. 250+ Seiten pro Anlagendokumentation
- Komplexe Abhängigkeiten: eine Maschine erzeugt Inhalte in mehreren Kapiteln (z. B. Risikomatrix in Kapitel Sicherheit, Beschreibung in Kapitel Anlagenbeschreibung, Schmierplan in Kapitel Wartung)
- Reines Markdown reicht nicht aus – erforderlich sind automatische Querverweise, Abbildungsnummerierung, Glossarverlinkung und komplexe Tabellenstrukturen

2.2 Mehrsprachigkeit

Die Dokumentation muss mehrsprachig erstellt werden können. Folgende Anforderungen wurden identifiziert:

- Primärsprachen: Deutsch und Englisch
- Weitere europäische Sprachen nach Bedarf (Französisch, Polnisch, etc.)
- Sprachspezifische Bilder: HMI-Screenshots, Risikographen und Typenschilder existieren je Sprache
- Normenverweise können je nach Zielregion abweichen (z. B. CE-Kennzeichnung vs. UKCA für UK)
- Fallback-Mechanismus: wenn eine Übersetzung fehlt, wird die deutsche Version verwendet

2.3 Datenquelle und Steuerung

Die Zusammenstellung der Dokumentation wird durch einen strukturierten Export aus dem ERP-System gesteuert:

- Format: JSON-Export, der von Mista GmbH spezifisch erstellt wird
- Inhalt: Liste der verbauten Maschinen mit Positionen, Varianten und Optionen
- Abhängigkeiten: Eine Maschine referenziert via meta.yaml, welche Kapitel und Unterkapitel sie benötigt
- Zusatzkapitel: Der JSON-Export kann projektspezifische Sonderkapitel referenzieren

2.4 Bildverwaltung

Bilder werden in drei Kategorien verwaltet:

Kategorie	Beschreibung	Beispiele
Sprachneutral	Technische Zeichnungen, Fotos, Schaltpläne ohne eingebetteten Text	DXF/SVG-Zeichnungen, Maschinenfotos, ISO-7010-Symbole
Sprachspezifisch	Bilder mit eingebettetem Text, der je Sprache variiert	HMI-Screenshots, Risikographen, Typenschilder, Bedienpanel-Aufkleber
Projektspezifisch	Einmalbilder, die nur für eine konkrete Anlage gelten	Aufstellpläne, Abnahmefotos, kundenspezifische Layouts

2.5 Firmenglossar

- Zentrales mehrsprachiges Glossar mit firmenspezifischen Fachbegriffen
- Verwaltung als CSV-Dateien (je Sprache), pflegbar auch für Nicht-Entwickler
- Automatische Verlinkung im Dokumenttext (**Sphinx**-Glossar-Direktive)
- Erscheint als Glossar-Verzeichnis im fertigen PDF

2.6 Versionierung und Zusammenarbeit

- Verwaltung in Git (Gitea auf Hetzner-Infrastruktur)
- Pflege durch einen technischen Redakteur (erweiterbar für Teams)
- Klare Trennung: allgemeines Ressourcen-Repo vs. projektspezifisches Repo
- Semver-Versionierung des Base-Repos mit Kompatibilitätsprüfung

3. Technische Umsetzung

Die Umsetzung basiert auf einer Zwei-Repo-Architektur mit Sphinx als Build-System. Das System wird vollständig in Python und **reStructuredText** realisiert und läuft auf der bestehenden Hetzner-Infrastruktur. Die Erzeugung des pdfs selbst erfolgt über Sphinx und einen Latex Compiler.

3.1 Zwei-Repo-Architektur

Die zentrale Designentscheidung ist die Trennung in zwei unabhängige Repositories:

Repo 1: transport-docs-base (Allgemeine Ressourcen)

Dieses Repository enthält alle wiederverwendbaren Inhalte und wird auf Gitea gepflegt. Es wird beim Build als Netzlaufwerk oder Mount-Punkt bereitgestellt.

