blocpress

document automation

# Element Design Concept

## Goals

### G-1: Automatisierte Template-basierte Dokumentengenerierung

**Description:** blocpress-render soll LibreOffice-Templates (ODT) mit strukturierten JSON-Daten automatisiert zusammenführen (Merge-Pipeline via blocpress-core) und Dokumente in den Formaten ODT, PDF und RTF generieren (Format-Export via LibreOffice). Die Generierung muss deterministisch, performant und über die öffentliche REST-API zugänglich sein.
*Systemziel: Automatisierte Dokumentengenerierung | Geschäftsprozess: Dokument automatisiert generieren*

### G-2: Zentrale Verwaltung von Templates und Bausteinen

**Description:** blocpress-workbench soll alle Templates, Bausteine und deren Versionen im Schema `workbench` speichern und verwalten. Änderungen müssen nachvollziehbar sein und der Freigabestatus jederzeit erkennbar.
*Systemziel: Zentrale Template-Verwaltung | Entities: Information Architecture*

### G-3: Workflow-gesteuerte Qualitätssicherung

**Description:** blocpress-proof soll einen mehrstufigen Freigabeprozess mit definiertem Vier-Augen-Prinzip (Template-Designer ≠ Prüfer) implementieren. Templates dürfen nur nach erfolgreicher Prüfung in das production-Schema übernommen werden.
*Systemziel: Workflow-gesteuerte Qualitätssicherung | Geschäftsprozess: Template erstellen und freigeben*

### G-4: Turnusmäßige Compliance-Reviews

**Description:** blocpress-proof soll automatisch Review-Zyklen (1, 3 oder 5 Jahre) überwachen, fällige Reviews identifizieren und für Prüfer sichtbar machen.

*Systemziel: Compliance-Unterstützung | Geschäftsprozess: Turnusmäßiges Compliance Review*

## G-5: Automatisierte Regressionstests

**Description:** blocpress-proof soll Testdaten verwalten, Test-PDFs generieren (via blocpress-render), mit Baselines vergleichen und Abweichungen erkennen.
*Systemziel: Umfassende Testunterstützung | Geschäftsprozess: Template testen und Regressionstests*

## G-6: Performante Inhaltssuche

**Description:** blocpress-workbench soll eine Integration mit Elasticsearch bereitstellen, um fachliche Konstrukte über alle Templates und Bausteine hinweg innerhalb von 2 Sekunden zu finden.
*Systemziel: Intelligente Inhaltssuche | Geschäftsprozess: Fachliche Konstrukte recherchieren*

# User Interfaces

## UI-1: Template-Verwaltungsoberfläche (blocpress-workbench)

**Description:** Web Component `<bp-workbench>` zum Hochladen, Bearbeiten, Anzeigen und Verwalten von Templates. Wird von Template-Designern genutzt. Geladen via Dynamic Import in blocpress-studio. Bietet Dashboard-Ansicht mit Status-Filtern und Workflow-Funktionen (UC-5).

**Actions:**

- Template hochladen (UC-1)
- Template zur Freigabe einreichen (UC-2)
- Template-Details anzeigen (UC-3)
- Template-Versionen vergleichen (UC-4)
- Template Management Dashboard: Status-Filter, Workflow-Übergänge, Kopieren, Re-Upload (UC-5)
- Baustein erstellen/bearbeiten (UC-5)

**Visible Data:**

- Template-Dashboard: Name, Status (DRAFT/SUBMITTED/APPROVED/REJECTED), Validierungsstatus, Erstellungsdatum (aus E-1: Template)
- Status-Filter: Übersicht nach Status mit Zählern
- Template-Details: ID, Name, Beschreibung, Version, Status, User-Fields, Wiederholungsgruppen, Bedingungen (aus E-1: Template)
- Validierungsergebnisse: Fehler, Warnungen (aus TF-1: Template validieren)
- Baustein-Liste: Name, Typ, verwendende Templates (aus E-2: Baustein)

# UI-2: Freigabe-Oberfläche (blocpress-proof)

**Description:** Bestandteil des Web Components `<bp-proof>` zur Prüfung und Freigabe von Templates. Wird von Prüfern genutzt. Geladen via Dynamic Import in blocpress-studio.

**Actions:**

- Template zur Prüfung öffnen (UC-6)
- Kommentar hinzufügen (UC-7)
- Template freigeben (UC-8)
- Template ablehnen (UC-9)
- Testdokument mit Template generieren (UC-10)

**Visible Data:**

- Eingereichte Templates: Name, Version, Einreicher, Einreichdatum (aus E-1: Template, E-4: Freigabeprozess)
- Template-Details zur Prüfung: Komplette Metadaten, Validierungsergebnisse (aus E-1: Template)
- Freigabe-Historie: Prüfer, Kommentare, Datum, Entscheidung (aus E-4: Freigabeprozess)

# UI-3: Compliance-Review-Oberfläche (blocpress-proof)

**Description:** Bestandteil des Web Components `<bp-proof>` zur Durchführung turnusmäßiger Reviews. Wird von Prüfern genutzt.

**Actions:**

- Fällige Reviews anzeigen (UC-11)
- Review durchführen (UC-12)
- Review-Ergebnis speichern (UC-13)

**Visible Data:**

- Review-Liste: Template-Name, Review-Typ, Fälligkeitsdatum, Status (aus E-5: Compliance Review)
- Template-Inhalte zur Review (aus E-1: Template)
- Review-Historie: Vergangene Reviews, Kommentare (aus E-5: Compliance Review)

# UI-4: Test-Management-Oberfläche (blocpress-proof)

**Description:** Bestandteil des Web Components `<bp-proof>` zur Verwaltung von Testdaten und Regressionstests. Wird von Prüfern genutzt.

**Actions:**

- Test Case erstellen (UC-14)
- Testdaten bearbeiten (UC-15)
- Test ausführen (UC-16)
- Baseline aktualisieren (UC-17)
- Testpool verwalten (UC-18)

**Visible Data:**

- Test Cases: Name, Template, Status, letzte Ausführung (aus E-6: Test Case)
- Testdaten: JSON-Struktur (aus E-6: Test Case)
- Test-Ergebnisse: Bestanden/Fehlgeschlagen, Abweichungen (aus E-6: Test Case)
- Testpools: Name, enthaltene Tests, Erfolgsrate (aus E-7: Testpool)

# UI-5: Such-Oberfläche (blocpress-workbench)

**Description:** Bestandteil des Web Components `<bp-workbench>` zur Recherche fachlicher Konstrukte. Wird von Template-Designern genutzt.

**Actions:**

- Nach Begriff suchen (UC-19)
- Suchergebnisse filtern (UC-20)
- Template an Fundstelle öffnen (UC-21)

**Visible Data:**

- Suchergebnisse: Template-Name, Baustein-Name, Kontext, Position (aus E-8: Content Index via TI-7)
- Suchfilter: Template-Typ, Baustein-Kategorie, Status (aus E-1, E-2)

# UI-6: Administrator-Oberfläche (blocpress-admin)

**Description:** Web Component `<bp-admin>` zur Verwaltung von Benutzern, Rollen und System-Konfiguration. Wird von Administratoren genutzt. Geladen via Dynamic Import in blocpress-studio.

