 Template

Januar 2023

**Über arc42**

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

Template Version DE. (basiert auf AsciiDoc Version), Januar 2023

Created, maintained and © by Dr. Peter Hruschka, Dr. Gernot Starke and contributors. Siehe <https://arc42.org>.

# Notizen

Anforderungen an die Architektur des virtuellen Agenten:

* Aktuell nur textuelle Infos, später Multimodal:
* Erweiterbarkeit nötig

# Einführung und Ziele

* 1. **Aufgabenstellung**

Das Projekt "Great Individual Tutor Embodiment" (GITTE) zielt darauf ab, ein personalisiertes und adaptives Lernerlebnis für Studierende der FernUniversität in Hagen zu unterstützen. Das System wird eine personalisierte, visuelle Repräsentation eines Intelligenten Lernassistenten (ILA) bieten, die auf den individuellen Studierendenmerkmalen basiert. Diese Merkmale können demografische Daten, Lernverhalten und weitere relevante Eigenschaften umfassen. Weitere Merkmale, die Einfluss auf die visuelle Repräsentation haben können, sollen später ergänzt werden können. Das System soll eine flexible und skalierbare Plattform bereitstellen, die dynamisch auf neue Erkenntnisse und Änderungen in den Studierendenmerkmalen reagieren kann. Die personalisierte Repräsentation soll als ein Aspekt eines personalisierten und adaptiven Lernerlebnisses integrierbar in Lernsysteme sein, die weitere Funktionalitäten bereitstellen. GITTE wird fortschrittliche Technologien der Künstlichen Intelligenz und des maschinellen Lernens nutzen, um die Personalisierung zu optimieren und das Lernen durch eine ansprechende Unterstützung durch einen ILA bieten.

**Verweis auf Anforderungsdokumente:**

* ArchitekturVision.docx (enthält die Vision und die architektonischen Überlegungen)
* Anforderungsdokumente der FernUniversität in Hagen (Versionsbezeichnung und Ablageort werden noch definiert)
* Tbd.

**Motivation:**

Die Einführung von GITTE soll das Lernerlebnis der Studierenden durch die Bereitstellung einer personalisierten und visuell ansprechenden Unterstützung verbessern. Durch die Anpassung der ILA-Repräsentation an die individuellen Merkmale der Studierenden wird erwartet, dass die Lernmotivation und weitere lernförderliche Faktoren gesteigert werden. Die Einbindung von Feedbackmechanismen soll eine kontinuierliche Verbesserung und Anpassung der Personalisierung ermöglichen.

**Form:**

Die Aufgabenstellung wird in einer textuellen Beschreibung zusammengefasst und durch tabellarische Use-Cases ergänzt. Diese enthalten spezifische Anwendungsfälle und Szenarien, die die verschiedenen Interaktionen und Funktionen des Systems detailliert darstellen. Verweise auf die entsprechenden Anforderungsdokumente werden in den entsprechenden Abschnitten angegeben.

## Ziele

Priorisierung: A = hoch, B = mittel.

Abnahme: Je Ziel sind konkrete Erfolgskriterien angegeben.

### Z-001 (A) Interaktive Lern-/Studienplattform

Beschreibung: End-to-end-Flow mit Login → Consent → Survey → Gestaltungsmerkmale → Chat (LLM) → Bildgenerierung.

**Erfolgskriterien:**

Alle Schritte als eigene UI-Module vorhanden und erreichbar.

Navigationsfluss ohne „tote Enden“ (Happy Path & Abbruchpfade getestet).

### Z-002 (A) Datenschutz & Consent-Gate

Beschreibung: Nutzung erst nach wirksamer Einwilligung; Widerruf jederzeit möglich.

**Erfolgskriterien:**

Ohne Consent kein Zugriff auf Chat/Bild.

Consent samt Timestamp/Version gespeichert; Widerruf sperrt Funktionen sofort.

### Z-003 (A) Nachvollziehbarkeit (Prompt-Audit)

Beschreibung: Lückenlose Speicherung von Input/Output, Modell, Parametern, Token-Usage, Latenz, Timestamps, parent-child-Bezug.

**Erfolgskriterien:**

Vollständigkeit ≥ 99 %/Monat; Export als CSV/JSON.

Jede Antwort eindeutig einer Sitzung/Nutzer-Pseudonym zuordenbar.

### Z-004 (A) PALD-Analyse & dynamische PALD-Struktur

Beschreibung: Extraktion strukturierter Merkmale (PALD), dynamisch erweiterbar (z. B. neues Feld „hair“).

**Erfolgskriterien:**

Eigenes PALD für die erste inhaltliche Nutzereingabe (nicht „greeting“).

Schema-Version wird mitgespeichert; neue Felder erscheinen in Speicher & Export.

### Z-005 (A) PALD-Vergleich & Vollständigkeitsmetriken

Beschreibung: Funktionen zum Diff zwischen zwei PALDs und zum Zählen (Gesamtfelder vs. gefüllte Felder).

**Erfolgskriterien:**

Service-API liefert (a) Feld-Diff, (b) Coverage-Prozentsatz.

Ergebnisse im Admin-Export verfügbar.

### Z-006 (A) Lokale Bildgenerierung

Beschreibung: Bildpipeline (z. B. Diffusers/SD) on-prem/GPU ohne Fremd-APIs.

**Erfolgskriterien:**

512×512 Bild ≤ 30 s (95-Perzentil).

Bildpfad/URI in Datenspeicher referenziert (lokal oder MinIO).

### Z-007 (A) Zentrale Konfiguration

Beschreibung: Alle Flags/Modelle/Prompts in config.py (Single Source of Truth).

**Erfolgskriterien:**

Umschalten per Flag ohne Codeänderungen: SAVE\_LLM\_LOGS, PALD\_ANALYSIS\_DEFERRED, Modellwahl u. a.

Änderung greift zur Laufzeit (nach Neustart) systemweit.

### Z-008 (A) Federated-Learning (FL) als schaltbare Option

Beschreibung: Basisbetrieb „zentral“; FL-Variante als alternative Pipeline, per Flag aktivierbar.

**Erfolgskriterien:**

UI/Use-Cases identisch; nur Trainings-/Aggregationspfad wechselt.

Datenflüsse dokumentiert; FL lässt sich ein/aus schalten ohne Codeänderung.

### Z-009 (A) Sauberer UI-Fluss im Chat

Beschreibung: Nach Name & Begrüßung wird der Bildschirm geleert; nur Chat sichtbar, Eingabefeld erscheint erst dann.

**Erfolgskriterien:**

UX-Test: kein Legacy-UI sichtbar, keine Doppel-Eingaben.

### Z-010 (A) Rollen & Administration

Beschreibung: Rollen „Admin“/„Teilnehmende“, Admin-Exports & Monitoring.

**Erfolgskriterien:**

Admin-Export filterbar (Zeitraum, user\_id, Modell, Kanal).

Basis-Statistiken: Antwortzeit, Fehlerquote, Token-Usage.

### Z-011 (B) Performance & Antwortqualität

Beschreibung: Chat-Reaktionsfähigkeit und Stabilität.

**Erfolgskriterien:**

Time-to-First-Token (Median) ≤ 2 s, 95-Perzentil ≤ 5 s.

Fehlerrate API < 1 % pro Tag; automatische Retries mit Backoff.

### Z-012 (A) Architekturkonformität (4-Layer)

Beschreibung: Strikte Trennung UI → Logic → Service → Data; Streamlit nur im UI-Layer.

**Erfolgskriterien:**

Code-Review/Static-Check: keine UI-Artefakte außerhalb UI-Layer, keine DB-Zugriffe außerhalb Data-Layer.

### Z-013 (B) Reproduzierbarkeit & Forschungstauglichkeit

Beschreibung: Seeds, Modell-/Config-Versionen, Umgebungs-Metadaten.

**Erfolgskriterien:**

Jeder Run protokolliert: Modell-Tag, Seed, Config-Hash; Ergebnis reproduzierbar.

### Z-014 (B) Accessibility & Usability

Beschreibung: Tastaturnavigation, Labels/ARIA, ausreichende Kontraste.

**Erfolgskriterien:**

Quick-Check gegen WCAG-Kriterien; Nutzertest mit 5 Teilnehmenden.

### Z-015 (A) Betroffenenrechte (DSGVO)

Beschreibung: Pseudonymisierung, Auskunft/Export/Löschung.

**Erfolgskriterien:**

Löschanforderung vollständig umgesetzt ≤ 72 h; Audit-Trail dokumentiert.

## Qualitätsziele

Die im Folgenden aufgeführten Qualitätsziele beschreiben das System in seinem wünschenswerten finalen Zustand. Dies bedeutet nicht, dass alle Faktoren umgesetzt werden. Sie sollen aber in jedem Schritt berücksichtigt werden.

### Benutzerfreundlichkeit & Barrierefreiheit

* **NFR-US-02 (A) Navigation & Konsistenz**  
  *Forderung:* Klare, konsistente Navigation & Terminologie.  
  *Kriterium:* 95 % Task-Success in 5-Personen-Usability-Test; <2 Klicks bis Chat/Feedback.
* **NFR-RSP-01 (A) Responsives UI**  
  *Forderung:* Desktop/Tablet/Smartphone unterstützt.  
  *Kriterium:* Kernflows (Login, Consent, Chat, Bild) bestehen auf ≥3 Breakpoints (≥320 px, ≥768 px, ≥1280 px) ohne horizontales Scrollen.
* **NFR-A11Y-01 (A) WCAG 2.1 AA**  
  *Forderung:* Erfüllung zentraler WCAG-Erfolgskriterien (Kontrast, Tastaturbedienung, Fokus).  
  *Kriterium:* WCAG-Checkliste „AA“ ohne kritische Mängel; Axe-Scan ohne Blocker.

### Personalisierung & Federated Learning

* **NFR-PERS-01 (A) Personalisierungs-Wirksamkeit**  
  *Forderung:* Visuelle Repräsentationen passen zu erfassten Merkmalen.  
  *Kriterium:* ≥85 % Übereinstimmung in Stichproben-Review (N≥50) zwischen PALD-Merkmalen und erzeugter Darstellung.
* **NFR-PERS-02 (A) Minimale Interaktion**  
  *Forderung:* Erste GITTE mit max. 1 Nutzereingabe.  
  *Kriterium:* 90 % der Sessions generieren die erste Darstellung nach ≤1 Eingabe + Greeting.
* **NFR-PERS-03 (B) Dynamische Aktualisierung**  
  *Forderung:* Neu erkannte Merkmale wirken ohne Neustart.  
  *Kriterium:* Anpassung in der **nächsten** Antwortrunde sichtbar (eine Interaktion Verzögerung).
* **NFR-FL-01 (B) Federated-Learning-Prinzip**  
  *Forderung:* *Modelle wandern, Daten bleiben* (bei aktiviertem FL).  
  *Kriterium:* Keine Rohdaten-Übertragung; nur aggregierte Updates; Opt-in/Opt-out dokumentiert in Legal-UI.

### Adaptivität & Erweiterbarkeit

* **NFR-ADPT-01 (B) Hot-Swap von Modulen**  
  *Forderung:* PALD-Extraktion/Prompt-Kompression/Konsistenzprüfung per config.py ohne Redeploy austauschbar.  
  *Kriterium:* Wechsel in <10 Min inkl. Smoke-Test.
* **NFR-ADPT-02 (B) Plugin-Schnittstellen**  
  *Forderung:* Dokumentierte Schnittstelle für neue Merkmals-Extractor.  
  *Kriterium:* Beispiel-Plugin integrierbar ohne Codeänderungen außerhalb Service-Layer.

### Skalierbarkeit

* **NFR-SCL-01 (B) Concurrent Users**  
  *Forderung:* System skaliert auf **N** gleichzeitige Nutzer.  
  *Kriterium:* Bei **N=100**: **NFR-PERF-01/02** werden weiterhin eingehalten.
* **NFR-SCL-02 (B) Caching/Pooling**  
  *Forderung:* DB-Pooling und Response-Caching für identische Prompts.  
  *Kriterium:* ≥20 % Kosteneinsparung bei Lasttest ggü. Baseline ohne Caching.

### Sicherheit/Datenschutz (Ergänzungen)

* **NFR-ENC-01 (A) Transportverschlüsselung**  
  *Forderung:* TLS 1.2+ für UI/API.  
  *Kriterium:* Alle externen Endpunkte liefern gültiges TLS-Zertifikat.
* **NFR-ENC-02 (A) Verschlüsselung ruhender Daten**  
  *Forderung:* Backups & Exportdateien verschlüsselt (AES-256).  
  *Kriterium:* Keys nicht im Code; Zugriff per Rollen.
* **NFR-SEC-AUD-01 (B) Sicherheitsüberprüfungen**  
  *Forderung:* Quartalsweise Security-Review (Dependencies, Secrets, OWASP).  
  *Kriterium:* Ticket + Nachweise pro Quartal.
* **NFR-AI-TRN-01 (A) KI-Transparenz**  
  *Forderung:* Informationsmechanismen zur KI-Nutzung in der UI (Legal-UI + Hilfe).  
  *Kriterium:* Nutzer sieht Zweck, Datenarten, Modelle, Opt-in/Opt-out.

### Zuverlässigkeit

* **NFR-AVAIL-01 (B) Verfügbarkeit**  
  *Forderung:* 99.5 % Monats-Uptime (geplante Wartung ausgenommen).  
  *Kriterium:* Monitoring-Bericht belegt Zielerreichung.
* **NFR-RCV-01 (B) RTO/RPO**  
  *Forderung:* RTO ≤4 h, RPO ≤1 h.  
  *Kriterium:* Wiederherstellungstest/Backup-Restore-Protokoll.
* **NFR-HLTH-01 (B) Health-Checks**  
  *Forderung:* Liveness/Readiness-Probes für UI, DB, LLM, Bild-Pipeline.  
  *Kriterium:* Automatisches Failover/Restart bei Fehler.

### Leistung (Erweiterungen)

* **NFR-DB-01 (B) DB-Performance**  
  *Forderung:* P95 Abfragezeit ≤50 ms für Kernqueries (Chat/PALD/Prompt-Audit).  
  *Kriterium:* Indizes + Pooling aktiviert; Messreport monatlich.
* **NFR-THR-01 (B) Durchsatz**  
  *Forderung:* ≥X Chat-Antworten/min und ≥Y Bilder/min bei N Nutzern (Werte projektieren).  
  *Kriterium:* Lasttest-Report mit Zielerreichung.

### Interoperabilität

* **NFR-INT-01 (B) API-Standards**  
  *Forderung:* Interne/externe APIs als **OpenAPI 3.1**, versionierte Endpunkte.  
  *Kriterium:* Maschine-lesbare Spezifikation veröffentlicht.
* **NFR-INT-02 (B) LMS/VU-Integration**  
  *Forderung:* **LTI 1.3** (oder Uni-Standard) für Single-Sign-On/Notenrückgabe (falls relevant).  
  *Kriterium:* End-to-End-Test gegen Test-LMS.

### Modularität

* **NFR-MOD-01 (B) Module/Microservices**  
  *Forderung:* Fachlich getrennte Services (Chat, PALD, Bild, Prompt-Audit) können unabhängig deployt werden.  
  *Kriterium:* Ein Service-Update verursacht keinen Deploy anderer Services.
* **NFR-CNT-01 (B) Containerisierung**  
  *Forderung:* Docker-Images pro Service; Compose/Orchestrierung.  
  *Kriterium:* docker compose up bringt Testsystem lauffähig.

### Datenqualität

* **NFR-DQ-01 (A) Schema-Validierung**  
  *Forderung:* PALD-JSON gegen pald\_schema.json validieren.  
  *Kriterium:* 100 % valide in Produktivdaten; Invaliden-Quote <1 % in Dev/QA.
* **NFR-DQ-02 (A) Konsistenz/Bias**  
  *Forderung:* Checks auf fehlende/konfligierende Merkmale + Bias-Indikatoren.  
  *Kriterium:* Monatlicher DQ-Report; Maßnahmenplan bei Abweichungen.
* **NFR-DQ-03 (B) Datenbereinigung**  
  *Forderung:* Quartalsweise Bereinigung/Archivierung.  
  *Kriterium:* Protokoll der gelöschten/archivierten Sätze.
* **NFR-ETH-01 (B) Ethische Leitlinien**  
  *Forderung:* Nutzung gemäß festgelegten Forschungsleitlinien.  
  *Kriterium:* Dokumentierte Freigabe vor Go-Live; jährlicher Review.

Diese Qualitätsziele bilden die Grundlage für die Entwicklung und Implementierung von GITTE, um sicherzustellen, dass das System den Anforderungen der Stakeholder gerecht wird und eine hohe Benutzerzufriedenheit erreicht.