- Sphinx-Extensions: anlagen_builder.py, localized_figure.py, glossary_loader.py
- LaTeX-Templates: Firmen-CI (Fonts, Farben, Kopf-/Fußzeilen, Titelseite)
- Module: Textbausteine pro Maschinentyp mit meta.yaml (Abhängigkeiten und Kapitelzuordnung)
- Assets: sprachneutrale und sprachspezifische Bilder (shared/ und localized/)
- Glossar: CSV-Dateien pro Sprache
- Rahmenkapitel: Gerüstkapitel mit Jinja2-Templates
- Hilfsskripte: DXF→SVG-Konvertierung, Bildoptimierung, Modul-Validierung

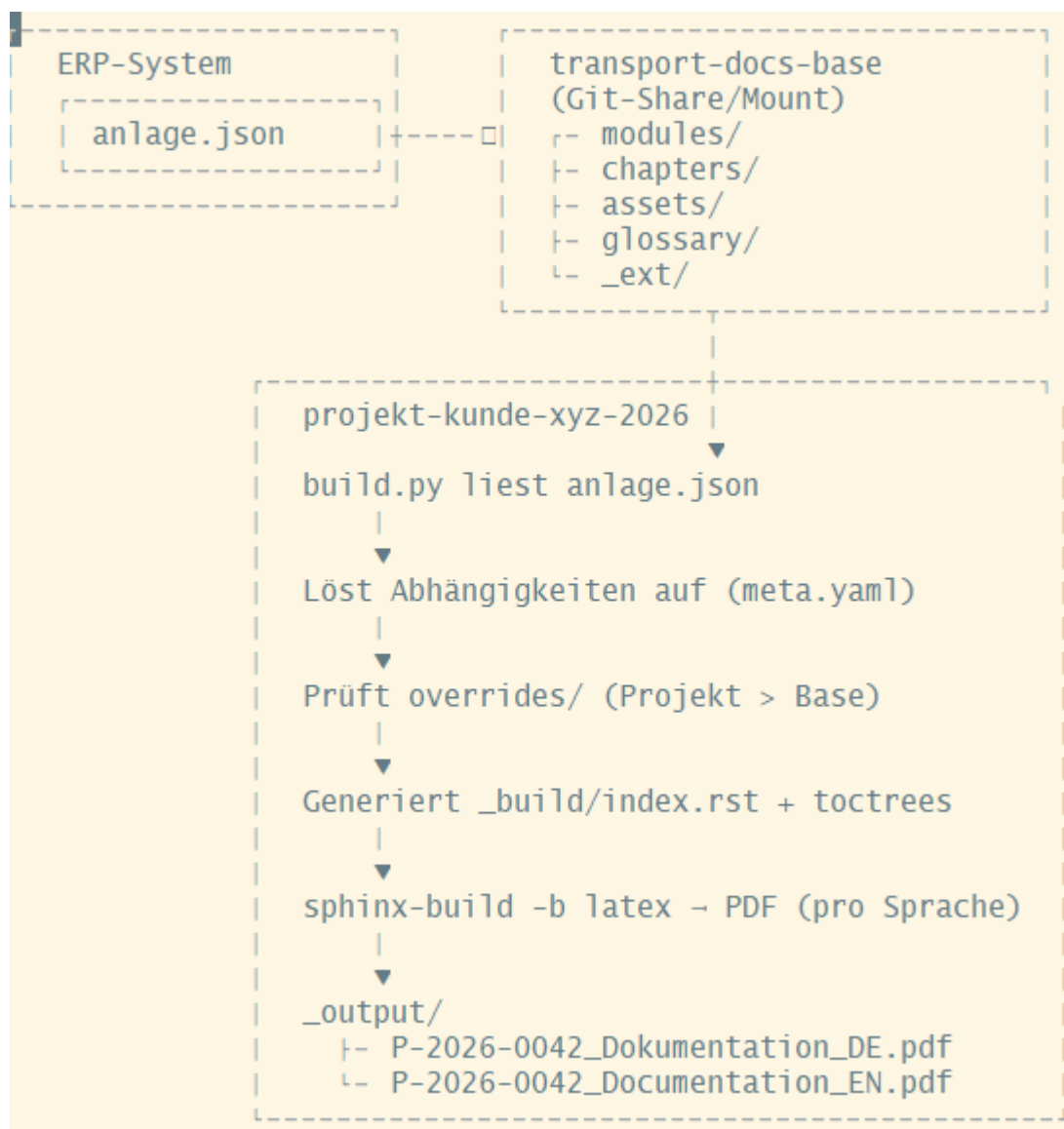
Repo 2: projekt-kunde-xyz (Projektspezifisch)