**Actions:**

- Benutzer verwalten (UC-22)
- Rollen zuweisen (UC-23)
- System-Konfiguration ändern (UC-24)

**Visible Data:**

- Benutzerliste: Name, E-Mail, Rollen, Status (aus E-9: Benutzer)

- Rollendefinitionen: Name, Berechtigungen (aus E-9: Benutzer)
- System-Parameter: Review-Zyklen, Limits, API-Keys (aus E-10: Konfiguration)

# Use Cases

## UC-1: Template hochladen

**Goal:** G-2: Zentrale Verwaltung von Templates und Bausteinen

**Prerequisites:**

- Benutzer ist authentifiziert
- Benutzer hat Rolle "Template-Designer"

**Actors:**

- Template-Designer (User Type aus System Design)
- `<bp-workbench>` Web Component (geladen in blocpress-studio)
- blocpress-workbench (SCS-Modul)

**Main Scenario:**

1. Template-Designer wählt über UI-1 die Aktion "Template hochladen"
2. Template-Designer wählt eine ODT-Datei vom lokalen Dateisystem
3. `<bp-workbench>` sendet die Datei über REST-API (POST /api/workbench/templates) an blocpress-workbench
4. blocpress-workbench ruft TF-1 (Template validieren) auf
5. TF-1 analysiert User-Fields, Wiederholungsgruppen und IF-Bedingungen
6. blocpress-workbench speichert Template-Binärdaten und Metadaten in E-1 (Template, Schema: workbench) mit Status "Entwurf"
7. blocpress-workbench sendet Indexierungs-Anfrage über TI-7 (Elasticsearch API) an Content Search Engine
8. blocpress-workbench gibt Template-ID und Validierungsergebnisse an `<bp-workbench>` zurück
9. `<bp-workbench>` zeigt Template-Details und Validierungsergebnisse auf UI-1 an

**Alternative Scenarios:**

4a. Validierung schlägt fehl (ungültige ODT-Struktur) 5a. blocpress-workbench gibt Fehlermeldung zurück 6a. `<bp-workbench>` zeigt Fehlermeldung auf UI-1 an 7a. Use Case endet ohne Template-Speicherung

# UC-2: Template zur Freigabe einreichen

**Goal:** G-3: Workflow-gesteuerte Qualitätssicherung

**Prerequisites:**

- UC-1 (Template hochladen) erfolgreich abgeschlossen
- Template hat Status "Entwurf"

**Actors:**

- Template-Designer (User Type aus System Design)
- `<bp-workbench>` Web Component (geladen in blocpress-studio)
- blocpress-workbench (SCS-Modul)
- blocpress-proof (SCS-Modul, Ziel der Stufenübergabe)

**Main Scenario:**

1. Template-Designer wählt Template auf UI-1 aus
2. Template-Designer wählt Aktion "Zur Freigabe einreichen"
3. `<bp-workbench>` sendet Anfrage über REST-API (POST /api/workbench/templates/{id}/submit) an blocpress-workbench
4. blocpress-workbench kopiert Template-Daten via interner REST-API an blocpress-proof (Stufenübergabe workbench → proof)
5. blocpress-proof erstellt Eintrag in E-4 (Freigabeprozess, Schema: proof) mit Status "Eingereicht"
6. blocpress-proof prüft, ob Benutzer berechtigt ist (Designer ≠ Prüfer)
7. blocpress-workbench bestätigt erfolgreiche Einreichung an `<bp-workbench>`
8. `<bp-workbench>` zeigt Bestätigung auf UI-1 an
9. Eingereichte Templates sind für Prüfer auf UI-2 sichtbar

**Alternative Scenarios:**

6a. Benutzer ist auch Prüfer (Vier-Augen-Prinzip verletzt) 7a. blocpress-proof gibt Fehlermeldung zurück 8a. `<bp-workbench>` zeigt Fehlermeldung auf UI-1 an 9a. Use Case endet ohne Statusänderung

# UC-3: Template-Details anzeigen

**Goal:** G-2: Zentrale Verwaltung von Templates und Bausteinen

**Prerequisites:**

- Benutzer ist authentifiziert
- Mindestens ein Template existiert

**Actors:**

- Template-Designer (User Type aus System Design)
- `<bp-workbench>` Web Component (geladen in blocpress-studio)
- blocpress-workbench (SCS-Modul)

**Main Scenario:**

1. Template-Designer wählt Template aus Liste auf UI-1 aus
2. `<bp-workbench>` sendet Anfrage über REST-API (GET /api/workbench/templates/{id}) an blocpress-workbench
3. blocpress-workbench lädt Template-Metadaten aus E-1 (Template, Schema: workbench)
4. blocpress-workbench lädt zugehörige Bausteine aus E-2 (Baustein, Schema: workbench)
5. blocpress-workbench gibt vollständige Template-Informationen an `<bp-workbench>` zurück
6. `<bp-workbench>` zeigt Template-Details auf UI-1 an (Name, Version, Status, User-Fields, Wiederholungsgruppen, Bausteine)

# UC-8: Template freigeben

**Goal:** G-3: Workflow-gesteuerte Qualitätssicherung

**Prerequisites:**

- Template hat Status "Eingereicht" oder "In Prüfung" (in Schema: proof)
- Benutzer hat Rolle "Prüfer"
- UC-2 (Template zur Freigabe einreichen) erfolgreich abgeschlossen

**Actors:**

- Prüfer (User Type aus System Design)
- `<bp-proof>` Web Component (geladen in blocpress-studio)
- blocpress-proof (SCS-Modul)
- blocpress-render (SCS-Modul, Ziel der Stufenübergabe)

**Main Scenario:**

1. Prüfer öffnet eingreichtes Template auf UI-2
2. Prüfer prüft Template-Inhalt und Validierungsergebnisse
3. Prüfer wählt Aktion "Freigeben"
4. `<bp-proof>` sendet Anfrage über REST-API (POST /api/proof/templates/{id}/approve) an blocpress-proof
5. blocpress-proof ruft TF-2 (Workflow-Status ändern) auf
6. TF-2 prüft, ob Benutzer berechtigt ist und nicht der ursprüngliche Designer ist
7. TF-2 ändert Status in E-1 (Template, Schema: proof) zu "Freigegeben"

8. TF-2 aktualisiert E-4 (Freigabeprozess, Schema: proof) mit Prüfer, Freigabedatum und Status "Freigegeben"
9. blocpress-proof kopiert freigegebenes Template via interner REST-API an blocpress-render (Stufenübergabe proof → production)
10. blocpress-proof ruft TF-4 (Review-Datum berechnen) auf
11. TF-4 erstellt Eintrag in E-5 (Compliance Review, Schema: proof) mit berechnetem nächsten Review-Datum
12. blocpress-proof bestätigt erfolgreiche Freigabe an `<bp-proof>`
13. `<bp-proof>` zeigt Bestätigung auf UI-2 an

**Alternative Scenarios:**

6a. Prüfer ist identisch mit Designer (Vier-Augen-Prinzip verletzt) 7a. TF-2 gibt Fehlermeldung zurück 8a. `<bp-proof>` zeigt Fehlermeldung auf UI-2 an 9a. Use Case endet ohne Freigabe

# UC-10: Testdokument mit Template generieren

**Goal:** G-3: Workflow-gesteuerte Qualitätssicherung

**Prerequisites:**

- Template existiert (in Schema: proof)
- Benutzer hat Rolle "Prüfer"

**Actors:**

- Prüfer (User Type aus System Design)
- `<bp-proof>` Web Component (geladen in blocpress-studio)
- blocpress-proof (SCS-Modul)
- blocpress-render (SCS-Modul, Dokumentengenerierung)

**Main Scenario:**

1. Prüfer öffnet Template auf UI-2
2. Prüfer wählt Aktion "Testdokument generieren"
3. Prüfer gibt JSON-Testdaten ein oder wählt vorhandene Testdaten
4. `<bp-proof>` sendet Anfrage über REST-API (POST /api/proof/templates/{id}/generate) an blocpress-proof
5. blocpress-proof validiert JSON-Daten gegen Template-Schema
6. blocpress-proof sendet Generierungsanfrage via interner REST-API an blocpress-render
7. blocpress-render ruft TF-5 (Dokument generieren) auf
8. TF-5 befüllt Template mit JSON-Daten (User-Fields, Wiederholungsgruppen, IF-Bedingungen)
9. TF-5 exportiert Dokument im gewünschten Format (ODT/PDF/RTF) via LibreOffice headless

