# Software‑Umsetzungsplan – Projektname

**Titel**: Projektname – Vollständiger Umsetzungsplan

**Untertitel**: Durchgängige Umsetzung von der Idee bis zum Betrieb (v1.0)

**Version/Datum**: 1.0 – 30.09.2025

**Autor:in**: Projektleitung (PMI/PRINCE2/Agile) & technischer Architekt

## 0 Deckblatt

| Element | Inhalt |
| --- | --- |
| Projekttitel | Projektname |
| Untertitel | Vollständiger Umsetzungsplan von der Idee bis zum Betrieb |
| Version | 1.0 |
| Datum | 30.09.2025 |
| Autor:in | Weltklasse‑Projektmanager:in & Senior Software‑Architekt:in |

## 0.1 Executive Summary

1. **Zielsetzung** – Das Projekt verfolgt das Ziel, eine moderne Web‑Applikation bereitzustellen, die soziale Interaktion und Informationsvermittlung ermöglicht. Die Plattform soll barrierearm, sicher und skalierbar sein. Die messbaren Key Performance Indicators (KPIs) umfassen **Verfügbarkeit ≥ 99,5 %**, **Ladezeit ≤ 1,5 s**, **User‑Growth > 20 %/Jahr** und **Fehlerquote < 1 %**.
2. **Scope & Zeitrahmen** – Der Scope reicht von der Produktidee über die Architektur‑ und Release‑Planung, Implementierung, Test, Security‑Hardening und Deployments bis zum Betrieb. Der geplante Zeitraum beträgt **12 Monate**, beginnend am 01.10.2025 und endend am 30.09.2026.
3. **Budget** – Das Projektbudget (geschätzt) liegt bei **1,2 Mio. €**, aufgeteilt auf Entwicklung, QA, DevOps, Infrastruktur und Sicherheitsmaßnahmen. Ein Puffer von 15 % ist für unvorhergesehene Kosten eingeplant.
4. **Team & Rollen** – Das Kernteam besteht aus **Product Owner**, **Tech Lead**, **4 Entwickler:innen**, **2 QA‑Engineers**, **DevOps/SRE**, **Security Engineer**, **UX‑Designer:in**, **Data Engineer** und **Stakeholder‑Vertretung**. Die RACI‑Matrix (Kapitel 4) regelt Verantwortlichkeiten.
5. **Delivery‑Modell** – Es wird ein **Scrum‑Modell** mit zweiwöchigen Sprints verwendet. Laut offizieller Scrum‑Guideline (Version 2020, unverändert bis 2025【606628840459431†L96-L107】) existiert ein gemeinsames Scrum‑Team mit Product Owner, Scrum Master und Entwicklern. Die Sprint‑Artefakte (Sprint Goal, Sprint Backlog, Increment) enthalten klare Commitments【606628840459431†L124-L156】.
6. **Tech‑Stack** – Frontend: **React 18/TypeScript**, Backend: **Python 3.11 (FastAPI)**, Datenbank: **PostgreSQL 15**, Authentifizierung via **JWT/OAuth 2.1**, Containerisierung mit **Docker** und Orchestrierung via **Kubernetes 1.30**, Infrastructure‑as‑Code (IaC) per **Terraform 1.6**, CI/CD über **GitHub Actions**, Observability mit **Prometheus/Grafana**, Logging per **Loki/ELK**. Die Auswahl berücksichtigt den aktuellen Trend zu containerisierten, cloud‑nativen Anwendungen【806704564751728†L391-L437】.
7. **Security‑ und Compliance‑Rahmen** – Supply‑Chain‑Security wird nach **SLSA** (Supply Chain Levels for Software Artifacts) Level 2/3 etabliert. SLSA definiert vier Reifestufen für Source‑Integrität und Build‑Provenance【94236474580611†L619-L628】. Es werden SBOMs erstellt und verifizierbare Build‑Attestationen generiert. DSGVO‑Compliance und ISO/IEC 27001 – die Norm für Informationssicherheits‑Managementsysteme【663759207839439†L140-L152】 – werden eingehalten.
8. **Hauptrisiken** – Hauptgefahren sind (a) **Veraltete Abhängigkeiten/Supply‑Chain‑Risiken**, wie in der Voranalyse (28 offene Dependabot‑PRs) gesehen【19†L20-L28】; (b) **Instabile Builds** aufgrund fehlgeschlagener TypeScript‑Compiles und Tests【19†L35-L39】; (c) **Fehlende Daten‑ und Zugriffskontrolle**; (d) **Unklare Anforderungen** und Scope‑Creep. Maßnahmen sind in der Risikomatrix (Kapitel 12) hinterlegt.
9. **Chancen** – Durch Modernisierung (Containerisierung, automatisierte Tests, Dependency‑Management, SBOMs, Performance‑Budgeting【1†L19-L22】) kann das Projekt die Qualität erhöhen und technische Schulden abbauen. Die Etablierung von DevEx‑Standards (Editorconfig, Pre‑Commit‑Hooks etc.) führt zu höherer Entwickler‑Effizienz【14†L35-L43】.
10. **Nächste Schritte** – (1) Kickoff‑Meeting & Scope‑Finalisierung, (2) Detail‑Anforderungsaufnahme mit Stakeholdern, (3) Einrichtung von Repository & CI/CD‑Grundgerüst, (4) Threat‑Modeling & Architektur‑Design, (5) Erstellen der ersten Epics/User Stories. Ein **Action Sheet** mit Top‑Prioritäten für die nächsten 14 Tage ist am Ende des Plans enthalten.

