blocpress

document automation

Architecture Documentation

1. Einführung und Ziele

blocpress ist ein template-basierter Print-Server zur automatisierten Generierung professioneller Dokumente. Das System führt LibreOffice-Templates (ODT) mit strukturierten JSON-Daten zusammen und erzeugt Dokumente in den Formaten ODT, PDF und RTF.

1.1. Aufgabenstellung

Kernaufgabe: blocpress ermöglicht die automatisierte, qualitätsgesicherte und compliance-konforme Erstellung von Dokumenten aus Templates.

Wesentliche funktionale Anforderungen:

- **Template-Verwaltung:** Zentrale Speicherung und Versionierung von LibreOffice-Templates und wiederverwendbaren Bausteinen
- **Dokumentengenerierung:** Automatisierte Befüllung von Templates mit JSON-Daten über REST-API
  - User-Fields in Punkt-Notation (z.B. `kunde.name`)
  - Wiederholungsgruppen für Arrays (Sections und Tables)
  - IF-Bedingungen für dynamische Inhalte
  - Export in ODT, PDF und RTF
- **Workflow-gesteuerte Freigabe:** Mehrstufiger Freigabeprozess mit Vier-Augen-Prinzip (Designer ≠ Freigeber)
- **Compliance-Management:** Automatische Überwachung turnusmäßiger Reviews (1, 3 oder 5 Jahre)
- **Test-Framework:** Verwaltung von Testdaten, Baseline-PDFs und automatisierte Regressionstests
- **Content Search:** Performante Volltextsuche nach fachlichen Konstrukten über alle Templates

Treibende Kräfte:

- Transformation von manueller, fehleranfälliger Dokumentenerstellung zu automatisiertem

Workflow

- Sicherstellung inhaltlicher Aktualität durch turnusmäßige Reviews
- Konsistenz über alle Dokumente durch zentrale Template-Verwaltung
- Qualitätssicherung durch Freigabeprozess und Regressionstests

Referenzen:

- Solution Design Concept: blocpress — Vision, Value Creation Architecture, Information Architecture, Business Processes
- System Design Concept: blocpress — Systemziele, Systemarchitektur, Szenarien
- Element Design Concept: blocpress (SCS-Module) — Goals, Use Cases, Technical Functions, Entities

1.2. Qualitätsziele

Priorität	Qualitätsziel	Szenario	Kapitel
1	**Performance**	Dokumentengenerierung für Standard-Dokumente (bis 20 Seiten) innerhalb von 5 Sekunden. Suche über alle Templates innerhalb von 2 Sekunden.	10.2
2	**Zuverlässigkeit (Determinismus)**	Identische Eingaben (Template + JSON-Daten) erzeugen immer identische Ausgabedokumente. Essentiell für Regressionstests.	10.1
3	**Wartbarkeit**	Templates und Code müssen versioniert sein. Änderungen nachvollziehbar. Storage Abstraction Layer ermöglicht Wechsel zu S3.	9, 10.3
4	**Sicherheit**	Vier-Augen-Prinzip (Designer ≠ Freigeber) technisch erzwungen. RBAC für alle Funktionen. API-Rate-Limiting.	2.3, 10.4
5	**Skalierbarkeit**	Horizontal skalierbar: Bis zu 100 gleichzeitige Dokumentengenerierungen. Asynchrone Verarbeitung über Message Queue.	8.1, 10.5

1.3. Stakeholder

Rolle	Kontakt	Erwartungshaltung
Template-Designer	Fachabteilungen	Einfaches Erstellen von Templates in vertrauter LibreOffice-Umgebung. Klare Validierungsmeldungen. Eigene Arbeitsumgebung (blocpress-workbench).
Prüfer (QM / Freigeber / Compliance)	QS- / Compliance-Abteilung	Strukturierter Freigabeprozess mit Vier-Augen-Prinzip. Testmöglichkeiten mit Beispieldaten. Übersichtliche Darstellung fälliger Compliance-Reviews. Eigene Arbeitsumgebung (blocpress-proof).
Entwicklungsteams	IT-Abteilung	Stabile REST-API für Dokumentengenerierung (blocpress-render). Gute Performance (< 5s). Klare Fehlermeldungen.
Administratoren	IT-Betrieb	Einfache Installation als Docker-Container. Monitoring-Integration. RBAC-Verwaltung (blocpress-admin).
Architekten	IT-Architektur	Klare Architektur. Modularer Aufbau. Technologie-Stack: Quarkus, PostgreSQL, LibreOffice.
Projektleitung	Management	Einhaltung von Compliance-Vorgaben. Wirtschaftlicher Betrieb (< 5.000 Dokumente in PostgreSQL).

2. Randbedingungen

2.1. Technische Randbedingungen

Detaillierte Constraints: Constraints im Element Design Concept | Constraints im Solution Design Concept

Randbedingung	Erläuterung
Quarkus Framework	Alle SCS-Module müssen mit Quarkus implementiert werden (schnelle Startup-Zeit, geringer Memory-Footprint, native Kompilierung möglich).
LibreOffice ≥ 24	LibreOffice Version 24 oder höher muss headless in blocpress-render integriert sein. Ältere Versionen nicht unterstützt.
PostgreSQL ≥ 18	Datenbank muss PostgreSQL Version 18 oder höher sein. JSONB-Indizes erforderlich.
Docker-Container	blocpress muss als Docker-Container betreibbar sein. Kubernetes-Deployment optional.
Template-Format	Nur ODT und OTT als Template-Eingabeformat. DOCX/DOC nicht unterstützt.
Export-Formate	Nur ODT, PDF und RTF als Ausgabeformat. DOCX/HTML nicht unterstützt.
Elasticsearch ≥ 7.x	Content Search Engine basiert auf Elasticsearch 7.x oder höher.
Message Queue	RabbitMQ oder Kafka für asynchrone Dokumentengenerierung.

2.2. Organisatorische Randbedingungen

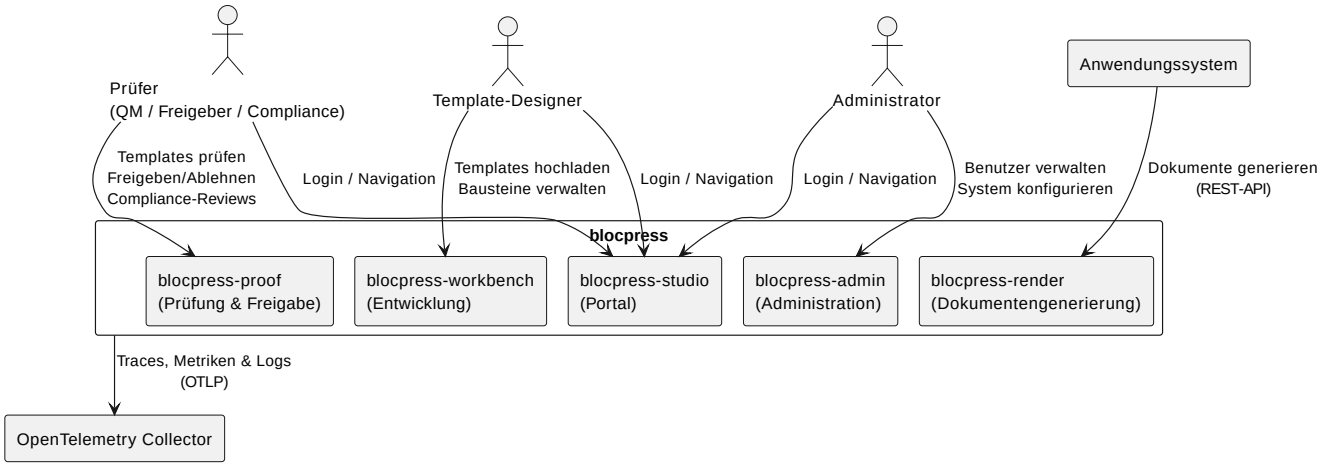
Randbedingung	Erläuterung
Vier-Augen-Prinzip	Designer darf nicht gleichzeitig Freigeber sein. Technisch erzwungen durch TF-2.
Compliance-Reviews	Templates mit rechtlicher/regulatorischer Relevanz müssen nach 1, 3 oder 5 Jahren zur Review vorgelegt werden.
Freigabe-Pflicht	Templates ohne Status "Freigegeben" dürfen nicht für Produktiv-Generierung verwendet werden.
JWT-basierte Authentifizierung	Benutzer-Authentifizierung erfolgt über JWT-Validierung. Jedes SCS-Modul prüft Signatur, Issuer und Ablaufdatum gegen einen konfigurierten Public Key. Kein externer Identity Provider zur Laufzeit erforderlich.

2.3. Konventionen

Konvention	Erläuterung
RESTful API Design	REST-API folgt RESTful Prinzipien: Ressourcen-orientiert, HTTP-Verben, JSON-Response.
JSON-Punkt-Notation	User-Fields referenzieren JSON-Pfade in Punkt-Notation: `kunde.name`, `vertrag.positionen[0].preis`
Storage Abstraction Layer	Zugriff auf Binärdaten nur über `StorageService` Interface. Ermöglicht zukünftigen Wechsel zu S3.
Audit-Logging	Alle Workflow-Änderungen, Freigaben und Reviews müssen audit-log-tauglich protokolliert werden.

3. Kontextabgrenzung

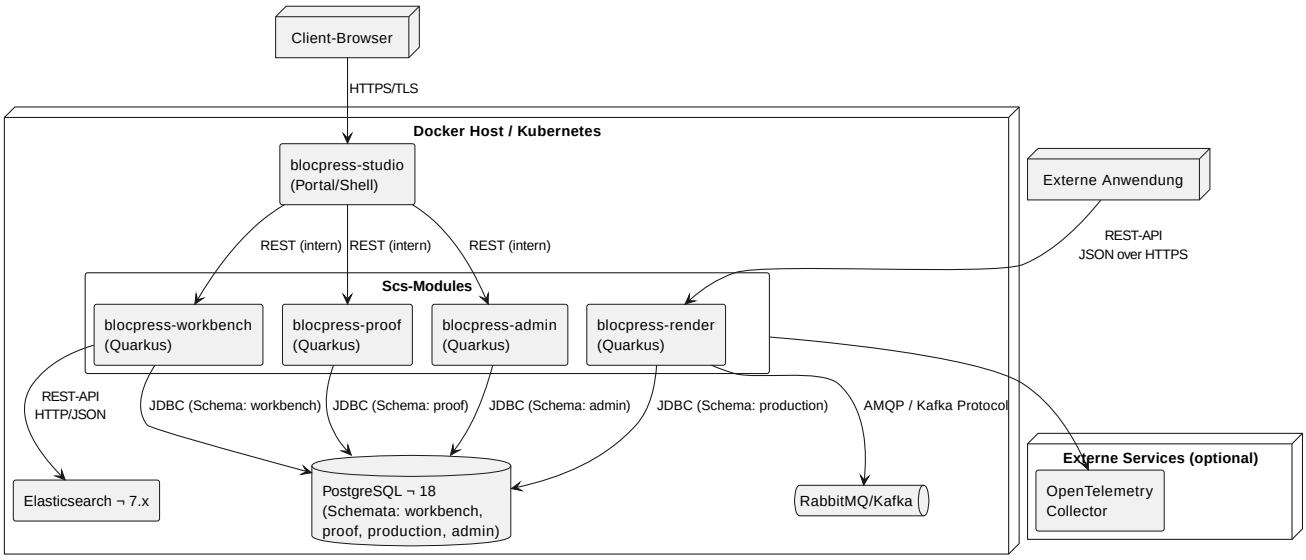
3.1. Fachlicher Kontext



Kommunikationsbeziehungen:

Partner	Eingabe	Ausgabe
Template-Designer	ODT-Templates, Bausteine, Metadaten	Validierungsergebnisse, Template-Status, Vorschau-Dokumente
Prüfer (QM / Freigeber / Compliance)	Prüfkommentare, Freigabe-/Ablehnungsentscheidung, Review-Ergebnisse, JSON-Testdaten, Baseline-Updates	Templates zur Prüfung, Testdokumente, fällige Reviews, Test-Ergebnisse, PDF-Vergleiche
Administrator	Benutzerdaten, Rollen, System-Konfiguration	Benutzerliste, Audit-Logs, System-Status
Anwendungssystem	Template-ID, JSON-Daten, gewünschtes Format	Generiertes Dokument (ODT/PDF/RTF), Generierungs-ID, Status
OpenTelemetry Collector (optional)	(keine Eingabe)	Traces, Metriken und Logs via OTLP-Protokoll

3.2. Technischer Kontext



Technische Schnittstellen:

Schnittstelle	Protokoll/Technologie	Mapping fachlicher Daten
Browser → blocpress-studio	HTTPS, Web Components (Dynamic Import)	Portal-Shell lädt Micro-Frontends der SCS-Module als Web Components
blocpress-studio → SCS-Module (intern)	REST-API, JSON (intern, nicht öffentlich)	UI-Aktionen werden an das jeweilige SCS-Modul delegiert
Externe Anwendung → blocpress-render	HTTPS, REST-API, JSON	Dokumentengenerierung: `POST /api/documents/generate` (einzige öffentliche API)
SCS-Module → PostgreSQL	JDBC, PostgreSQL Wire Protocol	Jedes Modul nutzt eigenes Schema (workbench, proof, production, admin)
Render → RabbitMQ/Kafka	AMQP (RabbitMQ) / Kafka Protocol	Asynchrone Dokumentengenerierung, Regressionstests
blocpress-workbench → Elasticsearch	REST-API, HTTP/JSON	Template-Indexierung, Volltextsuche (Content Search)
SCS-Module → OpenTelemetry Collector	OTLP (gRPC Port 4317 / HTTP Port 4318)	Traces (Request-Verarbeitung), Metriken (Latenz, Fehlerraten), Logs

4. Lösungsstrategie

4.1. Technologieentscheidungen

Entscheidung	Begründung	Qualitätsziel
Self-Contained Systems (SCS)	Fachliche Zerlegung in unabhängig deploybare Module mit eigener UI, eigenem Backend und eigenem Datenbank-Schema. Erleichtert Wartung und Team-Ownership.	Wartbarkeit, Skalierbarkeit
Quarkus Framework	Schnelle Startup-Zeit, geringer Memory-Footprint, Cloud-native, native Kompilierung möglich. Jedes SCS-Modul ist eine eigenständige Quarkus-Anwendung.	Performance, Skalierbarkeit
Micro-Frontends (Web Components)	Jedes SCS-Modul liefert eigene UI als Web Components (Custom Elements mit Shadow DOM). Portal-Shell (blocpress-studio) lädt diese via Dynamic Import.	Wartbarkeit, Unabhängigkeit
LibreOffice headless ≥ 24	Ausgereiftes ODT-Template-Processing, Export in ODT/PDF/RTF, kostenlos, Open Source	Funktionale Eignung, Kosteneffizienz
PostgreSQL ≥ 18 — Multi-Schema	Reife relationale DB. Jedes SCS-Modul nutzt ein eigenes Schema (workbench, proof, production, admin) in derselben Datenbank. Identische Tabellenstrukturen für Templates in workbench/proof/production ermöglichen kopie-basierte Stufenübergabe.	Zuverlässigkeit, Wartbarkeit
Storage Abstraction Layer	Ermöglicht zukünftigen Wechsel zu S3 ohne Code-Änderungen in Business-Logik	Wartbarkeit, Flexibilität
Elasticsearch für Content Search	Hochperformante Volltextsuche, Relevanz-Ranking, Highlighting	Performance (< 2s Suchzeit)
Message Queue (RabbitMQ/Kafka)	Asynchrone Verarbeitung zeitaufwändiger Operationen, Entkopplung, Skalierbarkeit	Performance, Skalierbarkeit
Docker-Container	Jedes SCS-Modul als eigener Container, unabhängig skalierbar	Übertragbarkeit, Betriebseffizienz

4.2. Top-Level-Zerlegung

Architekturmuster: Self-Contained Systems (SCS) mit Micro-Frontend-Integration

Die Anwendung ist in fünf fachlich geschnittene Module zerlegt:

SCS-Modul	Verantwortung	Datenstufe
blocpress-studio	Portal-Shell (Quarkus): Navigation, Authentifizierung (JWT), Integration der Micro-Frontends via Dynamic Import. Keine eigene Business-Logik.	—
blocpress-workbench	Template-Entwicklung: Upload, Validierung, Bearbeitung, Bausteinverwaltung, Content Search, Vorschau-Generierung, **Dashboard mit Status-Filtern und Workflow** (UC-5). Arbeitsumgebung für Template-Designer.	Schema `workbench`
blocpress-proof	Prüfung und Freigabe: Workflow (Vier-Augen-Prinzip), Testdaten, Regressionstests, Compliance-Reviews. Arbeitsumgebung für Prüfer.	Schema `proof`
blocpress-render	Dokumentengenerierung: Einzige öffentliche REST-API. Liest freigegebene Templates aus dem Production-Schema. Stateless.	Schema `production` (read-only)
blocpress-admin	Administration: Benutzerverwaltung, Rollenzuweisung, Systemkonfiguration, Audit-Logs.	Schema `admin`

Datenstufen und Stufenübergabe:

Templates durchlaufen drei Stufen mit kopie-basierter Übergabe:

1. **workbench** → Designer entwickelt und testet Templates
2. **proof** → Prüfer führt Qualitätssicherung, Freigabe und Compliance-Reviews durch
3. **production** → Freigegebene Templates stehen für die Dokumentengenerierung bereit

Die Übergabe erfolgt als Kopie: Das Template wird vom workbench-Schema in das proof-Schema kopiert (Übergabe in den Review-Prozess), nach Freigabe vom proof-Schema in das production-Schema. Für turnusmäßige Erneuerungen kann die aktuelle Baseline aus production in das proof-Schema gezogen werden.