10. blocpress-render gibt generiertes Dokument an blocpress-proof zurück
11. blocpress-proof speichert Ergebnis in E-3 (Dokumentengenerierung, Schema: proof) mit Status "Erfolgreich"
12. blocpress-proof gibt Download-Link an `<bp-proof>` zurück
13. `<bp-proof>` zeigt Download-Link auf UI-2 an

**Alternative Scenarios:**

5a. JSON-Daten entsprechen nicht dem Template-Schema 6a. blocpress-proof gibt Validierungsfehler zurück 7a. `<bp-proof>` zeigt Fehlermeldung auf UI-2 an 8a. Use Case endet ohne Dokumentengenerierung

9a. LibreOffice-Export schlägt fehl 10a. blocpress-render gibt Fehler an blocpress-proof zurück 11a. blocpress-proof aktualisiert Status in E-3 auf "Fehlgeschlagen" mit Fehlerprotokoll 12a. `<bp-proof>` zeigt Fehlermeldung auf UI-2 an 13a. Use Case endet mit Fehler

# UC-12: Review durchführen

**Goal:** G-4: Turnusmäßige Compliance-Reviews

**Prerequisites:**

- Compliance Review ist fällig (Status "Anstehend", in Schema: proof)
- Benutzer hat Rolle "Prüfer"

**Actors:**

- Prüfer (User Type aus System Design)
- `<bp-proof>` Web Component (geladen in blocpress-studio)
- blocpress-proof (SCS-Modul)

**Main Scenario:**

1. Prüfer öffnet fälliges Review auf UI-3
2. `<bp-proof>` sendet Anfrage über REST-API (GET /api/proof/reviews/{id}) an blocpress-proof
3. blocpress-proof lädt Review-Details aus E-5 (Compliance Review, Schema: proof)
4. blocpress-proof lädt zugehöriges Template aus E-1 (Template, Schema: proof)
5. `<bp-proof>` zeigt Template-Inhalte und Review-Informationen auf UI-3 an
6. Prüfer prüft inhaltliche Aktualität gegen rechtliche/fachliche Anforderungen
7. Prüfer gibt Review-Ergebnis ein ("Freigegeben" oder "Überarbeitung erforderlich")
8. Prüfer fügt Kommentare hinzu
9. `<bp-proof>` sendet Review-Ergebnis über REST-API (POST /api/proof/reviews/{id}/complete) an blocpress-proof
10. blocpress-proof aktualisiert E-5 (Compliance Review, Schema: proof) mit Reviewer, Review-

Datum, Ergebnis und Kommentaren

11. blocpress-proof ruft TF-4 (Review-Datum berechnen) auf
12. TF-4 berechnet nächstes Review-Datum und speichert es in E-5
13. blocpress-proof bestätigt erfolgreiche Review-Speicherung an `<bp-proof>`
14. `<bp-proof>` zeigt Bestätigung auf UI-3 an

**Alternative Scenarios:**

7a. Review-Ergebnis ist "Archivieren" 8a. blocpress-proof ändert Template-Status in E-1 (Schema: proof) zu "Archiviert" 9a. Template wird nicht mehr für Produktiv-Generierung verwendet 10a. Weiter mit Schritt 13

# UC-14: Test Case erstellen

**Goal:** G-5: Automatisierte Regressionstests

**Prerequisites:**

- Template existiert und ist freigegeben (in Schema: proof)
- Benutzer hat Rolle "Prüfer"

**Actors:**

- Prüfer (User Type aus System Design)
- `<bp-proof>` Web Component (geladen in blocpress-studio)
- blocpress-proof (SCS-Modul)
- blocpress-render (SCS-Modul, Dokumentengenerierung)

**Main Scenario:**

1. Prüfer wählt Template auf UI-4 aus
2. Prüfer wählt Aktion "Test Case erstellen"
3. Prüfer gibt Test-Name und Beschreibung ein
4. Prüfer erstellt oder lädt JSON-Testdaten
5. Prüfer wählt "Test ausführen und als Baseline speichern"
6. `<bp-proof>` sendet Anfrage über REST-API (POST /api/proof/testcases) mit Template-ID, Name, Beschreibung und JSON-Testdaten
7. blocpress-proof erstellt Eintrag in E-6 (Test Case, Schema: proof) mit Status "Baseline fehlt"
8. blocpress-proof sendet Generierungsanfrage via interner REST-API an blocpress-render
9. blocpress-render ruft TF-5 (Dokument generieren) auf und gibt PDF zurück
10. blocpress-proof speichert generiertes PDF als Baseline in E-6 (Test Case, Schema: proof)
11. blocpress-proof aktualisiert Status in E-6 auf "Bestanden"

12. blocpress-proof gibt Test-Case-ID an `<bp-proof>` zurück
13. `<bp-proof>` zeigt Test Case Details und Baseline-PDF auf UI-4 an

# UC-16: Test ausführen

**Goal:** G-5: Automatisierte Regressionstests

**Prerequisites:**

- Test Case existiert mit Baseline (in Schema: proof)
- Benutzer hat Rolle "Prüfer" oder Test wird automatisch ausgeführt

**Actors:**

- Prüfer (User Type aus System Design)
- `<bp-proof>` Web Component (geladen in blocpress-studio)
- blocpress-proof (SCS-Modul)
- blocpress-render (SCS-Modul, Dokumentengenerierung)

**Main Scenario:**

1. Prüfer wählt Test Case auf UI-4 aus
2. Prüfer wählt Aktion "Test ausführen"
3. `<bp-proof>` sendet Anfrage über REST-API (POST /api/proof/testcases/{id}/execute) an blocpress-proof
4. blocpress-proof lädt Test Case aus E-6 (Test Case, Schema: proof)
5. blocpress-proof sendet Generierungsanfrage via interner REST-API an blocpress-render
6. blocpress-render ruft TF-5 (Dokument generieren) auf und gibt aktuelles PDF zurück
7. blocpress-proof ruft TF-6 (PDF vergleichen) auf mit Baseline-PDF und aktuellem PDF
8. TF-6 vergleicht PDFs pixel- oder textbasiert
9. TF-6 gibt Vergleichsergebnis zurück (Bestanden/Fehlgeschlagen, Abweichungen)
10. blocpress-proof speichert aktuelles PDF und Vergleichsergebnis in E-6 (Test Case, Schema: proof)
11. blocpress-proof aktualisiert Status und letzte Ausführung in E-6
12. blocpress-proof gibt Test-Ergebnis an `<bp-proof>` zurück
13. `<bp-proof>` zeigt Test-Ergebnis auf UI-4 an (bei Abweichungen: visueller Vergleich)

**Alternative Scenarios:**

8a. PDFs sind identisch 9a. TF-6 gibt "Bestanden" zurück 10a. Weiter mit Schritt 10

8b. PDFs unterscheiden sich 9b. TF-6 gibt "Fehlgeschlagen" mit Abweichungsdetails zurück 10b. Weiter mit Schritt 10

# UC-19: Nach Begriff suchen

**Goal:** G-6: Performante Inhaltssuche

**Prerequisites:**

- Benutzer ist authentifiziert
- Content Search Engine ist aktiv und indexiert

**Actors:**

- Template-Designer (User Type aus System Design)
- `<bp-workbench>` Web Component (geladen in blocpress-studio)
- blocpress-workbench (SCS-Modul)
- Content Search Engine (Software Element aus System Design)