## Stakeholder

 **Teilnehmende**: einfache Bedienung, Transparenz zu Daten, Widerruf/Löschung.

 **Admin/Studienleitung**: Auswertungen, Logs, Export (CSV/JSON).

 **IT/Betrieb**: stabile Deployments, Monitoring, Backups.

| **Bezeichnung** | **Rolle** | **Kontakt** | **Erwartungshaltung** |
| --- | --- | --- | --- |
| Forschungsteam | Weiterentwicklung und Evaluierung | Forschungsgruppen und -leiter | - Zugriff auf anonymisierte Nutzungsdaten für Studienzwecke  - Unterstützung bei der Implementierung neuer Funktionen basierend auf Forschungsergebnissen  - Möglichkeiten zur Evaluierung der Wirksamkeit der Personalisierung |
| Studierende der FernUniversität | Primäre Nutzer | Studierendenvertretung, Einzelne Studierende | - Personalisierte und ansprechende visuelle Repräsentation des Lernassistenten  - Einfache Bedienung und Zugänglichkeit des Systems  - Verlässliche und sichere Handhabung persönlicher Daten |
| IT- und Support-Team | Technische Implementierung und Wartung | IT-Abteilung, Technischer Support | - Einfache Integration und Wartung des Systems  - Skalierbarkeit und Zuverlässigkeit der Architektur  - Ausführliche technische Dokumentation |
| Datenschutzbeauftragte | Einhaltung der Datenschutzvorschriften | Datenschutzbeauftragter der FernUniversität | - Konformität mit der DSGVO und anderen relevanten Datenschutzgesetzen  - Transparente Datenverarbeitungsprozesse  - Robuste Sicherheitsmaßnahmen zum Schutz der Studierendendaten |
| Bildungsadministratoren | Strategische Planung und Ressourcenallokation | Verwaltungsleitung, Studiengangskoordinatoren | - Kosteneffiziente Implementierung und Betrieb des Systems  - Unterstützung der Lehr- und Lernziele der Universität  - Langfristige Integration in das universitäre Angebot |
| Externe Technologiepartner | Bereitstellung spezieller Technologien und Unterstützung | Technologieanbieter, Dienstleister | - Reibungslose Zusammenarbeit und Integration externer Technologien  - Einhaltung von Service-Level-Agreements (SLA)  - Unterstützung bei technischen Herausforderungen |
| Ethikkommission | Überprüfung der ethischen Vertretbarkeit | Ethikrat der Universität | - Sicherstellung der ethischen Vertretbarkeit des Systems  - Berücksichtigung der Auswirkungen auf die Studierenden  - Einhaltung ethischer Standards in Bildung und Technologie |
| Lehrende und Dozenten | Indirekte Nutzer und Feedbackgeber | Fakultäten, Dozenten | - Verbesserung der Lernergebnisse durch die personalisierte Gestaltung  - Einblicke in die Lerngewohnheiten der Studierenden  - Unterstützung durch den Lernassistenten bei der Lehre |
| Gesellschaft und Bildungssektor | Interesse an den Entwicklungen und Ergebnissen | Bildungseinrichtungen, Öffentlichkeit | - Innovative Ansätze zur Verbesserung des digitalen Lernens  - Vorbildfunktion für andere Bildungseinrichtungen  - Beitrag zur digitalen Bildungslandschaft |

Diese detaillierte Liste der Stakeholder hilft sicherzustellen, dass die Anforderungen und Erwartungen aller relevanten Gruppen berücksichtigt werden, um eine erfolgreiche Implementierung und Nutzung von GITTE zu gewährleisten.

# Randbedingungen

## Technologische Randbedingungen:

### Verwendung etablierter Technologien:

Beschreibung: Es sollen nur Technologien eingesetzt werden, die sich bereits bewährt haben und eine breite Unterstützung sowie gute Dokumentation bieten.

Begründung: Dies erleichtert die Wartung und Weiterentwicklung des Systems und erhöht die Wahrscheinlichkeit

### Open Source Software:

Beschreibung: Vorzugsweise sollen Open Source Technologien verwendet werden.

Begründung: Reduzierung von Lizenzkosten und Abhängigkeiten von spezifischen Anbietern.

## Organisatorische Randbedingungen:

### Budgetbeschränkungen:

Beschreibung: Das Projektbudget ist begrenzt und erfordert eine kosteneffiziente Lösung.

Begründung: Effiziente Mittelverwendung und Einhaltung des vorgegebenen Budgets sind notwendig.

### Zeitliche Beschränkungen:

Beschreibung: Das Projekt muss innerhalb von drei Jahren abgeschlossen sein.

Begründung: Ein strikter Zeitrahmen zur Erfüllung der Projektziele ist aus Finanzierungssicht notwendig.

### Organisatorische/Regulatorische Randbedingungen

* REQ-ORG-001: **DSGVO**: Datenminimierung, Zweckbindung, Löschkonzept.
* REQ-ORG-002: Einwilligung (Consent) vor Start der Datenerhebung verpflichtend.
* REQ-ORG-003: Rollenmodell: **Admin** vs. **Teilnehmende**.

## Rechtliche und regulatorische Randbedingungen:

### Einhaltung der DSGVO:

Beschreibung: Das System muss vollständig mit der Datenschutz-Grundverordnung (DSGVO) konform sein.

Begründung: Sicherstellung des Schutzes personenbezogener Daten der Studierenden.

### Sicherheitsstandards:

Beschreibung: Einhaltung von IT-Sicherheitsstandards wie ISO/IEC 27001.

Begründung: Schutz der Daten und Systeme vor unbefugtem Zugriff und anderen Bedrohungen.

## Technische Randbedingungen:

### Integration bestehender Systeme:

Beschreibung: GITTE muss in die vorhandene IT-Infrastruktur der FernUniversität integriert werden.

Begründung: Nutzung bestehender Systeme und Daten zur Effizienzsteigerung.

### Multiplattformunterstützung:

Beschreibung: Das System muss auf verschiedenen Endgeräten (Desktop, Tablet, Smartphone) zugänglich sein.

Begründung: Gewährleistung einer breiten Zugänglichkeit für die Studierenden.

### Technische Randbedingungen

* REQ-SYS-001: Datenbank **PostgreSQL ≥13**, DB-Name <data\_collector>, UTF-8.
* REQ-SYS-002: GPU-Runtime für Bildgenerierung (**CUDA**, NVIDIA RTX 30xx empfohlen).
* REQ-SYS-003: Optionales Objekt-Storage **MinIO** (S3-kompatibel) für Bilder/Artefakte.
* REQ-SYS-004: **Python ≥3.10** (venv), Paketverwaltung via pip.
* REQ-SYS-005: Betrieb **on-prem** oder in **Private Cloud**; keine Drittanbieter-APIs für personenbezogene Daten.

## Politische und gesellschaftliche Randbedingungen:

### Ethische Standards:

Beschreibung: Das System muss ethischen Standards in der Bildung und Technologie entsprechen.

Begründung: Sicherstellung der ethischen Vertretbarkeit und Akzeptanz des Systems.

### Inklusion und Barrierefreiheit:

Beschreibung: Das System muss barrierefrei gestaltet sein und den Web Content Accessibility Guidelines (WCAG) entsprechen.

Begründung: Zugänglichkeit für alle Studierenden, einschließlich Menschen mit Behinderungen.

## Entwicklungsprozess:

### Agile Entwicklungsmethoden:

Beschreibung: Verwendung agiler Methoden wie Scrum zur Entwicklung des Systems.

Begründung: Flexibilität und schnelle Anpassungsfähigkeit an sich ändernde Anforderungen.

## Interne Standards und Vorgaben:

### Dokumentationsanforderungen:

Beschreibung: Ausführliche technische und nutzungsorientierte Dokumentation muss erstellt werden.

Begründung: Sicherstellung der Nachvollziehbarkeit und Unterstützung der Nutzer sowie des IT-Teams.

Diese Randbedingungen definieren die Rahmenbedingungen für die Entwicklung und Implementierung von GITTE und helfen sicherzustellen, dass das System den Erwartungen und Anforderungen der Stakeholder gerecht wird und innerhalb der festgelegten Einschränkungen bleibt.

# Kontextabgrenzung

## Fachlicher Kontext

Im Folgenden sind die Systeme aufgeführt, die vermutlich mit GITTE interagieren müssen, da

1. Die darin enthaltenen Daten für GITTE relevant sein können oder
2. umgekehrt, eine Plattform darstellt, die GITTE für eine reibungslose Funktionalität benötigt.

| **Bezeichnung** | **Beschreibung** | **Schnittstellenfunktion** | **Interaktionen** |
| --- | --- | --- | --- |
| Lernmanagementsystem (LMS) | Das LMS der FernUniversität in Hagen ist die zentrale Plattform für Kursmaterialien, Diskussionsforen, Aufgaben und Bewertungen. | GITTE muss Daten über Module/Kurse, abrufen können, wenn dies für die Personalisierung notwendig ist. | Synchronisation Kursinformationen; |
| Die Virtuelle Universität (VU) | Die Virtuelle Universität (VU) enthält wesentliche Daten über die Studierenden, einschließlich ihrer Registrierungsdaten, Studienfortschritte und akademischen Leistungen. | GITTE muss mit der VU interagieren, um aktuelle Informationen über die Studierenden zu erhalten und bei der Erstellung personalisierter Repräsentationen berücksichtigen zu können, wenn dies für die Personalisierung notwendig ist. | Abruf von z.B. demografischen Daten und akademischen Informationen; Ggf. Anpassung der ILA-Repräsentation basierend auf Studienfortschritten und Änderungen im akademischen Status. |
| Lernplattformen | Systeme für die Bearbeitung der Lernmaterialien, z.B. Moodle, Foren, Chat-Plattformen. | Die Systeme müssen in der Lage sein, GITTE darzustellen und zu steuern. GITTE muss in der Lage sein, Kommunikationsdaten zu analysieren, um das Lernverhalten und die Interaktionsmuster der Studierenden zu verstehen und darauf zu reagieren. | Darstellung und Analyse von Kommunikationsdaten zur Anpassung der ILA-Repräsentation; Bereitstellung von personalisierten Kommunikationsvorschlägen und -unterstützung. |
| Externe Datenquellen und APIs | Externe Datenquellen und APIs, die zusätzliche Informationen oder Funktionalitäten bereitstellen. | GITTE kann externe Datenquellen nutzen, um zusätzliche Kontextinformationen oder fortschrittliche Funktionen (z.B. KI-gestützte Analyse) zu integrieren. | Abruf und Integration externer Daten zur Verbesserung der GITTE-Personalisierung; Nutzung externer APIs für erweiterte Funktionen und Analysen. |
| ILA-Komponenten  (Mimik, Gestik, Didaktik, etc.) | Systeme, die zum Gesamtkonzept ILA gehören und weitere Komponenten des ILA. | GITTE muss mit den übrigen Komponenten des ILA interagieren, um eine kohärente Darstellung des ILA zu gewährleisten. | Synchronisation der visuellen Darstellung mit Mimik, Gestik, Didaktik, etc. Analyse der Informationen aus den Komponenten zur Anpassung der ILA-Repräsentationen.  Bereitstellung von Informationen über die visuelle Repräsentation, zur Nutzung durch andere Komponenten des ILA. |

## Technischer Kontext

Der technische Kontext von GITTE beschreibt die verschiedenen technischen Schnittstellen, über die das System mit seiner Umgebung kommuniziert. Dies umfasst Kanäle, Übertragungsmedien sowie die Zuordnung fachlicher Ein- und Ausgaben zu den entsprechenden technischen Kanälen. Ziel ist es, die externen Schnittstellen des Systems klar zu definieren und sicherzustellen, dass alle notwendigen Interaktionen reibungslos und effizient ablaufen.

### Übersicht der technischen Schnittstellen und Mapping

1. **Lernmanagementsystem (LMS)**
   * **Kanäle**: tbd. (REST-API, …)
   * **Übertragungsmedien**: tbd. (HTTPS, …)
   * **Funktion**: Synchronisation von Kursinformationen, Abruf von Kursmaterialien und Diskussionsdaten zur Personalisierung.
   * **Fachliche Ein- und Ausgaben**: Kursinformationen, Diskussionsbeiträge, etc.
   * **Mapping**: Kursdaten (fachliche Eingabe) werden über die technischen Kanäle synchronisiert.
2. **Die Virtuelle Universität (VU)**
   * **Kanäle**: tbd. (REST-API, …)
   * **Übertragungsmedien**: tbd. (HTTPS, …)
   * **Funktion**: Abruf von demografischen Daten, Studienfortschritten und akademischen Leistungen.
   * **Fachliche Ein- und Ausgaben**: Studierendendaten, akademische Leistungsdaten.
   * **Mapping**: Demografische Daten und Studienfortschritte (fachliche Eingabe) werden über die technischen Kanäle abgerufen.
3. **Lernplattformen (z.B. Moodle, Foren, Chat-Plattformen)**
   * **Kanäle**: tbd. (REST-API, …)
   * **Übertragungsmedien**: tbd. (HTTPS, …)
   * **Funktion**: Darstellung und Steuerung von GITTE, Analyse von Kommunikationsdaten zur Anpassung der ILA-Repräsentation.
   * **Fachliche Ein- und Ausgaben**: Kommunikationsdaten, Interaktionsmuster.
   * **Mapping**: Kommunikationsdaten (fachliche Eingabe) werden über die technischen Kanäle analysiert.
4. **Externe Datenquellen und APIs**
   * **Kanäle**: tbd. (REST-API)
   * **Übertragungsmedien**: tbd. (HTTPS, …)
   * **Funktion**: Integration zusätzlicher Kontextinformationen oder fortschrittlicher Funktionen (z.B. KI-gestützte Analyse).
   * **Fachliche Ein- und Ausgaben**: Kontextinformationen, Analyseresultate.
   * **Mapping**: Externe Kontextinformationen (fachliche Eingabe) werden über die technischen Kanäle integriert.
5. **ILA-Komponenten (Mimik, Gestik, Didaktik etc.)**
   * **Kanäle**: tbd. (REST-API)
   * **Übertragungsmedien**: tbd. (HTTPS, …)
   * **Funktion**: Interaktion mit weiteren ILA-Komponenten zur kohärenten Darstellung.
   * **Fachliche Ein- und Ausgaben**: Visuelle Darstellungen, Mimik, Gestikdaten.
   * **Mapping**: Visuelle Darstellungen und Mimikdaten (fachliche Ein- und Ausgaben) werden über die technischen Kanäle synchronisiert.

### 1.3 Technischer Kontext & externe Schnittstellen

* LLM-Endpoint (lokal, z. B. Ollama/HTTP).
* Optionales MinIO (S3 API).
* E-Mail/SMTP (optional) für Admin-Benachrichtigungen.
* OS-Dateisystem für temporäre Artefakte.

### Diagramm zur Veranschaulichung (optional)

Später könnte ein Deployment Diagramm ergänzt werden.

# Lösungsstrategie

In diesem Kapitel wird die technologische Basis der Implementierung des GITTE-Systems mit Fokus auf die Verwendung von **Streamlit** als Web-Framework und **Ollama** als Plattform zur Nutzung moderner KI-Modelle wie **Llama 3** und **Llava** vorgestellt. Diese Tools bieten eine robuste und flexible Umgebung zur Umsetzung der personalisierten Lernassistenten, ermöglichen eine einfache Interaktion und bieten Skalierbarkeit sowie die Einbindung neuester KI-Technologien.

## Streamlit Technologische Implementierung mit Streamlit und Ollama

**Streamlit** wurde als Hauptframework für die Benutzeroberfläche des GITTE-Systems gewählt. Dieses Python-basierte Open-Source-Tool bietet eine einfache Möglichkeit, Webanwendungen zu erstellen, die auf maschinellem Lernen und Datenverarbeitung beruhen. Die Entscheidung für Streamlit wurde aufgrund folgender Eigenschaften getroffen:

* **Schnelle Entwicklung**: Streamlit ermöglicht die schnelle Erstellung von Prototypen und Anwendungen, die auf maschinellem Lernen basieren, ohne dass umfangreiche Web-Entwicklungskenntnisse erforderlich sind. Die interaktive Benutzeroberfläche kann innerhalb weniger Zeilen Code erstellt und angepasst werden.
* **Echtzeit-Interaktionen**: Streamlit unterstützt Echtzeit-Updates, was es den Studierenden ermöglicht, sofortige Rückmeldungen über ihre Eingaben und Anpassungen an den GITTEs zu erhalten. Diese Dynamik ist entscheidend für die kontinuierliche Personalisierung der Lernassistenten.
* **Skalierbarkeit und einfache Bereitstellung**: Durch die Integration in Cloud-basierte Systeme kann das GITTE-System über Streamlit leicht skaliert werden, um eine große Anzahl von Studierenden gleichzeitig zu unterstützen. Zudem ist es einfach, die Anwendung für externe Nutzer bereitzustellen, was es möglich macht, die Funktionalitäten über die FernUniversität Hagen hinaus zu erweitern.

Streamlit ist optimal für die visuelle Darstellung und Verwaltung der GITTEs geeignet, da es sowohl die Eingabe von Benutzerpräferenzen als auch die Anpassung der Lernassistenten über eine interaktive, leicht zugängliche Web-Oberfläche ermöglicht. In zukünftigen Implementierungen könnte die Funktionalität von Streamlit weiter ausgebaut werden, um sprachbasierte Interaktionen oder emotionale Reaktionen in Echtzeit zu erfassen.

## Federated Learning (FL) – einfache Nutzenden‑Variante

**Ziel:** Personalisierung ohne Rohdatenabfluss. Studierende interagieren wie gewohnt; lokal werden aus strukturierten Zwischenständen (z. B. PALD, Klick‑Signale) kleine Modell‑Updates („Deltas“) berechnet. Der Server aggregiert nur diese Deltas (**FedAvg**), optional mit **Differential Privacy**.