Micro-Frontend-Architektur:

- blocpress-studio ist die Portal-Shell (HTML-Seite mit Navigation und Auth-Handling)

- Jedes SCS-Modul liefert seine UI als Web Component (Custom Element mit Shadow DOM)
- Studio lädt die Web Components per Dynamic Import von den jeweiligen SCS-Modulen (z.B. `import('http://workbench:8081/components/bp-workbench.js')`)
- JWT wird als Property an jedes Web Component weitergereicht: `<bp-workbench jwt="${token}"></bp-workbench>`
- Shadow DOM kapselt CSS und DOM — Module beeinflussen sich nicht gegenseitig

Interne Kommunikation:

- SCS-Module kommunizieren untereinander über interne REST-APIs (nicht öffentlich exponiert)
- blocpress-render kann Templates sowohl aus dem production-Schema lesen als auch direkt per REST von workbench/proof empfangen (für Vorschau/Test-Generierung)

4.3. Erreichung wichtigster Qualitätsziele

Q1: Performance

- LibreOffice headless läuft in blocpress-render (keine IPC-Overhead)
- Asynchrone Dokumentengenerierung über Message Queue (nicht blockierend)
- Elasticsearch für schnelle Volltextsuche
- PostgreSQL-Indizes auf häufig abgefragte Spalten (template_id, status, user_id)

Q2: Zuverlässigkeit (Determinismus)

- LibreOffice-Verarbeitung strikt sequenziell für gleiche Template-Version
- Keine Zufallswerte, keine zeitabhängigen Werte in Template-Processing
- Versionierung von Templates: Gleiche Version = gleiche Binärdaten
- Regressionstests validieren Determinismus automatisch

Q3: Wartbarkeit

- SCS-Architektur: Module unabhängig entwickelbar, testbar und deploybar
- Eigene Datenbank-Schemata: Keine Datenkopplung zwischen Modulen
- Storage Abstraction Layer (`StorageService` Interface) entkoppelt Business-Logik von Storage-Implementierung
- OpenTelemetry für Traces, Metriken und Logs (optional, vendor-neutral)

Q4: Sicherheit

- TF-2 (Workflow-Status ändern) prüft technisch: `userId != template.erstellerId`
- RBAC: Rollen im admin-Schema, Prüfung bei jedem API-Call
- API-Rate-Limiting für blocpress-render: 1.000 Requests/h pro API-Key
- HTTPS/TLS für alle externen Verbindungen

- JWT-Weitergabe an Web Components über Properties (kein LocalStorage)

Q5: Skalierbarkeit

- SCS-Module unabhängig skalierbar: blocpress-render kann separat horizontal skaliert werden
- Message Queue entkoppelt Request-Handling von Dokumentengenerierung
- PostgreSQL Connection Pooling (HikariCP) pro Modul
- Caching von Templates (Caffeine Cache) in blocpress-render

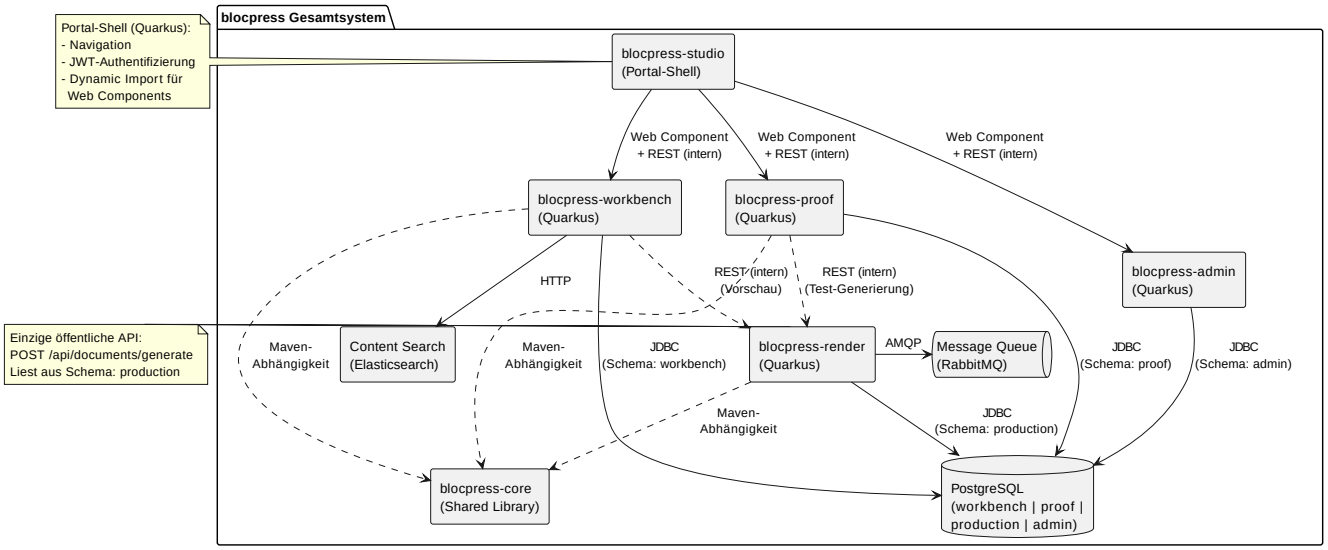
4.4. Organisatorische Entscheidungen

- **Entwicklungsprozess:** Agil Kanban
- **CI/CD:** Automatische Tests bei jedem Commit, je SCS-Modul
- **Dokumentation:** arc42 für Architektur, Javadoc für Code, OpenAPI für REST-API
- **Team-Ownership:** Jedes SCS-Modul kann von einem eigenen Team verantwortet werden

5. Bausteinsicht

Detaillierte Beschreibung der Bausteine: Technical Functions | Technical Interfaces | Use Cases

5.1. Whitebox Gesamtsystem



Begründung:

Die Zerlegung folgt dem Self-Contained-Systems-Ansatz (SCS):

- **blocpress-studio** ist die Portal-Shell — lädt Micro-Frontends der fachlichen Module als Web Components
- **blocpress-workbench** enthält die Logik für Template-Entwicklung (Dev-Stufe)
- **blocpress-proof** enthält die Logik für Prüfung, Freigabe und Compliance (Test/QA-Stufe)
- **blocpress-render** ist die einzige öffentliche API für Dokumentengenerierung (Production-Stufe)
- **blocpress-admin** verwaltet Benutzer, Rollen und Systemkonfiguration
- **PostgreSQL** nutzt getrennte Schemata pro Modul — identische Tabellenstrukturen für Templates ermöglichen kopie-basierte Stufenübergabe
- **Message Queue** ermöglicht asynchrone Verarbeitung
- **Elasticsearch** ist spezialisiert auf Volltextsuche (genutzt von blocpress-workbench)

Enthaltene Bausteine:

Name	Verantwortung
blocpress-studio	Portal-Shell (Quarkus): Authentifizierung (JWT-Handling), Navigation, Laden der Micro-Frontends via Dynamic Import. Keine eigene Business-Logik — liefert statische HTML/JS-Assets aus und lädt Web Components dynamisch von den SCS-Modulen.

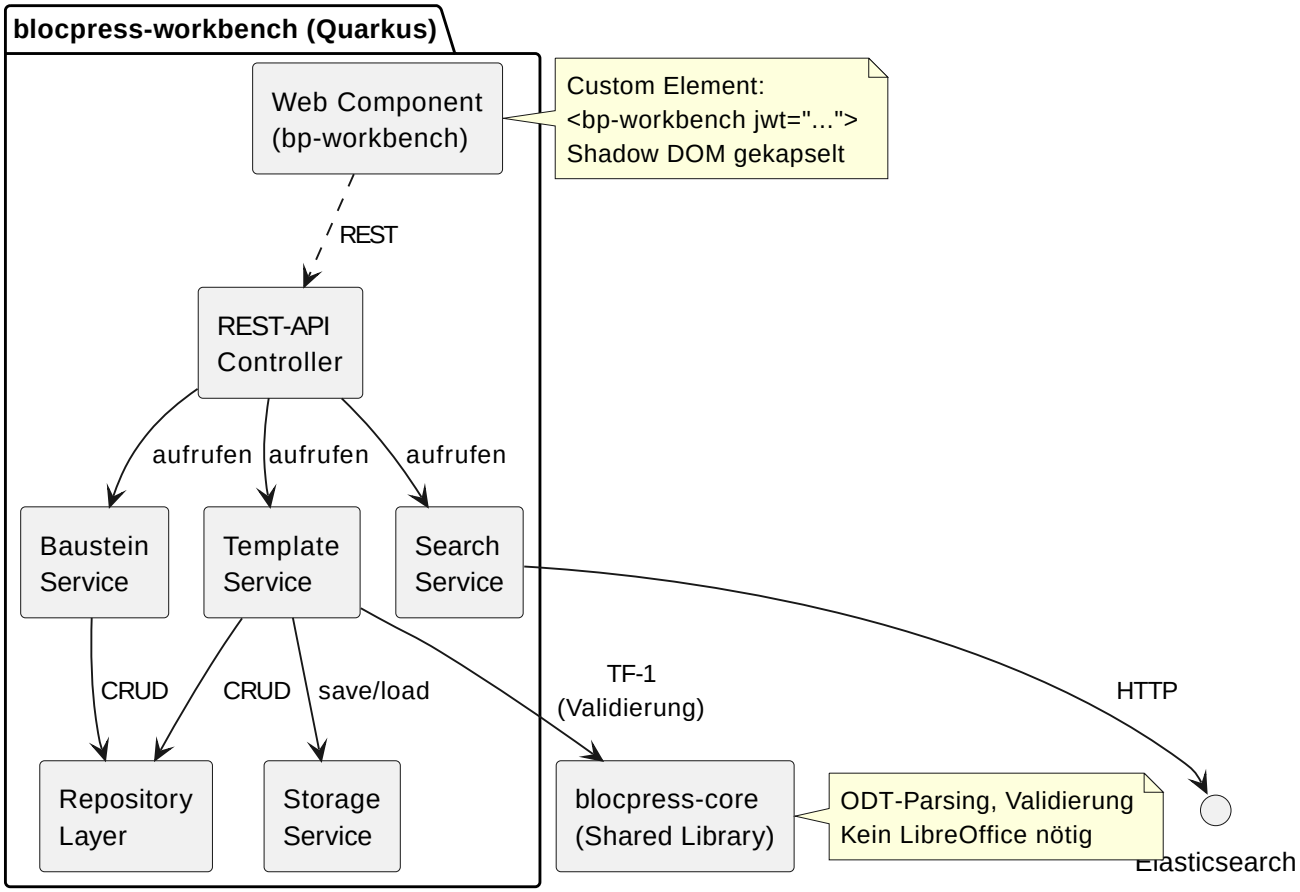
Name	Verantwortung
blocpress-workbench	Template-Entwicklung: Upload, Validierung (TF-1 via blocpress-core), Bearbeitung, Bausteinverwaltung, Content Search, Vorschau-Generierung. Use Cases: UC-1 bis UC-5, UC-19 bis UC-21.
blocpress-proof	Prüfung und Freigabe: Workflow (TF-2, Vier-Augen-Prinzip), Testdaten (UC-14 bis UC-18), Regressionstests (TF-6), Compliance-Reviews (TF-7, UC-11 bis UC-13), Freigabe (UC-6 bis UC-9). Stufenübergabe proof → production.
blocpress-render	Dokumentengenerierung: Einzige öffentliche REST-API (TI-1). Liest freigegebene Templates aus Schema `production`. Nutzt blocpress-core (Merge-Pipeline) und LibreOffice (Format-Export). Stateless, horizontal skalierbar.
blocpress-admin	Administration: Benutzerverwaltung (UC-22 bis UC-24), Rollenzuweisung, Systemkonfiguration, Audit-Logs.
PostgreSQL Datenbank	Persistente Speicherung mit getrennten Schemata: `workbench`, `proof`, `production`, `admin`. Identische Template-Tabellenstrukturen in workbench/proof/production.
Message Queue	Asynchrone Verarbeitung zeitaufwändiger Operationen (Dokumentengenerierung, Regressionstests). Entkopplung.
blocpress-core	Shared Java-Library (Maven-Abhängigkeit, kein eigenständiger Container): ODT-Parsing (odfdom), Template-Validierung, Merge-Pipeline (Text-Block-Expansion, Bedingungen, Schleifen, Feldersetzung). Wird von blocpress-workbench (Validierung) und blocpress-render (Merge + Generierung) genutzt. Keine LibreOffice-Abhängigkeit.
Content Search (Elasticsearch)	Indexierung von Template-Inhalten, Volltextsuche nach fachlichen Konstrukten, Highlighting. Genutzt von blocpress-workbench.