## 0.2 Inhaltsverzeichnis (automatisch)

1. Deckblatt & Executive Summary
2. Einleitung & Business‑Kontext
3. Produkt‑ & Anforderungen
4. Architektur & Design
5. Delivery‑Setup & Governance
6. DevEx & Code‑Standards
7. CI/CD & Environments
8. Sicherheit & Datenschutz
9. Infrastruktur & Betrieb
10. Qualitätssicherung & Test
11. Roadmap, Planung & Aufwand
12. Budget, Kosten & FinOps
13. Risiken & Chancen
14. Kommunikation & Enablement
15. Betrieb & Übergabe
16. Recherche & Evidence
17. Anhänge

## 1 Einleitung & Business‑Kontext

### 1.1 Ziele (SMART) & Nichtziele

| Kategorie | Ziel (SMART formuliert) | Nichtziele |
| --- | --- | --- |
| **Produkt** | Bis **30.09.2026** steht eine funktionsfähige Web‑Plattform mit responsive UI bereit. Die Plattform ermöglicht Nutzer:innen die Registrierung, Konsumption und Einreichung von Inhalten. | Kein Aufbau eines nativen Mobile‑Clients im ersten Release. |
| **Performance** | Seiten laden < **1,5 s** (p95) mit Zielscore **≥ 90** im Lighthouse‑Performance‑Budget【1†L19-L22】. | Keine Optimierung für sehr langsame 2G‑Verbindungen. |
| **Sicherheit** | Erfüllung von **OWASP Top 10** & SLSA Level 2 in den ersten sechs Monaten【94236474580611†L619-L628】. SBOMs werden für alle Releases erstellt【57†L158-L165】. | Kein Ausschluss aller Drittanbieter‑Bibliotheken – Auswahl basiert auf Sicherheits‑Bewertung. |
| **Compliance** | DSGVO‑Konformität: minimale Datenspeicherung, vollständige Dokumentation der Datenverarbeitungsprozesse【50†L151-L159】. ISO/IEC 27001‑Konformität anstreben (Audit bis Q4 2026)【663759207839439†L140-L152】. | Keine Zertifizierung nach PCI‑DSS, da keine Kreditkartenzahlungen verarbeitet werden. |
| **Business** | Steigerung der aktiven Nutzer:innenanzahl von 0 auf 10 000 innerhalb von 12 Monaten. Monetarisierung über Spendenkampagnen: 50 000 € Jahresziel. | Keine Werbeeinblendungen im Produkt. |
| **Team** | Aufbau eines funktionsfähigen Dev‑Teams innerhalb eines Monats, Onboarding‑Zeit ≤ 14 Tage pro Entwickler:in, durch Onboarding‑Guide & Devcontainer【12†L39-L47】. | Kein Outsourcing der Kernentwicklung. |

### 1.2 Scope (In / Out), Annahmen & Constraints

**Im Scope:**

* Entwicklung einer React‑basierten Web‑Applikation mit serverseitiger API (FastAPI) und PostgreSQL.
* Integration eines Drittanbieter‑CRM (z. B. CiviCRM) für Kontakt‑ und Newsletter‑Management.
* Aufbau einer CI/CD‑Pipeline inkl. SAST/DAST, SBOM‑Generierung und automatischem Deployment auf Staging/Prod.
* Erstellung der notwendigen Dokumentation (API‑Dok, Architektur‑Diagramme, Contributing Guide).
* Implementation von Security‑Maßnahmen (JWT‑Auth, RBAC, Secrets‑Management, Logging/Monitoring【50†L151-L159】).
* Einrichtung eines Observability‑Stacks (Metriken, Logs, Tracing) mit Alerting.
* Erstellung von Tests (Unit‑, Integration‑, E2E‑, Lasttests) und definierte Abnahmekriterien.

**Out of Scope / Nichtziele:**

* Entwicklung nativer Mobile‑Apps.
* Migration bestehender Legacy‑Datenbanken anderer Projekte (nur definierte Imports).
* Aufbau eines On‑Premise‑Rechenzentrums; es wird Cloud‑Hosting genutzt.
* Aufbau eines eigenen Payment‑Systems (Integration externer Zahlungsanbieter erst in späteren Releases).