**Nicht gesendet:** Rohtexte, Bilder, komplette Chatverläufe.

### **Gelearnte Module (MVP):**

1. **PALD‑Extractor** (Text→PALD‑Slots, robust gegen Eigenjargon).
2. **Prompt‑Kompressor** (PALD→kompakter Bild‑Prompt, Stilpräferenzen).
3. **Konsistenz‑Checker** (Text↔Bild „passt/nicht“).
4. **Antwort‑Stiladapter** (lange vs. kurze Antworten, Tonfall).
   1. **Signals (lokal → Delta):** strukturierte PALD‑Felder, implizites Feedback (Re‑Generate‑Klick, Speichern, Daumen), Konsistenz‑Label; **keine** Rohtexte/Bilder.
   2. **Konfiguration (zentral):** FL\_ENABLED, Client‑Sampling, Clip‑Norm, DP‑Sigma.
   3. **Rechtlich/Transparenz:** explizites Opt‑in in der Legal‑UI; Hinweis auf lokale Verarbeitung und rein aggregierte Serverupdates.
   4. **Warum hier sinnvoll:** Das System arbeitet bereits mit strukturierten PALDs und Feedbackschleifen; FL ergänzt die bestehende Personalisierung ohne Änderung des Nutzungsablaufs.

## Einsatz und Integration von Ollama

Ollama ist die primäre Plattform für die Nutzung und Verwaltung generativer KI-Modelle wie llama3.2, llama-llava3, mistral und CodeLlama, wie in den Modulen LLM\_API.py und GITTE\_logic.py umgesetzt.

Die aktuelle Implementierung umfasst:

* **Chat-Generierung**: Das Modell llama3.2 liefert generative Textantworten für die Chatinteraktion und gewährleistet die Verarbeitung natürlicher Sprache.
* **Bildbeschreibung**: Das multimodale Modell llama-llava3 erstellt Beschreibungen von generierten Bildern, um die visuelle Validität zu überprüfen.
* **Prompt-Komprimierung**: Das Modell mistral wird verwendet, um aus PALD-Strukturen textuelle Prompts für Stable Diffusion zu generieren.
* **PALD-Vergleich**: Das Modell CodeLlama dient der logischen Überprüfung und dem Vergleich von PALD-Strukturen zur Sicherstellung der Konsistenz.

Ollama erlaubt die lokale Nutzung der Modelle und reduziert die Abhängigkeit von externen Anbietern, wodurch die Sicherheit und der Datenschutz verbessert werden.

## Integration und Nutzung von Stable Diffusion

Stable Diffusion ist als Bildgenerierungswerkzeug in das System integriert und wird über das Modul Pic\_API.py verwendet. Die Umsetzung erfolgt über die StableDiffusionPipeline, um auf Grundlage präziser Prompts visuelle Darstellungen der Intelligent Learning Assistants (ILAs) zu erzeugen.

Die konkreten Schritte umfassen:

* **Prompt-Erzeugung**: Mittels PALD-Strukturen generierte Prompts werden direkt an Stable Diffusion übergeben.
* **Bildgenerierung**: Bilder werden lokal generiert, automatisch auf Korrektheit überprüft und anschließend gespeichert.
* **Bildspeicherung**: Unterstützung sowohl für lokale Speicherung als auch für Cloud-basierte Speicherung über MinIO (siehe GITTE\_storage.py).
* **Fallback-Verhalten**: Implementierung eines Fehlermanagements mit standardisierten Fallback-Bildern, um Ausfälle zu minimieren.

Diese Integration unterstützt die Personalisierung der visuellen Merkmale der GITTEs und trägt erheblich zur Benutzerfreundlichkeit bei.

## Potenzielle Nutzung von LangChain

LangChain bietet in diesem Projekt optional erweiterte Funktionalitäten zur Integration und Verkettung komplexer NLP-Komponenten. Der Einsatz von LangChain ist über die Konfigurationsvariable USE\_LANGCHAIN steuerbar und derzeit noch optional, bietet jedoch folgende zukünftige Potenziale:

* **Flexible Integration von Large Language Models (LLMs)**: Unterstützung verschiedener Modelle, einschließlich externer Dienste oder selbst gehosteter Modelle.
* **Modulare Komponentennutzung**: Möglichkeit zur schrittweisen Integration komplexer NLP-Komponenten wie Dokumentenverarbeitung, Codeanalyse und Inhaltszusammenfassung.
* **Robustes Debugging und Testen**: Nutzung von LangChain-Werkzeugen wie LangSmith für einfacheres Debugging und umfassende Tests der LLM-Anwendungen.

Aktuell ist LangChain als optionaler Wrapper um die direkten API-Calls zu Ollama konzipiert, um mittelfristig eine noch flexiblere und skalierbare Integration externer und interner NLP-Dienste zu ermöglichen. Die endgültige Aktivierung und Nutzung richtet sich nach den Performance-Tests und der Evaluation der zusätzlichen Komplexität.

## Technische Randbedingungen

* Ollama wird ausschließlich lokal gehostet, um Datenschutz und Performance sicherzustellen.
* Stable Diffusion ist in einer Version (runwayml/stable-diffusion-v1-5) verfügbar, die lokal oder in Cloud-Speichern betrieben wird.
* LangChain ist optional aktivierbar und soll bei steigendem Bedarf nach komplexeren NLP-Pipelines oder besseren Debugging- und Testverfahren aktiviert werden.

Diese Integration der KI-Technologien ermöglicht die Personalisierung des Embodiments der Intelligent Learning Assistants (ILAs) auf jeweils aktuellem technischen Niveau, wobei Erweiterbarkeit, Performanz und Datenschutz stets berücksichtigt werden.

# Bausteinsicht

## Whitebox Gesamtsystem

Die Architektur des Gesamtsystems „Great Individual TuTor Embodiment“ (GITTE) ist klar in vier definierte Schichten strukturiert, die jeweils spezifische Aufgaben übernehmen und nur mit der jeweils übergeordneten oder untergeordneten Schicht interagieren. Diese Schichten gewährleisten eine hohe Modularität, Wartbarkeit und einfache Erweiterbarkeit:

***<Übersichtsdiagramm>***

## Schichtenarchitektur

Die Architektur des Gesamtsystems sind wie folgt definiert.

### UI-Schicht (Benutzeroberfläche)

* Implementiert mit **Streamlit**
* Direkte Interaktion mit Nutzenden
* Verantwortlich für Eingabe, Ausgabe und Darstellung der Informationen
* Keine Geschäftslogik, nur Darstellungsfunktionalitäten

### Logic-Schicht (Geschäftslogik)

* Zentrale Verarbeitungsebene für Anwendungslogik
* Konsolidierung und Verarbeitung der Eingaben aus der UI-Schicht
* Kommunikation mit Service-Schicht
* PALD-Generierung, Feedback-Verarbeitung, Konsistenzprüfung und Entscheidung über nachfolgende Schritte

### Service-Schicht (Dienste und APIs)

* Abstraktionsebene zur Integration externer KI-Systeme (z. B. Ollama, Stable Diffusion, LangChain)
* Zugriffsschicht für Datenspeicherung und -abruf
* Bereitstellung von APIs zur Kommunikation zwischen Logic- und Persistenzschicht

### Persistence-Schicht (Datenhaltung)

* Realisiert mittels PostgreSQL und optional MinIO
* Speicherung von PALDs, Studierendeninformationen, Chatverläufen, Entscheidungen und Bilddaten
* Unterstützung lokaler und Cloud-basierter Speicherung

Jede dieser Schichten spielt eine wesentliche Rolle im Gesamtbild der Architektur und interagiert mit der jeweils nur mit der darunter liegenden Schicht, um die gewünschte Funktionalität bereitzustellen.

## Begründung:

Die Architektur ist modular aufgebaut, um zukünftige Erweiterungen wie neue Interaktionsmethoden, zusätzliche Geräteunterstützung oder neue Technologien integrieren zu können. Gleichzeitig soll diese Unterschiedlichkeit transparent bleiben.

Die Verwendung einer Schichtenarchitektur in der angehängten Architektur ist aus mehreren Gründen sinnvoll:

* Modularität und Trennung der Zuständigkeiten: Die Schichtenarchitektur ermöglicht eine klare Trennung der Funktionalitäten in separate Schichten, was die Wartbarkeit und Erweiterbarkeit des Systems und die Testbarkeit verbessert, da die klare Trennung der Funktionalitäten das Testen einzelner Komponenten erleichtert.
* Flexibilität und Anpassungsfähigkeit: Durch die modulare Struktur können einzelne Schichten einfacher ausgetauscht oder aktualisiert werden, ohne das gesamte System zu beeinträchtigen. Die rasante Entwicklung der generativen künstlichen Intelligenz macht es notwendig, diese Flexibilität und Austauschbarkeit sicher zu stellen. Da das System in unterschiedlichen Lernumgebungen eingesetzt werden können soll, sollten ebenfalls austauschbare Schnittstellenmodule zu diesen Systemen bereitgestellt werden.
* Skalierbarkeit: Die Schichtenarchitektur erleichtert die horizontale Skalierung des Systems, indem einzelne Schichten unabhängig voneinander skaliert werden können. Dies ist nicht nur für den breiten Einsatz zur Datenerhebung sinnvoll, sondern auch für die Bereitstellung über das Internet oder alternativ lokal.
* Wiederverwendbarkeit: Komponenten innerhalb einer Schicht können leichter in anderen Projekten wiederverwendet werden. GITTE stellt einen Teil eines gesamten ILA dar und muss entsprechend in ein ILA-Gesamtsystem integrierbar sein.

### Aktuell nicht berücksichtig

#### Sicherheitsmanagement

Umfasst Mechanismen zur Authentifizierung, Autorisierung, Datenverschlüsselung und weiteren Sicherheitsmaßnahmen, um die Einhaltung der Datenschutz-Grundverordnung (DSGVO) und weiterer Sicherheitsstandards zu gewährleisten.

#### Analyse und Reporting

Werkzeuge zur Überwachung der Systemleistung, zur Analyse der Nutzungsmuster und zur Erstellung von Berichten, die helfen, das System kontinuierlich zu verbessern und die Personalisierung weiter zu verbessern.

### Beschreibungsmuster für die Bausteine (<Name Blackbox 1>)

*<Zweck/Verantwortung>*

*<Schnittstelle(n)>*

*<(Optional) Qualitäts-/Leistungsmerkmale>*

*<(Optional) Ablageort/Datei(en)>*

*<(Optional) Erfüllte Anforderungen>*

*<(optional) Offene Punkte/Probleme/Risiken>*

## Beschreibung der Schichten und ihrer Komponenten

### Benutzeroberfläche (UI Layer)

Die Benutzeroberfläche (UI Layer) bildet den zentralen Zugangspunkt, über den die Studierenden mit dem GITTE-System interagieren. Sie ist dafür verantwortlich, eine benutzerfreundliche und intuitive Bedienung auf verschiedenen Endgeräten (Desktop, Tablet, Smartphone) zu gewährleisten. Zudem soll sie sicherstellen, dass alle Funktionen des Systems barrierefrei zugänglich sind und die Inhalte konsistent und ansprechend präsentiert werden. Die Benutzeroberfläche basiert auf Streamlit und enthält aktuell folgende Module:

#### Login UI (login\_ui.py)

* Verantwortlich für die Benutzeranmeldung und Authentifizierung
* Implementierung eines klaren Anmeldeformulars mit Benutzername und Passwort
* Nutzung des Moduls auth.py zur Prüfung der Zugangsdaten über Service-Schicht
* Erfolgreicher Login führt automatisch zur Auswahl weiterer Interaktionsschritte, bei Fehlschlag folgt klare Rückmeldung mit erneuter Eingabeaufforderung

#### Legal UI (legal\_ui.py)

* Explizite Zustimmungserhebung zu Datenschutz, Studienbeteiligung und Nutzung künstlicher Intelligenz
* Darstellung der Zustimmungstexte in klar strukturierter Form mit optionalen Erklärungen in ausklappbaren Bereichen
* Speicherung der Zustimmung durch Aufruf der Logic-Schicht (LegalLogic) und persistente Sicherung über die Service-Schicht
* Prozessverzweigung abhängig von vollständiger Zustimmung oder Abbruch bei unvollständiger Zustimmung

#### Study Participation UI (study\_participation\_ui.py)

* Initiale Erstellung eines individuellen Pseudonyms als Schlüssel für anonymisierte Studierendenidentifikation
* Automatische Initialisierung der notwendigen Datenbankstrukturen (init\_all\_db) bei erster Interaktion
* Sicherung der generierten Pseudonyme in der Datenbank, Vorbereitung der anschließenden Interaktionsschritte (Legal UI und Fragebogen)

#### GITTE Chat UI (GITTE\_chat\_ui.py)

* Realisierung des interaktiven Chats zur Gestaltung der Verkörperung (Embodiment) des Lernassistenten (GITTE)
* Dynamische Verwaltung des Chatverlaufs, Speicherung und Darstellung der Nachrichten
* Integrierter Ablauf mit expliziter Begrüßung, Feedback-Schleifen, kontinuierlicher PALD-Erstellung und Konsistenzprüfung

#### Kurs-Fake-UI

Diese Komponente simuliert den Aufruf eines Kurses, den die Studierenden bearbeiten möchten. Sie zeigt den gewählten Kurs an und ermöglicht optional die Bearbeitung mit einer **personalisierten GITTE**. Die Komponente dient als Testumgebung, um die Integration von GITTE in die Kursumgebung zu simulieren.

#### Survey-UI

Diese Komponente erfasst alle relevanten **Informationen über die Studierenden**, die für die Personalisierung der GITTE notwendig sind. Dazu gehören demografische Daten, Lernverhalten und individuelle Präferenzen. Die gesammelten Daten werden anschließend für die **Generierung und Anpassung** der GITTE verwendet.

#### Begründung

Der **modulare Aufbau** der Benutzeroberfläche gewährleistet eine hohe Flexibilität und Erweiterbarkeit. Dies ermöglicht es, **neue Interaktionsmethoden** oder die Unterstützung zusätzlicher Endgeräte ohne grundlegende Änderungen in der Struktur des Systems zu integrieren.

Die Gestaltung des UI-Layers fokussiert sich auf die Bedürfnisse der Studierenden. Durch die Bereitstellung einer **einfachen und einheitlichen Oberfläche** wird sichergestellt, dass die Lerninhalte und Interaktionen im Vordergrund stehen. Die klare Strukturierung und intuitive Bedienung minimieren Ablenkungen und erleichtern die Nutzung des Systems.

### Logikschicht (Logic Layer)

Die Logikschicht dient der **Verarbeitung der Benutzerinteraktionen** sowie der **Verwaltung** von Informationen, die Einfluss auf die Gestaltung der GITTE haben. Dazu gehört insbesondere die Koordination der Schritte, die zur Erfüllung der jeweiligen Aufgabe notwendig sind. Dies kann z.B. die **Verwaltung und Auswertung** von Rückmeldungen der Studierenden, die in den Optimierungsprozess der GITTE einfließen sein, um eine **dynamische Anpassung** der Lernassistenten zu ermöglichen und so GITTE an die individuellen Präferenzen und Lernstile der Studierenden anzupassen. Die zentrale Verarbeitung und Geschäftslogik erfolgt in folgenden Modulen:

#### GITTE\_logic (GITTE\_logic.py)

* Klar strukturierte Verarbeitungsschritte:
  + Umwandlung der Nutzereingabe in PALD-Strukturen (JSON)
  + Erzeugung von Prompts für Text- und Bildgenerierung
  + Feedback-Verarbeitung und iterative Anpassung der erzeugten PALDs
  + Konsistenzprüfung zwischen Textbeschreibungen und generierten Bildern
* Trennung der Verantwortlichkeiten:
  + Texterzeugung durch Ollama (über LLM-API)
  + Bildgenerierung mittels Stable Diffusion (über Pic-API)

#### Legal Logic (legal\_logic.py)

* Verwaltung und Verarbeitung der erteilten Zustimmungen
* Eindeutige und nachvollziehbare Speicherung der Zustimmungen (Datenschutz, Studienbeteiligung, KI-Nutzung)
* Ermittlung zulässiger Funktionen auf Basis der gegebenen Zustimmungen

#### Login Logic (auth.py)

* Externe Authentifizierungslogik klar definiert und von UI getrennt
* Überprüfung von Benutzerdaten und Abwicklung der Anmeldungen

#### Student\_logic

Die Komponente **Student\_logic** steuert die **Erfassung der Studierendenmerkmale**, die für die Personalisierung der GITTE relevant sind. Basierend auf dem PACO-Modell von Heidig (2011) werden **emotionale, motivationale, volitionale** und **kognitive** Faktoren (teilweise zukünftig) berücksichtigt, die Einfluss auf die visuelle Gestaltung und das Verhalten der GITTE haben. Darüber hinaus werden auch das **Aussehen, die Persönlichkeit** und **kulturelle Faktoren** einbezogen, da die Literatur zeigt, dass diese ebenfalls eine wichtige Rolle bei der Personalisierung spielen. In der ersten Phase werden zunächst nur die demographischen Daten und Persönlichkeitsmerkmale erhoben.