Wichtige Schnittstellen:

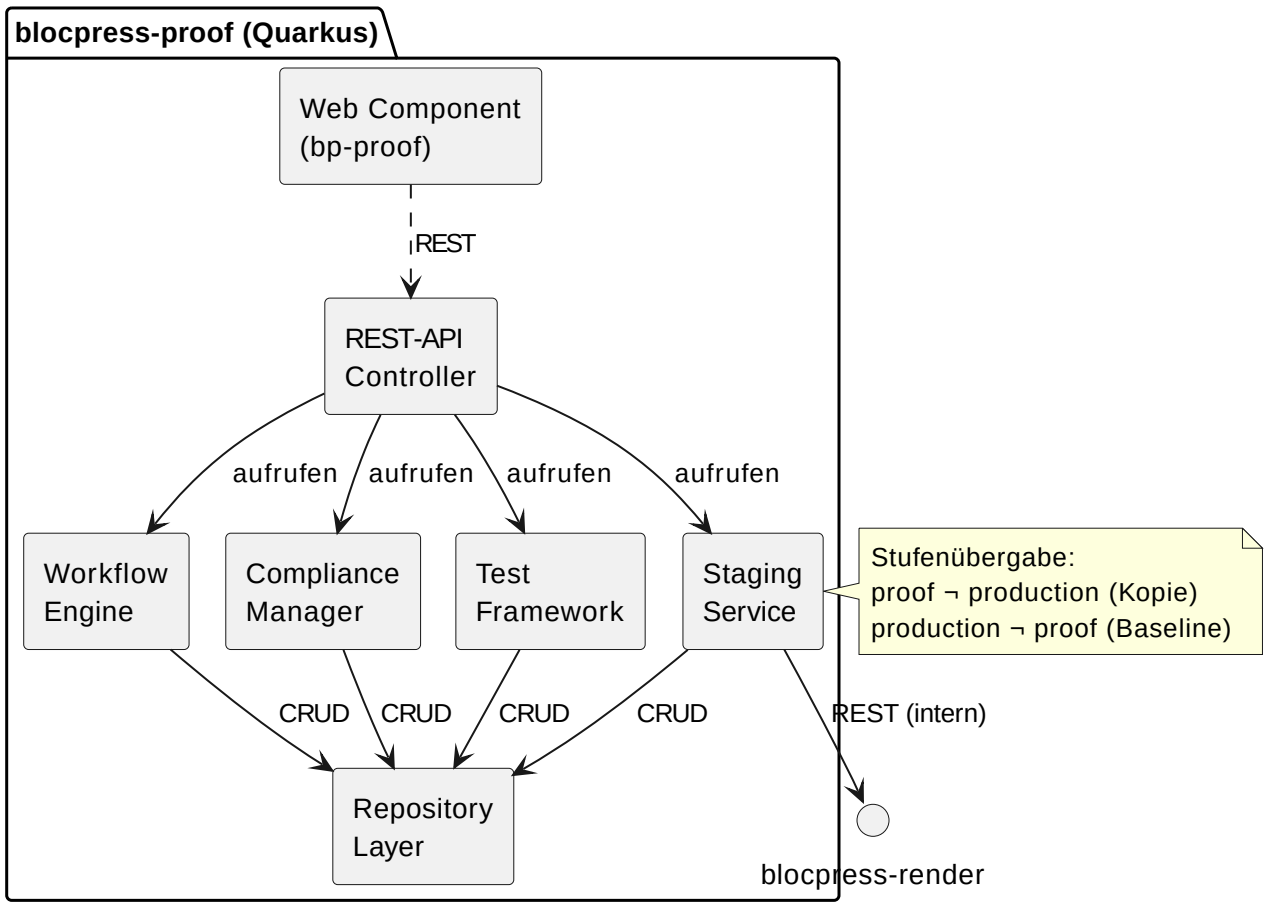
- **TI-1 (REST-API):** Öffentliche API von blocpress-render für externe Anwendungen
- **Interne REST-APIs:** Kommunikation zwischen SCS-Modulen (nicht öffentlich exponiert)
- **TI-2 (Datenbank-Interface):** JDBC-Zugriff pro Modul auf eigenes PostgreSQL-Schema
- **TI-5 (Message Queue Interface):** AMQP-Zugriff auf RabbitMQ
- **TI-7 (Elasticsearch API):** HTTP-REST-Zugriff auf Elasticsearch

5.2. Ebene 2

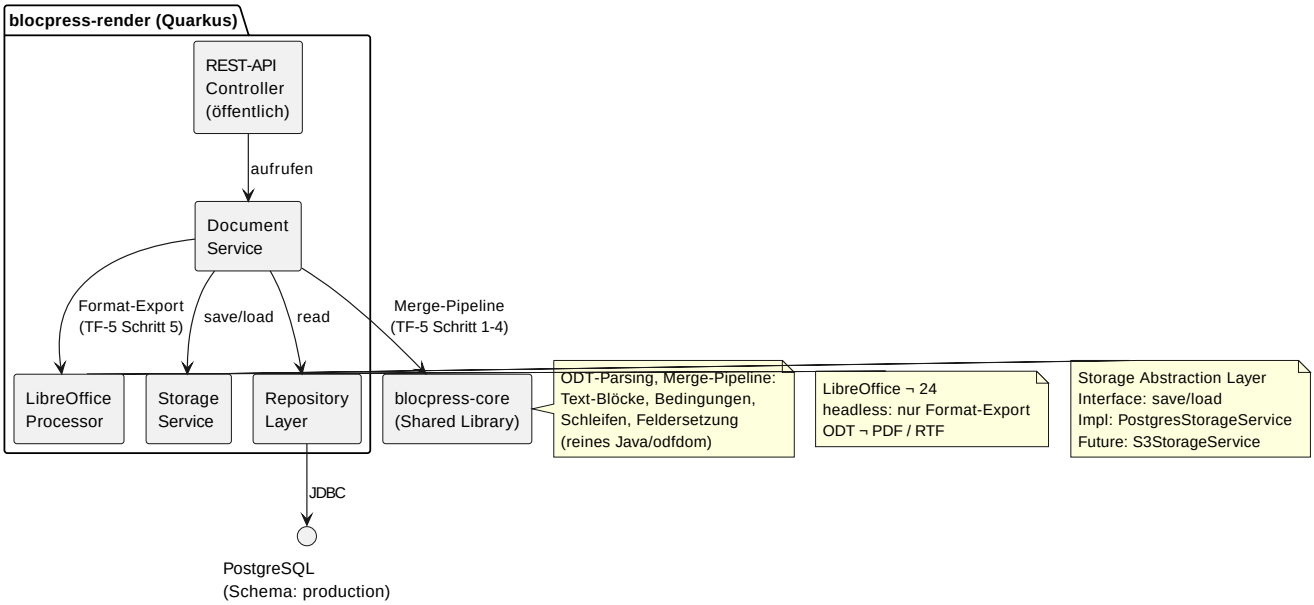
5.2.1. Whitebox blocpress-workbench



5.2.2. Whitebox blocpress-proof



5.2.3. Whitebox blocpress-render



Begründung der internen Zerlegung:

Jedes SCS-Modul folgt intern einer Schichtenarchitektur mit Dependency Inversion:

- REST-API Controller delegieren an fachliche Services
- Fachliche Services orchestrieren Technical Functions
- Technical Functions nutzen Infrastructure Services (LibreOffice, Storage, Repository)
- Storage Abstraction Layer ermöglicht Austausch der Storage-Implementierung

5.2.4. Blackbox blocpress-core (Shared Library)

Zweck/Verantwortung:

Gemeinsame Java-Library für ODT-Parsing, Template-Validierung und Merge-Pipeline. Wird als Maven-Abhängigkeit von blocpress-workbench und blocpress-render genutzt. Keine LibreOffice-Abhängigkeit — rein odfdom-basiert.

Schnittstellen:

- `RenderEngine.mergeTemplate(URL template, JsonNode data): OdtTemplateDocument` — Merge-Pipeline (Text-Block-Expansion, Bedingungen, Schleifen, Feldersetzung)
- `TemplateValidator.validate(byte[] odtBinary): ValidationResult` — Struktur-Analyse, User-Fields, Wiederholungsgruppen, IF-Bedingungen extrahieren
- `OdtTemplateDocument` / `OdtTemplateElement` — Abstraktionen über odfdom

Qualitäts-/Leistungsmerkmale:

- Deterministisch: Gleiche Eingabe → gleiche Ausgabe (Merge-Ergebnis)
- Kein externes Prozess-Management: Reine In-Process-Verarbeitung
- Testbar ohne LibreOffice-Installation

Ablageort/Datei:

- Maven-Modul: `blocpress-core`
- Package: `io.github.flaechsig.blocpress.core`
- Hauptklassen: `RenderEngine`, `OdtTemplateDocument`, `OdtTemplateElement`, `JexlConditionEvaluator`

5.2.5. Blackbox LibreOffice Processor

Zweck/Verantwortung:

Format-Konvertierung von ODT-Dokumenten nach PDF und RTF via LibreOffice headless. Wird in blocpress-render (Dokumentengenerierung) eingesetzt. Blocpress-workbench nutzt LibreOffice nicht — die Validierung läuft vollständig über blocpress-core.

Schnittstellen:

- `exportDocument(byte[] mergedOdt, OutputFormat format): byte[]` — Konvertiert ein bereits gemergtes ODT in das Zielformat (PDF/RTF/ODT)

Qualitäts-/Leistungsmerkmale:

- Deterministisch: Gleiche Eingabe → gleiche Ausgabe
- Performance: < 5s für Standard-Dokumente (20 Seiten)
- Thread-Safe: Mehrere parallele LibreOffice-Instanzen möglich

Ablageort/Datei:

- Maven-Modul: `blocpress-render`
- Package: `io.github.flaechsig.blocpress.render`
- Hauptklasse: `LibreOfficeProcessor`

5.2.6. Blackbox Storage Service

Zweck/Verantwortung:

Abstraktionsschicht für Speicherung von Binärdaten (Templates, generierte Dokumente, Test-PDFs). Wird in jedem SCS-Modul mit eigenem Schema eingesetzt.

Schnittstellen:

- `save(byte[] data, String key): StorageId`
- `load(StorageId id): byte[]`
- `delete(StorageId id): void`

Qualitäts-/Leistungsmerkmale:

- Austauschbar: Interface erlaubt Wechsel von PostgreSQL zu S3 ohne Code-Änderungen in Business-Logik

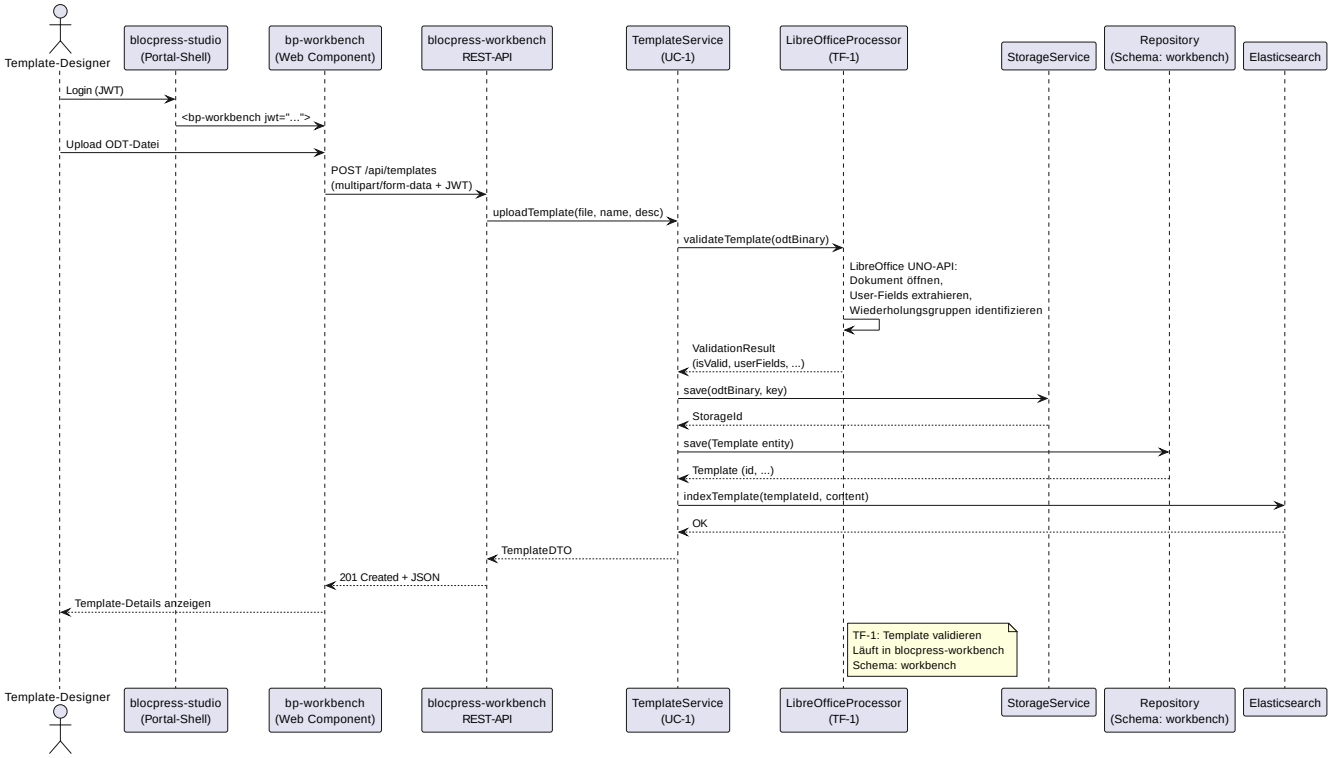
- Performance: Optimiert für < 5.000 Dokumente in PostgreSQL
- Transaktional: Speicherung in PostgreSQL erfolgt transaktional mit Metadaten

6. Laufzeitsicht

Systemszenarien: System Scenarios im System Design Concept | Business Processes: Business Processes im Solution Design Concept

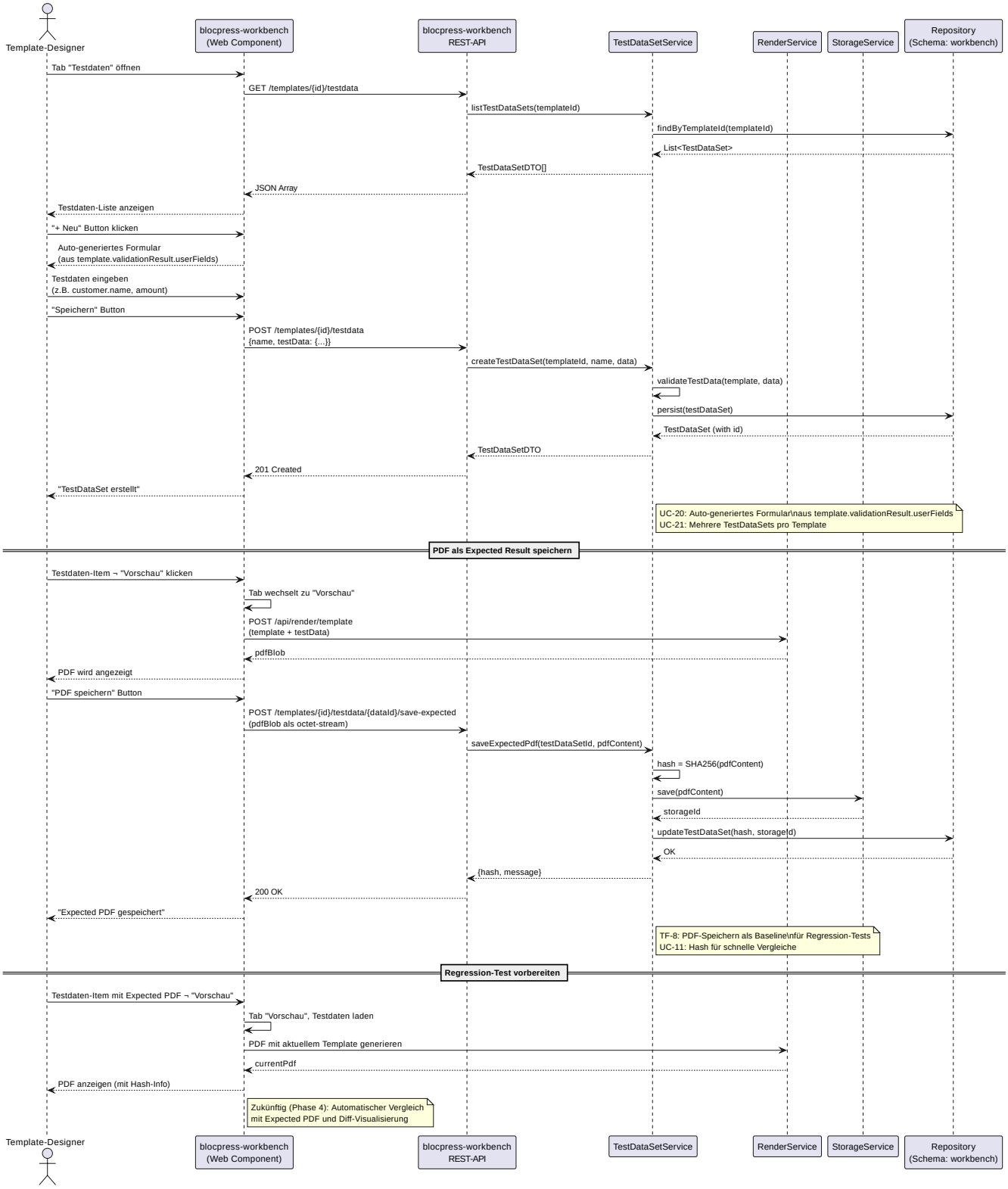
6.1. Template hochladen und validieren

Szenario: Template-Designer lädt neues ODT-Template in blocpress-workbench hoch (UC-1)