**Annahmen & Constraints:**

* Externe Dienste (CiviCRM, Payment) stehen stabil zur Verfügung.
* Teammitglieder stehen zu 100 % zur Verfügung; Urlaubspuffer wird berücksichtigt.
* Budgetfreigaben erfolgen rechtzeitig.
* Technologieentscheidungen basieren auf vorangegangener Forschung und modernen Best Practices【94236474580611†L619-L628】【806704564751728†L391-L437】.

### 1.3 Erfolgsmessung (Produkt‑ & Delivery‑KPIs)

| KPI | Definition | Zielwert | Messpunkt |
| --- | --- | --- | --- |
| **Verfügbarkeit** | Prozentsatz der Zeit, in der die Plattform fehlerfrei nutzbar ist | ≥ 99,5 % | Monitoring via SLOs |
| **Ladezeit** | p95‑Ladezeit der Startseite | ≤ 1,5 s | Lighthouse‑Berichte【1†L19-L22】 |
| **Conversion Rate** | Anteil registrierter Besucher | > 30 % | Analytics |
| **Security‑Incidents** | Meldungen kritischer Sicherheitsverletzungen pro Quartal | 0 | Security‑Monitoring【50†L151-L159】 |
| **Technical Debt** | Anzahl offener P1‑Issues & kritischer CVEs | ≤ 5 | Backlog (Dependabot)【19†L20-L28】 |
| **Delivery Predictability** | Ratio „Committed Story Points vs. Abgeschlossene Story Points“ | ≥ 90 % | Sprint Reports |
| **Testabdeckung** | Anteil durch automatisierte Tests abgedeckter Codezeilen | ≥ 80 % | CI‑Coverage Reports |

## 2 Produkt & Anforderungen

### 2.1 Nutzer‑ / Stakeholder‑Analyse, Personas & User‑Journeys

| Stakeholder/Nutzergruppe | Ziele/Bedürfnisse | Einfluss/Interesse | Bemerkungen |
| --- | --- | --- | --- |
| **Endnutzer:innen** | Informationen erhalten, sich registrieren, Kommentare abgeben, an Aktionen teilnehmen | Hoch / Hoch | Nutzende stehen im Fokus; Barrierefreiheit (WCAG 2.1) wird sichergestellt. |
| **Redakteur:innen** | Inhalte verwalten (Artikel, Events), Nutzer:innen moderieren | Mittel / Hoch | Benötigen Admin‑Backend mit Rollenmodellen. |
| **PO & Produktteam** | Business‑Ziele erreichen, Roadmap definieren | Hoch / Hoch | Entscheiden über Prioritäten und Features. |
| **IT‑Administration** | Systeme betreiben, Infrastruktur sichern | Hoch / Mittel | Verantwortlich für Deployments, Backup, DR. |
| **Datenschutzbeauftragte:r** | Einhaltung DSGVO, Löschfristen | Mittel / Hoch | Prüft Daten‑Flows & Einwilligungen【50†L151-L159】. |
| **Externe Partner (CRM)** | Schnittstellen bereitstellen, Daten konsistent halten | Mittel / Mittel | Integrationstests & SLA‑Überwachung notwendig. |

**User‑Journey Beispiel**: „Registrierung“ – Person besucht Startseite → klickt „Registrieren“ → füllt Formular aus → erhält Double‑Opt‑In‑Mail → bestätigt Mail → Profil wird angelegt. Diese Journey erfordert End‑to‑End‑Tests und Zugriffssicherheit (JWT, RBAC)【19†L35-L39】.

### 2.2 Funktionale Anforderungen (Epics → User Stories)

**Epics (E)**

1. **E1 – Account & Authentifizierung** – Registrierung, Login, Profilverwaltung, Passwort‑Reset, Social Login (optional).
2. **E2 – Content Management** – Erstellung/Bearbeitung/Löschung von Artikeln, Events, Kommentaren, Tagging, Mediathek.
3. **E3 – CRM‑Integration** – Synchronisierung von Kontakten, Newsletter‑Opt‑Ins, Abgleichen von Nutzer:innen mit dem CRM.
4. **E4 – Spenden & Aktionen** – Darstellung von Kampagnen, Spenden via integrierter Anbieter, Gamification‑Elemente.
5. **E5 – Administration & Rollen** – Rollen‑/Rechte‑Management, Moderation, Nutzer‑Support.
6. **E6 – Reporting & Analytics** – Dashboards für Team, Export von Nutzungsdaten (DSGVO konform), Feedback‑Funktion.