#### Context\_Logic

Die Komponente **Context\_Logic** erfasst den **Kontext**, in dem die Studierenden mit der GITTE interagieren. Dies umfasst beispielsweise das **Learning Environment** (Lernumgebung), das im PACO-Modell als zentraler Faktor für das Lernerlebnis hervorgehoben wird. Diese Komponente ist für zukünftige Erweiterungen vorgesehen, um den Lernkontext besser zu erfassen und in die Personalisierungslogik einfließen zu lassen.

#### Interaction\_Logic

Die **Interaction\_Logic** steuert den **Personalisierungsprozess** der GITTE auf Grundlage der erfassten **Studierendenmerkmale** und **Kontextinformationen**. In einem ersten Schritt erfolgt die Sammlung von Daten zu den Interaktionen zwischen Studierenden und GITTE, ohne dass bereits eine umfassende Auswertung vorgenommen wird. Sobald ausreichend Daten vorliegen, werden diese analysiert, um die GITTE dynamisch an die individuellen Bedürfnisse der Studierenden anzupassen. Daraus werden die unterschiedlichen Modelle abgeleitet.

Die **Feedback-Komponente** dieser Logik erfasst die Rückmeldungen der Studierenden und verarbeitet sie, um den **Personalisierungsprozess** weiter zu optimieren. Dabei wird Feedback in Bezug auf Studierendenmerkmale, Kontext und GITTE-Gestaltungselemente geprüft und ggf. integriert.

#### Course\_logic

Die **Course\_logic** steuert den **Aufruf und die Integration von Kursinformationen** in das GITTE-System. Sie sorgt dafür, dass die Inhalte der Kurse korrekt dargestellt und personalisierte GITTEs entsprechend den spezifischen Anforderungen des Kurses angepasst werden.

#### Begründung

Die Logikschicht ist so gestaltet, dass sie eine **zentrale Steuerung** aller wichtigen Prozesse und Entscheidungslogiken innerhalb des GITTE-Systems ermöglicht. Sie sorgt für eine klare **Trennung** zwischen der Präsentationsschicht (UI-Layer) und der Datenverarbeitung in der Persistenzschicht, um die Wartbarkeit und Erweiterbarkeit des Systems zu gewährleisten.

Durch den modularen Aufbau der Logikschicht ist es möglich, zukünftige Erweiterungen wie zusätzliche **Gestaltungselemente** der GITTE oder **neue Studierendenmerkmale** ohne grundlegende Änderungen an der Architektur zu integrieren. Gleichzeitig stützt sich die Logikschicht auf den Service Layer, der die notwendigen Schnittstellen zu externen Systemen bereitstellt.

### Dienstleistungsschicht (Service Layer)

Die Dienstleistungsschicht (Service Layer) des GITTE-Systems ermöglicht die **Anpassungsfähigkeit** und **Erweiterbarkeit** des Systems in Bezug auf neue Technologien oder Umgebungen. Sie fungiert als Schnittstelle zwischen der internen Logik des GITTE-Systems und externen Systemen, wie beispielsweise Lernmanagementsystemen (LMS), KI-Diensten oder anderen webbasierten Anwendungen, sowie der Datenhaltungsschicht. Diese Schicht gewährleistet, dass das System flexibel auf neue technologische Entwicklungen und Einsatzszenarien reagieren kann, und ermöglicht so eine Integration externer Dienste.

#### Zweck und Funktionalität

Der primäre Zweck der Dienstleistungsschicht besteht darin, das GITTE-System zukunftssicher und flexibel zu gestalten. Dies bedeutet, dass das System an neue **technologische Entwicklungen** und pädagogische Anforderungen angepasst werden kann. Insbesondere ermöglicht die Dienstleistungsschicht perspektivisch die **Integration externer Systeme**, wie Lernmanagementsysteme (LMS), KI-Dienste oder Plattformen anderer Bildungseinrichtungen, um den Einsatz von GITTE außerhalb der FernUniversität Hagen zu unterstützen.

Die Schicht unterstützt die Anpassung an neue Szenarien, indem sie APIs (Application Programming Interfaces) bereitstellt, die es dem System ermöglichen, mit externen Plattformen zu kommunizieren und deren Daten zu nutzen. Gleichzeitig erlaubt sie die Integration neuerer und möglicherweise besserer KI-Modelle, die zur **Verbesserung der Personalisierung** und **der Lernprozesse** genutzt werden können.

Diese Schicht bietet Schnittstellen zu externen Diensten und Datenquellen:

#### LLM API (LLM\_API.py)

* Nutzung von Ollama zur Texterzeugung
* Explizite Definition verwendeter Modelle (z. B. Llama 3, Llava, Mistral)
* Verbesserte Fehlerbehandlung mit detaillierten Fehlermeldungen
* Optionale Nutzung von LangChain zur erweiterten Modellintegration (über Konfiguration USE\_LANGCHAIN)

#### Pic API (Pic\_API.py)

* Integration der Stable Diffusion Pipeline zur automatisierten Bildgenerierung
* Speicherung erzeugter Bilder lokal oder alternativ in MinIO-Objektspeicher
* Robuste Fehlerbehandlung und Fallback-Mechanismen (Dummy-Bilder)

#### PALD Service (PALD\_service.py)

* Klare Trennung von Geschäftslogik (Logic-Schicht) und persistenter Speicherung (Data-Layer)
* Speicherung, Abruf und Löschung von PALDs über klar definierte Schnittstellen

#### Student Service (student\_service.py)

* Verwaltung von Studierendendaten inklusive Erstellung, Aktualisierung und Löschung
* Erfassung und persistente Speicherung der Zustimmungen und Entscheidungen der Studierenden
* Nutzung eindeutiger Datenbankoperationen und Transaktionssicherheit

#### Chat Service (chat\_service.py)

* Persistente Speicherung der Chatnachrichten (inklusive Bildern und PALD-Verknüpfungen)
* Bereitstellung umfangreicher Funktionen zur Verwaltung und Abfrage des Chatverlaufs

#### Parsing-Rendering

Die **Parsing-Rendering**-Komponente dient der **Konvertierung und Verarbeitung** von GITTE-Designs in verschiedenen Formaten, um deren **Flexibilität** und **Anpassungsfähigkeit** sicherzustellen. Die Komponente unterstützt mehrere Konvertierungsprozesse, die in dieser Phase auf dem PALD-Modell basiert.:

* **PALD2JSON**: Konvertiert PALD-Strukturen in JSON-Format, um die Speicherung und Verarbeitung der Daten zu erleichtern.
* **PALD2Image**: Wandelt PALD-Strukturen in Bilder um, um visuelle Darstellungen der GITTE zu erstellen.
* **Image2PALD**: Ermöglicht die Rückkonvertierung von Bildern in PALD-Strukturen, um Vergleiche und Analysen durchzuführen.
* **Text2PALD**: Wandelt natürlichsprachliche Texteingaben in PALD-Strukturen um, um eine flexible und intuitive Dateneingabe zu gewährleisten.
* **ComparePALD**: Vergleicht verschiedene PALD-Strukturen, um Unterschiede, Ähnlichkeiten oder fehlende Elemente zu identifizieren.
* **CombinePALD**: Kombiniert mehrere PALD-Strukturen zu einer integrierten Struktur, die die Grundlage für die Erstellung einer umfassenderen GITTE darstellt.

Diese Funktionen ermöglichen es, GITTE-Designs effizient zu verarbeiten, zu analysieren und in weitere Formate neben dem PALD-Modell bereitzustellen, was die Anpassbarkeit des Systems erhöht.

#### LMS API

Die **LMS API** stellt eine **Schnittstelle zu Lernmanagementsystemen** wie Moodle bereit und ermöglicht den **Datenaustausch** zwischen dem GITTE-System und externen Bildungsplattformen. Diese Schnittstelle erleichtert die Integration von Kursmaterialien, Studierendendaten und Lernverhalten in die GITTE-Personalisierung. Sie ermöglicht eine dynamische Anpassung der Lernassistenten basierend auf den im LMS verfügbaren Informationen.

#### Surveys

Die **Surveys**-Komponente verwaltet und verarbeitet **Fragebögen**, die zur Erhebung von Studierendenmerkmalen genutzt werden. Die Umfragen erfassen wesentliche Informationen über die Präferenzen und Eigenschaften der Studierenden und tragen so zur **Personalisierung der GITTE** bei. Die Auswertung der Fragebögen ermöglicht eine tiefere Analyse und Anpassung des Systems an individuelle Lernbedürfnisse.

#### Weitere ILA-Komponenten

Die Dienstleistungsschicht unterstützt zudem die Integration weiterer **Intelligenter Lernassistenten (ILA)**-Komponenten. Dies betrifft z.B. die Interaktion mit den Studierenden, didaktische Module, etc. die zur Erweiterung der Funktionalitäten beitragen. Diese Komponenten können je nach Bedarf an unterschiedliche Lernumgebungen und Studierendenpräferenzen angepasst werden, um eine noch individuellere Lernunterstützung zu ermöglichen.

#### Begründung

Die Dienstleistungsschicht spielt eine zentrale Rolle bei der **Erweiterbarkeit** des GITTE-Systems. Durch ihren modularen Aufbau kann das System problemlos auf neue Lernumgebungen und Bildungseinrichtungen ausgeweitet werden, ohne dass grundlegende Änderungen an der Systemarchitektur erforderlich sind. Sie ermöglicht die **Anpassung an verschiedene externe Systeme**, indem sie Schnittstellen für die Integration bereitstellt.

Mit dieser Schicht kann das System kontinuierlich aktualisiert und um **neue Technologien**, insbesondere im Bereich der Künstlichen Intelligenz, erweitert werden. Sie stellt sicher, dass GITTE mit externen Plattformen kooperieren kann und so eine **interdisziplinäre** und **institutionenübergreifende Nutzung** des Systems ermöglicht wird.

Darüber hinaus erleichtert die Dienstleistungsschicht die **Integration pädagogischer Neuerungen** in das System, indem sie den Austausch von Informationen zwischen GITTE und anderen bildungsrelevanten Systemen wie Lernmanagementplattformen und Didaktiktools unterstützt.

#### **Zu ergänzen für Federated Learning & PALDSchema-Ergänzungen:**

* federated.FL\_client (Logic→Service‑Aufruf): startet **nicht‑blockierend** lokale Kurz‑Updates; exportiert **Adapter‑Deltas** (z. B. LoRA).
* federated.FL\_server: nimmt Deltas entgegen, **FedAvg + DP** + Versionierung; veröffentlicht globale Modellstände.
* schema\_service: lädt/speichert **pald\_schema.json**, führt **Schema‑Patches** aus, verwaltet Versionen.
* federated.FL\_schema: aggregiert **Schema‑Kandidaten** aus Clients und triggert Schema‑Patches.
* **Schnittstellen (vereinfacht):**
* submit\_update(delta), aggregate\_and\_publish(), submit\_schema\_suggestions(payload).
* **Datenhaltung (Data‑Layer):**
* Tabellen/Files für **Model‑Registry** (Version, Artefaktpfad) und **FL‑Metriken** (runde, name, value).

### Persistenzschicht (Persistence Layer)

Die Persistenzschicht ist für die dauerhafte und zuverlässige Speicherung aller relevanten Daten innerhalb des GITTE-Systems verantwortlich. Sie stellt sicher, dass sowohl Studierendenmerkmale als auch spezifische GITTE-Gestaltungsmerkmale, Feedback-Daten und Interaktionslogs sicher und konsistent gespeichert werden. Durch die Verwendung dieser Schicht wird der **schnelle und sichere Zugriff** auf die gespeicherten Daten ermöglicht, wodurch die Personalisierung und Optimierung der GITTEs kontinuierlich verbessert werden kann.

#### Erfassung und Verwaltung von Daten

Die Persistenzschicht gewährleistet die **Erfassung, Verwaltung und langfristige Speicherung** der Studierendendaten sowie der Informationen zur visuellen Gestaltung der GITTEs. Neben den typischen Profildaten soll es in zukünftigen Ausbaustufen möglich sein, auch **sprachliche und emotionale Interaktionen** der Studierenden zu erfassen, die in die Personalisierungsprozesse einfließen können.

Diese Schicht ermöglicht es, Daten aus den verschiedenen Interaktionen der Studierenden zu speichern und für zukünftige **Analysen und Anpassungen** zu verwenden. Die Erfassung der Interaktionen und Chatverläufe bietet dabei die Grundlage für eine detaillierte Analyse des Lernverhaltens und unterstützt somit die fortlaufende Optimierung des Lernassistenten.

#### Organisatorische Funktionen

Neben der Datenverwaltung unterstützt die Persistenzschicht auch **organisatorische Prozesse**. Dazu zählt insbesondere die Verwaltung von grundlegenden Funktionalitäten wie der **Login-Prozess**, die **Einhaltung von Informationspflichten** sowie das Einholen und Verwalten der erforderlichen **Zustimmungen** der Studierenden zur Nutzung ihrer Daten. Diese organisatorischen Aufgaben gewährleisten die **Rechtskonformität**, insbesondere im Hinblick auf die Datenschutz-Grundverordnung (DSGVO).

Die persistente Speicherung wird mit PostgreSQL-Datenbanken und optional MinIO realisiert:

#### PALD Storage (PALD\_storage.py)

* Präzisierte Tabellenstruktur mit Referenzen (Foreign Keys) zu Studierenden und Eltern-PALDs
* Optimierte Abfragen durch klar definierte Indizes und Schlüsselbeziehungen

#### Student Storage (student\_storage.py)

* Klare Tabellen für Studierende, inklusive personenbezogener Daten (pseudonymisiert), Zustimmungen, Entscheidungen
* Verknüpfung der Studierendeninformationen mit PALDs und Chatnachrichten über eindeutige Schlüsselbeziehungen

#### Chat Storage (Chat\_storage.py)

* Neu eingeführte Datenbankstruktur zur persistenten Speicherung des Chatverlaufs
* Speicherung von Textnachrichten, Bildern und Verknüpfung mit PALDs
* Abrufbarkeit von Nachrichtenverläufen nach Studierenden, Zeitstempeln und PALDs

#### GITTE Storage (GITTE\_storage.py)

* Speicherung erzeugter Bilder entweder lokal oder mittels MinIO-Cloudspeicher
* Robuste Mechanismen zur Sicherstellung der Erreichbarkeit und Integrität der gespeicherten Bilder

#### Systemkonfigurationen:

Beinhaltet die Konfigurationsdaten des Systems, die für den Betrieb und die Verwaltung des GITTE-Systems erforderlich sind.

#### Prompt-Templates:

Diese Vorlagen für Eingabeaufforderungen, die in der Interaktion mit generativen KI-Systemen verwendet werden, sind ebenfalls Teil der in dieser Schicht verwalteten Daten.

#### Begründung

Die Persistenzschicht gewährleistet die **Sicherheit, Konsistenz** und **Nachvollziehbarkeit** aller gespeicherten Daten. Durch die Trennung der Persistenzschicht von den darüber liegenden Schichten bleibt die **Datenintegrität** auch dann gewahrt, wenn in anderen Schichten Änderungen oder Fehler auftreten.

* **Datensicherheit und Konsistenz**: Die Persistenzschicht sorgt dafür, dass alle systemrelevanten Daten sicher und zuverlässig gespeichert werden. Datenverluste werden vermieden, indem künftig regelmäßige Backups und andere Sicherungsmaßnahmen implementiert werden.
* **Nachvollziehbarkeit**: Diese Schicht erlaubt die langfristige Speicherung von Daten, die für die **kontinuierliche Verbesserung** und **Personalisierung** der GITTEs entscheidend sind. Insbesondere durch die Aufzeichnung von Interaktionsdaten und Feedback der Studierenden wird es möglich, die Lernprozesse kontinuierlich zu analysieren und anzupassen.