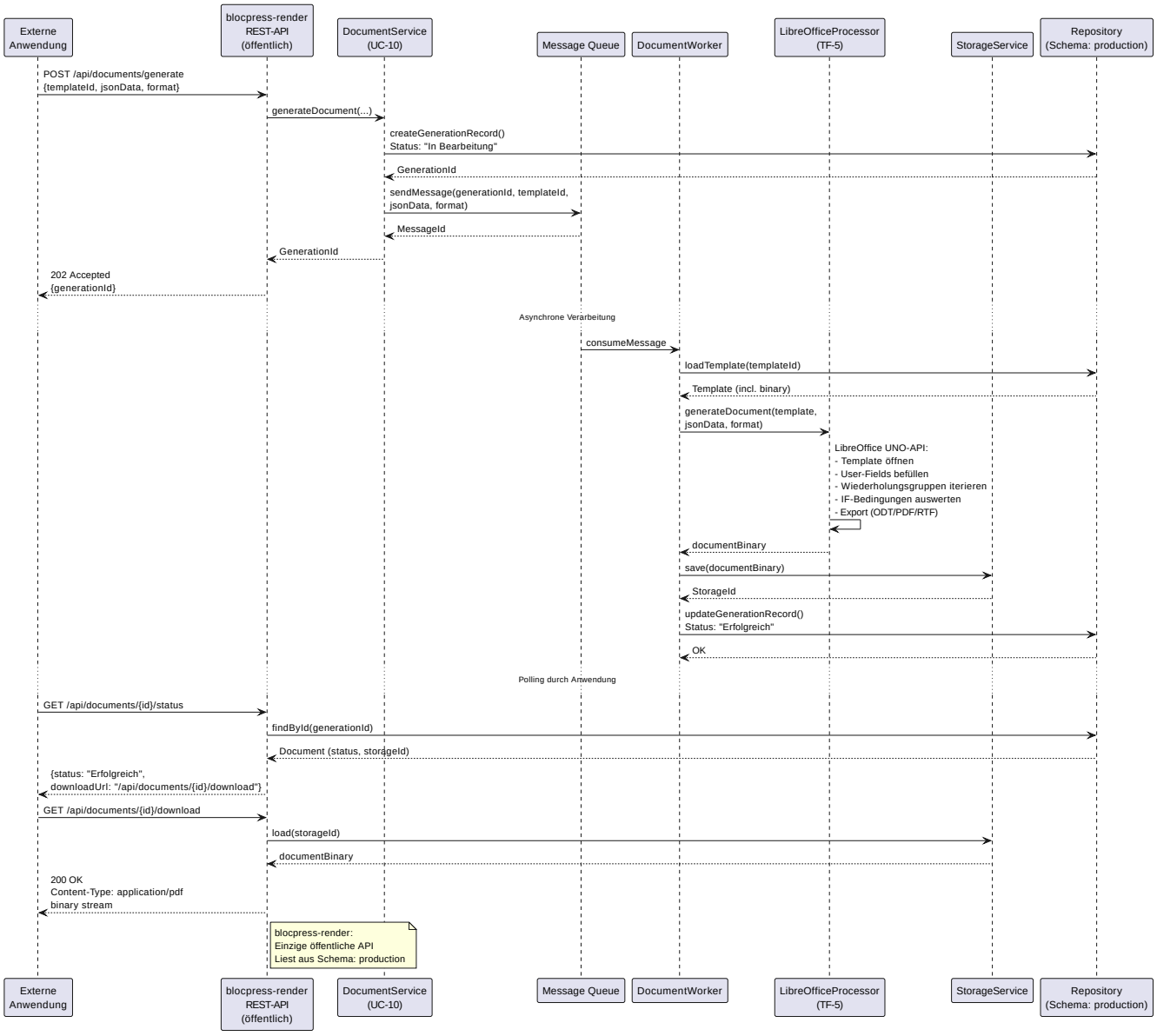
6.2. TestDataSet-Verwaltung und lokale Regression-Tests (blocpress-workbench)

Szenario: Template-Designer erstellt Test-Datensätze und speichert PDF-Baselines für lokale Regression-Tests (UC-20, UC-21, TF-8, UC-11)



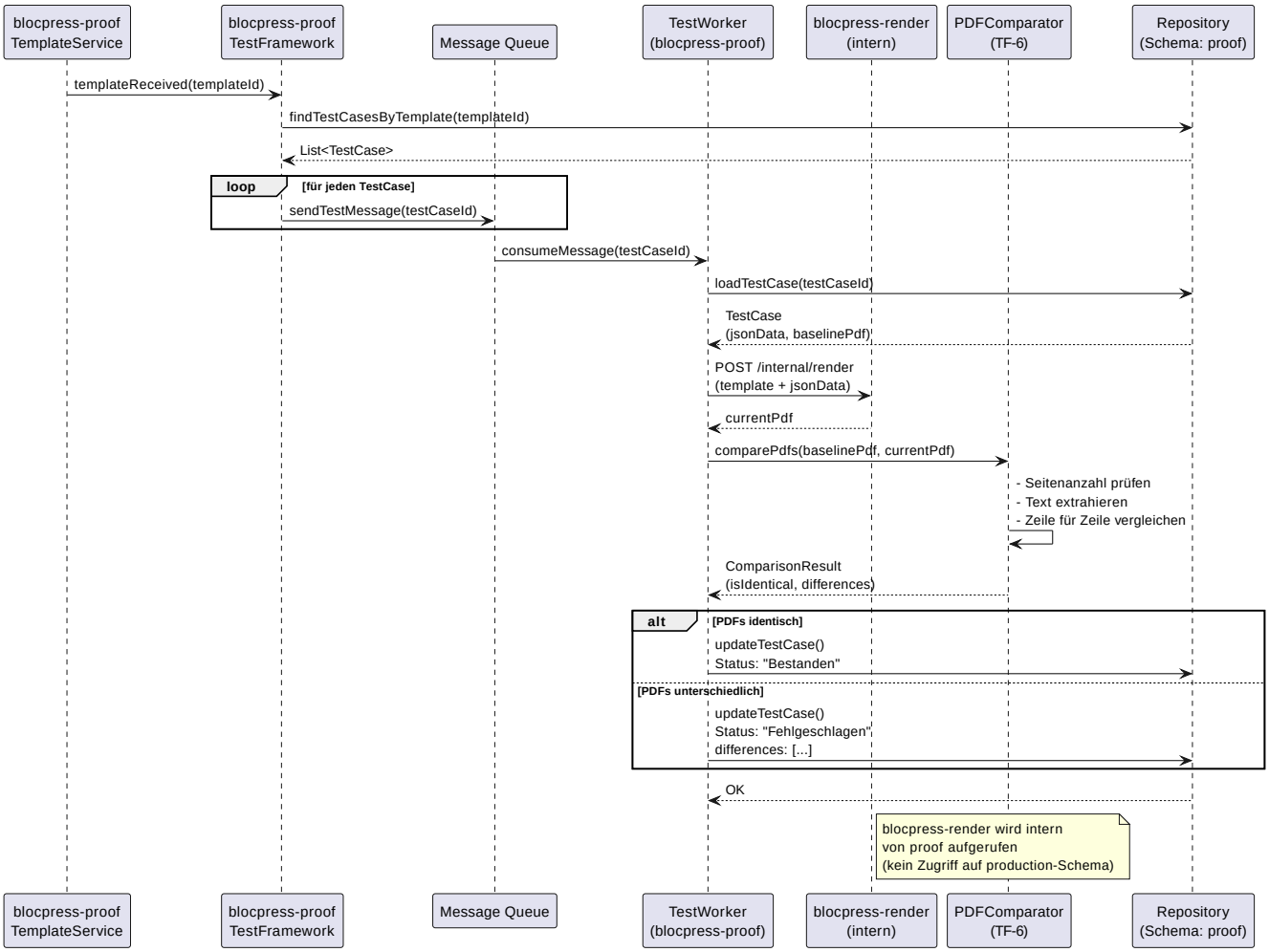
6.3. Dokument über API generieren (asynchron)

Szenario: Externe Anwendung generiert Dokument über blocpress-render (UC-10, TF-5)



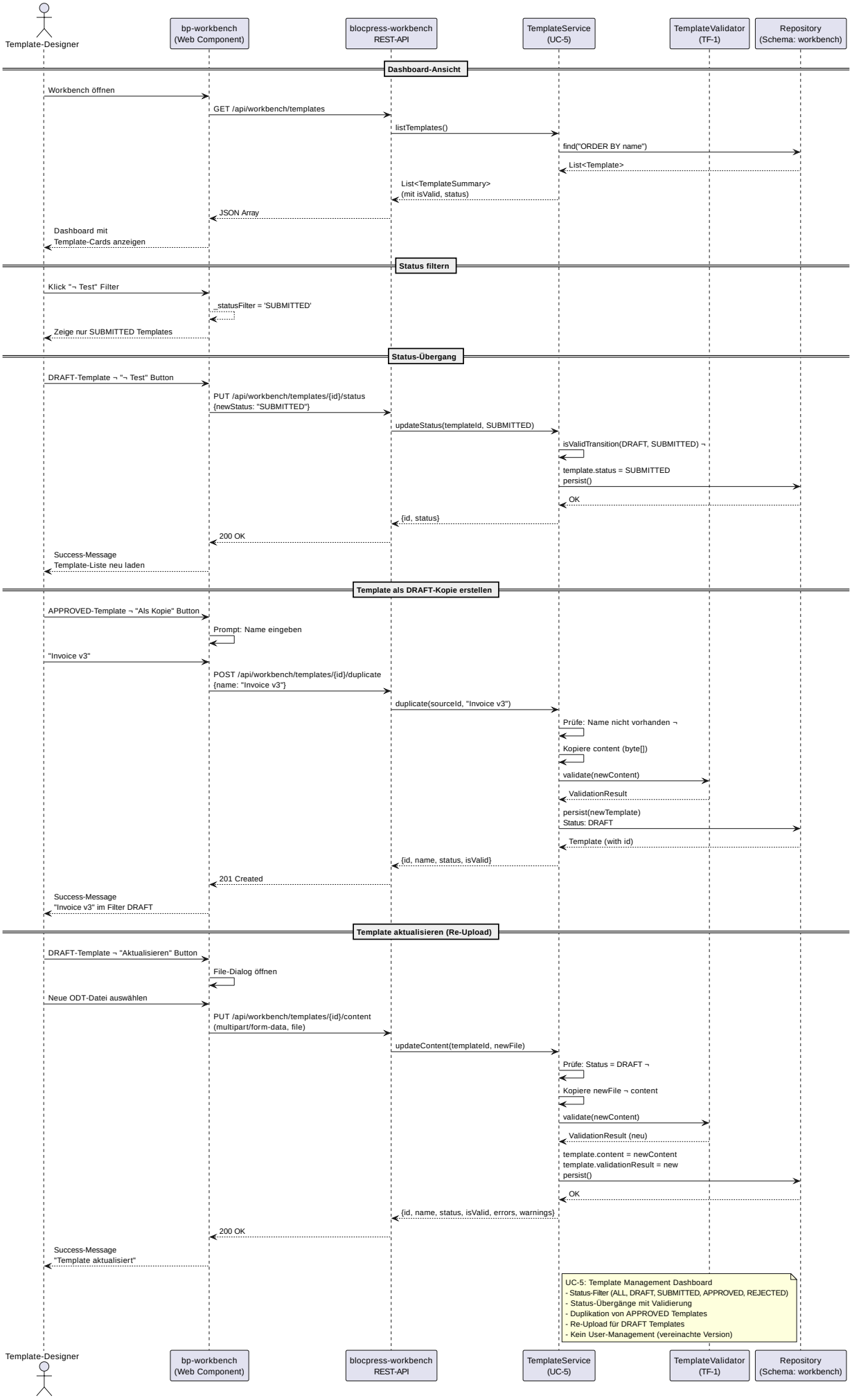
6.4. Automatischer Regressionstest

Szenario: Nach Übergabe eines Templates in blocpress-proof werden Regressionstests ausgeführt (UC-16, TF-6)



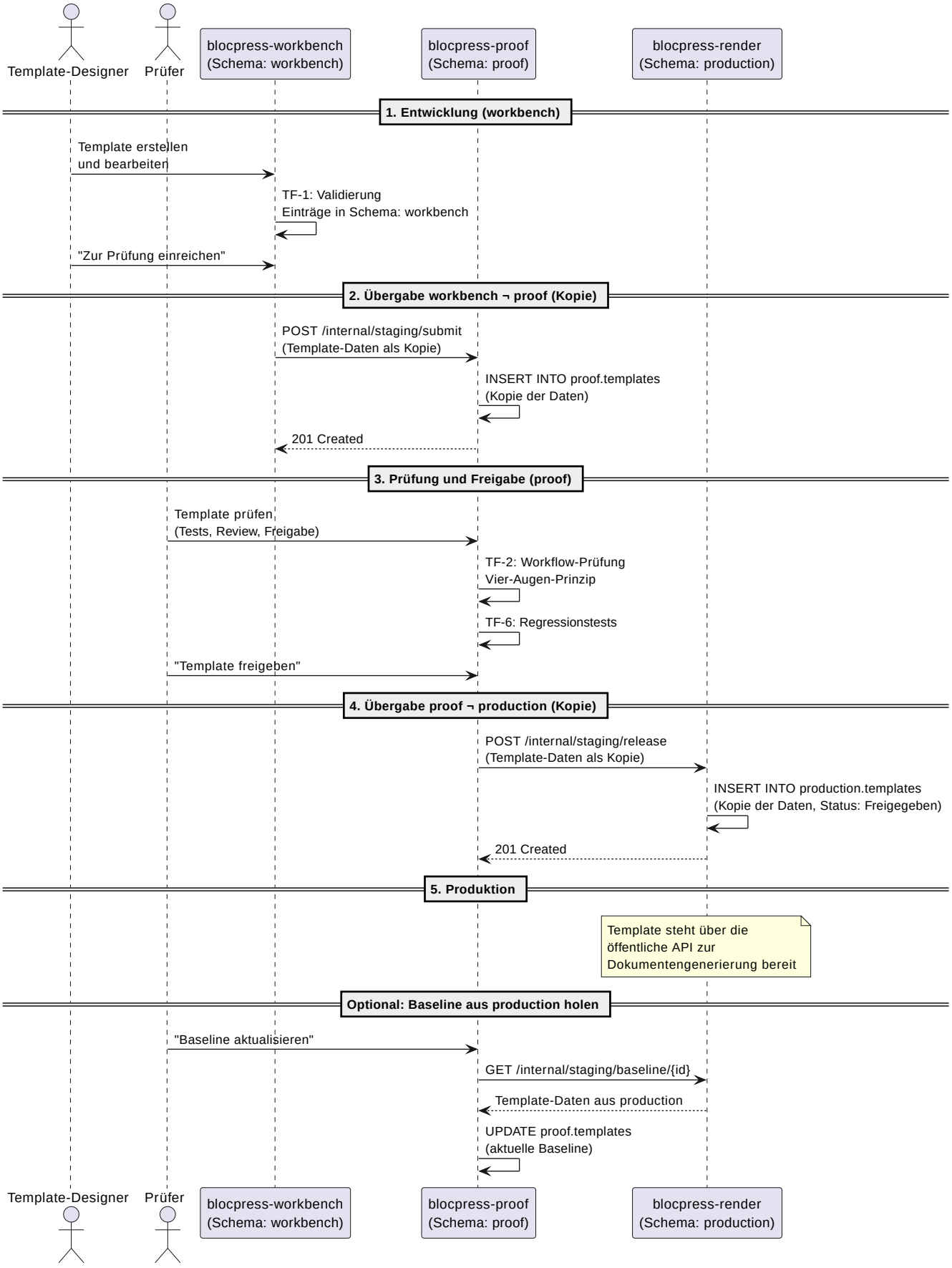
6.5. Template Management Dashboard — Status-Filter und Workflow (UC-5)