Pro Kundenanlage wird ein eigenes Repository angelegt. Es enthält nur die projektspezifischen Dateien:

- anlage.json: ERP-Export mit der Liste verbauter Maschinen, Sprachen und Normenregion
- Projektbilder: Aufstellpläne, Abnahmefotos, Kundenlogo
- Overrides: Projektspezifische Überschreibungen einzelner Textbausteine oder Kapitel
- Zusatzkapitel: Sonderkapitel, die nur für dieses Projekt gelten
- conf.py: Importiert die Basiskonfiguration und setzt Projektmetadaten
- build.py / Makefile: Steuert den Build-Prozess

3.2 Technologie

Komponente	Technologie und Zweck
Build-System	Sphinx (Python) – Dokumentationsgenerator mit LaTeX-Backend
Textformat	reStructuredText (rST) – Querverweise, Glossar, Direktiven
Templating	Jinja2 – Dynamische Inhalte in Rahmenkapiteln und Titelseite
PDF-Erzeugung	LaTeX (pdflatex/lualatex) – Professioneller Satz
Versionierung	Git (Gitea) – Nachvollziehbarkeit, Branching, Reviews
Konfiguration	JSON (ERP-Export) + YAML (Modul-Metadaten)
Infrastruktur	Hetzner-Server mit Gitea, n8n (CI-Automatisierung), Nextcloud
Bildkonvertierung	Python (ezdxf → SVG), Pillow (Optimierung)



Hier die entsprechenden Links zu den eingesetzten Technologien

Latex: <https://www.latex-project.org/get/>

Sphinx: <https://www.sphinx-doc.org/en/master/>

rst: <https://de.wikipedia.org/wiki/ReStructuredText>

3.3 Build-Prozess

Der Build-Prozess läuft vollautomatisch ab und besteht aus vier Phasen:

Phase 1: Konfiguration einlesen

- build.py liest anlage.json (ERP-Export)
- Prüft Kompatibilität der Base-Repo-Version (VERSION vs. base_version)
- Lädt meta.yaml aller referenzierten Module aus dem Base-Repo

Phase 2: Abhängigkeiten auflösen

- Erstellt die vollständige Kapitelliste basierend auf den meta.yaml-Einträgen
- Löst transitive Abhängigkeiten auf (z. B. Weiche benötigt Antrieb-Modul)
- Prüft auf fehlende Module, Bilder und Übersetzungen
- Wendet Override-Logik an: Projekt-Dateien überschreiben Base-Dateien

Phase 3: Sphinx-Quellen generieren

- Generiert index.rst mit vollständigen toctree-Einträgen
- Kopiert/verlinkt Module und Bilder in den Build-Ordner
- Erzeugt Glossar-Direktiven aus CSV-Dateien
- Setzt Projektmetadaten (Kunde, Projektnummer, Datum) in Jinja2-Templates ein

Phase 4: PDF erzeugen (pro Sprache)

- sphinx-build -b latex erzeugt LaTeX-Quellen mit Firmen-CI
- pdflatex/lualatex compiliert das finale PDF
- Ausgabe: P-2026-0042_Dokumentation_DE.pdf, P-2026-0042_Dokumentation_EN.pdf

3.4 Modul-Architektur (meta.yaml)

Jedes Maschinenmodul wird durch eine meta.yaml-Datei beschrieben, die seine Einordnung in die Buchstruktur und seine Abhängigkeiten definiert. Beispiel für eine Weiche Typ A:

Feld	Beschreibung
------	--------------

module	Eindeutiger Bezeichner (z. B. weiche_typ_a)
display_name	Mehrsprachiger Anzeigename (de: Weiche Typ A, en: Switch Type A)
chapters	Zuordnung zu Buchkapiteln mit Angabe der Sections (z. B. 02_sicherheit → risikomatrix)
images	Referenzen auf shared, localized und modulspezifische Bilder
requires	Abhängigkeiten auf andere Module (z. B. antrieb_sew)

3.5 Mehrsprachigkeitskonzept

Die Mehrsprachigkeit wird auf drei Ebenen gelöst:

Ebene 1: Textbausteine

Jeder Textbaustein existiert als sprachspezifische Datei mit dem Schema `dateiname.{lang}.rst`. Der Build wählt automatisch die richtige Sprachversion basierend auf der Zielsprache. Fehlende Übersetzungen fallen auf die deutsche Version zurück.

Ebene 2: Bilder

Eine Custom-Sphinx-Direktive `localized-figure` löst Bildpfade automatisch auf: `assets/localized/{lang}/pfad/bild.png`. Bei fehlender Sprachversion wird das deutsche Bild verwendet. Für SVG-Diagramme mit Text wird geprüft, ob ein Jinja2-basierter SVG-Generator sinnvoller ist als separate Bilddateien.

Ebene 3: Glossar

Pro Sprache existiert eine CSV-Datei mit dem Schema `Begriff;Definition;Kategorie`. Die `glossary_loader.py`-Extension erzeugt daraus automatisch Sphinx-Glossardirektiven. Im Text wird mit der Rolle `:term:`Fachbegriff`` verlinkt.

3.6 Verzeichnisstruktur

3.6.1 Allgemein

```

transport-docs-base/
|-- conf_base.py                # Sphinx-Basiskonfiguration (wird importiert)
|-- _ext/
|   |-- anlagen_builder.py      # Hauptextension: JSON-Buchstruktur
|   |-- localized_figure.py     # Sprachauflösung für Bilder
|   |-- glossary_loader.py      # CSV-Glossar-Direktive
|-- _templates/
|   |-- latex/
|       |-- preamble.tex        # Firmen-CI: Fonts, Farben, Kopfzeilen
|       |-- titlepage.tex       # Titelseite-Template (Jinja2)
|       |-- maketitle.tex
|   |-- common/
|       |-- header_footer.rst
|-- assets/
|   |-- shared/
|       |-- photos/
|           |-- weiche_typ_a_foto_01.jpg
|           |-- hubstation_foto_01.jpg
|           |-- antrieb_sew_foto_01.jpg
|       |-- drawings/
|           |-- weiche_typ_a_layout.svg
|           |-- hubstation_schnitt.pdf
|       |-- symbols/            # ISO 7010 Sicherheitssymbole
|           |-- W001_warnung.svg
|           |-- M004_augenschutz.svg
|       |-- diagrams/
|           |-- stromlaufplan_weiche_a.pdf
|   |-- localized/
|       |-- de/
|           |-- screenshots/
|               |-- hmi_hauptmenu.png
|           |-- diagramme/
|               |-- risikograph_weiche_a.svg
|       |-- en/
|           |-- screenshots/
|               |-- hmi_hauptmenu.png
|           |-- diagramme/
|               |-- risikograph_weiche_a.svg
|       |-- fr/

```

Mehrsprachige Bilder, Latexvorlagen und die Skripte zur Verwaltung der Basistruktur

```
--- modules/                                     # □□ ALLE MASCHINENTYPEN-BAUSTEINE □□
|   |-- weiche_typ_a/
|   |   |-- meta.yaml
|   |   |-- beschreibung.de.rst
|   |   |-- beschreibung.en.rst
|   |   |-- risikomatrix.de.rst
|   |   |-- risikomatrix.en.rst
|   |   |-- wartung.de.rst
|   |   |-- images/
|   |       |-- montage_detail_01.jpg
|   |-- hubstation/
|   |   |-- meta.yaml
|   |   |-- beschreibung.de.rst
|   |   |-- technische_daten.de.rst
|   |   |-- wartung.de.rst
|   |   |-- images/
|   |       |-- hubmechanik_detail.jpg
|   |-- antrieb_sew/
|   |   |-- meta.yaml
|   |   |-- ...
|   |-- steuerung_siemens_s7/
|   |   |-- meta.yaml
|   |-- ...
```