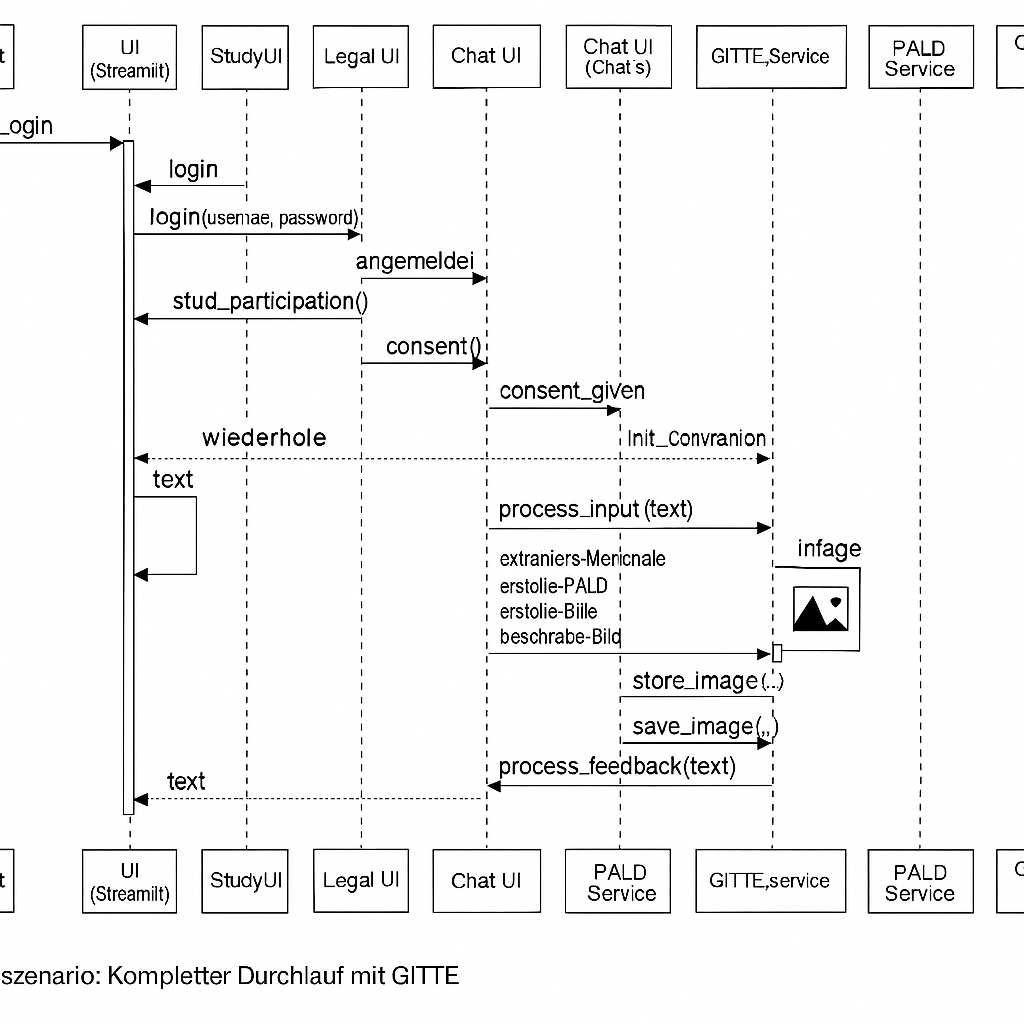
#### Integration und Schnittstellen

Die Persistenzschicht verfügt über Schnittstellen zu verschiedenen externen Systemen, die für den Betrieb des GITTE-Systems von zentraler Bedeutung sind:

1. **Externe Datenquellen und APIs**: Die Persistenzschicht ermöglicht es, externe Datenquellen anzubinden, um zusätzliche **Kontextinformationen** für die Repräsentation der GITTE zu verwenden. Diese externen Daten können dazu beitragen, die **Personalisierung** weiter zu verfeinern und GITTE an individuelle Lernumgebungen oder externe Wissensressourcen anzupassen.

# Laufzeitsicht (ui\_level\_main)

Die Laufzeitsicht beschreibt das dynamische Verhalten des GITTE-Systems anhand der wesentlichen Interaktionsschritte, die ein Studierender bei der Nutzung durchläuft. Nachfolgend wird der Ablauf detailliert dargestellt, so wie er in der zentralen Steuerungsdatei (ui\_level\_main.py) vorgesehen ist, einschließlich geplanter, aber noch nicht vollständig implementierter Schritte:



## Übersicht der Laufzeitszenarien

Die GITTE-Anwendung läuft in klar definierten und aufeinander aufbauenden Phasen ab, die alle Interaktionen zwischen den Studierenden und dem System umfassen:

### 4.1 Use-Cases (Kurzform, MUSS)

* **UC-01 Login/Registrierung**: Nutzer:in registriert sich, erhält Rolle; Passwort-Hashing.
* **UC-02 Consent**: Anzeige Rechtstexte → Zustimmung persistieren → Freischalten.
* **UC-03 Chat mit LLM**: Prompt eingeben → Antwort streamen → Verlauf speichern.
* **UC-04 Bildgenerierung**: Beschreibung → SD-Pipeline → Bild anzeigen/speichern.
* **UC-05 Prompt-Audit**: Jede I/O wird unveränderlich gespeichert, parent-child-Link.
* **UC-06 Export/Reporting** (Admin): CSV/JSON Export der Studien-/Chatdaten.

### 4.2 Nicht-funktionale MUSS-Anforderungen (Auszug)

* **NFR-SEC-01**: Passwort-Hashing via bcrypt, Rollenprüfung pro Aktion.
* **NFR-LOG-01**: Strukturierte Logs (DEBUG→ERROR), Korrelation via request\_id.
* **NFR-I18N-01**: Texte/Labels zentral in config.py pflegbar.

### Initialisierung und Login

 ui\_level\_main.py

* prüft st.session\_state, z. B. log\_in, GITTE\_choice, legal\_choice, etc.
* ruft nacheinander folgende UIs auf:
  + login\_ui.login()
  + study\_participation\_ui.study\_participation()
  + legal\_ui.consent()
  + GITTE\_yes\_no()
  + GITTE\_chat\_ui.run\_chat()

 **Login-Phase**

* login\_ui.py: Eingabe von Benutzername und Passwort
* Authentifizierung über auth.py (oder student\_service.check\_credentials)
* Session-State wird gesetzt (log\_in = False, GITTE\_choice = True)

### Entscheidung über GITTE-Nutzung (GITTE\_choice)

**GITTE Entscheidung**

* Nutzer:in wählt „Ja“ zur Gestaltung → Session-State GITTE = True

### Teilnahmebestätigung und Erzeugung eines Studierenden-Pseudonyms

**Studienteilnahme (pseudonymisiert)**

* Nutzer:in erzeugt pseudonymen Schlüssel
* student\_service.create\_student() wird aufgerufen
* Session-State student\_id wird gesetzt

### Einholen von Zustimmungen zu rechtlichen Bedingungen (Legal\_choice)

**Einwilligungen**

* legal\_ui.py zeigt Einwilligungstexte
* Auswahloptionen für Datenschutz, Studienbeteiligung, KI-Nutzung
* Speicherung der Einwilligungen in DB über LegalLogic und student\_service.set\_consent

### Erhebung initialer Studierendenmerkmale (Survey)

### Gestaltungsabfrage des Lernassistenten (GITTE-Chat)

 **Chat-Initialisierung**

* GITTE\_chat\_ui.ChatInterface fragt nach Name
* GITTE\_logic.init\_conversation() erzeugt Begrüßung + leeres PALD
* Begrüßung wird im Chatverlauf angezeigt

 **Erster Prompt (Designwunsch)**

* process\_input() in GITTE\_logic:
  + Extrahiert Designmerkmale → erstellt initiales PALD
  + Generiert Prompt für Bildgenerator (aus PALD)
  + Bild wird erzeugt → in MinIO oder lokal gespeichert
  + Chat speichert Bild als Nachricht
  + Beschreibung des Bildes → erzeugt weiteres PALD
  + Konsistenzprüfung zwischen Original und Beschreibung (optional)
  + Antwort-Text wird generiert → zurück an Studierende:n

### Feedback-Schleifen zur schrittweisen Verbesserung des Lernassistenten

**Feedbackschleife**

* Nutzer:in gibt Feedback
* process\_feedback() → neues PALD
* erneute Bildgenerierung + Beschreibung + Konsistenzcheck

### Federated Learning

**Schritt X: Föderiertes Mini‑Update (optional, bei Opt‑in)**

Nach Speicherung der Antwort wird ein nicht‑blockierender Call an FL\_client ausgelöst. Eingaben: **nur** strukturierte Signale (PALD‑JSON, Klicks/Thumbs, Konsistenzlabel). Ausgabe: **Delta** → FL\_server.

**Schritt Y: Dynamische Schema‑Kandidaten (optional)**

Erkennt der Parser bislang nicht modellierte Attribute (z. B. hair.color), sendet der Client eine **Schema‑Kandidatenmeldung** (Key, Typ, Count) an FL\_schema. Ab Erreichen der Schwellen (Clients, Support, Beispiele) erstellt der schema\_service eine neue **PALD‑Schema‑Version**; die Clients übernehmen sie beim nächsten Start.

### Abschluss der Interaktion und Übergang in die Kursumgebung

Im Folgenden wird der Ablauf mit diesen Schritten ausführlich erläutert.

### Ablaufbeschreibung im Detail

#### Schritt 1: Initialisierung und Login (UI-Ebene)

* Beim Aufruf der Anwendung wird eine Überprüfung der Session-State-Variablen durchgeführt.
* Falls noch kein gültiger Login vorliegt (log\_in = True), erfolgt zunächst der Aufruf der Login-Seite (login\_ui.py).
* Die Authentifizierung erfolgt durch Eingabe von Benutzername und Passwort und wird über die Logic- (auth.py) und Service-Schicht (student\_service.py) abgewickelt.

#### Schritt 2: Entscheidung über GITTE-Nutzung (GITTE\_choice)

* Nach erfolgreichem Login wird abgefragt, ob der/die Studierende eine eigene Gestaltung des Lernassistenten vornehmen möchte.
* Die Entscheidung wird persistiert und beeinflusst den weiteren Ablauf wesentlich (gespeichert in student\_decisions).

#### Schritt 3: Teilnahmebestätigung und Erzeugung eines Studierenden-Pseudonyms

* Der Studierende wird aufgefordert, ein persönliches, anonymes Pseudonym zu erstellen (study\_participation\_ui.py).
* Die Datenbank-Tabellen werden an dieser Stelle initialisiert, falls nicht bereits erfolgt (db\_service.init\_all\_db()).
* Das Pseudonym und eine eindeutige student\_id werden generiert und persistiert (student\_service.create\_student()).

#### Schritt 4: Einholen von Zustimmungen (Legal\_choice)

* Der Studierende erhält rechtliche Informationen zu Datenschutz, Studienbeteiligung und KI-Einsatz (legal\_ui.py).
* Er muss allen Bedingungen explizit zustimmen, andernfalls erfolgt ein Abbruch und eine Weiterleitung zur normalen Kursseite.
* Zustimmungen werden über die Logic-Schicht (legal\_logic.py) verarbeitet und durch student\_service persistiert.

#### Schritt 5: Erhebung initialer Studierendenmerkmale (Survey)

* In dieser Phase erfolgt eine erste Erfassung relevanter Studierendenmerkmale mithilfe von Fragebögen.
* Diese Funktionalität ist aktuell noch nicht vollständig implementiert, aber in survey\_ui.py vorgesehen.
* Ergebnisse werden zukünftig persistiert und dienen als Grundlage für die personalisierte Erstellung der GITTE.

#### Schritt 6: Gestaltungsabfrage des Lernassistenten (GITTE-Chat)

* Die Interaktion mit der GITTE findet in Form eines Chatdialogs statt (GITTE\_chat\_ui.py).
* Initial erfolgt die Begrüßung und Abfrage gewünschter Gestaltungselemente durch den Lernassistenten (Logic-Schicht: GITTE\_logic.py).
* Aus der Benutzereingabe werden automatisch Gestaltungselemente extrahiert (PALD-Erstellung), ein Bild generiert (Stable Diffusion via Pic\_API.py), und das Bild dem Studierenden angezeigt.

#### Schritt 7: Feedback-Schleifen zur schrittweisen Verbesserung des Lernassistenten

* Der Studierende gibt Feedback zu generierten Bildern.
* Jede Rückmeldung erzeugt eine aktualisierte PALD-Struktur und ggf. ein angepasstes Bild.
* Der Zyklus aus Eingabe –> Bildgenerierung –> Feedback –> Anpassung wiederholt sich so lange, bis der Studierende zufrieden ist.
* Diese Feedbackschleifen sind durch entsprechende Flags (awaiting\_feedback) im Session-State gesteuert und über die Logic- und Service-Schichten realisiert.

#### Schritt 8: Abschluss der Interaktion und Übergang in die Kursumgebung

* Nach Abschluss der GITTE-Gestaltung wird der/die Studierende auf die eigentliche Kursseite weitergeleitet.
* Der Kurs erhält die Informationen zur personalisierten GITTE, um diese als Lernassistenten nutzen zu können.
* Dieser Übergang erfolgt automatisiert abhängig vom Zustand der Session-State-Variablen (participation\_complete).

## Besonderheiten im Zusammenspiel der Bausteine:

* **Session-State als zentrale Steuerungskomponente:**
  + Der gesamte Ablauf wird über Statusvariablen im Session-State gesteuert. Die Übergänge zwischen Phasen erfolgen anhand dieser Zustände.
* **Lazy-Loading und Caching:**
  + Um Ressourcen zu schonen, erfolgt die Initialisierung der LLM-Modelle (via LLM\_API.py) und der Stable Diffusion Pipeline (Pic\_API.py) erst bei Bedarf und wird über Streamlit gecacht.
* **Persistenz und Konsistenz der Daten:**
  + Für alle Datenbankaktionen sorgt db\_service.init\_all\_db() für die Initialisierung der Tabellen (einmalig bei App-Start).
  + Die Speicherung der Daten erfolgt streng getrennt von der Logik (via dedizierte Storage-Module wie student\_storage.py, PALD\_storage.py, Chat\_storage.py).
  + Jeder PALD-Schritt wird mit Typ und optionalem Parent-ID gespeichert.
  + Jedes generierte Element (Textnachrichten, PALDs, Bilder) wird persistent gespeichert und mit klar definierten IDs verknüpft.
* **Geplante Erweiterungen (noch nicht vollständig implementiert):**
  + Die Erhebung detaillierter Studierendenmerkmale (Schritt 5).
  + Dynamische Anpassung des Lernassistenten auf Basis eines kontinuierlichen Lernmodells.
  + Integration mit Lernmanagementsystemen und externen Bildungsplattformen zur umfassenden Nutzung der GITTE in der Lernumgebung.

# 🖧 Verteilungssicht (Deployment View)

## Zweck

Die Verteilungssicht beschreibt die technische Infrastruktur, auf der das GITTE-System lokal ausgeführt wird. Sie zeigt, welche Systembausteine auf welchen physischen oder logischen Knoten installiert bzw. ausgeführt werden, wie diese Knoten verbunden sind und welche Kommunikationsbeziehungen bestehen.

## 1.4 Verteilungssicht (Deployment-Annahmen)

* **Knoten A (App-Server)**: Streamlit-UI, Logic-, Service-Layer.
* **Knoten B (DB)**: PostgreSQL.
* **Knoten C (GPU-Node)**: Diffusers/Stable-Diffusion Pipeline (kann = Knoten A).
* Ports/Firewalls: nur interne Kommunikation zwischen A–B–C; TLS empfohlen.

## Infrastruktur-Ebene 1: Lokale Einzelplatznutzung (Entwicklungs-/Studienmodus)

### Komponenten

| **Knoten** | **Beschreibung** |
| --- | --- |
| 🖥️ **Lokaler Rechner** | Arbeitsstation oder Laptop der Entwickler:in oder Studien­teilnehmenden. Führt alle Systemkomponenten lokal aus. |

### Software-Verteilung auf dem lokalen Rechner

| **Komponente** | **Beschreibung** | **Technologie** |
| --- | --- | --- |
| **Streamlit-Frontend** | Darstellung der UI-Komponenten (Login, Legal, Fragebogen, Chat) | streamlit, Python |
| **Logikschicht** | Verarbeitung von Benutzerinteraktionen, Steuerung des Ablaufs, Prompt-Generierung, Konsistenzprüfung | Python, eigene Klassen |
| **Service-Schicht** | Zugriff auf KI-Modelle, Bildgenerierung, Speicherung | HTTP-Calls zu lokalem Ollama & lokale Stable-Diffusion |
| **Datenhaltung (Persistence)** | PostgreSQL-Datenbank data\_collector mit Tabellen für Studierende, PALDs, Chats etc. | PostgreSQL, lokal installiert |
| **Ollama-Server** | Localhost-Server für LLM-Modelle (Llama3, Llava3, Mistral etc.) | Ollama, CLI |
| **Stable Diffusion** | Bildgenerierung lokal über Diffusers | diffusers, torch, Modell: runwayml/stable-diffusion-v1-5 |
| **Dateispeicher für Bilder** | Lokales Verzeichnis generated\_images/local/ als Fallback zu MinIO | Dateisystem |

### Netzwerkverbindungen

* Alle Komponenten kommunizieren **lokal**:
  + localhost:8501 für die Streamlit-Webapp
  + localhost:11434 für die LLM-API (Ollama)
  + localhost:5432 für PostgreSQL
* Kein Internetzugang notwendig bei vorinstallierten Modellen

### **Zusatzkomponenten Federated Learning (optional):**

* FL‑Aggregator (leichtgewichtig): Prozess/Thread im Backend, der lokale Test‑Deltas sammelt und FedAvg simuliert (für Studienmodus).
* Schema‑Registry (Datei): schemas/pald\_schema.json mit Versionierung.  
  **Kommunikation:** intern (In‑Process/localhost); keine externen Datenflüsse.