Szenario: Template-Designer verwaltet Templates über Dashboard mit Status-Filtern, führt Status-Übergänge durch, dupliziert Produktiv-Templates und aktualisiert DRAFT-Templates


Template-Designer
bp-workbench (Web Component)
blocpress-workbench REST-API
TemplateService (UC-5)
TemplateValidator (TF-1)
Repository (Schema: workbench)
Dashboard-Ansicht
Workbench öffnen
GET /api/workbench/templates
listTemplates()
find("ORDER BY name")
List<Template>
List<TemplateSummary> (mit isValid, status)
JSON Array
Dashboard mit Template-Cards anzeigen
Status filtern
Klick "¬ Test" Filter
statusFilter = 'SUBMITTED'
Zeige nur SUBMITTED Templates
Status-Übergang
DRAFT-Template ¬ "¬ Test" Button
PUT /api/workbench/templates/{id}/status {newStatus: "SUBMITTED"}
updateStatus(templateId, SUBMITTED)
isValidTransition(DRAFT, SUBMITTED) ¬
template.status = SUBMITTED
persist()
OK
{id, status}
200 OK
Success-Message
Template-Liste neu laden
Template als DRAFT-Kopie erstellen
APPROVED-Template ¬ "Als Kopie" Button
Prompt: Name eingeben
"Invoice v3"
POST /api/workbench/templates/{id}/duplicate {name: "Invoice v3"}
duplicate(sourceId, "Invoice v3")
Prüfe: Name nicht vorhanden ¬
Kopiere content (byte[])
validate(newContent)
ValidationResult
persist(newTemplate)
Status: DRAFT
Template (with id)
{id, name, status, isValid}
201 Created
Success-Message
"Invoice v3" im Filter DRAFT
Template aktualisieren (Re-Upload)
DRAFT-Template ¬ "Aktualisieren" Button
File-Dialog öffnen
Neue ODT-Datei auswählen
PUT /api/workbench/templates/{id}/content (multipart/form-data, file)
updateContent(templateId, newFile)
Prüfe: Status = DRAFT ¬
Kopiere newFile ¬ content
validate(newContent)
ValidationResult (neu)
template.content = newContent
template.validationResult = new
persist()
OK
{id, name, status, isValid, errors, warnings}
200 OK
Success-Message
"Template aktualisiert"
UC-5: Template Management Dashboard
- Status-Filter (ALL, DRAFT, SUBMITTED, APPROVED, REJECTED)
- Status-Übergänge mit Validierung
- Duplikation von APPROVED Templates
- Re-Upload für DRAFT Templates
- Kein User-Management (vereinachte Version)

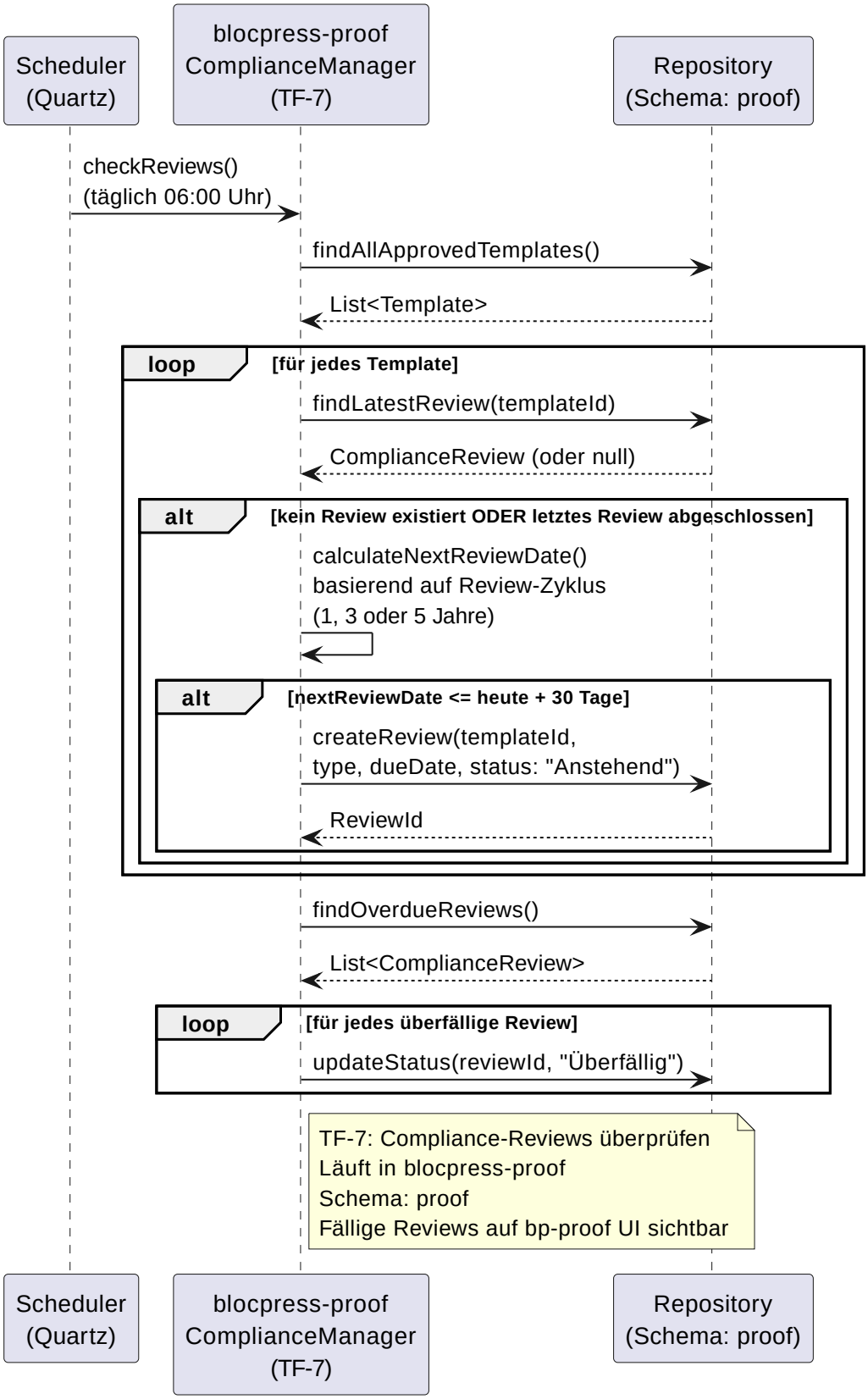
6.6. Stufenübergabe (workbench → proof → production)

Szenario: Template durchläuft den kompletten Lebenszyklus von Entwicklung über Prüfung bis zur Produktionsfreigabe



6.7. Compliance Review-Überwachung

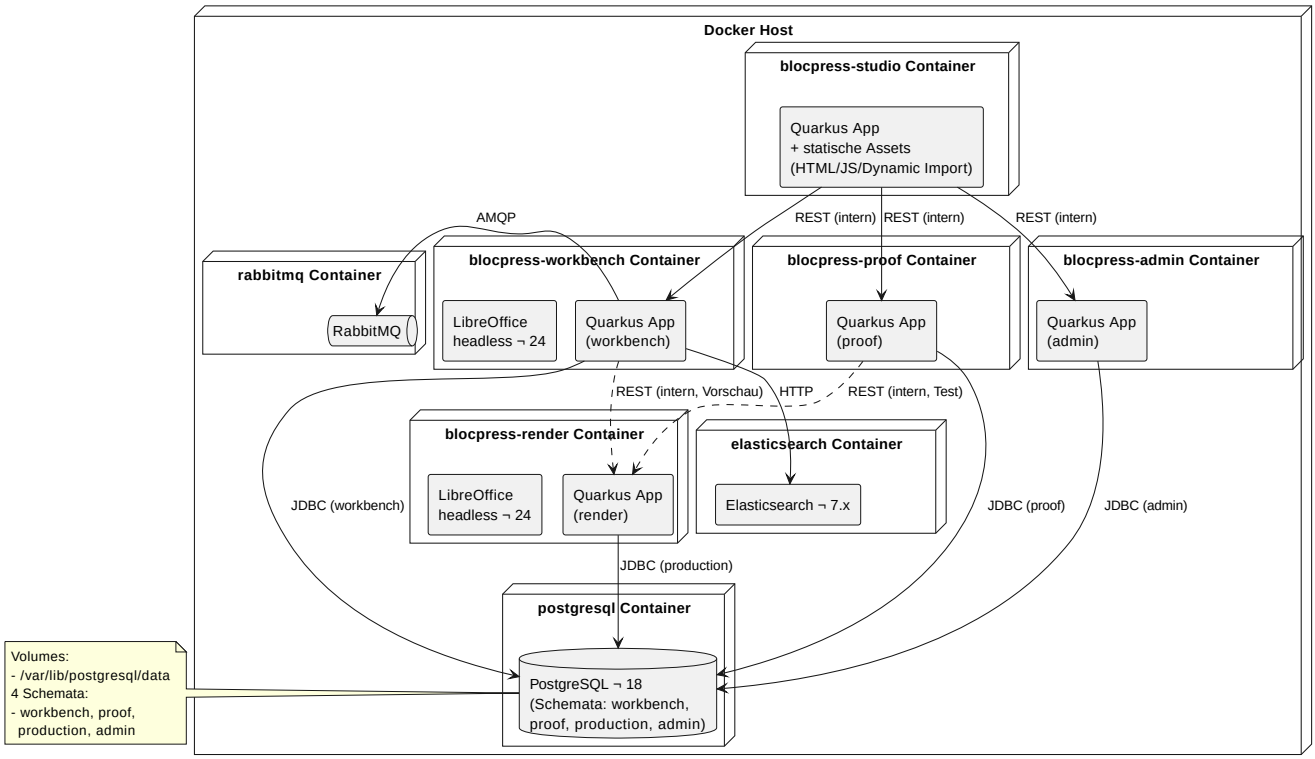
Szenario: Geplanter Job in blocpress-proof prüft täglich fällige Reviews (TF-7, UC-12)


Scheduler (Quartz)
blocpress-proof ComplianceManager (TF-7)
Repository (Schema: proof)
checkReviews()
(täglich 06:00 Uhr)
findAllApprovedTemplates()
List<Template>
loop
[für jedes Template]
findLatestReview(templateId)
ComplianceReview (oder null)
alt
[kein Review existiert ODER letztes Review abgeschlossen]
calculateNextReviewDate()
basierend auf Review-Zyklus
(1, 3 oder 5 Jahre)
alt
[nextReviewDate <= heute + 30 Tage]
createReview(templateId,
type, dueDate, status: "Anstehend")
ReviewId
findOverdueReviews()
List<ComplianceReview>
loop
[für jedes überfällige Review]
updateStatus(reviewId, "Überfällig")
TF-7: Compliance-Reviews überprüfen
Läuft in blocpress-proof
Schema: proof
Fällige Reviews auf bp-proof UI sichtbar
Scheduler (Quartz)
blocpress-proof ComplianceManager (TF-7)
Repository (Schema: proof)

7. Verteilungssicht

Systemarchitektur und Hardware-Elemente: System Architecture im System Design Concept

7.1. Infrastruktur Ebene 1 (Docker-Compose)



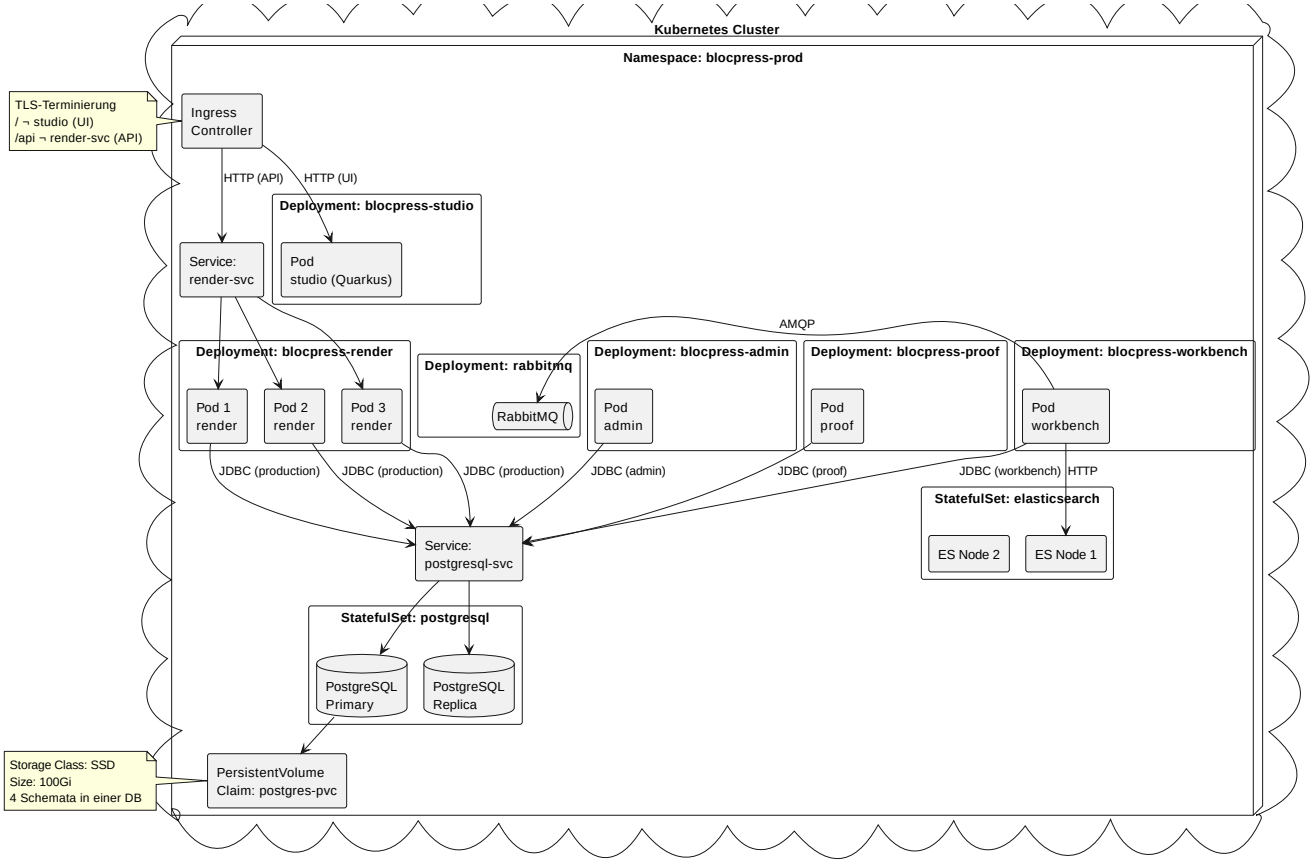
Deployment-Konfiguration (docker-compose.yml):

```yaml
version: '3.8'
services:
  postgresql:
    image: postgres:18-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: blocpress
      POSTGRES_USER: blocpress
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    ports:
      - "5432:5432"

  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.11.0
```

```
    environment:
      - discovery.type=single-node
      - ES_JAVA_OPTS=-Xms512m -Xmx512m
    ports:
      - "9200:9200"

  blocpress-studio:
    build: ./blocpress-studio
    ports:
      - "8082:8082"
    depends_on:
      - blocpress-workbench
      - blocpress-proof
      - blocpress-admin

  blocpress-workbench:
    build: ./blocpress-workbench
    depends_on:
      - postgresql
      - rabbitmq
      - elasticsearch
    environment:
      QUARKUS_DATASOURCE_JDBC_URL:
jdbc:postgresql://postgresql:5432/blocpress?currentSchema=workbench
      RABBITMQ_HOST: rabbitmq
      ELASTICSEARCH_URL: http://elasticsearch:9200
      LIBREOFFICE_HOME: /usr/lib/libreoffice
      BLOCPRESS_RENDER_URL: http://blocpress-render:8083
      # OpenTelemetry (optional)
      # QUARKUS_OTEL_EXPORTER_OTLP_ENDPOINT: http://otel-collector:4317
    ports:
      - "8081:8081"

  blocpress-proof:
    build: ./blocpress-proof
    depends_on:
      - postgresql
    environment:
      QUARKUS_DATASOURCE_JDBC_URL:
jdbc:postgresql://postgresql:5432/blocpress?currentSchema=proof
      BLOCPRESS_RENDER_URL: http://blocpress-render:8083
      # QUARKUS_OTEL_EXPORTER_OTLP_ENDPOINT: http://otel-collector:4317
    ports:
      - "8082:8082"

  blocpress-render:
    build: ./blocpress-render
    depends_on:
      - postgresql
    environment:
      QUARKUS_DATASOURCE_JDBC_URL:
```

```
jdbc:postgresql://postgresql:5432/blocpress?currentSchema=production
      LIBREOFFICE_HOME: /usr/lib/libreoffice
      # QUARKUS_OTEL_EXPORTER_OTLP_ENDPOINT: http://otel-collector:4317
    ports:
      - "8083:8083"
    volumes:
      - libreoffice_cache:/tmp/libreoffice

  blocpress-admin:
    build: ./blocpress-admin
    depends_on:
      - postgresql
    environment:
      QUARKUS_DATASOURCE_JDBC_URL:
jdbc:postgresql://postgresql:5432/blocpress?currentSchema=admin
      # QUARKUS_OTEL_EXPORTER_OTLP_ENDPOINT: http://otel-collector:4317
    ports:
      - "8084:8084"

volumes:
  postgres_data:
  libreoffice_cache:
```