**Main Scenario:**

1. Template-Designer gibt Suchbegriff (z.B. "Beitragsanpassung") in Suchfeld auf UI-5 ein
2. `<bp-workbench>` sendet Such-Anfrage über REST-API (GET /api/workbench/search?query=...) an blocpress-workbench
3. blocpress-workbench sendet Such-Anfrage über TI-7 (Elasticsearch API) an Content Search Engine
4. Content Search Engine durchsucht Index und gibt Treffer zurück (Template-ID, Baustein-ID, Kontext, Position)
5. blocpress-workbench reichert Treffer mit Metadaten aus E-1 (Template, Schema: workbench) und E-2 (Baustein, Schema: workbench) an
6. blocpress-workbench gibt Suchergebnisse an `<bp-workbench>` zurück
7. `<bp-workbench>` zeigt Suchergebnisse auf UI-5 an (gruppiert nach Templates und Bausteinen)

**Alternative Scenarios:**

4a. Keine Treffer gefunden 5a. Content Search Engine gibt leere Liste zurück 6a. blocpress-workbench gibt "Keine Ergebnisse" an `<bp-workbench>` zurück 7a. `<bp-workbench>` zeigt "Keine Ergebnisse gefunden" auf UI-5 an

# UC-22: Benutzer verwalten

**Goal:** G-3: Workflow-gesteuerte Qualitätssicherung (unterstützend)

**Prerequisites:**

- Benutzer hat Rolle "Administrator"

**Actors:**

- Administrator (User Type aus System Design)
- `<bp-admin>` Web Component (geladen in blocpress-studio)
- blocpress-admin (SCS-Modul)

**Main Scenario:**

1. Administrator öffnet Benutzerverwaltung auf UI-6
2. `<bp-admin>` sendet Anfrage über REST-API (GET /api/admin/users) an blocpress-admin
3. blocpress-admin lädt Benutzerliste aus E-9 (Benutzer, Schema: admin)
4. `<bp-admin>` zeigt Benutzerliste auf UI-6 an
5. Administrator wählt Benutzer aus und ändert Rollen
6. `<bp-admin>` sendet Update über REST-API (PUT /api/admin/users/{id}/roles) an blocpress-admin
7. blocpress-admin aktualisiert Rollen in E-9 (Benutzer, Schema: admin) und speichert das JWT Subject (sub-Claim) als Referenz
8. blocpress-admin bestätigt erfolgreiche Aktualisierung an `<bp-admin>`
9. `<bp-admin>` zeigt Bestätigung auf UI-6 an

# Technical Functions

## TF-1: Template validieren (blocpress-workbench, via blocpress-core)

**Input:**

- templateBinary: byte[] (ODT-Datei)

**Output:**

- validationResult: ValidationResult
- isValid: boolean
- errors: List<ValidationError>
- warnings: List<ValidationWarning>
- userFields: List<UserField>
- repetitionGroups: List<RepetitionGroup>
- conditions: List<Condition>

**Main Functional Flow:**

1. Lade templateBinary als ODT-Dokument über blocpress-core (odfdom)
2. Prüfe, ob Dokument gültige ODT-Struktur hat
3. Extrahiere alle User-Fields aus dem Dokument

4. Für jedes User-Field: Prüfe, ob Name in gültiger Punkt-Notation ist (z.B. "kunde.name")
5. Identifiziere alle Sections und Tables im Dokument
6. Für jede Section/Table: Prüfe, ob sie als Wiederholungsgruppe markiert ist (enthält Array-Referenz)
7. Extrahiere alle IF-Bedingungen (OpenDocument-Felder mit Condition)
8. Validiere, dass alle referenzierten Felder in Bedingungen existieren
9. Erstelle ValidationResult mit isValid=true/false, Liste von Fehlern, Warnungen und extrahierten Elementen
10. Gebe ValidationResult zurück

**Alternative Functional Flows:**

2a. ODT-Struktur ist ungültig (keine valide ZIP-Datei oder fehlendes content.xml) 3a. Setze isValid=false 4a. Füge Fehler "Ungültige ODT-Datei" zu errors hinzu 5a. Gebe ValidationResult zurück

4a. User-Field hat ungültige Punkt-Notation (z.B. enthält Leerzeichen) 5a. Füge Warnung zu warnings hinzu 6a. Fahre mit nächstem Field fort

8a. IF-Bedingung referenziert nicht existierendes Field 9a. Füge Fehler zu errors hinzu 10a. Setze isValid=false

# TF-2: Workflow-Status ändern (blocpress-proof)

**Input:**

- templateId: Long
- newStatus: String ("Entwurf", "Eingereicht", "In Prüfung", "Freigegeben", "Abgelehnt", "Archiviert")
- userId: Long
- comment: String (optional)

**Output:**

- success: boolean
- errorMessage: String (optional)

**Main Functional Flow:**

1. Lade Template mit templateId aus E-1 (Template)
2. Lade aktuellen Status des Templates
3. Lade Benutzer mit userId aus E-9 (Benutzer, Schema: admin) via blocpress-admin-API oder JWT-Claims
4. Prüfe, ob Statusübergang erlaubt ist (z.B. "Entwurf" → "Eingereicht" erlaubt)

5. Prüfe Vier-Augen-Prinzip: Wenn newStatus="Freigegeben", prüfe dass userId ≠ Template.erstellerId
6. Aktualisiere Status in E-1 (Template) auf newStatus
7. Wenn Freigabeprozess in E-4 (Freigabeprozess) für Template existiert, aktualisiere ihn
8. Sonst erstelle neuen Eintrag in E-4 mit templateId, Status, userId, Datum
9. Wenn comment vorhanden, speichere in E-4
10. Setze success=true
11. Gebe success zurück

**Alternative Functional Flows:**

1a. Template mit templateId existiert nicht 2a. Setze success=false, errorMessage="Template nicht gefunden" 3a. Gebe success und errorMessage zurück

4a. Statusübergang nicht erlaubt (z.B. "Freigegeben" → "Entwurf") 5a. Setze success=false, errorMessage="Statusübergang nicht erlaubt" 6a. Gebe success und errorMessage zurück

5a. Vier-Augen-Prinzip verletzt (userId = Template.erstellerId) 6a. Setze success=false, errorMessage="Designer darf nicht freigeben (Vier-Augen-Prinzip)" 7a. Gebe success und errorMessage zurück

# TF-4: Review-Datum berechnen (blocpress-proof)

**Input:**

- templateId: Long
- reviewCycle: Integer (1, 3 oder 5 Jahre)

**Output:**

- nextReviewDate: LocalDate

**Main Functional Flow:**

1. Lade Template mit templateId aus E-1 (Template)
2. Lade aktuelles Datum (heute)
3. Berechne nextReviewDate = heute + reviewCycle Jahre
4. Gebe nextReviewDate zurück

# TF-5: Dokument generieren (blocpress-render, Merge via blocpress-core)

**Input:**

- templateId: Long

- templateVersion: Integer (optional, default: neueste freigegebene Version)
- jsonData: String (JSON-Struktur)
- outputFormat: String ("ODT", "PDF", "RTF")

**Output:**

- documentBinary: byte[]
- success: boolean
- errorMessage: String (optional)

**Main Functional Flow:**

1. Lade Template mit templateId und templateVersion aus E-1 (Template, Schema: production)
2. Prüfe, ob Template Status "Freigegeben" hat
3. Lade Template-Binärdaten aus E-1
4. Parse jsonData zu JSON-Objekt
5. **Merge via blocpress-core (Schritte 5-8 laufen ohne LibreOffice):**
6. Für jedes User-Field im Template: Extrahiere Wert aus JSON via Punkt-Notation und setze Field-Wert
7. Für jede Wiederholungsgruppe (Section/Table): Iteriere über Array aus JSON und dupliziere Gruppe
8. Für jede IF-Bedingung: Evaluiere Bedingung basierend auf JSON-Daten und zeige/verstecke Inhalt
9. **Format-Export via LibreOffice Processor:**
10. Exportiere gemergtes ODT im gewünschten outputFormat (ODT, PDF oder RTF) über LibreOffice headless
11. Konvertiere exportiertes Dokument zu byte[]
12. Setze success=true
13. Gebe documentBinary und success zurück