## Verteilungssicht – Visualisierung (Textuell)

+------------------------+ +----------------------+

| | | |

| 🖥️ Lokaler Rechner |<-------->| PostgreSQL DB |

| | | "data\_collector" |

+-----------+------------+ +----------+-----------+

| ^

v |

+--------+---------+ |

| Streamlit App | |

| (UI-Ebene) | |

+--------+---------+ |

| |

v |

+--------+---------+ |

| Logik-Ebene | |

+--------+---------+ |

| |

v |

+--------+------------------------------+ |

| Service-Layer: | |

| - LLM\_API → Ollama (localhost:11434) | |

| - Pic\_API → Stable Diffusion local | |

| - PALD-, Chat-, Student-Services | |

+--------------------------------------+ |

## Qualitäts- und Leistungsmerkmale dieser Konfiguration

| **Merkmal** | **Beschreibung** |
| --- | --- |
| **Einfachheit** | Alle Komponenten laufen auf einem Rechner – einfache Installation |
| **Offline-Fähigkeit** | Kein Internet erforderlich nach Download der Modelle |
| **Performance** | Begrenzte Skalierbarkeit, aber ausreichend für Einzelstudien oder Tests |
| **Datenschutz** | Keine externen Datenflüsse; vollständige lokale Kontrolle |
| **Reproduzierbarkeit** | Zustand lässt sich durch Datenbank-Dumps und Logs dokumentieren |

## Zuordnung von Bausteinen zu Infrastruktur

| **Baustein** | **Infrastrukturknoten** |
| --- | --- |
| UI-Level (alle UIs) | Lokaler Rechner (Streamlit) |
| Logic-Level | Lokaler Rechner (Python) |
| Service-Level | Lokaler Rechner (Python, HTTP) |
| Ollama / LLMs | Lokaler Rechner (localhost) |
| Stable Diffusion | Lokaler Rechner (diffusers) |
| PALD- und Chat-Datenbank | Lokale PostgreSQL-Instanz |
| Bildspeicher | Lokales Dateisystem |

Hier ist die **Verteilungssicht für eine zukünftige Cloud-/MinIO-Variante des GITTE-Systems**, wie sie sich aus dem aktuellen Code ergibt – insbesondere durch Unterstützung für MinIO, Ollama-APIs und eine modularisierte Architektur mit klar getrennten Schichten:

## ☁️ Verteilungssicht – Cloud/MinIO-Szenario

### Zweck

Diese Verteilungssicht beschreibt die Deployment-Architektur für eine produktionsnahe oder skalierte Cloud-basierte Nutzung von GITTE. Ziel ist die Trennung der Komponenten auf mehrere physische oder virtuelle Knoten, inklusive der Nutzung von MinIO zur Bildspeicherung und ggf. Remote-Modellen (z. B. Ollama als Container, LangChain, Cloud-LLMs).

### Infrastruktur-Ebene 1: Containerisierte Microservice-Architektur (Cloud oder On-Prem)

| **Knoten** | **Beschreibung** |
| --- | --- |
| 🧑‍🎓 **Client-Browser** | Endnutzer:in greift über Webbrowser auf das GITTE-Frontend zu |
| 🌐 **Frontend-Server (Streamlit)** | Hostet die Streamlit-App, die die UI bereitstellt |
| 🧠 **LLM-Backend (Ollama / LangChain)** | Containerisierte LLMs (Llama, Mistral etc.) via HTTP erreichbar |
| 🎨 **Bildgenerator (Stable Diffusion)** | Separater Container oder Server mit GPU zur Bildgenerierung |
| 🗄️ **MinIO-Server** | Objektspeicher für generierte Bilder |
| 🛢️ **PostgreSQL-DB** | Zentrale relationale Datenbank |
| 🔒 **Reverse Proxy / Auth Layer** | Optional: NGINX, Authentifizierung, TLS, Load Balancer |

### Deployment-Skizze (textuell)

+------------------------+ HTTP/WebSocket +---------------------+

| 🧑‍🎓 Client-Browser | <---------------------> | 🌐 Streamlit-Server |

+------------------------+ | (UI + Logic Layer) |

+----------+----------+

|

v

+-------------------------------+ +------------------------+

| 🧠 Ollama / LangChain | | 🎨 Stable Diffusion |

| (Text & Bildbeschreibung) | | (Bild-Generator) |

+-------------------------------+ +-------------------------+

|

v

+----------------------+

| 🗄️ MinIO Image Store |

| (HTTP, S3-kompatibel)|

+----------------------+

^

|

+----------------------------+

| 🛢️ PostgreSQL DB |

| (Students, PALDs, ...) |

+----------------------------+

### Zuordnung von Bausteinen

| **Baustein** | **Infrastrukturknoten** |
| --- | --- |
| UI-Level | Streamlit-Webserver |
| Logic-Level | Streamlit-Webserver |
| Service-Level: LLM\_API | Ollama / LangChain-Container |
| Service-Level: Pic\_API | Stable Diffusion-Server (GPU) |
| Service-Level: GITTE\_storage | MinIO |
| Data-Level (alle \*\_storage.py) | PostgreSQL |

### Kommunikationswege (typisch via HTTPS)

| **Von** | **An** | **Protokoll** | **Zweck** |
| --- | --- | --- | --- |
| Browser | Streamlit | HTTPS | Benutzerinteraktion |
| Streamlit | PostgreSQL | TCP | Datenhaltung |
| Streamlit | MinIO | HTTPS (S3) | Upload/Download von Bildern |
| Streamlit | Ollama / LangChain | HTTP | Promptgenerierung, Bildbeschreibung |
| Streamlit | Stable Diffusion | HTTP oder interner API-Call | Bildgenerierung |

### Qualitäts- und Leistungsmerkmale

| **Merkmal** | **Beschreibung** |
| --- | --- |
| **Skalierbarkeit** | Jeder Service (LLM, Bild, DB) kann getrennt skaliert werden (horizontal oder vertikal) |
| **Wartbarkeit** | Containerisierte Microservices sind unabhängig update- und restartbar |
| **Zugänglichkeit** | Nutzer:innen benötigen nur einen Webbrowser |
| **Flexibilität** | Modell-Austausch über LLM-API möglich; Bildgenerator konfigurierbar |
| **Speicherauslagerung** | Generierte Bilder in MinIO, nicht im Backend selbst |
| **Mehrbenutzerbetrieb** | Parallele Sessions mit eigenen student\_ids und pald\_id-Ketten |

### Hinweise zur Umsetzung

* **Docker-Setup empfehlenswert**:
  + streamlit, postgres, ollama, minio, ggf. nginx als Proxy
* **GPU-Zugang für Bildgenerierung** zwingend für Stable Diffusion
* **MinIO-Konfiguration** wird bereits im Code erkannt (Umgebungsvariablen)
* **Datenbankverbindung über ENV/Secrets steuerbar**
* **Reverse Proxy mit TLS** für produktiven Einsatz (HTTPS, OAuth2 etc.)

### Zusatzknoten für Federated Learning:

* **FL‑Server/Aggregator** (Container): HTTP‑Endpoint für Delta‑Uploads, Batch‑Aggregation, DP‑Rauschen, Model‑Registry.
* **Schema‑Service** (Container): verwaltet **pald\_schema.json** und API für Clients (Version‑Check).  
  **Kommunikation:** UI/Logic → Services (HTTPS); Clients senden nur **Deltas/Kandidaten**; **keine** Rohdaten/Assets.

## Template Infrastruktur Ebene 1

***<Übersichtsdiagramm>***

Begründung

*<Erläuternder Text>*

Qualitäts- und/oder Leistungsmerkmale

*<Erläuternder Text>*

Zuordnung von Bausteinen zu Infrastruktur

*<Beschreibung der Zuordnung>*

# 🧩 Querschnittliche Konzepte

Dieses Kapitel beschreibt systemweite Konzepte, die unabhängig von einer konkreten Schicht oder Komponente sind, aber für mehrere Teile des Systems relevant sind. Sie erhöhen die Wiederverwendbarkeit, Kohärenz und Wartbarkeit des Systems.

* **Konfiguration**: Alle Flags/Modelle/Prompts zentral in config.py.
* **Logging & Tracing**: strukturiertes Logging (JSON möglich), Korrelation per request\_id.
* **Persistenz**: strikte 4-Layer-Trennung: UI → Logic → Service → Data (CRUD).
* **Security**: Passwort-Hashing (bcrypt), Rollenprüfung im Logic-Layer.
* **Internationalisierung (i18n)**: Texte zentral konfigurierbar (DE Standard).
* **Fehler-/Retry-Konzept**: Backoff bei LLM/Bild-API, Fallbacks loggen.

## 🔐 Authentifizierung & Autorisierung

**Zweck:** Sicherstellen, dass nur berechtigte Benutzer:innen Zugriff auf das System haben.

### Umsetzung:

* Aktuell einfache Pseudonym-basierte Authentifizierung (auth.py bzw. student\_service.py)
* Kombination von Benutzername und Passwort (ggf. später OAuth/SSO integrierbar)
* Session-Management über st.session\_state
* Entscheidungen und Zustimmungen sind nutzergebunden über student\_id

### Ausblick:

* Einführung einer rollenbasierten Zugriffskontrolle (RBAC) bei Mehrbenutzerszenarien
* Optional: Absicherung über zentrale Auth-Instanz bei Cloud-Betrieb

## 🔐 Datenschutz & DSGVO-Konformität

**Zweck:** Schutz personenbezogener Daten gemäß EU-DSGVO.

### Umsetzung:

* Nutzer:innen erstellen pseudonymisierte IDs (kein Klarname)
* Zustimmung zu drei Kernpunkten über legal\_ui und legal\_logic:
  + Datenschutz
  + Studienbeteiligung
  + KI-Nutzung
* Speicherung der Zustimmungen mit Timestamp und Typ in student\_consents

### Verwaltung:

* Trennung von Zustimmung und Nutzung
* Bei Abbruch werden Daten nicht gespeichert oder automatisch gelöscht

### Ausblick:

* Lösch- und Exportfunktion für Nutzer:innendaten
* Transparenzbericht zur KI-Verwendung

## Federated Learning & Datenschutz

**Prinzip:** Modell‑Updates statt Daten. Clients trainieren kurze lokale Schritte auf strukturierten Signalen (PALD, Feedback‑Events).

**Aggregation:** Server wendet **FedAvg** an; optional **Secure Aggregation** und **Differential Privacy** (Rauschen σ, Gradient‑Clipping).

**Opt‑in:** Einwilligung in Legal‑UI; Widerruf beendet Teilnahme.

**Protokollierung:** Nur aggregierte Metriken (Runden‑Loss, Anzahl Beispiele); keine personenbezogenen Rohdaten.

**Fail‑safe:** Bei Opt‑out oder Fehlern fällt der Client auf das globale Modell zurück.

**Bezug zu Qualitätszielen:** Datenschutz, Personalisierung, Datenqualität.

## Dynamische PALD‑Schema‑Entwicklung (Schema Discovery & Versionierung)

**Motivation:** Neue Nutzermerkmale (z. B. Haare) sollen ohne Big‑Bang‑Refactoring ergänzt werden.

### **Ablauf:**

1. **Lokal** erkannte, bislang nicht modellierte Attribute (z. B. hair.color=red, hair.length=long) werden als **Kandidaten** zusammengefasst (nur Schlüssel, Typ, Count, wenige Beispielwerte).
2. **Föderierte Aggregation** konsolidiert Kandidaten (Mindest‑Support, Mindestanzahl, DP‑geeignete Zählung).
3. **Schema‑Service** vergibt kanonische Keys, erhöht **PALD\_SCHEMA\_VERSION** und veröffentlicht die neue **pald\_schema.json**.
4. **Abwärtskompatibel:** Neue Felder sind optional; bestehende Flows bleiben funktional.
5. **Integration:** Prompt‑Generator nutzt neue Felder „best effort“; Parser mappt frühere \_aux‑Treffer auf neue Keys.
6. **Nutzen:** Erfüllung der Anforderung „Erweiterbarkeit der Studierendenmerkmale/Strukturierungsschemas“ ohne zentrale Rohdatensammlung.

## 🧠 Personalisierung & Feedback-Zyklus

**Zweck:** Erstellung personalisierter GITTEs auf Basis von Textfeedback und iterativer Bildverarbeitung.

### Kernelemente:

* Nutzung des **PALD-Modells** (Pedagogical Agent Level of Design) zur Strukturierung der Designmerkmale
* Feedback-basierte Iteration: Nutzerkommentare → Anpassung → neues Bild
* Speicherung aller Schritte (initial → prompt → Bildbeschreibung → Feedback) mit parent\_id-Verknüpfung

### Besonderheit:

* Nachvollziehbare Kette aller Entscheidungen in palds und chat\_messages

## 🧠 KI-Integration & Modellmanagement

**Zweck:** Modularer und austauschbarer Einsatz von LLMs und Bildgeneratoren

### Umsetzung:

* Textbasierte Interaktion: über Ollama / LangChain mit konfigurierbarem Modell (z. B. llama3.2)
* Bildgenerierung: über Stable Diffusion (lokal oder später remote)
* Promptgenerierung über eigenes Modul Pald2ImagePrompt.py
* Bildbeschreibung und Konsistenzprüfung optional (abschaltbar über Flags)

**Konfigurierbar via:** config.py

### Ausblick:

* Dynamischer Modellwechsel per UI
* Nutzung kommerzieller APIs (z. B. GPT-4 Vision) optional möglich

## 💾 Persistenzstrategie

**Zweck:** Nachvollziehbare, strukturierte Speicherung aller Daten und Zwischenschritte

### Speicherorte:

* **PostgreSQL**: zentrale Speicherung von
  + Studierendendaten
  + Zustimmungen & Entscheidungen
  + PALDs
  + Chatverläufen
* **MinIO** oder lokales Filesystem: Speicherung von Bilddateien (generated\_images/local/)

### Besonderheit:

* Trennung von strukturierten Daten (DB) und unstrukturierten Assets (Bilder)

## „Prompt-Audit“ (Persistenz & Logging)

### 5.1 Zweck

Sicherstellen, dass **jede** Interaktion mit LLM/Bildmodellen vollständig und revisionssicher nachvollziehbar ist (Forschung, Debugging, Compliance).

### 5.2 Datenmodell (Pflichtfelder je Eintrag)

* prompt\_id (UUID), parent\_prompt\_id (UUID|NULL)
* timestamp\_in, timestamp\_out (UTC ISO-8601)
* actor (user|system|assistant)
* channel (chat|image)
* model (z. B. llama3.1:8b, sd-1.5)
* params (JSON: temperature, max\_tokens, guidance\_scale, steps …)
* input\_text (TEXT|NULL), input\_media\_ref (Pfad/URL|NULL)
* output\_text (TEXT|NULL), output\_media\_ref (Pfad/URL|NULL)
* token\_usage (JSON: prompt, completion, total|NULL)
* latency\_ms (INT)
* status (ok|error), error\_msg (TEXT|NULL)
* session\_id (UUID) / user\_id (UUID|Pseudonym)

### 5.3 Prozessregeln

* R-PA-01: **Write-ahead**: Datensatz wird **vor** Aufruf initialisiert, nach Antwort finalisiert.
* R-PA-02: **Unveränderlichkeit**: Updates nur für status/latency/outputs; keine Löschung ohne Datenschutz-Ticket.
* R-PA-03: **Verknüpfung**: parent\_prompt\_id bildet Kette (z. B. Feedback-Runden).
* R-PA-04: **Konfigurierbarkeit**: Schalter SAVE\_LLM\_LOGS in config.py (default: ON).
* R-PA-05: **Export**: Admin-Export (CSV/JSON) filterbar nach Zeitraum, user\_id, model.

### 5.4 Messgrößen (Verweis auf Q-S01)

* Vollständigkeit ≥99 %/Monat, Konsistenzprüfung (NOT NULL-Pflicht), Alarme bei Lücken.

## 📊 Analyse & Reporting (vorgesehen)

**Zweck:** Nachvollziehbarkeit der Interaktionen, Optimierung der Personalisierung

### Umsetzung (geplant):

* Zugriff auf gespeicherte PALD-Verläufe für Forschende
* Exportfunktionen für Feedback und Bildgeneration
* Basis für Evaluation: Typisierung von GITTEs, Clusterbildung

## ⚙️ Modularität & Erweiterbarkeit

**Zweck:** Einfache Integration neuer Funktionen und Technologien

### Maßnahmen:

* Strikte Einhaltung der Schichtenarchitektur
* Trennung von UI, Logik, Services und Speicher
* Erweiterbare Parser-/Promptlogik (z. B. neue PALD-Ebenen)
* Klare JSON-Formate zur Kommunikation

## 🌐 Internationalisierung & Barrierefreiheit (vorgesehen)

**Zweck:** Systemnutzung durch diverse Nutzergruppen

### Umsetzung (geplant):

* Mehrsprachige UI-Texte (via config.py)
* Barrierefreie Darstellung gemäß WCAG
* Text-to-Speech und Spracheingabe in zukünftigen Versionen

# Architekturentscheidungen

In diesem Kapitel werden zentrale und explizite Architekturentscheidungen dokumentiert, die während der Entwicklung des GITTE-Systems getroffen wurden. Jede Entscheidung ist mit ihrer Begründung, den betrachteten Alternativen und möglichen Auswirkungen versehen.

## ADR-001 — Lokale KI-Modelle statt externer SaaS

Status: Accepted • Datum: 10.08.2025

### Entscheidung

LLM (z. B. Ollama/HTTP) und Bildmodelle (Diffusers/Stable Diffusion) laufen lokal/on‑prem (GPU). Keine Übermittlung personenbezogener Daten an Drittanbieter.