7.2. Infrastruktur Ebene 2 (Kubernetes - Produktiv)



Kubernetes-Konfiguration (deployment.yaml — Beispiel blocpress-render):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blocpress-render
  namespace: blocpress-prod
spec:
  replicas: 3
  selector:
    matchLabels:
      app: blocpress-render
  template:
    metadata:
      labels:
        app: blocpress-render
    spec:
      containers:
      - name: blocpress-render
        image: blocpress/render:1.0.0
        ports:
        - containerPort: 8083
        env:
        - name: QUARKUS_DATASOURCE_JDBC_URL
          value: jdbc:postgresql://postgresql-
svc:5432/blocpress?currentSchema=production
        - name: LIBREOFFICE_HOME
          value: /usr/lib/libreoffice
        # OpenTelemetry (optional)
        # - name: QUARKUS_OTEL_EXPORTER_OTLP_ENDPOINT
        #   value: http://otel-collector-svc:4317
        resources:
          requests:
            memory: "2Gi"
            cpu: "1"
          limits:
            memory: "4Gi"
            cpu: "2"
        livenessProbe:
          httpGet:
            path: /q/health/live
            port: 8083
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /q/health/ready
            port: 8083
          initialDelaySeconds: 10
          periodSeconds: 5
```

NOTE	Die übrigen SCS-Module (workbench, proof, admin) werden analog konfiguriert, jeweils mit eigenem Schema und Port. blocpress-render ist das einzige Modul mit mehreren Replicas, da es die lastintensive Dokumentengenerierung übernimmt.

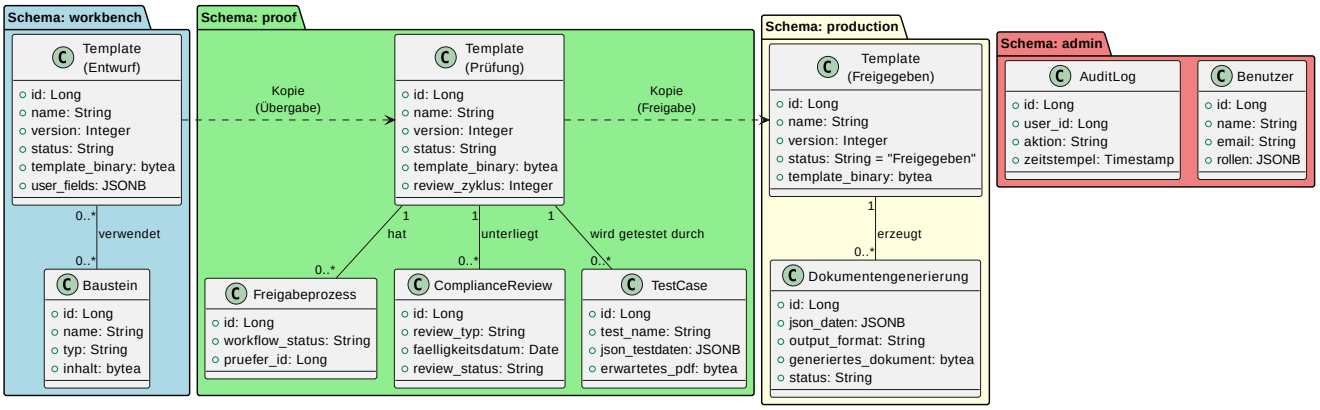
Qualitätsmerkmale der Verteilungssicht:

- **Skalierbarkeit:** blocpress-render separat skalierbar (3+ Replicas), übrige Module mit 1 Replica
- **Verfügbarkeit:** PostgreSQL mit Primary-Replica-Setup, Elasticsearch mit 2 Nodes
- **Performance:** SSD-Storage für PostgreSQL, Connection Pooling pro Modul, Load Balancing für render
- **Überwachbarkeit:** Liveness-/Readiness-Probes je Modul, OpenTelemetry für Traces/Metriken/Logs (optional)
- **Unabhängigkeit:** Jedes SCS-Modul kann unabhängig deployed und aktualisiert werden

8. Querschnittliche Konzepte

Detaillierte Entities: *Entities im Element Design Concept* | *Geschäftsentitäten:* *Information Architecture im Solution Design Concept*

8.1. Domänenmodell



NOTE	Die Template-Tabelle existiert mit identischer Struktur in allen drei fachlichen Schemata (workbench, proof, production). Die Stufenübergabe erfolgt als Datenkopie zwischen den Schemata.

8.2. Benutzerverwaltung und Sicherheit

Authentifizierung (JWT-basiert):

- blocpress-studio (Portal-Shell) übernimmt die Authentifizierung und hält das JWT
- Jedes SCS-Modul validiert JWTs eigenständig via Quarkus SmallRye JWT Extension
- JWT wird von der Studio-Shell als Property an die Web Components weitergereicht: `<bp-workbench jwt="${token}"></bp-workbench>`
- Web Components senden das JWT als `Authorization: Bearer` Header bei REST-Aufrufen an ihr Backend
- Issuer und Public Key werden in jeder `application.properties` konfiguriert (`mp.jwt.verify.issuer`, `mp.jwt.verify.publickey.location`)
- Kompatibel mit jedem IdP, der JWTs ausstellt — nur Public Key und Issuer müssen konfiguriert werden

Autorisierung:

Rollenbasierte Zugriffskontrolle (RBAC) — konsolidierte Rollen:

Rolle	Berechtigungen	SCS-Modul / Use Cases
Template-Designer	Template erstellen, bearbeiten, einreichen, Bausteine verwalten, Suche	blocpress-workbench: UC-1 bis UC-5, UC-19 bis UC-21

Rolle	Berechtigungen	SCS-Modul / Use Cases
Prüfer (QM / Freigeber / Compliance)	Templates prüfen, freigeben/ablehnen, Compliance-Reviews durchführen, Tests verwalten und ausführen	blocpress-proof: UC-6 bis UC-9, UC-11 bis UC-18
Administrator	Benutzer verwalten, Rollen zuweisen, System konfigurieren	blocpress-admin: UC-22 bis UC-24
API-Konsument	Dokumente generieren (nur via öffentliche API)	blocpress-render: POST /api/documents/generate

Sicherheitskonzepte:

- **Vier-Augen-Prinzip:** Technisch erzwungen in TF-2 (blocpress-proof): `userId != template.erstellerId`
- **HTTPS/TLS:** Alle externen Verbindungen verschlüsselt
- **JWT-Weitergabe:** Über Web Component Properties — kein LocalStorage, kein Cookie
- **Schema-Isolation:** Jedes SCS-Modul hat nur Zugriff auf sein eigenes Datenbank-Schema
- **SQL-Injection-Schutz:** Prepared Statements, JPA/Hibernate
- **DSGVO-Compliance:** Löschfunktion für personenbezogene Daten in E-3, E-6

8.3. Persistenz und Storage

Multi-Schema-Architektur:

Alle SCS-Module teilen sich eine PostgreSQL-Instanz, nutzen aber jeweils ein eigenes Schema:

Schema	SCS-Modul	Enthaltene Entities
`workbench`	blocpress-workbench	Template (Entwurf), Baustein, Template-Baustein-Zuordnung
`proof`	blocpress-proof	Template (Prüfung), Freigabeprozess, ComplianceReview, TestCase, Testpool
`production`	blocpress-render	Template (Freigegeben), Dokumentengenerierung
`admin`	blocpress-admin	Benutzer, AuditLog

Die Template-Tabelle hat in allen drei fachlichen Schemata eine identische Struktur. Dies ermöglicht die kopie-basierte Stufenübergabe: `INSERT INTO proof.templates SELECT … FROM workbench.templates WHERE id = ?`.

Storage Abstraction Layer:

```
public interface StorageService {
    StorageId save(byte[] data, String key);
```

```
    byte[] load(StorageId id);
    void delete(StorageId id);
}
```

Jedes SCS-Modul nutzt eine eigene `StorageService`-Instanz, die auf das jeweilige Schema zugreift.

Datenbank-Schema (Auszug — Schema: workbench):

```
CREATE SCHEMA workbench;

CREATE TABLE workbench.templates (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    version INTEGER NOT NULL,
    status VARCHAR(50) NOT NULL,
    ersteller_id BIGINT,          -- Referenz auf admin.benutzer (JWT sub)
    template_binary BYTEA,
    user_fields JSONB,
    review_zyklus INTEGER,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Identische Struktur in proof und production:
-- CREATE SCHEMA proof;
-- CREATE TABLE proof.templates (...);
-- CREATE SCHEMA production;
-- CREATE TABLE production.templates (...);
```

Schema-Konfiguration pro Modul:

```
# blocpress-workbench
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/blocpress?currentSchema=w
orkbench

# blocpress-proof
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/blocpress?currentSchema=p
roof

# blocpress-render
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/blocpress?currentSchema=p
roduction
```

Transaktionsmanagement:

- JTA-Transaktionen für atomare Operationen innerhalb eines Schemas
- Stufenübergabe (Cross-Schema-Kopie) erfolgt über interne REST-APIs, nicht über Cross-Schema-SQL

- Isolation Level: READ_COMMITTED
- Rollback bei Fehlern in Use Cases

8.4. Micro-Frontend-Architektur (Web Components)

blocpress-studio ist die Portal-Shell, die Micro-Frontends der SCS-Module als Web Components integriert.

Prinzip:

- Jedes SCS-Modul liefert ein JavaScript-Bundle mit Custom Elements (z.B. `<bp-workbench>`, `<bp-proof>`, `<bp-admin>`)
- Die Web Components nutzen Shadow DOM für vollständige CSS- und DOM-Kapselung
- Studio lädt die Bundles dynamisch via Dynamic Import (`import()`) von den jeweiligen SCS-Modulen

Dynamic Import (Beispiel):

```
// Studio lädt Web Components per Dynamic Import von den SCS-Modulen
async function loadWorkbench() {
  const workbenchUrl = 'http://localhost:8081'; // konfigurierbar
  await import(`${workbenchUrl}/components/bp-workbench.js`);
}
```

JWT-Weitergabe:

```
<!-- Studio-Shell: JWT als Property an Web Component -->
<bp-workbench jwt="${currentUser.token}"></bp-workbench>
```

```
// Web Component: JWT empfangen und für REST-Aufrufe verwenden
class BpWorkbench extends HTMLElement {
  static get observedAttributes() { return ['jwt']; }

  attributeChangedCallback(name, oldVal, newVal) {
    if (name === 'jwt') this._jwt = newVal;
  }

  async fetchTemplates() {
    const res = await fetch('/api/workbench/templates', {
      headers: { 'Authorization': `Bearer ${this._jwt}` }
    });
    // ...
  }
}
customElements.define('bp-workbench', BpWorkbench);
```

Vorteile:

- Vollständige UI-Isolation: CSS und DOM eines Moduls beeinflussen andere Module nicht
- Unabhängiges Deployment: Web Component-Bundles können einzeln aktualisiert werden
- Framework-Freiheit: Jedes Modul kann intern ein beliebiges UI-Framework verwenden
- JWT-Sicherheit: Token wird als Property übergeben, nicht im LocalStorage

8.5. Template-Verarbeitung mit LibreOffice

LibreOffice-Integration:

```
public class LibreOfficeProcessor {
    private XComponentLoader componentLoader;

    public ValidationResult validateTemplate(byte[] odtBinary) {
        // 1. Lade ODT über LibreOffice UNO-API
        XComponent doc = componentLoader.loadComponentFromURL(...);

        // 2. Extrahiere User-Fields
        XTextFieldsSupplier fieldsSupplier = (XTextFieldsSupplier) doc;
        XNameAccess fields = fieldsSupplier.getTextFields();

        // 3. Identifiziere Wiederholungsgruppen (Sections/Tables)
        XTextSectionsSupplier sectionsSupplier = (XTextSectionsSupplier) doc;

        // 4. Validiere Punkt-Notation in Field-Namen
        // z.B. "kunde.name" ist valide, "kunde name" ist invalide

        return new ValidationResult(...);
    }

    public byte[] generateDocument(Template template,
                                   String jsonData,
                                   OutputFormat format) {
        // 1. Parse JSON zu Map/Object
        JsonNode json = objectMapper.readTree(jsonData);

        // 2. Öffne Template
        XComponent doc = loadTemplate(template.getTemplateBinary());

        // 3. Befülle User-Fields
        for (UserField field : template.getUserFields()) {
            String value = resolveJsonPath(json, field.getName());
            setFieldValue(doc, field.getName(), value);
        }

        // 4. Iteriere Wiederholungsgruppen
        for (RepetitionGroup group : template.getRepetitionGroups()) {
```

```
            JsonNode array = resolveJsonPath(json, group.getArrayRef());
            duplicateAndFillGroup(doc, group, array);
        }

        // 5. Evaluiere IF-Bedingungen
        evaluateConditions(doc, json);