**Alternative Functional Flows:**

2a. Template hat nicht Status "Freigegeben" 3a. Setze success=false, errorMessage="Template nicht freigegeben" 4a. Gebe success und errorMessage zurück

4a. jsonData ist kein valides JSON 5a. Setze success=false, errorMessage="Ungültige JSON-Daten" 6a. Gebe success und errorMessage zurück

6a. Referenziertes Feld in Punkt-Notation existiert nicht in JSON 7a. Setze Field-Wert auf "" (leerer String) oder Default-Wert 8a. Fahre mit nächstem Field fort

9a. LibreOffice-Export schlägt fehl 10a. Setze success=false, errorMessage="Export fehlgeschlagen: " + Fehlerdetails 11a. Schließe LibreOffice-Dokument 12a. Gebe success und errorMessage zurück

# TF-6: PDF vergleichen (blocpress-proof)

**Input:**

- baselinePdf: byte[]
- currentPdf: byte[]

**Output:**

- comparisonResult: ComparisonResult
- isIdentical: boolean
- differences: List<Difference> (Seite, Position, Beschreibung)

**Main Functional Flow:**

1. Lade baselinePdf und currentPdf als PDF-Dokumente
2. Prüfe, ob beide PDFs gleiche Seitenanzahl haben
3. Für jede Seite: Extrahiere Text aus baselinePdf und currentPdf
4. Vergleiche Text Zeile für Zeile
5. Bei Unterschieden: Erstelle Difference-Objekt mit Seitennummer, Position und Beschreibung
6. Füge Difference zu differences-Liste hinzu
7. Nach Vergleich aller Seiten: Wenn differences leer, setze isIdentical=true, sonst false
8. Erstelle ComparisonResult mit isIdentical und differences
9. Gebe ComparisonResult zurück

**Alternative Functional Flows:**

2a. Seitenanzahl unterschiedlich 3a. Setze isIdentical=false 4a. Erstelle Difference "Seitenanzahl unterschiedlich: Baseline X Seiten, Current Y Seiten" 5a. Gebe ComparisonResult zurück

1a. baselinePdf oder currentPdf ist ungültiges PDF 2a. Setze isIdentical=false 3a. Erstelle Difference "PDF-Vergleich fehlgeschlagen: Ungültiges PDF" 4a. Gebe ComparisonResult zurück

# TF-7: Compliance-Reviews überprüfen (blocpress-proof)

**Input:**

- keines (läuft als geplanter Job in blocpress-proof)

**Output:**

- fälligeReviews: List<ComplianceReview>

**Main Functional Flow:**

1. Lade aktuelles Datum (heute)
2. Lade alle Templates mit Status "Freigegeben" aus E-1 (Template)
3. Für jedes Template: Lade zugehörige Compliance Reviews aus E-5 (Compliance Review)
4. Wenn kein Review existiert oder letztes Review abgeschlossen: Berechne nächstes Review-Datum via TF-4
5. Wenn nächstes Review-Datum ⇐ heute + 30 Tage oder bereits überschritten:
6. Erstelle neuen Review-Eintrag in E-5 mit Status "Anstehend"
7. Füge Review zu fälligeReviews-Liste hinzu
8. Fällige Reviews sind für Prüfer auf UI-3 sichtbar
9. Gebe fälligeReviews zurück

# Technical Interfaces

## TI-1: REST-APIs (pro SCS-Modul)

**Description:** Jedes SCS-Modul stellt eigene REST-Endpoints bereit. Nur blocpress-render ist öffentlich zugänglich; die anderen Module werden über Web Components in blocpress-studio konsumiert. Interne Kommunikation zwischen SCS-Modulen erfolgt über interne REST-APIs.

**Öffentliche API (blocpress-render):**

- `POST /api/render/template/merge` - Template mit JSON-Daten zusammenführen und als ODT/PDF/RTF zurückgeben
- Parameter: file (multipart/form-data), data (JSON), outputFormat (String)

**Interne API (blocpress-workbench, Port 8081):**

Template-Management: - `POST /api/workbench/templates` - Template hochladen - `GET /api/workbench/templates/{id}` - Template-Binärdaten abrufen - `GET /api/workbench/templates` - Template-Liste mit Status und Validierung abrufen - `GET /api/workbench/templates/{id}/details` - Template-Details mit Validierungsergebnis - `POST /api/workbench/templates/{id}/submit` - Template zur Freigabe einreichen (löst Stufenübergabe an proof aus) - `PUT /api/workbench/templates/{id}/status` - Template-Status ändern (DRAFT→SUBMITTED, etc., UC-5) - `POST /api/workbench/templates/{id}/duplicate` - Produktives Template als DRAFT-Kopie erstellen (UC-5) - `PUT /api/workbench/templates/{id}/content` - Template-Inhalt aktualisieren/neu hochladen (UC-5) - `DELETE /api/workbench/templates/{id}` - Template löschen - `GET /api/workbench/search` - Nach Begriff suchen

TestDataSet-Management (UC-20, UC-21, TF-8, UC-11): - `GET /api/workbench/templates/{id}/testdata` - Alle TestDataSets für Template abrufen - `POST /api/workbench/templates/{id}/testdata` - Neues TestDataSet erstellen {name, testData} - `PUT /api/workbench/templates/{id}/testdata/{testDataId}` - TestDataSet aktualisieren - `DELETE /api/workbench/templates/{id}/testdata/{testDataId}` - TestDataSet löschen - `POST /api/workbench/templates/{id}/testdata/{testDataId}/save-expected` - PDF als Expected Result speichern - `GET`

`/api/workbench/templates/{id}/testdata/{testDataId}/expected-pdf` - Expected PDF abrufen

**Interne API (blocpress-proof, Port 8082):**

- `POST /api/proof/templates/{id}/approve` - Template freigeben (löst Stufenübergabe an production aus)
- `GET /api/proof/reviews/{id}` - Review-Details abrufen
- `POST /api/proof/reviews/{id}/complete` - Review abschließen
- `POST /api/proof/testcases` - Test Case erstellen
- `POST /api/proof/testcases/{id}/execute` - Test ausführen
- `POST /api/proof/templates/{id}/generate` - Testdokument generieren (delegiert an blocpress-render)

**Interne API (blocpress-admin, Port 8084):**

- `GET /api/admin/users` - Benutzerliste abrufen
- `PUT /api/admin/users/{id}/roles` - Rollen aktualisieren

**Stufenübergabe-APIs (intern, nicht öffentlich):**

- `POST /api/proof/stage/receive` - Template von workbench empfangen
- `POST /api/render/stage/receive` - Template von proof empfangen

**Output:**

JSON-Response je nach Endpoint (Template-Objekte, Generierungs-Ergebnisse, Test-Ergebnisse, Suchergebnisse)

**Action:**

Jedes SCS-Modul ruft seine eigenen Technical Functions auf:

- blocpress-workbench: TF-1 (Template validieren), Schreibt E-1 (Schema: workbench), verwaltet Status-Übergänge (UC-5: Dashboard mit Workflow)
- blocpress-proof: TF-2 (Workflow-Status ändern), TF-4 (Review-Datum), TF-6 (PDF vergleichen), TF-7 (Reviews prüfen)
- blocpress-render: TF-5 (Dokument generieren)
- blocpress-admin: CRUD auf E-9 (Benutzer), E-10 (Konfiguration)

# TI-2: Datenbank-Interface (PostgreSQL, Multi-Schema)

**Description:** Zugriff auf eine gemeinsame PostgreSQL-Instanz mit getrennten Schemata pro SCS-Modul. Jedes Modul greift ausschließlich auf sein eigenes Schema zu; Cross-Schema-Zugriff erfolgt über REST-APIs (Stufenübergabe), nicht über SQL.