Textvorlagen zur Beschreibung von einzelnen Maschinen und Baugruppen

```

|-- ...
|-- chapters/                                # □□ RAHMENKAPITEL (Gerüst) □□
|   |-- 01_einleitung/
|   |   |-- template.de.rst                  # Jinja2-Template mit Platzhaltern
|   |   |-- template.en.rst
|   |-- 02_sicherheit/
|   |   |-- allgemein.de.rst                # Immer dabei
|   |   |-- allgemein.en.rst
|   |   |-- normen_ce.de.rst               # EU-Konformität
|   |   |-- normen_ukca.en.rst            # UK-spezifisch
|   |-- 03_anlagenbeschreibung/
|   |   |-- template.de.rst                # Hier werden Module eingefügt
|   |-- 04_betrieb/
|   |   |-- template.de.rst
|   |-- 05_wartung/
|   |   |-- allgemein.de.rst
|   |   |-- schmierplan_template.de.rst
|   |-- 06_ersatzteile/
|   |   |-- template.de.rst
|   |-- 07_anhang/
|   |   |-- template.de.rst
|-- glossary/
|   |-- terms.de.csv                       # "Begriff;Definition;Kategorie"
|   |-- terms.en.csv
|   |-- terms.fr.csv
|-- styles/
|   |-- firma_logo.pdf
|   |-- firma_logo_sw.pdf
|   |-- color_scheme.yaml                  # CI-Farben für LaTeX und SVG-Generi
|-- scripts/
|   |-- dxf2svg.py                         # DXF-SVG Konvertierung
|   |-- optimize_images.py                 # Bildoptimierung vor Commit
|   |-- validate_modules.py                # Prüft meta.yaml Konsistenz
|-- VERSION                                ■ # Semver der Dokumentationsbasis

```

Kapitelstruktur, Glossar, Logo und Stile. Helferprogramme zur Aufbereitung der Bilder

3.6.2 Projektspezifische Angaben

```
projekt-kunde-xyz-2026/
|-- conf.py                # □□ IMPORTIERT conf_base.py □□
|-- anlage.json            # □□ ERP-EXPORT □□
|-- images/               # Projektspezifische Bilder
|   |-- aufstellplan_halle3.pdf
|   |-- foto_abnahme_01.jpg
|   |-- kundenlogo.pdf
|-- overrides/            # Projektspezifische Überschreibungen
|   |-- modules/
|       |-- weiche_typ_a/
|           |-- beschreibung.de.rst # Überschreibt Basis wenn vorhanden
|       |-- chapters/
|           |-- 01_einleitung/
|               |-- template.de.rst # Kundenspezifische Einleitung
|-- additional/           # Kapitel die NUR dieses Projekt hat
|   |-- sonderausfuehrung_xyz.de.rst
|   |-- sonderausfuehrung_xyz.en.rst
|-- build.py              # Hauptbuild-Skript
|-- Makefile
|-- README.md
```

Angaben, die projektspezifisch Dinge beschreiben oder ergänzen

4. Projektplan

Personen:

A. Ja: Anton Jakob
 A.Sch: Annette Schopper
 D.Fe: Doris Fetzer
 F.Ve: Ferdinand Veith
 M.St: Michael Stangl
 M.Ro: Max Rost
 N.Sch: Nikolai Schwechten
 S.Hensch: Simon Hensch
 Y.Wang: Yuelin Wang

Stakeholder (Diese Leute geht das Ergebnis an, bzw. entscheiden über Form und Inhalt:

M.Rost, N.Schwechten, Y.Wang/S.Hensch

-> hier muß definiert welches Ergebnis man erwartet bzw. entschieden werden, ob das Ziel erreicht wurde

Development Team: Redakteure und Entwickler

0. Use-Case-Definition & Anforderungen

Phase	Beschreibung	Aufwand (gesch.)	Beteiligte
0.1	Story & Teaser: Zieldokumentation beschreiben, Prosa-Story je Produktfamilie	2 Tage	D.Fe, A.Sch, M.St
0.2	Stakeholder-Analyse: Alle Akteure und deren Interessen/Ziele auflisten	1 Tag	A.Sch, D.Fe, F.Ve, M.St
0.3	Standard-Ablauf entwerfen: Build-Pipeline, Redaktionsworkflow, Freigabeprozess	2 Tage	M.St
0.4	Ausnahmen sammeln: Sonderfälle (Mehrfachverwendung der 129 Nummern, Varianten, fehlende Übersetzungen)	1 Tag	F.Ve, A.Sch, M.St, D.Fe
0.5	Ausnahmen detaillieren, in User Stories schneiden	2 Tage	A.Scho, F.Ve, M.St
0.6	Aufwandsschätzung & Business Value für alle Stories	1 Tag	F.Ve, A.Sch, M.St, D.Fe
0.7	Review & DOR: Prüfung aller Interessen, Freigabe für Umsetzung	1 Tag	Stakeholder

1. PoC: Proof of Concept

Phase	Beschreibung	Aufwand (gesch.)	Beteiligte
-------	--------------	------------------	------------

1.1	Ein Modul (z.B. OMNIFLO Weiche), ein Kapitel, zwei Sprachen (DE/EN), PDF-Ausgabe	2 Tage	M.St, F.Ve
1.2	LaTeX-Template mit Firmen-CI erstellen und validieren	2 Tage	M.St, F.Ve, A.Sch
1.3	PoC-Review mit Stakeholdern	0,5 Tage	Stakeholder

2. Base-Repo & Infrastruktur

Phase	Beschreibung	Aufwand (gesch.)	Beteiligte
2.1	Repo-Struktur anlegen (transport-docs-base), Sphinx-Projekt initialisieren	1 Tag	M.St
2.2	Sphinx-Extensions implementieren (anlagen_builder, localized_figure, glossary_loader)	4 Tage	M.St
2.3	build.py: JSON-Parsing, Abhängigkeitsauflösung, Override-Mechanismus	3 Tage	F.Ve, M.St
2.4	CI/CD: Gitea Actions für automatischen Build	2 Tage	M.St

3. Produktprogramm-Strukturierung

Phase	Beschreibung	Aufwand (gesch.)	Beteiligte
3.1	Modulstruktur & meta.yaml-Schema je Produktfamilie definieren (5 Familien)	3 Tage	A.Sch, M.St, D.Fe
3.2	Nummernkreis-Mapping: 1388 Produkte auf Module abbilden, Gruppierung festlegen	3 Tage	A.Sch, F.Ve, M.St
3.3	Vorlage-Templates pro Kapiteltyp erstellen (Beschreibung, Wartung, Ersatzteile)	2 Tage	A.Sch, M.St

4. Content-Migration (Hauptaufwand)

Phase	Beschreibung	Aufwand (gesch.)	Beteiligte
4.1	CPC-Familie: 126 Produkte in 11 Gruppen – rST-Bausteine erstellen	2 Wochen	A.Sch, F.Ve
4.2	GERÜST-Familie: 263 Produkte in 14 Gruppen – rST-Bausteine erstellen	1 Wochen	A.Sch, F.Ve
4.3	ILS 2100-Familie: 396 Produkte in 27 Gruppen – rST-Bausteine erstellen	3 Wochen	A.Sch, F.Ve