        // 6. Exportiere
        return exportDocument(doc, format); // ODT/PDF/RTF
    }
}
```

Determinismus:

- Keine Zufallswerte, keine Zeitstempel in Template-Verarbeitung
- Sortierung von Arrays im JSON konsistent (z.B. nach ID)
- LibreOffice-Version fixiert (≥ 24)
- Gleiche Template-Binärdaten + gleiche JSON-Daten = gleiches Ausgabedokument

8.6. Asynchrone Verarbeitung

Message Queue Pattern:

```
// Producer (UC-10: Testdokument generieren)
public class DocumentService {
    @Inject
    MqttClient mqttClient;

    public GenerationId generateDocument(UUID templateId,
                                         String jsonData,
                                         OutputFormat format) {
        // 1. Erstelle Generierungs-Record
        Dokumentengenerierung gen = new Dokumentengenerierung();
        gen.setStatus("In Bearbeitung");
        repository.save(gen);

        // 2. Sende Message an Queue
        GenerationMessage msg = new GenerationMessage(
            gen.getId(), templateId, jsonData, format);
        mqttClient.send("document.generation", msg);

        return gen.getId();
    }
}

// Consumer (Worker)
@ApplicationScoped
public class DocumentWorker {
```

```
    @Inject
    LibreOfficeProcessor processor;

    @Incoming("document.generation")
    public void processGeneration(GenerationMessage msg) {
        try {
            Template template = loadTemplate(msg.getTemplateId());
            byte[] doc = processor.generateDocument(
                template, msg.getJsonData(), msg.getFormat());

            storageService.save(doc, ...);
            updateStatus(msg.getGenerationId(), "Erfolgreich");
        } catch (Exception e) {
            updateStatus(msg.getGenerationId(), "Fehlgeschlagen");
            log.error("Generation failed", e);
        }
    }
}
```

Vorteile:

- Nicht-blockierend: API gibt sofort GenerationId zurück
- Skalierbar: Worker-Pool kann unabhängig skaliert werden
- Fehlertoleranz: Retry-Mechanismus bei Fehlern
- Entkopplung: Producer und Consumer unabhängig deploybar

8.7. Observability (OpenTelemetry)

Alle SCS-Module nutzen **Quarkus OpenTelemetry** (`quarkus-opentelemetry`) für die Erfassung von Traces, Metriken und Logs. Die Integration ist vendor-neutral und erfordert vom Betreiber lediglich die Konfiguration eines OTLP-Endpunkts.

Auto-Instrumentation:

Quarkus instrumentiert automatisch:

- REST-API-Aufrufe (Eingangs-Requests und ausgehende HTTP-Calls)
- JDBC-Datenbankzugriffe (PostgreSQL)
- Message-Queue-Operationen (RabbitMQ/Kafka)

Für diese Standardmetriken und Traces ist kein eigener Code erforderlich.

Konfiguration:

```
# OpenTelemetry aktivieren (optional – ohne Endpoint werden keine Daten exportiert)
quarkus.otel.exporter.otlp.endpoint=http://otel-collector:4317
```

```
# Service-Name für Traces und Metriken
quarkus.otel.resource.attributes=service.name=blocpress-workbench  # bzw. -proof,
-render, -admin