### Begründung

DSGVO/Datensouveränität, Kostenkontrolle, Offline‑Betrieb; konsistente Debug‑Möglichkeiten (Prompt/Seed/Params).

### Alternativen

A) Cloud‑LLM/Bild‑APIs (schnell, aber datenschutzkritisch); B) Hybride Lösung (mehr Betriebsaufwand).

### Konsequenzen

+ Datenschutz, + Kostenkappe, + deterministischere Debugbarkeit; – GPU‑Ressourcen & Modellpflege lokal.

### Implementierung/Impact

- `config.py`: `LLM\_PROVIDER="ollama"`, Modell‑Mapping; `SD\_MODEL\_NAME` für Diffusers.

- Services `LLM\_API`, `Pic\_API` nutzen lokale Endpunkte/Runtime (CUDA).

### Test/Verifikation

Smoke‑Tests: Prompt→Antwort, Bild ≤ 30 s; Netzwerkmonitoring: kein externer Traffic.

## ADR-002 — Strikte 4‑Layer‑Architektur

Status: Accepted • Datum: 10.08.2025

### Entscheidung

UI (Streamlit) ↔ Logic ↔ Service ↔ Data; keinerlei Cross‑Layer‑Zugriffe.

### Begründung

Wartbarkeit, klare Verantwortlichkeiten, Testbarkeit, optionale spätere Services‑Auslagerung.

### Alternativen

A) 3‑Layer (Service+Data vermischt), B) monolithischer Streamlit‑Code, C) klassische MVC.

### Konsequenzen

+ Austauschbarkeit, + Tests, – etwas mehr Boilerplate.

### Implementierung/Impact

- UI: reine Darstellung/Events (keine DB/HTTP).

- Logic: Prozesssteuerung (Consent‑Gate, Routing, Feedback‑Loops).

- Service: Adapter (LLM, SD, Prompt/Chat/DB‑Services).

- Data: CRUD + Schema‑Checks.

### Test/Verifikation

Unit‑Tests je Layer (Service mit Mocks), Integrationsfluss UI→DB.

## ADR-003 — Prompt‑Audit als Write‑Ahead‑Log

Status: Accepted • Datum: 10.08.2025

### Entscheidung

Jede LLM/Bild‑Interaktion wird vor dem Call initialisiert und nach Antwort finalisiert (input/output, model, params, token\_usage, Latenz, parent\_id).

### Begründung

Forschung, Debugging, Compliance; vollständige Nachvollziehbarkeit.

### Alternativen

A) Nur Fehler loggen; B) Sampling.

### Konsequenzen

+ Transparenz, – mehr Storage.

### Implementierung/Impact

- `config.py`: `SAVE\_LLM\_LOGS=True`.

- `prompt\_service.save\_prompt(...)` → `data.llm\_chat\_storage.store(...)`.

- Parent‑Child‑Ketten mittels `parent\_prompt\_id`.

### Test/Verifikation

Qualitätsziel Q‑S01: Vollständigkeit ≥ 99 %/Monat; Pflichtfelder geprüft.

## ADR-004 — Single Source of Truth in config.py + Dependency Injection

Status: Accepted • Datum: 10.08.2025

### Entscheidung

Alle Flags/Model‑Mappings/Resource‑Texte zentral in `config.py`; Services erhalten Konfiguration per Konstruktor (DI).

### Begründung

Änderbarkeit ohne Code‑Eingriffe; reproduzierbare Umgebungen; i18n‑Fähigkeit.

### Alternativen

Implizite Globals/ENV‑Streuung; harte Strings im Code.

### Konsequenzen

+ Transparenz, + Testbarkeit (Config‑Doubles), – einmaliger DI‑Aufwand.

### Implementierung/Impact

Helper `t("key")` im UI; `FEATURE\_FLAGS`/`LLM\_MODELS`/i18n‑Ressourcen in `config.py`; Services werden mit `cfg` instanziiert.

### Test/Verifikation

Unit‑Tests injizieren Test‑Configs; kein direkter Tiefen‑Import auf `config`.

## ADR-005 — PostgreSQL als Primärspeicher + Migrationen mit Alembic

Status: Proposed • Datum: 10.08.2025

### Entscheidung

Relationale Persistenz (PostgreSQL) mit schema‑versionierten Migrationen (Alembic).

### Begründung

ACID, klare Foreign‑Keys (z. B. `pald\_id`), reproduzierbare Deployments.

### Alternativen

A) Roh‑SQL ohne Migrationen; B) sofortiger ORM‑Wechsel.

### Konsequenzen

+ Stabilität bei Schema‑Änderungen, – Setup‑Aufwand.

### Implementierung/Impact

Alembic‑Baseline; Migrationsskripte für `students`, `palds`, `llm\_chats`, `chat\_messages` …

### Test/Verifikation

CI führt `alembic upgrade head` gegen Test‑DB aus.

## ADR-006 — Consent‑Gate vor Datenverarbeitung

Status: Accepted • Datum: 10.08.2025

### Entscheidung

Ohne dokumentierte Einwilligung kein Zugriff auf Chat/Bild/Analyse.

### Begründung

DSGVO‑Konformität und Nachweisbarkeit.

### Alternativen

Späterer Consent.

### Konsequenzen

+ Compliance, – zusätzlicher Schritt.

### Implementierung/Impact

Logic prüft `has\_consent(user\_id)`; UI zeigt Rechts‑UI → Service persistiert Consent.

### Test/Verifikation

E2E‑Test: Nutzer ohne Consent wird blockiert (Q‑S04 = 100 %).

## ADR-007 — UUIDv4‑IDs & konsistente Sessions

Status: Accepted • Datum: 10.08.2025

### Entscheidung

`user\_id`, `session\_id`, `prompt\_id`, `image\_id` als UUIDv4; Sessions trennen Chat‑Zyklen; Feedback‑Runden via `parent\_prompt\_id`.

### Begründung

Eindeutigkeit, Pseudonymisierung, Verkettung.

### Alternativen

Autoincrement‑IDs.

### Konsequenzen

+ Skalierbarkeit, – längere Schlüssel.

### Implementierung/Impact

DB: `uuid\_generate\_v4()`/Python‑UUID; Indizes auf `session\_id`.

### Test/Verifikation

DB‑Constraints + FK‑Konsistenztests.

## ADR-008 — Objektspeicher (MinIO) mit lokalem Fallback

Status: Accepted • Datum: 10.08.2025

### Entscheidung

Bild‑Artefakte primär S3‑kompatibel (MinIO). Fehlt `MINIO\_\*`, Speicherung lokal im Dateisystem.

### Begründung

Austauschbare Backends; einfache Dev‑Erfahrung.

### Alternativen

Nur FS; nur S3.

### Konsequenzen

+ Flexibilität, – zwei Pfade testen.

### Implementierung/Impact

`Pic\_API` kapselt Write/Read; Referenzen (Pfad/URL) im Data‑Layer konsistent verwalten.

### Test/Verifikation

Adapter‑Tests für beide Modi; Pfad‑/URL‑Validierung.

## ADR-009 — Fehlerbehandlung, Retries & strukturiertes Logging

Status: Accepted • Datum: 10.08.2025

### Entscheidung

Exponentielles Backoff bei LLM/Bild‑Calls; strukturierte Logs (Level, `request\_id`, Dauer, Size).

### Begründung

Robustheit, Ursachenanalyse.

### Alternativen

Fire‑and‑forget.

### Konsequenzen

+ Stabilität, – minimal höhere Latenzen bei Retries.

### Implementierung/Impact

Gemeinsamer `retry\_call(fn, policy)`; zentrale `logging`‑Config (JSON‑fähig).

### Test/Verifikation

Fault‑Injection‑Tests (Timeouts).

## ADR-010 — Token‑Budget‑Kontrollen & Prompt‑Guards

Status: Accepted • Datum: 10.08.2025

### Entscheidung

`tiktoken`‑basierte Token‑Schätzung vor Call; Limits aus `config.py`; ggf. Auto‑Truncation/Heuristik.

### Begründung

Vermeidbare Abbrüche/Fehler; planbare Latenzen/Kosten.

### Alternativen

Keine Guards.

### Konsequenzen

+ Stabilität, – potenziell gekürzte Kontexte.

### Implementierung/Impact

`prompt\_service.ensure\_budget(prompt, cfg)`; Loghinweis bei Kürzung.

### Test/Verifikation

Unit‑Tests mit synthetisch langen Prompts.

## ADR-011 — Modell‑Routing & Feature‑Flags

Status: Accepted • Datum: 10.08.2025

### Entscheidung

Routing über `config.LLM\_MODELS` und `FEATURE\_FLAGS` (z. B. `USE\_LANGCHAIN=False`).

### Begründung

Experimentieren ohne Risiko für Stable‑Pfad.

### Alternativen

Hartverdrahtete Modelle.

### Konsequenzen

+ Umschaltbarkeit, – Disziplin in Config nötig.

### Implementierung/Impact

Flags/Modelle in `config.py`; Services konsumieren via DI.

### Test/Verifikation

Matrix‑Tests (Flags an/aus).

## ADR-012 — Teststrategie pro Layer + Export‑Verifikation

Status: Accepted • Datum: 10.08.2025

### Entscheidung

Unit‑ und Integrationstests; Metadaten‑Checks statt ‚Golden Image‘; Export‑Validierung.

### Begründung

Determinismus selten; Verträge/Metadaten sind stabiler.

### Alternativen

Snapshot‑Vergleiche.

### Konsequenzen

+ Stabil, – keine Pixel/Token‑Gleichheit.

### Implementierung/Impact

Pytest‑Fixtures (Temp‑DB/Storage), Export‑CSV‑Validator.

### Test/Verifikation

CI‑Pipeline mit Coverage‑Gate.

## ADR-013 — Antwort‑Streaming & Progress‑Feedback (UI)

Status: Proposed • Datum: 10.08.2025

### Entscheidung

Token‑Streaming falls unterstützt; sonst Progress‑Bar/Spinner mit Zwischen‑Logs.

### Begründung

Bessere UX bei langen Läufen.

### Alternativen

Vollständige Antwort am Ende.

### Konsequenzen

+ UX, – etwas komplexere UI‑Logik.

### Implementierung/Impact

UI‑Abstraktion `render\_stream(response\_iter)`; Service liefert Generator.

### Test/Verifikation

Manuelle UX‑Checks; TTFT‑Messung.

## ADR-014 — Sicherheit: Hashing, Rollen, Secrets & Löschkonzept

Status: Accepted • Datum: 10.08.2025

### Entscheidung

`bcrypt`‑Hashing, Role‑Checks im Logic, Secrets via ENV, DSGVO‑Lösch-/Auskunfts‑Funktionen.

### Begründung

DSGVO & Best Practices.

### Alternativen

Schwache Hashes, Secrets im Code.

### Konsequenzen

+ Compliance, – mehr Admin‑Pflege.

### Implementierung/Impact

`users(role)`, `can(action)` Guards; `gdpr\_delete(user\_id)`; Secrets‑Store.

### Test/Verifikation

Pen‑Test‑Checkliste; Rechte‑Unit‑Tests.

## ADR-015 — Internationalisierung über zentrale Texte

Status: Accepted • Datum: 10.08.2025

### Entscheidung

UI‑Texte/Labels konfigurierbar (DE Standard, EN optional) in `config.py`/Resources; Zugriff nur im UI.

### Begründung

Mehrsprachigkeit ohne Code‑Änderungen.

### Alternativen

Hardcoded Strings.

### Konsequenzen

+ Flexibel, – Pflege von Ressourcen.

### Implementierung/Impact

`t("login.title")` Helper im UI; Ressourcen‑Maps in Config/Files.

### Test/Verifikation

Smoke‑Test EN/DE; fehlende Keys melden.

## ADR-016 — Federated Learning (opt‑in)

Status: Proposed • Datum: 10.08.2025

### Entscheidung

Personalisierungsmodelle optional via Federated Learning: Clientseitiges Kurztraining, serverseitige Aggregation; nur aggregierte Updates werden übertragen.

### Begründung

Datenschutz (keine Rohdaten), Nutzung vorhandener Artefakte (PALD), experimentell per Feature‑Flag zuschaltbar.

### Alternativen

Zentrales Training (pseudonymisierte Rohdaten); On‑Device‑Only ohne Aggregation.

### Konsequenzen

+ Privacy‑Preserving, + Skalierbarkeit, – zusätzlicher Aggregationsdienst, – Heterogenität der Clients.

### Implementierung/Impact

`FEATURE\_FLAGS["USE\_FL"] = False` (Default); FL‑Orchestrator als separater Service; opt‑in UI.

### Test/Verifikation

Pilot‑Studie mit Offline‑Validierung; Messgrößen: Teilnahmequote, Modellgüte Δ, Privacy‑Audit.

## ADR-017 — PALD‑Governance & Schema‑Versionierung

Status: Proposed • Datum: 10.08.2025

### Entscheidung

Neue Merkmale werden via Discovery vorgeschlagen, kuratiert (Governance) und als Schema‑Patch mit Versionssprung ausgerollt; Parser und Prompt‑Generator bleiben abwärtskompatibel.

### Begründung

Erweiterbarkeit der Merkmale bei stabiler Verarbeitung; reproduzierbare Analysen.

### Alternativen

Statisch gepflegte Schemas; freie Zusatzfelder ohne Kanonisierung.

### Konsequenzen

+ Evolutionär, + Konsistenz, – Bedarf für Registry/Review.

### Implementierung/Impact

Schema‑Registry (JSON‑Schema Versionen); Migrationslogik im Parser; Versionstempel in PALD‑Datensätzen.

### Test/Verifikation

Schema‑Validierung in CI; Backward‑Compatibility‑Tests (alt→neu).

# Qualitätsanforderungen

In diesem Kapitel werden die priorisierten Qualitätsanforderungen beschrieben, die für das GITTE-System relevant sind. Diese leiten sich aus den funktionalen und nicht-funktionalen Zielen des Projekts sowie aus Stakeholder-Erwartungen und regulatorischen Rahmenbedingungen (z. B. DSGVO) ab.

**Weiterführende Informationen**

Siehe [Qualitätsanforderungen](https://docs.arc42.org/section-10/) in der online-Dokumentation (auf Englisch!).

## Qualitätsbaum

Qualitätsanforderungen

├── Funktionalität

│ ├── Personalisierung

│ ├── Rückverfolgbarkeit

│ └── Konsistenz

├── Zuverlässigkeit

│ ├── Verfügbarkeit

│ ├── Fehlertoleranz

│ └── Wiederherstellbarkeit

├── Benutzbarkeit

│ ├── Intuitive UI

│ ├── Barrierefreiheit

│ └── Rückmeldung im Dialog

├── Effizienz

│ ├── Antwortzeiten

│ ├── Bildgenerierung ≤ 30s

│ └── Ressourcennutzung (lokal)

├── Änderbarkeit

│ ├── Austauschbarkeit von Modellen

│ ├── Modularer Code

│ └── Erweiterbarkeit der PALD-Struktur

├── Übertragbarkeit

│ ├── Deployment lokal & Cloud

│ └── Konfigurierbare APIs

└── Sicherheit

├── Datenschutz (DSGVO)

├── Zugriffskontrolle

└── ProtokollierungQualitätsanforderungen

├── Funktionalität

│ ├── Personalisierung

│ ├── Rückverfolgbarkeit

│ └── Konsistenz

├── Zuverlässigkeit

│ ├── Verfügbarkeit

│ ├── Fehlertoleranz

│ └── Wiederherstellbarkeit

├── Benutzbarkeit

│ ├── Intuitive UI

│ ├── Barrierefreiheit

│ └── Rückmeldung im Dialog

├── Effizienz

│ ├── Antwortzeiten

│ ├── Bildgenerierung ≤ 30s

│ └── Ressourcennutzung (lokal)

├── Änderbarkeit

│ ├── Austauschbarkeit von Modellen

│ ├── Modularer Code

│ └── Erweiterbarkeit der PALD-Struktur

├── Übertragbarkeit

│ ├── Deployment lokal & Cloud

│ └── Konfigurierbare APIs

└── Sicherheit

├── Datenschutz (DSGVO)

├── Zugriffskontrolle

└── Protokollierung

## Qualitätsszenarien

### 3.4 Qualitätsszenarien (messbar)

* **Q-S01 Prompt-Audit**: Für jede Chat-/Bild-Interaktion sind *input, output, model, params, token\_usage, latency, timestamps* vorhanden; **Vollständigkeit ≥99%/Monat**.
* **Q-S02 Performance Chat**: Median Time-to-first-token ≤ **2 s**, 95-Perzentil ≤ **5 s**.
* **Q-S03 Bildgenerierung**: End-to-End ≤ **30 s** (95-Perzentil) bei 512×512.
* **Q-S04 Consent**: Ohne gültigen Consent kein Zugriff auf Chat/Bild; **Blockierungsrate 100%**.
* **Q-S05 Datenschutz**: Löschanforderung wird innerhalb **72 h** vollständig umgesetzt.