**Schema-Zuordnung:**

- blocpress-workbench → Schema `workbench` (E-1, E-2, E-8)
- blocpress-proof → Schema `proof` (E-1, E-3, E-4, E-5, E-6, E-7)
- blocpress-render → Schema `production` (E-1, E-3)
- blocpress-admin → Schema `admin` (E-9, E-10)

**Input:**

SQL-Queries über JPA/Hibernate mit `?currentSchema=<schema>` in der JDBC-URL

**Output:**

Entity-Objekte oder Anzahl betroffener Zeilen

**Action:**

CRUD-Operationen auf Entities des jeweiligen Schemas

# TI-3: LibreOffice API (nur blocpress-render)

**Description:** Zugriff auf LibreOffice headless (Version ≥ 24) ausschließlich für Format-Export (ODT → PDF/RTF). Wird intern nur von TF-5 (Schritt Format-Export) in blocpress-render genutzt. Die Merge-Schritte (Feldersetzung, Schleifen, Bedingungen) laufen über blocpress-core ohne LibreOffice.

**Input:**

- Gemergtes ODT-Dokument (byte[], bereits von blocpress-core verarbeitet)
- Export-Format (PDF/RTF)

**Output:**

- Exportiertes Dokument (byte[])

**Action:**

Ruft LibreOffice UNO-API auf:

- Dokument öffnen: `XComponentLoader.loadComponentFromURL()`
- Dokument exportieren: `XStorable.storeToURL()` mit Filter für PDF/RTF

# TI-4: JWT-Authentifizierung (alle SCS-Module)

**Description:** blocpress-studio hält den JWT nach Login und übergibt ihn als Attribut an die Web Components (`<bp-workbench jwt="⋯">`, `<bp-proof jwt="⋯">`, `<bp-admin jwt="⋯">`). Jedes SCS-Modul validiert den JWT eigenständig über die Quarkus SmallRye JWT Extension (`smallrye-jwt`). Alle Module teilen denselben Public Key und Issuer (konfiguriert in `application.properties`).

**Input:**

- JWT Bearer Token (im `Authorization`-Header jeder REST-Anfrage)

**Output:**

- Authentifizierungsergebnis (success/failure)
- Benutzerinformationen aus JWT-Claims (sub, name, email, groups)

**Action:**

Jedes SCS-Modul validiert JWT-Signatur gegen den konfigurierten Public Key, prüft Issuer und Expiration. Extrahiert das Subject (`sub`-Claim) und gleicht es mit E-9 (Benutzer, Schema: admin) ab — blocpress-admin stellt hierfür eine interne API bereit, oder die Rolleninformation wird direkt aus dem JWT-`groups`-Claim gelesen.

# TI-5: Message Queue Interface (blocpress-workbench, blocpress-proof)

**Description:** Asynchrone Kommunikation für zeitaufwändige Operationen (Indexierung in blocpress-workbench, Regressionstests in blocpress-proof). Wird intern von den jeweiligen SCS-Modulen genutzt.

**Input:**

- Queue-Name (String)
- Message-Payload (JSON)

**Output:**

- Message-ID

**Action:**

Sendet Message an RabbitMQ Queue, Consumer im jeweiligen SCS-Modul verarbeitet Message asynchron

# TI-7: Elasticsearch API

**Description:** Zugriff auf Content Search Engine zur Indexierung und Suche. Wird von blocpress-workbench für UC-19 (Nach Begriff suchen) genutzt.

**Input:**

- Indexierungs-Request: templateId (Long), content (String)
- Such-Request: query (String)

**Output:**

- Indexierungs-Bestätigung

- Such-Ergebnisse: List<SearchHit> mit templateId, baustein_id, kontext, position

**Action:**

Kommuniziert mit Elasticsearch über REST-API:

- Indexierung: `PUT /templates/_doc/{id}`
- Suche: `GET /templates/_search?q={query}`

# Entities

## E-1: Template

**Description:** Speichert Metadaten und Binärdaten von LibreOffice-Templates sowie deren Versionen und Status. Existiert mit identischer Struktur in drei Schemata: `workbench` (Entwurf), `proof` (Prüfung), `production` (Freigegeben).
*Geschäftsentität: Template in der Information Architecture*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige Template-ID (Primary Key, auto-generiert) |
| 2 | name | String(255) | Name des Templates |
| 3 | description | String(2000) | Beschreibung des Templates |
| 4 | version | Integer | Versionsnummer (1, 2, 3, ...) |
| 5 | status | String(50) | Status: "Entwurf", "Eingereicht", "In Prüfung", "Freigegeben", "Abgelehnt", "Archiviert" |
| 6 | ersteller_id | UUID | Foreign Key zu E-9 (Benutzer) |
| 7 | erstellungsdatum | Timestamp | Erstellungszeitpunkt |
| 8 | freigeber_id | UUID | Foreign Key zu E-9 (Benutzer), nullable |
| 9 | freigabedatum | Timestamp | Freigabezeitpunkt, nullable |

| ID | Name | Data Type | Description |
|---|---|---|---|
| 10 | template_binary | bytea | ODT-Datei als Binärdaten (PostgreSQL BLOB) |
| 11 | user_fields | JSONB | Extrahierte User-Fields aus Validierung |
| 12 | wiederholungsgruppen | JSONB | Extrahierte Wiederholungsgruppen |
| 13 | bedingungen | JSONB | Extrahierte IF-Bedingungen |
| 14 | review_zyklus | Integer | Review-Zyklus in Jahren (1, 3 oder 5) |

# E-2: Baustein

**Description:** Speichert wiederverwendbare Dokumentfragmente, die in mehreren Templates verwendet werden können. Schema: `workbench`.
*Geschäftsentität: Baustein in der Information Architecture*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige Baustein-ID (Primary Key, auto-generiert) |
| 2 | name | String(255) | Name des Bausteins |
| 3 | description | String(2000) | Beschreibung |
| 4 | typ | String(50) | Typ: "Kopfzeile", "Fußzeile", "Klausel", "Abschnitt" |
| 5 | inhalt | bytea | ODT-Fragment als Binärdaten |
| 6 | version | Integer | Versionsnummer |
| 7 | ersteller_id | UUID | Foreign Key zu E-9 (Benutzer) |
| 8 | erstellungsdatum | Timestamp | Erstellungszeitpunkt |

# E-3: Dokumentengenerierung

**Description:** Speichert Metadaten und Ergebnisse von Dokumentengenerierungen. Schema: `production`.
*Geschäftsentität: Dokumentengenerierung in der Information Architecture*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige Generierungs-ID (Primary Key, auto-generiert) |
| 2 | template_id | UUID | Foreign Key zu E-1 (Template) |
| 3 | template_version | Integer | Version des verwendeten Templates |
| 4 | json_daten | JSONB | Eingabe-JSON-Daten |
| 5 | output_format | String(10) | "ODT", "PDF" oder "RTF" |
| 6 | generiertes_dokument | bytea | Generiertes Dokument als Binärdaten |
| 7 | zeitstempel | Timestamp | Generierungszeitpunkt |
| 8 | status | String(50) | "In Bearbeitung", "Erfolgreich", "Fehlgeschlagen" |
| 9 | fehlerprotokoll | Text | Fehlerdetails bei Fehlschlag, nullable |
| 10 | requestor_id | UUID | Foreign Key zu E-9 (Benutzer) oder API-Key |