# Logging: Standard Quarkus JSON-Logging, wird durch OTel-Kontext (Trace-ID, Span-ID)
angereichert
quarkus.log.console.format=%d{yyyy-MM-dd HH:mm:ss} %-5p traceId=%X{traceId} [%c]
%s%e%n
```

Optionalität:

Wird kein `quarkus.otel.exporter.otlp.endpoint` konfiguriert, ist OpenTelemetry deaktiviert — die SCS-Module laufen ohne externe Monitoring-Abhängigkeit. Der Betreiber kann jeden OTLP-kompatiblen Collector einsetzen (z.B. Grafana Alloy, Jaeger, Datadog Agent).

Audit-Logging:

Audit-Logging für Compliance-relevante Aktionen (Freigaben, Reviews, Statusänderungen) ist ein eigenständiges Konzept und unabhängig von OpenTelemetry. Audit-Einträge werden in einer separaten, nicht modifizierbaren Datenbanktabelle gespeichert (User-ID, Zeitstempel, Aktion, alte/neue Werte).

Health-Checks:

Quarkus stellt Liveness- und Readiness-Probes automatisch bereit (`/q/health/live`, `/q/health/ready`). Diese sind unabhängig von OpenTelemetry und erfordern keine zusätzliche Konfiguration.

9. Architekturentscheidungen

Zugehörige Constraints: Constraints im Element Design Concept | Constraints im System Design Concept

9.1. ADR-001: Binärdaten in PostgreSQL statt S3

Status: Akzeptiert

Kontext:

Wir müssen entscheiden, wo Template-Binärdaten (ODT), generierte Dokumente (PDF/RTF) und Test-Baselines (PDF) gespeichert werden. Optionen: PostgreSQL (bytea) oder S3-kompatibler Object Storage.

Entscheidung:

Speicherung in PostgreSQL bytea-Spalten. Erwartetes Datenvolumen: < 5.000 Dokumente + Templates.

Begründung:

- **Einfachheit:** Keine zusätzliche Infrastruktur-Komponente
- **Transaktionalität:** Metadaten + Binärdaten in einer Transaktion
- **Performance:** Bei < 5.000 Dokumenten ist PostgreSQL performant genug
- **Kosteneffizienz:** Kein separater S3-Service notwendig
- **Zukunftssicherheit:** Storage Abstraction Layer ermöglicht späteren Wechsel zu S3

Konsequenzen:

- Positiv: Einfacheres Deployment, weniger Moving Parts, ACID-Garantien
- Negativ: PostgreSQL-Backup größer, bei > 5.000 Dokumenten Migration zu S3 notwendig
- Mitigation: Storage Abstraction Layer implementiert, Migration zu S3 vorbereitet

9.2. ADR-002: Quarkus statt Spring Boot

Status: Akzeptiert

Kontext:

Wahl des Java-Frameworks für die SCS-Module. Optionen: Spring Boot, Quarkus, Micronaut.

Entscheidung:

Quarkus Framework

Begründung:

- **Performance:** Schnellere Startup-Zeit (< 1s vs. 5-10s bei Spring Boot)
- **Memory-Footprint:** Geringerer RAM-Verbrauch (wichtig bei Container-Deployment)
- **Cloud-Native:** Entwickelt für Kubernetes, native Kompilierung möglich (GraalVM)
- **Developer Experience:** Live-Reload, Dev-Services, gute Dokumentation
- **Standards:** Jakarta EE, MicroProfile (nicht vendor-locked)

Konsequenzen:

- Positiv: Bessere Container-Performance, niedrigere Infrastruktur-Kosten
- Negativ: Kleineres Ökosystem als Spring Boot, weniger Extensions

9.3. ADR-003: Asynchrone Dokumentengenerierung

Status: Akzeptiert

Kontext:

Dokumentengenerierung kann 5-30 Sekunden dauern. Soll der REST-API-Call blockieren oder asynchron sein?

Entscheidung:

Asynchrone Verarbeitung über Message Queue (RabbitMQ/Kafka)

Begründung:

- **Responsiveness:** API gibt sofort (< 100ms) GenerationId zurück
- **Skalierbarkeit:** Worker-Pool kann unabhängig von API-Servern skaliert werden
- **Fehlertoleranz:** Retry-Mechanismus bei LibreOffice-Fehlern
- **Monitoring:** Queue-Länge zeigt System-Last

Konsequenzen:

- Positiv: Bessere User Experience, horizontal skalierbar
- Negativ: Komplexere Architektur, Client muss pollen oder WebSocket nutzen
- Mitigation: Klares API-Design (202 Accepted, Polling-Endpoint)

9.4. ADR-004: Elasticsearch für Content Search

Status: Akzeptiert

Kontext:

Volltextsuche über alle Templates nach fachlichen Konstrukten. Optionen: PostgreSQL Full-Text-Search, Elasticsearch, Apache Solr.

Entscheidung:

Elasticsearch ≥ 7.x

Begründung:

- **Performance:** Sub-2-Sekunden-Suche auch bei 1.000+ Templates
- **Relevanz:** Besseres Ranking als PostgreSQL FTS
- **Highlighting:** Zeigt Treffer im Kontext
- **Skalierbarkeit:** Horizontal skalierbar (mehrere Nodes)
- **Reife:** Bewährte Technologie, große Community

Konsequenzen:

- Positiv: Exzellente Suchperformance und -qualität
- Negativ: Zusätzliche Infrastruktur-Komponente, höhere Komplexität
- Risiko: Eventual Consistency (Index kann wenige Sekunden verzögert sein)

9.5. ADR-005: LibreOffice headless statt Cloud-Conversion-Service

Status: Akzeptiert

Kontext:

Template-Verarbeitung und Dokumentengenerierung. Optionen: LibreOffice headless (self-hosted), Cloud-Conversion-Service (z.B. Aspose, CloudConvert).

Entscheidung:

LibreOffice ≥ 24 headless, integriert in blocpress-render (Format-Export). blocpress-workbench nutzt blocpress-core für Validierung ohne LibreOffice

Begründung:

- **Kosteneffizienz:** Keine Lizenzkosten, keine API-Kosten
- **Datenschutz:** Templates verlassen nicht die eigene Infrastruktur
- **Kontrolle:** Vollständige Kontrolle über Verarbeitungs-Pipeline
- **Determinismus:** LibreOffice-Version fixiert, reproduzierbare Ergebnisse
- **ODT-Support:** Native Unterstützung für OpenDocument-Format

Konsequenzen:

- Positiv: Keine externe Abhängigkeit, DSGVO-konform, kosteneffizient
- Negativ: LibreOffice-Deployment komplex, Memory-Verbrauch höher

- Risiko: LibreOffice-Bugs, Performance-Tuning notwendig

9.6. ADR-006: Self-Contained Systems (SCS) statt Monolith

Status: Akzeptiert

Kontext:

blocpress wurde initial als monolithische Quarkus-Anwendung (blocpress-server) konzipiert. Mit wachsender fachlicher Komplexität (Template-Entwicklung, Prüfung/Freigabe, Compliance, Administration, Dokumentengenerierung) entstehen enge Kopplung und erschwerte Wartbarkeit. Es soll eine Zerlegung in unabhängig wartbare Module erfolgen.

Entscheidung:

Zerlegung in fünf Self-Contained Systems (SCS) mit eigener UI, eigenem Backend und eigenem Datenbank-Schema:

- **blocpress-studio** — Portal-Shell (Navigation, Auth, Micro-Frontend-Integration)
- **blocpress-workbench** — Template-Entwicklung (Dev-Stufe)
- **blocpress-proof** — Prüfung, Freigabe, Compliance, Tests (QA-Stufe)
- **blocpress-render** — Dokumentengenerierung (Einzige öffentliche API, Production-Stufe)
- **blocpress-admin** — Benutzerverwaltung, Systemkonfiguration

Gemeinsame Logik (ODT-Parsing, Validierung, Merge-Pipeline) wird in **blocpress-core** als Shared Java-Library bereitgestellt und von blocpress-workbench und blocpress-render als Maven-Abhängigkeit genutzt.

Drei Datenstufen mit kopie-basierter Übergabe: workbench → proof → production.

Begründung:

- **Wartbarkeit:** Jedes Modul kann unabhängig entwickelt, getestet und deployed werden
- **Team-Ownership:** Module können von verschiedenen Teams verantwortet werden
- **Skalierbarkeit:** blocpress-render (lastintensiv) kann separat skaliert werden
- **Fachliche Kohäsion:** Jedes Modul kapselt einen klar abgegrenzten fachlichen Bereich
- **UI-Isolation:** Web Components mit Shadow DOM verhindern CSS-/DOM-Konflikte

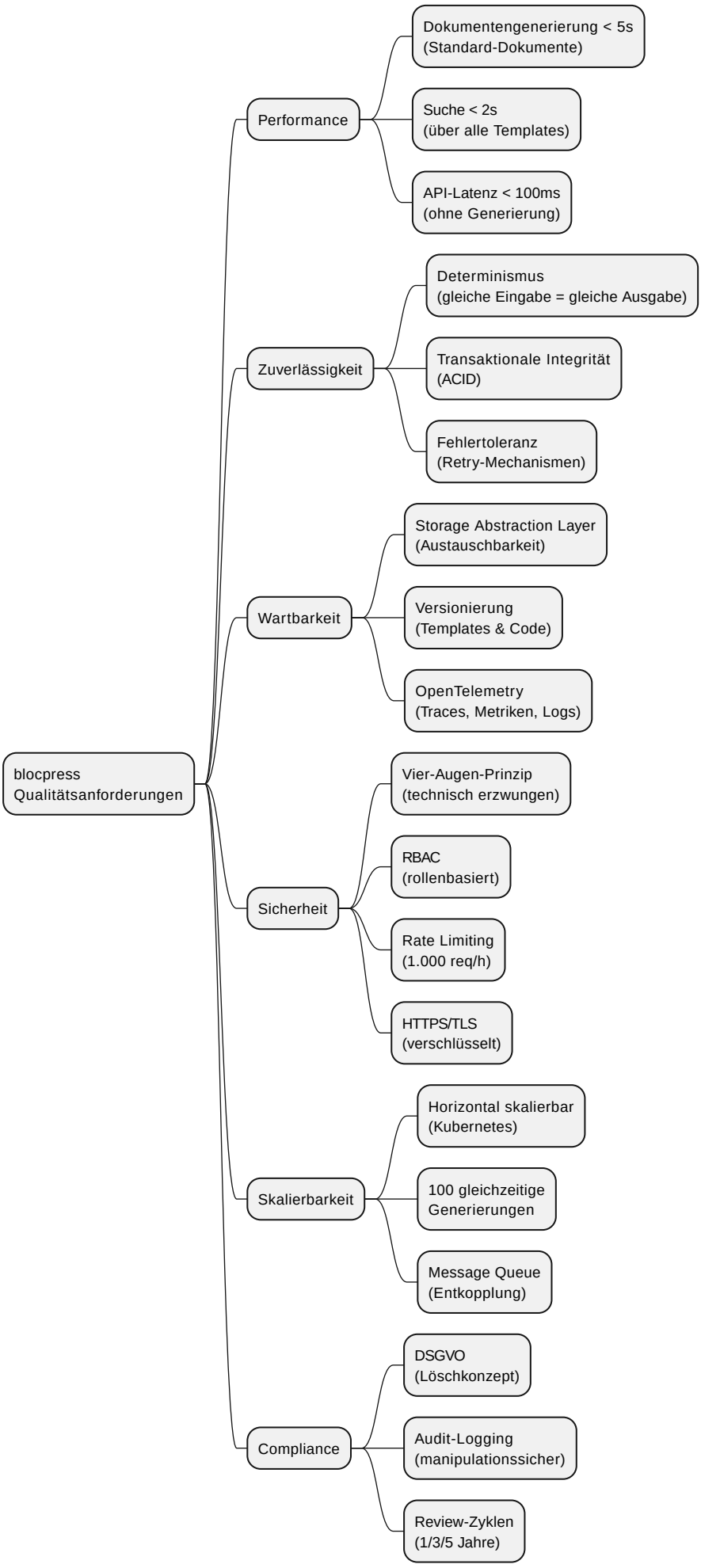
Konsequenzen:

- Positiv: Bessere Wartbarkeit, unabhängige Deployments, gezielte Skalierung, klare Ownership
- Negativ: Höhere Deployment-Komplexität (5 Container statt 1), Datenredundanz durch Kopien, Cross-Schema-Kommunikation über REST statt direkte DB-Zugriffe
- Mitigation: Gemeinsame PostgreSQL-Instanz reduziert Infrastruktur-Overhead, Docker-Compose/Kubernetes-Konfiguration automatisiert Deployment

10. Qualitätsanforderungen

Qualitätsanforderungen auf Fachebene: Quality Requirements im Solution Design Concept | auf Systemebene: Quality Requirements im System Design Concept | auf Elementebene: Quality Requirements im Element Design Concept

10.1. Qualitätsbaum


blocpress
Qualitätsanforderungen
Performance
Dokumentengenerierung < 5s
(Standard-Dokumente)
Suche < 2s
(über alle Templates)
API-Latenz < 100ms
(ohne Generierung)
Zuverlässigkeit
Determinismus
(gleiche Eingabe = gleiche Ausgabe)
Transaktionale Integrität
(ACID)
Fehlertoleranz
(Retry-Mechanismen)
Wartbarkeit
Storage Abstraction Layer
(Austauschbarkeit)
Versionierung
(Templates & Code)
OpenTelemetry
(Traces, Metriken, Logs)
Sicherheit
Vier-Augen-Prinzip
(technisch erzwungen)
RBAC
(rollenbasiert)
Rate Limiting
(1.000 req/h)
HTTPS/TLS
(verschlüsselt)
Skalierbarkeit
Horizontal skalierbar
(Kubernetes)
100 gleichzeitige
Generierungen
Message Queue
(Entkopplung)
Compliance
DSGVO
(Löschkonzept)
Audit-Logging
(manipulationssicher)
Review-Zyklen
(1/3/5 Jahre)

10.2. Qualitätsszenarien

Qualitätsziel	Szenario	Stimulus	Response
Performance	Standard-Dokumentengenerierung	Externe Anwendung sendet Request mit Template (20 Seiten) und JSON-Daten	Dokument wird innerhalb von 5 Sekunden generiert und zurückgegeben
Performance	Volltextsuche	Template-Designer sucht nach "Beitragsanpassung" (1.000 Templates im Index)	Suchergebnisse werden innerhalb von 2 Sekunden angezeigt
Zuverlässigkeit	Deterministische Generierung	Gleiches Template + gleiche JSON-Daten, zweimal aufgerufen	Beide generierten PDFs sind Byte-für-Byte identisch (Regressionstests bestätigen)
Zuverlässigkeit	Transaktionale Integrität	Template-Freigabe schlägt fehl (z.B. DB-Fehler nach Status-Update)	Komplettes Rollback: Template behält alten Status, kein Freigabeprozess-Eintrag erstellt
Wartbarkeit	Storage-Austausch	Migration von PostgreSQL bytea zu S3	Storage Abstraction Layer ermöglicht Implementierungswechsel ohne Änderungen in Use Cases/TFs
Wartbarkeit	Template-Versionierung	Template wird geändert	Neue Version wird erstellt, alte Version bleibt verfügbar, Änderungen nachvollziehbar
Sicherheit	Vier-Augen-Prinzip	Designer versucht, sein eigenes Template freizugeben	TF-2 verweigert Freigabe mit Fehlermeldung "Designer darf nicht freigeben (Vier-Augen-Prinzip)"
Skalierbarkeit	Horizontale Skalierung	150 gleichzeitige Dokumentengenerierungen	Kubernetes skaliert auf 5 blocpress-render Pods, Queue puffert Last, alle Requests erfolgreich
Skalierbarkeit	Message Queue Entkopplung	100 Generierungsanfragen in 10 Sekunden	API nimmt alle Anfragen an (< 100ms Response), Queue verarbeitet asynchron

Qualitätsziel	Szenario	Stimulus	Response
Compliance	DSGVO-Löschung	Administrator löscht personenbezogene Daten in Test Cases	JSON-Testdaten werden aus E-6 entfernt, Test-PDFs anonymisiert oder gelöscht
Compliance	Turnusmäßiges Review	Template mit 3-Jahres-Zyklus ist seit 3 Jahren freigegeben	Compliance Manager erstellt automatisch Review-Aufgabe, fälliges Review ist auf UI-3 sichtbar
Benutzbarkeit	Template-Upload mit Validierung	Designer lädt Template mit ungültigen User-Fields hoch	System zeigt klare Validierungsfehler ("Field 'kunde name' enthält Leerzeichen, Punkt-Notation erforderlich")
Übertragbarkeit	Docker-Deployment	blocpress soll in neuer Umgebung deployed werden	docker-compose up startet alle Services, keine manuellen Konfigurationsschritte

11. Risiken und technische Schulden

11.1. Risiken

ID	Risiko	Wahrscheinlichkeit	Auswirkung	Maßnahme
R-1	**LibreOffice Memory Leaks**: LibreOffice headless kann bei Langzeitbetrieb Memory Leaks entwickeln	Mittel	Hoch (Out-of-Memory, Systemausfall)	- Regelmäßiger Neustart von LibreOffice-Instanzen nach X Generierungen - Memory-Monitoring - Container-Restart bei Memory-Limit
R-2	**PostgreSQL bytea Performance**: Bei > 5.000 Dokumenten kann Performance von PostgreSQL bytea degradieren	Hoch	Mittel (Langsamere Queries)	- Storage Abstraction Layer bereits implementiert - Migration zu S3 vorbereitet - Monitoring der DB-Performance
R-3	**Determinismus-Verlust**: LibreOffice-Updates können Rendering ändern, Determinismus brechen	Mittel	Hoch (Regressionstests schlagen fehl)	- LibreOffice-Version in Dockerfile fixieren - Regressionstests bei jedem LibreOffice-Update - Baseline-PDFs bei bewusstem Update neu generieren
R-4	**Message Queue Single Point of Failure**: Wenn RabbitMQ ausfällt, keine asynchrone Verarbeitung möglich	Niedrig	Hoch (Keine Dokumentengenerierung)	- RabbitMQ Cluster (3 Nodes) - Persistent Messages - Monitoring & Alerting
R-5	**Elasticsearch Eventual Consistency**: Index kann wenige Sekunden verzögert sein	Hoch	Niedrig (Nutzer findet neu hochgeladenes Template nicht sofort)	- User informieren ("Index wird aktualisiert, Template erscheint in ca. 10 Sekunden") - Refresh-Button in UI
R-6	**Compliance Review-Backlog**: Viele Templates werden gleichzeitig fällig	Mittel	Mittel (Compliance-Reviewer überlastet)	- Frühwarnung (30 Tage im Voraus) - Priorisierung nach Kritikalität - Eskalation bei Überschreitung

ID	Risiko	Wahrscheinlichkeit	Auswirkung	Maßnahme
R-7	**LibreOffice-API Breaking Changes**: LibreOffice-Updates können UNO-API ändern	Niedrig	Hoch (Code-Anpassungen notwendig)	- Versionsfixierung - Umfassende Integration-Tests - Update-Strategie dokumentiert

11.2. Technische Schulden

ID	Technische Schuld	Auswirkung	Refactoring-Plan
TD-1	**Fehlende Caching-Strategie**: Templates werden bei jeder Generierung aus DB geladen	Performance: Unnötige DB-Last, langsamere Generierung	- Caffeine Cache für Templates implementieren - Cache-Invalidierung bei Template-Update - TTL: 1 Stunde
TD-2	**Keine Batch-Generierung**: API unterstützt nur Single-Document-Generation	Effizienz: Viele API-Calls für Batch-Szenarien	- Batch-Endpoint hinzufügen: `POST /api/documents/batch` - Array von Generierungs-Requests - Transaktional verarbeiten
TD-3	**Hartcodierte LibreOffice-Pfade**: LibreOffice-Installation fest in Code	Wartbarkeit: Erschwert Deployment in verschiedenen Umgebungen	- Pfad als Environment-Variable - Auto-Detection von LibreOffice - Fallback-Mechanismus
TD-4	**Fehlende WebSocket-Unterstützung**: Client muss für Generierungs-Status pollen	Benutzbarkeit: Höhere Last, langsameres Feedback	- WebSocket-Endpoint für Status-Updates - Server-Sent Events (SSE) als Alternative
Geplant für Release 2.0	TD-5	**Kein PDF-Differenz-Highlighting**: TF-6 gibt nur Text-Unterschiede zurück	Testbarkeit: Test-Manager muss Unterschiede manuell finden
- PDF-Differenz visuell markieren (z.B. rote Umrandung) - Screenshot-Diff als Zusatzfeature	Geplant für Release 1.5	TD-6	**Fehlende Template-Preview**: Nutzer sehen Template erst nach Download

ID	Technische Schuld	Auswirkung	Refactoring-Plan
Benutzbarkeit: Umständlicher Workflow	- Template-Preview in Web-UI (PDF-Rendering) - Thumbnail-Generierung	Nice-to-have	TD-7
Keine Versionsverwaltung für Bausteine: Bausteine haben Version, aber keine Historie	Wartbarkeit: Änderungen an Bausteinen nicht nachvollziehbar	- Baustein-Versionierung wie bei Templates - Historie-View in UI	Geplant für Release 2.0

12. Glossar

Begriff	Definition
Baustein	Wiederverwendbares LibreOffice-Fragment (ODT), das in mehreren Templates eingebunden werden kann (z.B. Standardfußzeile, AGBs). Verwaltet in blocpress-workbench. Entity E-2.
Baseline-PDF	Erwartetes PDF-Ergebnis eines Test Cases, gegen das aktuelle Generierungen verglichen werden. Verwaltet in blocpress-proof.
blocpress-admin	SCS-Modul für Administration: Benutzerverwaltung, Rollenzuweisung, Systemkonfiguration, Audit-Logs. Schema: `admin`.
blocpress-core	Shared Java-Library (Maven-Abhängigkeit, kein eigenständiger Container): ODT-Parsing (odfdom), Template-Validierung, Merge-Pipeline (Text-Block-Expansion, Bedingungen, Schleifen, Feldersetzung). Wird von blocpress-workbench (Validierung) und blocpress-render (Merge + Generierung) genutzt. Keine LibreOffice-Abhängigkeit.
blocpress-proof	SCS-Modul für Prüfung und Freigabe: Workflow, Compliance-Reviews, Testdaten, Regressionstests. Arbeitsumgebung des Prüfers. Schema: `proof`.
blocpress-render	SCS-Modul für Dokumentengenerierung: Einzige öffentliche REST-API. Liest freigegebene Templates aus dem production-Schema. Stateless, horizontal skalierbar.
blocpress-studio	Portal-Shell (Quarkus): Keine eigene Business-Logik. Verantwortlich für Navigation, JWT-Authentifizierung und Integration der Micro-Frontends (Web Components) via Dynamic Import.
blocpress-workbench	SCS-Modul für Template-Entwicklung: Upload, Validierung, Bearbeitung, Bausteinverwaltung, Content Search. Arbeitsumgebung des Template-Designers. Schema: `workbench`.
Compliance Review	Turnusmäßige Prüfung eines Templates auf inhaltliche Aktualität. Review-Zyklen: 1, 3 oder 5 Jahre. Verwaltet in blocpress-proof. Entity E-5.
Content Search Engine	Elasticsearch-basierte Komponente zur Indexierung und Volltextsuche über alle Templates und Bausteine. Genutzt von blocpress-workbench.
Deterministische Generierung	Eigenschaft, dass identische Eingaben (Template-Version + JSON-Daten) immer identische Ausgabedokumente erzeugen. Essentiell für Regressionstests.
Dokumentengenerierung	Prozess der automatisierten Befüllung eines Templates mit JSON-Daten (Merge via blocpress-core) und Export als ODT/PDF/RTF (Format-Konvertierung via LibreOffice in blocpress-render). Entity E-3, TF-5.
Freigabeprozess	Workflow zur Qualitätssicherung von Templates: Eingereicht → In Prüfung → Freigegeben/Abgelehnt. Gesteuert von blocpress-proof. Entity E-4, UC-8, UC-9.
IF-Bedingung	Bedingung in einem Template, die Inhalte basierend auf JSON-Daten dynamisch ein- oder ausblendet.

Begriff	Definition
Dynamic Import	JavaScript `import()`-Ausdruck zum dynamischen Laden von ES-Modulen zur Laufzeit. Verwendet von blocpress-studio zum Laden der Web Component-Bundles von den jeweiligen SCS-Modulen.
LibreOffice headless	LibreOffice-Installation ohne grafische Oberfläche, die über UNO-API programmgesteuert bedient wird. Version ≥ 24 erforderlich. Eingesetzt ausschließlich in blocpress-render für Format-Export (ODT → PDF/RTF). Validierung und Merge laufen über blocpress-core ohne LibreOffice.
Message Queue	RabbitMQ oder Kafka zur asynchronen Verarbeitung zeitaufwändiger Operationen (Dokumentengenerierung, Regressionstests).
Micro-Frontend	Architekturmuster, bei dem die UI eines SCS-Moduls als eigenständige Einheit (Web Component) in eine gemeinsame Portal-Shell integriert wird.
ODT	OpenDocument Text Format - Standardformat für LibreOffice-Textdokumente. Template-Eingabeformat für blocpress.
Prüfer	Konsolidierte Rolle: Vereint Qualitätsmanager, Freigeber, Test-Manager und Compliance-Reviewer. Arbeitet in blocpress-proof.
Punkt-Notation	Notation zur Referenzierung von JSON-Pfaden in User-Fields, z.B. `kunde.name` oder `vertrag.positionen[0].preis`.
Regressionstest	Automatisierter Test, der prüft, ob Template-Änderungen ungewollte Auswirkungen auf generierte Dokumente haben. Ausgeführt in blocpress-proof. UC-16, TF-6.
Self-Contained System (SCS)	Architekturmuster: Unabhängig deploybare Einheit mit eigener UI, eigenem Backend und eigener Datenhaltung. blocpress besteht aus fünf SCS.
Storage Abstraction Layer	Interface (`StorageService`) zur Entkopplung von Binärdaten-Speicherung. Ermöglicht Wechsel von PostgreSQL bytea zu S3. Wird in jedem SCS-Modul eingesetzt.
Stufenübergabe	Kopie-basierte Überführung von Templates zwischen Datenstufen: workbench → proof (zur Prüfung) → production (nach Freigabe). Optional: Baseline-Pull von production → proof für turnusmäßige Erneuerung.
Template	LibreOffice-Dokument (ODT) mit Platzhaltern (User-Fields), Wiederholungsgruppen und Bedingungen. Existiert in drei Stufen (workbench/proof/production). Entity E-1.
Test Case	Testfall für ein Template mit JSON-Testdaten und Baseline-PDF. Verwaltet in blocpress-proof. Entity E-6.
Testpool	Sammlung von Test Cases für gemeinsame Ausführung (z.B. alle Tests für ein Template). Verwaltet in blocpress-proof. Entity E-7.
User-Field	Platzhalter in einem Template, der durch JSON-Daten befüllt wird. Name in Punkt-Notation (z.B. `kunde.name`).

Begriff	Definition
Vier-Augen-Prinzip	Sicherheitskonzept: Template-Designer darf nicht gleichzeitig Freigeber sein. Technisch erzwungen in TF-2 (blocpress-proof).
Web Component	Browser-Standard (Custom Elements + Shadow DOM) für wiederverwendbare UI-Komponenten. Jedes SCS-Modul liefert seine UI als Web Component (z.B. `<bp-workbench>`), die von blocpress-studio geladen wird.
Wiederholungsgruppe	Section oder Table in einem Template, die für jedes Element eines JSON-Arrays dupliziert und befüllt wird.
Workflow-Engine	Komponente in blocpress-proof zur Steuerung von Freigabeprozessen und Statusübergängen. Prüft Berechtigungen und Vier-Augen-Prinzip.

13. Über arc42

arc42, das Template zur Dokumentation von Software- und Systemarchitekturen.

Template Version 8.2 DE. (basiert auf AsciiDoc Version) Erstellt, gepflegt und © von Dr. Peter Hruschka, Dr. Gernot Starke und Mitwirkenden. Siehe https://arc42.org.

13.1. Lizenz