### BEschreibungen

| **Nr.** | **Szenario** | **Qualitätsattribut** | **Priorität** | **Beschreibung** |
| --- | --- | --- | --- | --- |
| Q1 | Eine:n Nutzer:in äußert Designwünsche im Chat, die korrekt strukturiert, gespeichert und visualisiert werden. | Funktionalität: Personalisierung | 🟢 Hoch | Die vom Nutzer erzeugte PALD-Struktur führt zu einem angepassten GITTE-Bild und nachvollziehbarer Kommunikation. |
| Q2 | Die GITTE-Instanz ist lokal verfügbar und antwortet innerhalb von 1–2 Sekunden mit Text, Bild ≤ 30 Sekunden. | Effizienz: Antwortzeit | 🟢 Hoch | Nutzer:innen sollen nicht durch Wartezeiten im Lernfluss gestört werden. |
| Q3 | Nach Feedback äußert der:die Nutzer:in neue Wünsche, das System verarbeitet diese und speichert alle Versionen nachvollziehbar. | Funktionalität: Rückverfolgbarkeit | 🟢 Hoch | Die GITTE-Historie (inkl. Korrekturschleifen) bleibt rekonstruierbar. |
| Q4 | Ein Systemupdate (z. B. neues LLM-Modell) soll erfolgen, ohne das gesamte System neu zu schreiben. | Änderbarkeit: Modulaustausch | 🟡 Mittel | Dank Kapselung über Service-Layer können Modelle wie Llama oder Mistral einfach getauscht werden. |
| Q5 | Eine:n Nutzer:in verweigert eine Einwilligung – die Verarbeitung wird korrekt abgebrochen und dokumentiert. | Sicherheit: Datenschutz | 🟢 Hoch | Die rechtlichen Anforderungen der DSGVO werden strikt umgesetzt. |
| Q6 | Ein:e Forscher:in möchte alle anonymisierten Chatverläufe und PALD-Daten exportieren. | Übertragbarkeit: Analyse | 🟡 Mittel | Forschungsauswertung wird durch strukturierte Datenhaltung unterstützt. |
| Q7 | Der Zugriff erfolgt von einem Tablet mit Screenreader – die Inhalte sind semantisch korrekt und barrierefrei. | Benutzbarkeit: Barrierefreiheit | 🟠 Niedrig (aber relevant) | Zukünftig soll WCAG-Kompatibilität gesichert sein. |

## ⚙️ Technische Maßnahmen zur Qualitätssicherung

 **CI/Lint**: black, flake8, isort.

 **Tests**: Unit-Tests je Layer (Service mit Mocks), DB-Tests gegen Test-Schema.

 **Metriken**: Antwortzeit Chat, Bild-Durchlaufzeit, Fehlerquote, Token-Usage.

| **Kategorie** | **Maßnahme** |
| --- | --- |
| Testbarkeit | Modultests für Logic- und Service-Komponenten geplant |
| Nachvollziehbarkeit | Versionierte Speicherung aller PALD-Schritte mit Parent-Verknüpfung |
| Wiederverwendbarkeit | Standardisierte Prompt-Extraktion & Speicherung über zentrale Parser |
| Modularität | Klar getrennte Pfade & Imports für jede Schicht |
| Performanz | Caching (st.cache\_data), Latenz-Monitoring geplant |
| Konfigurierbarkeit | config.py mit zentraler Steuerung von Modellwahl, Texten, Flags |

# Risiken und technische Schulden

Dieses Kapitel dokumentiert bekannte technische Risiken, Architekturgrenzen sowie bewusste technische Schulden, die im Projektverlauf eingegangen wurden, um Entwicklungszeit zu sparen oder Funktionalität zu priorisieren. Sie sollten regelmäßig überprüft und bei Bedarf abgebaut werden.

## 🛑 Technische Risiken

| **Nr.** | **Risiko** | **Beschreibung** | **Mögliche Auswirkung** | **Gegenmaßnahme** |
| --- | --- | --- | --- | --- |
| R1 | **LLM-Reaktionszeit** | Die lokale Nutzung von LLMs (Ollama) kann bei großen Modellen zu langen Antwortzeiten führen. | Nutzer:innen verlieren Geduld oder brechen ab. | Modelle optimieren (Quantisierung), Konsistenzprüfungen deaktivieren (Flag), ggf. kleineres Modell verwenden. |
| R2 | **GPU-Verfügbarkeit** | Die Bildgenerierung via Stable Diffusion erfordert eine leistungsfähige GPU. | System funktioniert nicht wie erwartet ohne CUDA. | Fallback auf Dummy-Bilder implementiert, Hinweistext für Nutzer:innen erforderlich. |
| R3 | **Konsistenzprüfung unzuverlässig** | Die automatische Bewertung der Bild-Text-Konsistenz ist noch rudimentär. | Schleifen wiederholen sich unnötig oder falsche Ergebnisse. | Mechanismus deaktivierbar über Flag ENABLE\_CONSISTENCY\_CHECK, Logging zur späteren Optimierung. |
| R4 | **Datei-/Bildspeicher wächst unkontrolliert** | Lokale Speicherung in generated\_images/ oder MinIO kann unbemerkt zu Speicherüberlauf führen. | Systemausfälle, schwer nachvollziehbare Fehler. | Regelmäßige Bereinigung, automatische Löschung veralteter Bilder (Cronjob/Batch). |
| R5 | **Datenbankstruktur fixiert** | PALD-Tabellenstruktur ist aktuell nicht migrationsfähig (keine Versionierung). | Erweiterungen am Modell erfordern manuelle DB-Eingriffe. | Einführung von Schema-Migrationstools wie Alembic empfohlen. |
| R6 | **Mehrsprachigkeit nicht vorbereitet** | Alle UI-Texte sind derzeit nur in Englisch/Deutsch angelegt, aber nicht dynamisch übersetzbar. | Internationalisierung nur durch Codeänderung möglich. | Einführung einer Übersetzungstabelle oder i18n-Modul (z. B. gettext). |

## 🧾 Technische Schulden

| **Nr.** | **Bereich** | **Beschreibung** | **Aktueller Workaround** | **Empfehlung** |
| --- | --- | --- | --- | --- |
| TS1 | **Auth-System** | Nur einfache Benutzerprüfung über Klartext-Passwörter oder Dummy-Login. | auth.py nutzt Platzhalterfunktion | Passwort-Hashing mit bcrypt, später OAuth2 einführen |
| TS2 | **Datenvalidierung** | Keine zentralen Schema-Validierungen für pald\_json beim Speichern. | Vertrauen in Parserstruktur | JSON Schema + Validierungsfunktion integrieren |
| TS3 | **Fehlendes Logging** | Keine einheitliche Protokollierung von Fehlern oder Nutzerinteraktionen. | Print-Statements | Logging via logging-Modul einführen, mit Loglevels |
| TS4 | **Unit Tests fehlen** | Bisher keine automatisierten Tests für Logic- oder Service-Komponenten. | Manuelle Tests über UI oder Integrationstest | Testabdeckung mit pytest priorisieren |
| TS5 | **Hardcodierte Pfade** | Einige Pfade (z. B. generated\_images) sind im Code verankert. | Nutzung von os.path zur Abmilderung | Konfigurierbar über .env oder zentrale config.py |
| TS6 | **Modelldefinition im Code** | Modellnamen wie llama3.2 sind mehrfach im Code verteilt | Teilweise Verwendung von config.py | Vollständige Zentralisierung aller Modellnamen in config.py sicherstellen |
| TS7 | **UI-Caching inkonsistent** | @st.cache\_data wird selektiv verwendet | Teilweise redundante Berechnungen | Einheitliches Caching-Konzept mit Invalidation |

## 📌 Umgang mit Risiken und Schulden

* **Transparenz**: Alle Risiken/Schulden werden dokumentiert und kommuniziert.
* **Priorisierung**: Technische Schulden mit hohem Einfluss (z. B. Auth, Performance) werden bevorzugt abgebaut.
* **Sprint-Retention**: Je nach Ressourcen sollte in jedem Sprint eine Schuldenposition bearbeitet werden.
* **Test- und Loggingstrategie**: wird gesondert geplant (empfohlen bis Herbst 2025)

# Anhang

## Glossar

|  |  |
| --- | --- |
| **Begriff / Abkürzung** | **Definition / Bedeutung** |
| **GITTE** | *Great Individual Tutor Embodiment* – ein personalisierter, visuell gestalteter Lernassistent |
| **ILA** | *Intelligent Learning Assistant* – allgemeiner Begriff für KI-gestützte Lernbegleiter |
| **PALD** | *Pedagogical Agent Level of Design* – Strukturmodell zur Beschreibung von GITTE-Merkmalen |
| **PALD-Ebene** | Ebene im Designmodell: Global (Typ), Mittel (Realismus, Darstellung), Detail (Alter etc.) |
| **pald\_id** | Eindeutiger Datenbankschlüssel für einen PALD-Schritt oder eine Zwischenstruktur |
| **parent\_id** | Referenz auf den übergeordneten PALD-Schritt, für Rückverfolgbarkeit und Chronologie |
| **student\_id** | Pseudonymisierte Kennung einer teilnehmenden Person |
| **Prompt** | Texteingabe zur Steuerung eines KI-Modells (Text, Bild etc.) |
| **Stable Diffusion** | Text-zu-Bild-Generator, lokal eingebunden über HuggingFace Diffusers |
| **Ollama** | Lokaler Server zur Ausführung von LLMs (Llama3, Mistral, etc.) über HTTP |
| **LLM** | *Large Language Model* – Sprachmodell für die Verarbeitung natürlicher Sprache |
| **LLMWrapper** | Python-Klasse zur Modellabstraktion (LangChain oder direkte API-Nutzung) |
| **LangChain** | Python-Framework zur Modularisierung und Verkettung von LLM-Komponenten |
| **MinIO** | S3-kompatibler Objektspeicher zur Speicherung von generierten Bildern |
| **Studi** | Session-spezifische Variable mit Name und Angaben der Nutzenden |
| **st.session\_state** | Globale Streamlit-Speicherstruktur für Ablaufsteuerung und Nutzerinformationen |
| **Chatverlauf** | Zeitlich geordnete Nachrichtenliste zwischen GITTE und Nutzer:in |
| **Feedback** | Textlicher Kommentar eines/einer Studierenden zur Anpassung eines GITTE-Designs |
| **Konsistenzprüfung** | Abgleich, ob ein generiertes Bild zum zuvor formulierten Textbeschreibung passt |
| **Consent / Einwilligung** | Zustimmung zur Datennutzung (Datenschutz, KI, Studie) durch den/die Teilnehmende |
| **legal\_ui / legal\_logic** | Module zur Darstellung und Auswertung der Einwilligungen |
| **study\_participation\_ui** | Modul zur Erfassung des Pseudonyms und Aktivierung des Studieneinstiegs |
| **LLM\_API / Pic\_API** | Service-Module zur Kommunikation mit den Text- und Bildgeneratoren |
| **Chat\_service** | Service-Modul zur Speicherung von Chatnachrichten, Rollen und PALD-Verknüpfung |
| **data\_collector** | Name der verwendeten PostgreSQL-Datenbank |
| **config.py** | Zentrale Konfigurationsdatei mit Flags, Modellnamen und UI-Texten |
| **Fallback-Image** | Platzhalterbild für nicht erzeugte oder fehlerhafte GITTE-Bilder |

## Use-Cases (, MoSCoW, prüfbar)

Legende: M=MUSS, S=SOLL, K=KANN. Jeder Use-Case ist kurz, messbar und testbar beschrieben.

### UC-01 (M) Pseudonym/Teilnahmeschlüssel erzeugen

Akteur: Nutzer:in • Vorbed.: App läuft, DB initialisiert • Nachbed.: student\_id in Session  
Ablauf: Formular → Pseudonym eingeben → Schlüssel generieren → in DB speichern → student\_id in Session.  
Erfolgskriterien: Schlüssel ist eindeutig; Hinweistext angezeigt; Weiterleitung zum Consent.

### UC-02 (M) Rechtliche Einwilligungen erfassen (Consent-Gate)

Akteur: Nutzer:in • Vorbed.: student\_id vorhanden • Nachbed.: Consent(s) persistiert  
Ablauf: Rechtstexte anzeigen → Checkboxen (Datenschutz/Studie/AI) → Bestätigen/Abbruch → Persistenz.  
Erfolgskriterien: Ohne vollständigen Consent kein Zugang zu Chat/Bild; Audit-Eintrag erstellt.

### UC-03 (S) Registrierung/Login/Logout

Akteur: Nutzer:in/Administrator:in • Vorbed.: Login-Tabellen vorhanden  
Ablauf: Registrieren (PW-Doppelcheck, Hash) → Login (Prüfung Hash) → Logout.  
Erfolgskriterien: Passwort gehasht (bcrypt); Fehlversuche geloggt; Rolle gesetzt.

### UC-04 (M) Gespräch initialisieren (Greeting + Initial-PALD)

Akteur: Nutzer:in • Vorbed.: Consent ok • Nachbed.: Begrüßung + erster PALD-Eintrag gespeichert  
Ablauf: Name abfragen → init\_conversation() → Greeting in Chat + PALD(type=initial) persistent.  
Erfolgskriterien: neue pald\_id erzeugt; Greeting im Verlauf sichtbar.

### UC-05 (M) Erste Nutzereingabe → Text-zu-PALD (JSON)

Akteur: Nutzer:in • Vorbed.: UC-04 • Nachbed.: PALD(type=initial) mit extrahierten Merkmalen  
Ablauf: User-Text → JSON-LLM extrahiert Merkmale → PALD speichern → Prompt-Audit speichern.  
Erfolgskriterien: valide JSON-Struktur (Schema); Fehlversuche werden retried/geloggt.

### UC-06 (M) Bild generieren aus PALD (Text2Image)

Akteur: System • Vorbed.: UC-05 • Nachbed.: Bild gespeichert + im Chat angezeigt  
Ablauf: PALD → Prompt komprimieren/formatieren → SD-Pipeline → Bild speichern (MinIO oder lokal) → Chat-Nachricht (type=image).  
Erfolgskriterien: 512×512 oder konfiguriert; Speicherort protokolliert; Fallback-Bild bei Fehler.

### UC-07 (M) Feedback-Schleife (MAX\_FEEDBACK\_ROUNDS)

Akteur: Nutzer:in/System • Vorbed.: UC-06 • Nachbed.: Feedback-PALDs, ggf. neue Bilder  
Ablauf: Nutzerfeedback → PALD(type=feedback) → optional neue Bildgenerierung → Runden zählen.  
Erfolgskriterien: Rundenzähler korrekt; parent-child-Link zwischen Prompts/PALDs; Abbruch bei Limit.

### UC-08 (S) Session-Ende bestätigen

Akteur: Nutzer:in • Vorbed.: Ende Feedback-Loop • Nachbed.: Rückkehr zum Einstieg  
Ablauf: Abschlussnachricht → explizite Bestätigung (Button/Enter) → Aufräumen Session → Rückkehr.  
Erfolgskriterien: Nutzer sieht „Danke“; kein sofortiges Umschalten; Zustände zurückgesetzt.

### UC-09 (M) Prompt-Audit speichern (Write-Ahead + Finalize)

Akteur: System • Vorbed.: bei jedem LLM/Bild-Call • Nachbed.: vollständiger Audit-Datensatz  
Ablauf: Datensatz vor Call initialisieren (input, params, model, parent\_id) → nach Antwort finalisieren (output, usage, latency, status).  
Erfolgskriterien: Vollständigkeit ≥99 %/Monat; Parent-Ketten rekonstruierbar.

### UC-10 (S) PALD vergleichen & Feldabdeckung zählen

Akteur: Admin/Forschung • Vorbed.: mind. zwei PALDs • Nachbed.: Vergleichsreport  
Ablauf: zwei pald\_id wählen → Vergleich (Differenzen, Konflikte) → total fields vs. filled fields.  
Erfolgskriterien: deterministischer Report; Exportierbar (CSV/JSON).

### UC-11 (S) Chat-/Studien-Daten exportieren (Admin)

Akteur: Admin • Vorbed.: Rolle=Admin • Nachbed.: Exportdatei  
Ablauf: Filter (Zeitraum, user, model) → Export generieren → Download.  
Erfolgskriterien: UTF-8 CSV/JSON; Felder dokumentiert; Datenschutzfilter anwendbar.

### UC-12 (K) Bildbeschreibung (Image→Text)

Akteur: Nutzer:in • Vorbed.: Bild hochgeladen • Nachbed.: Textbeschreibung gespeichert/angezeigt  
Ablauf: Bild → multimodales LLM → Beschreibung → Audit.  
Erfolgskriterien: Limit Dateigröße geprüft; Fehlerfälle geloggt.

### UC-13 (S) Passwort ändern / Nutzer verwalten (Admin)

Akteur: Nutzer:in/Admin • Ablauf: Passwort ändern; Admin: User löschen/deaktivieren.  
Erfolgskriterien: Hash aktualisiert; Historie/Audit vorhanden.

### UC-14 (S) Datenschutz-Löschung anstoßen (DSGVO)

Akteur: Nutzer:in/Admin • Ablauf: Löschanfrage → Verifikation → Löschung aller referenzierten Datensätze.  
Erfolgskriterien: Frist ≤72 h; Nachweis/Protokoll; nicht rückgängig ohne Backup-Recovery.

## Traceability