# E-4: Freigabeprozess

**Description:** Speichert Workflow-Status und Historie von Template-Freigaben. Schema: `proof`.
*Geschäftsentität: Freigabeprozess in der Information Architecture*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige Prozess-ID (Primary Key, auto-generiert) |
| 2 | template_id | UUID | Foreign Key zu E-1 (Template) |

| ID | Name | Data Type | Description |
|---|---|---|---|
| 3 | workflow_status | String(50) | "Eingereicht", "In Prüfung", "Freigegeben", "Abgelehnt" |
| 4 | pruefer_id | UUID | Foreign Key zu E-9 (Benutzer), nullable |
| 5 | pruefkommentare | Text | Kommentare des Prüfers, nullable |
| 6 | pruefdatum | Timestamp | Prüfzeitpunkt, nullable |
| 7 | freigeber_id | UUID | Foreign Key zu E-9 (Benutzer), nullable |
| 8 | freigabedatum | Timestamp | Freigabezeitpunkt, nullable |

# E-5: Compliance Review

**Description:** Speichert turnusmäßige Reviews von Templates. Schema: `proof`.
*Geschäftsentität: Compliance Review in der Information Architecture*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige Review-ID (Primary Key, auto-generiert) |
| 2 | template_id | UUID | Foreign Key zu E-1 (Template) |
| 3 | review_typ | String(20) | "1-Jahres-Review", "3-Jahres-Review", "5-Jahres-Review" |
| 4 | faelligkeitsdatum | Date | Geplantes Review-Datum |
| 5 | review_status | String(50) | "Anstehend", "In Bearbeitung", "Abgeschlossen", "Überfällig" |
| 6 | reviewer_id | UUID | Foreign Key zu E-9 (Benutzer), nullable |
| 7 | review_datum | Timestamp | Tatsächliches Review-Datum, nullable |

| ID | Name | Data Type | Description |
|---|---|---|---|
| 8 | review_ergebnis | String(50) | "Freigegeben", "Überarbeitung erforderlich", "Archivieren", nullable |
| 9 | kommentare | Text | Review-Kommentare, nullable |
| 10 | naechstes_review_datum | Date | Nächstes geplantes Review-Datum, nullable |

# E-6: Test Case

**Description:** Speichert Test Cases mit Testdaten und Baseline-PDFs für Regressionstests. Schema: `proof`.
*Geschäftsentität: Test Case in der Information Architecture*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige Test-ID (Primary Key, auto-generiert) |
| 2 | template_id | UUID | Foreign Key zu E-1 (Template) |
| 3 | template_version | Integer | Getestete Template-Version |
| 4 | test_name | String(255) | Name des Tests |
| 5 | beschreibung | String(2000) | Beschreibung |
| 6 | json_testdaten | JSONB | Test-JSON-Daten |
| 7 | erwartetes_pdf | bytea | Baseline-PDF als Binärdaten, nullable |
| 8 | aktuelles_pdf | bytea | Zuletzt generiertes PDF, nullable |
| 9 | test_status | String(50) | "Bestanden", "Fehlgeschlagen", "Baseline fehlt" |
| 10 | ersteller_id | UUID | Foreign Key zu E-9 (Benutzer) |
| 11 | erstellungsdatum | Timestamp | Erstellungszeitpunkt |

| ID | Name | Data Type | Description |
|---|---|---|---|
| 12 | letzte_ausfuehrung | Timestamp | Zeitpunkt der letzten Ausführung, nullable |
| 13 | test_typ | String(50) | "Abnahmetest", "Regressionstest" |
| 14 | abweichungen | JSONB | Details zu Abweichungen bei Fehlschlag, nullable |

# E-7: Testpool

**Description:** Gruppiert mehrere Test Cases für gemeinsame Ausführung. Schema: `proof`.
*Geschäftsentität: Testpool in der Information Architecture*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige Pool-ID (Primary Key, auto-generiert) |
| 2 | name | String(255) | Name des Testpools |
| 3 | beschreibung | String(2000) | Beschreibung |
| 4 | letzte_ausfuehrung | Timestamp | Zeitpunkt der letzten Ausführung, nullable |
| 5 | erfolgsrate | Decimal(5,2) | Erfolgsrate in Prozent (0.00 - 100.00), nullable |

# E-8: Content Index

**Description:** Referenz auf Elasticsearch-Index für fachliche Konstrukte. Wird nicht in PostgreSQL gespeichert, sondern nur als Metadaten-Verweis.
*Geschäftsentität: Content Index in der Information Architecture*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | String | Elasticsearch Document ID |
| 2 | template_id | UUID | Referenz zu E-1 (Template) |
| 3 | baustein_id | UUID | Referenz zu E-2 (Baustein), nullable |

| ID | Name | Data Type | Description |
|---|---|---|---|
| 4 | inhalt | Text | Volltext-Inhalt (in Elasticsearch indexiert) |
| 5 | letzte_aktualisierung | Timestamp | Zeitpunkt der letzten Indexierung |

*Hinweis: E-8 wird physisch in Elasticsearch gespeichert, nicht in PostgreSQL.*

# E-9: Benutzer

**Description:** Speichert Benutzerkonten und deren Rollen. Schema: `admin`.
*Geschäftsentität: Benutzer & Rolle in der Information Architecture*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige Benutzer-ID (Primary Key, auto-generiert) |
| 2 | name | String(255) | Vollständiger Name |
| 3 | email | String(255) | E-Mail-Adresse |
| 4 | rollen | JSONB | Array von Rollen: ["Template-Designer", "Prüfer", "Administrator", "API-Konsument"] |
| 5 | berechtigungen | JSONB | Array von Berechtigungen: ["Template erstellen", "Template bearbeiten", "Template prüfen", "Template freigeben", "API nutzen", "Review durchführen", "Tests verwalten"] |
| 6 | external_id | String(255) | JWT Subject (sub-Claim) |
| 7 | aktiv | Boolean | Account aktiv/deaktiviert |
| 8 | erstellungsdatum | Timestamp | Account-Erstellungszeitpunkt |

# E-10: Konfiguration

**Description:** Speichert System-Konfiguration und API-Keys. Schema: `admin`.

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige Konfigurations-ID (Primary Key, auto-generiert) |
| 2 | schluessel | String(255) | Konfigurations-Schlüssel (z.B. "default_review_cycle") |
| 3 | wert | String(2000) | Konfigurations-Wert |
| 4 | datentyp | String(50) | "String", "Integer", "Boolean", "JSON" |
| 5 | beschreibung | String(2000) | Beschreibung des Parameters |
| 6 | aenderbar | Boolean | Kann zur Laufzeit geändert werden |

# E-11: TestDataSet (blocpress-workbench)

**Description:** Speichert Test-Datensätze und optionale PDF-Baselines für Template-Designer zur lokalen Validierung. Schema: `workbench`. Separate von E-6 (Test Case in proof), da Designer ihre eigenen Testdaten verwalten und PDFs als "Expected Results" speichern können für zukünftige Regression-Tests.
*Stories: UC-20 (Auto-generiertes Formular), UC-21 (Mehrere TestDataSets), TF-8 (PDF speichern), UC-11 (Regression-Vorbereitung)*