4.4	OMNIFLO-Familie: 453 Produkte in 14 Gruppen – rST-Bausteine erstellen	4 Wochen	A.Sch, F.Ve
4.5	Trolleyprogramm: 191 Produkte in 4 Gruppen – rST-Bausteine erstellen	2 Wochen	A.Sch, F.Ve
4.6	KI-gestützte Textgenerierung: Vorlagen automatisiert befüllen (Beschreibungen, techn. Daten, Logische Schalter)	parallel	F.Ve, M.St
4.7	Bildmaterial: Fotos/Zeichnungen zuordnen, optimieren, Pfadstruktur aufbauen	parallel	A.Sch, F.Ve

5. Glossar & Mehrsprachigkeit

Phase	Beschreibung	Aufwand (gesch.)	Beteiligte
5.1	Glossar DE aufbauen (Fachbegriffe aus allen 5 Produktfamilien)	1 Woche	A.Sch, D.Fe, M.St
5.2	Glossar EN übersetzen	1 Woche	A.Sch, D.Fe, M.St, V.Fe
5.3	Stichproben-Übersetzung: 10% der Module auf EN migrieren	2 Wochen	A.Sch, D.Fe, F.Ve

6. ERP-Integration & Automatisierung

Phase	Beschreibung	Aufwand (gesch.)	Beteiligte
6.1	JSON-Export-Format mit ERP abstimmen (anlage.json-Schema)	2 Tage	M.St, A.Jakob
6.2	Validierungsskripte: meta.yaml-Konsistenz, fehlende Dateien/Übersetzungen	2 Tage	M.St
6.3	Hilfsskripte: dxf2svg, optimize_images	2 Tage	F.Ve

7. Pilotprojekt

Phase	Beschreibung	Aufwand (gesch.)	Beteiligte
7.1	Erste vollständige Anlagendokumentation (reales Kundenprojekt) generieren	1 Woche	A.Sch, M.St, F.Ve
7.2	Review mit Stakeholdern, Korrekturrunde	3 Tage	Stakeholder
7.3	Lessons Learned, Backlog-Pflege	1 Tag	F.Ve, A.Sch, M.St, D.Fe

Aufwandszusammenfassung

Phase	Dauer (geschätzt)
0 – Use-Case-Definition	~2 Wochen
1 – PoC	~1 Woche
2 – Base-Repo & Infrastruktur	~2 Wochen
3 – Produktprogramm-Strukturierung	~2 Wochen
4 – Content-Migration	~10-12 Wochen (2 Redakteure parallel + KI-Unterstützung)
5 – Glossar & Mehrsprachigkeit	~4 Wochen
6 – ERP-Integration	~1 Woche
7 – Pilotprojekt	~2 Wochen
Gesamt	~24 Wochen (6 Monate)

Anmerkungen:

- **Phase 4** ist der **Hauptaufwand**. Bei 1388 Produkten und 2 Redakteurinnen (A.Sch, F.Ve) parallel ergibt sich ca. 10 Wochen netto für die Content-Erstellung. Dies ist aktuell nur eine Schätzung aufgrund des *heutigen Produktprogrammes*. Wird als Detaillierungsgrad für den Kunden eine geringe Anzahl von Produkten gewählt (z.B. Nummern die als Rückmeldung bei Reparaturen genannt werden sollen), kann dieser auch drastisch reduziert sein
- KI-gestützte Textgenerierung und Standardisierung (F.Ve) kann den Aufwand in Phase 4 um geschätzt 30% reduzieren.
- Es sollten die bereits vorhandenen Glossare aus der Automation ebenfalls eingebunden werden.
- Phasen 4, 5 und 6 können teilweise parallelisiert werden, wodurch die Gesamtlaufzeit auf ca. 5 Monate verkürzt werden kann.
- Die Schätzung basiert auf ~20 Produkte/Woche pro Redakteur bei mittlerer Komplexität (Beschreibung, techn. Daten, Wartungshinweise als rST-Bausteine).
- Man sollte zu Beginn noch prüfen inwieweit das Team zur Überarbeitung der Inhalte anstatt durch D.Fetzer durch S.Hensch und Y.Wang unterstützt werden können, da D.Fetzer durch diverse anfragen an Konstruktionen ausgelastet wird