**Attributes:**

| ID | Name | Data Type | Description |
|---|---|---|---|
| 1 | id | UUID | Eindeutige TestDataSet-ID (Primary Key, auto-generiert) |
| 2 | template_id | UUID | Foreign Key zu E-1 (Template), Schema workbench |
| 3 | name | String(255) | Sprechender Name (z.B. "Standardfall", "Grenzfall") |

| ID | Name | Data Type | Description |
|---|---|---|---|
| 4 | test_data | JSONB | Test-JSON-Daten (Struktur folgt template.validationResult.userFields) |
| 5 | expected_pdf | bytea | Optional: Baseline-PDF als Binärdaten, nullable |
| 6 | pdf_hash | String(64) | SHA-256 Hash des expected_pdf für schnelle Vergleiche, nullable |
| 7 | created_at | Timestamp | Erstellungszeitpunkt |
| 8 | updated_at | Timestamp | Letzter Aktualisierungszeitpunkt, nullable |

**Relationships:**

- One-to-Many: Template (E-1) ← → TestDataSet (E-11)
- Cascade Delete: Wenn Template gelöscht, werden auch alle TestDataSets gelöscht

**Usage in Components:**

- **bp-workbench (UI):** Zeigt TestDataSet-Liste im Tab "Testdaten", erlaubt Create/Edit/Delete
- **blocpress-workbench (Backend):** TI-1 REST-Endpoints für CRUD-Operationen, SHA-256 Hash-Berechnung
- **Future (Phase 4):** PDF-Vergleich mit E-6 (Test Case in proof) für umfassende Regression-Tests

# Quality Requirements

## QR-1: Performance bei Dokumentengenerierung

**Description:** TF-5 (Dokument generieren) muss für Standard-Dokumente (bis 20 Seiten, bis 10 Wiederholungsgruppen) innerhalb von 5 Sekunden abgeschlossen sein. Dies ist kritisch für das Benutzererlebnis bei UC-10 (Testdokument generieren) und die API-Performance bei TI-1.

## QR-2: Suchgeschwindigkeit

**Description:** UC-19 (Nach Begriff suchen) muss innerhalb von 2 Sekunden Ergebnisse liefern, auch bei 1.000+ Templates. TI-7 (Elasticsearch API) muss entsprechend optimiert sein mit geeigneten Indizes.

# QR-3: Deterministische Dokumentengenerierung

**Description:** TF-5 (Dokument generieren) muss deterministisch sein: Identische Eingaben (Template-Version + JSON-Daten) müssen zu identischen Ausgabedokumenten führen. Dies ist essentiell für TF-6 (PDF vergleichen) und die Regressionstests in UC-16.

# QR-4: Transaktionale Integrität bei Workflow-Änderungen

**Description:** TF-2 (Workflow-Status ändern) muss atomar sein: Entweder alle Änderungen (E-1, E-4) werden committed oder keine. Bei Fehler muss vollständiger Rollback erfolgen.

# QR-5: Vier-Augen-Prinzip

**Description:** TF-2 (Workflow-Status ändern) muss sicherstellen, dass Designer ≠ Freigeber. Diese Prüfung ist zwingend erforderlich für UC-8 (Template freigeben) und darf nicht umgangen werden können.

# QR-6: Skalierbarkeit der Message Queue

**Description:** TI-5 (Message Queue Interface) muss bis zu 100 gleichzeitige Dokumentengenerierungen in der Queue verwalten können ohne Performance-Degradation.

# QR-7: Datensicherheit bei Binärdaten

**Description:** E-1.10 (template_binary), E-3.6 (generiertes_dokument), E-6.7 (erwartetes_pdf) und E-6.8 (aktuelles_pdf) müssen verschlüsselt in PostgreSQL gespeichert werden, wenn personenbezogene Daten enthalten sein können. Zugriff nur über authentifizierte TI-2 (Datenbank-Interface).

# QR-8: API-Rate-Limiting

**Description:** TI-1 (REST-API) muss Rate Limiting implementieren: Maximal 1.000 Requests pro Stunde pro API-Key. Dies verhindert Missbrauch und gewährleistet faire Ressourcennutzung.

# QR-9: Audit-Logging

**Description:** Alle Aufrufe von TF-2 (Workflow-Status ändern), UC-8 (Template freigeben), UC-12 (Review durchführen) müssen in einem Audit-Log protokolliert werden (separate Tabelle, nicht modifizierbar). Log muss User-ID, Zeitstempel, Aktion und alte/neue Werte enthalten.

# QR-10: Backup & Recovery für Binärdaten

**Description:** PostgreSQL-Datenbank inklusive aller Binärdaten (E-1.10, E-3.6, E-6.7, E-6.8) muss täglich gesichert werden. Recovery Time Objective: 4h, Recovery Point Objective: 24h.

# Constraints

## C-1: Quarkus Framework

**Description:** Alle SCS-Module (blocpress-workbench, blocpress-proof, blocpress-render, blocpress-admin) müssen mit Quarkus Framework implementiert werden. Dies ermöglicht schnelle Startup-Zeiten, geringen Memory-Footprint und einheitliche Konfiguration (SmallRye JWT, Hibernate ORM, RESTEasy).

*Referenz: Quarkus.io Dokumentation*

## C-2: LibreOffice Version

**Description:** TF-5 (Dokument generieren, Format-Export in blocpress-render) muss LibreOffice Version ≥ 24 headless nutzen. Ältere Versionen werden nicht unterstützt. TF-1 (Template validieren) läuft über blocpress-core und benötigt kein LibreOffice.

*Referenz: LibreOffice Release Notes*

## C-3: PostgreSQL Version

**Description:** TI-2 (Datenbank-Interface) muss PostgreSQL Version ≥ 18 nutzen. JSONB-Indizes und bytea-Speicherung sind erforderlich.

*Referenz: PostgreSQL 18 Release Notes*

## C-4: Template-Format

**Description:** UC-1 (Template hochladen) akzeptiert nur ODT- und OTT-Formate. Andere Formate (DOCX, DOC) werden nicht unterstützt.

*Referenz: OpenDocument Format Specification (OASIS)*

## C-5: Export-Formate

**Description:** TF-5 (Dokument generieren) unterstützt nur Export nach ODT, PDF und RTF. Andere Formate (DOCX, HTML) werden nicht unterstützt.

## C-6: Datenschutz (DSGVO)

**Description:** Alle Entities mit personenbezogenen Daten (E-3.4 json_daten, E-6.6 json_testdaten) müssen DSGVO-konform verarbeitet werden. Löschfunktion muss implementiert sein.

*Referenz: DSGVO Art. 17 (Recht auf Löschung)*

## C-7: Storage Abstraction Layer

**Description:** Zugriff auf Binärdaten (E-1.10, E-3.6, E-6.7, E-6.8) muss über eine Storage Abstraction Layer erfolgen, die zukünftigen Wechsel von PostgreSQL bytea zu S3-kompatiblem Object Storage ermöglicht. Interface: `StorageService.save(byte[])` und `StorageService.load(id)`.

## C-8: Datenvolumen

**Description:** Das System ist dimensioniert für < 5.000 Dokumente + Templates als Binärdaten in PostgreSQL. Bei Überschreitung muss Migration zu S3 über C-7 (Storage Abstraction Layer) erfolgen.

## C-9: Docker-Container (SCS-Module)

**Description:** Jedes SCS-Modul (blocpress-studio, blocpress-workbench, blocpress-proof, blocpress-render, blocpress-admin) wird als eigener Docker-Container betrieben. Eine gemeinsame `docker-compose.yml` orchestriert alle Container inkl. PostgreSQL, RabbitMQ und Elasticsearch. blocpress-render kann unabhängig skaliert werden.

*Referenz: Docker Best Practices*

## C-10: Elasticsearch Version

**Description:** TI-7 (Elasticsearch API) benötigt Elasticsearch Version ≥ 7.x. Kompatibilität mit Version 8.x muss gewährleistet sein.

*Referenz: Elasticsearch Compatibility Matrix*