**Beispiel User Stories (US)** – jede Story verfügt im Backlog über Akzeptanzkriterien, Definition of Ready (DoR) & Definition of Done (DoD):

| Epic | User Story | Beschreibung | Akzeptanzkriterien |
| --- | --- | --- | --- |
| **E1** | **US‑1: Als nicht registrierte:r Nutzer:in** möchte ich mich registrieren, um ein Profil zu erstellen. | Formulareingabe mit Validierung, Double‑Opt‑In‑Email. | E‑Mail wird versendet, Link gilt 24 h; nach Bestätigung wird JWT generiert und Session gestartet. |
| **E1** | **US‑2: Als registrierte:r Nutzer:in** möchte ich mich einloggen, um Zugriff auf meine Inhalte zu erhalten. | Eingabe von E‑Mail & Passwort; Authentifizierung über API; falsche Logins werden protokolliert【50†L151-L159】. |  |
| **E2** | **US‑3: Als Redakteur:in** möchte ich einen Artikel erstellen, damit Nutzer:innen informiert werden. | Editor mit Markdown; Speichern erstellt Draft; Veröffentlichen setzt Status auf „Published“. |  |
| **E3** | **US‑4: Als System** möchte ich neue Kontakte an das CRM übertragen, damit Newsletter‑Abonnent:innen synchron sind. | Bei Registrierung wird ein API‑Call an CiviCRM ausgelöst; Fehler werden erfasst und retried. |  |
| **E4** | **US‑5: Als Nutzer:in** möchte ich an einer Spendenkampagne teilnehmen, um die Organisation zu unterstützen. | Auswahl eines Betrags, Weiterleitung an Zahlungsanbieter; Bestätigungsseite; Daten werden anonymisiert gespeichert. |  |
| **E5** | **US‑6: Als Admin** möchte ich Benutzerrollen zuweisen können, um Funktionen zu steuern. | Rollenmodell (Admin, Editor, Nutzer), RBAC‑Durchsetzung auf API‑Ebene. |  |
| **E6** | **US‑7: Als Produktteam** möchte ich Berichte über Nutzerinteraktionen exportieren, um Entscheidungen zu treffen. | Filter nach Zeitraum, CSV‑Export; DSGVO‑Konform: personenbezogene Daten werden pseudonymisiert【50†L151-L159】. |  |

### 2.3 Nichtfunktionale Anforderungen (NFRs)

| NFR‑Kategorie | Vorgabe | Evidence‑ID | Bemerkung |
| --- | --- | --- | --- |
| **Performance** | p95‑Ladezeit < 1,5 s; Build‑Pipeline ≤ 15 min; Response‑Zeit API ≤ 300 ms. | 【19†L35-L39】【1†L19-L22】 | Instabile Builds müssen vermieden werden; Performance‑Budgeting via Lighthouse【1†L19-L22】. |
| **Skalierbarkeit** | Horizontale Skalierung durch Kubernetes; Auto‑Scaling ≥ 2 Instances; DB‑Replikation. | 【66†L39-L47】 | Containerisierung erhöht Isolation und Skalierbarkeit. |
| **Sicherheit** | Einhaltung OWASP Top 10; Secrets‑Management; MFA für Admins; SLSA 2/3【94236474580611†L619-L628】; Logging ohne PII【50†L151-L159】. | 【50†L151-L159】【94236474580611†L619-L628】 | Supply‑Chain‑Security, Logging & Monitoring. |
| **Verfügbarkeit** | 99,5 % Uptime; RPO < 1 h; RTO < 4 h. | 【50†L151-L159】 | Backup & DR‑Prozesse definieren. |
| **Compliance** | DSGVO‑, ISO 27001‑Erfüllung, Barrierefreiheit nach WCAG 2.1 AA; Standardkonforme Cookie‑Consent‑Banner. | 【50†L151-L159】【663759207839439†L140-L152】 | Logging anonymisieren, Consent‑Management. |
| **Interoperabilität** | API‑Versionierung (v1, v2), Unterstützung für GraphQL (optional), JSON:API‑Konventionen. |  | Schnittstellen klar dokumentieren. |
| **Internationalisierung** | Unterstützung für DE/EN, spätere Sprachen; i18n‑Library; l10n‑Funktionalität. |  | Sprach‑Dateien zentral verwalten. |
| **Barrierefreiheit** | Mindestens WCAG 2.1 AA; Farben/ Kontraste; Tastatur‑Navigation; Screenreader‑Labels. |  | Lighthouse‑Accessibility‑Audits (≥ 90). |
| **Observability** | Metriken, Logs, Tracing; SLO/SLI definieren; Alerts bei Ausfällen【50†L151-L159】. | 【50†L151-L159】 | Nutzung von Prometheus, Grafana, Loki. |

### 2.4 Feature‑Flag & Experimentierstrategie

Um Releases risikoarm zu gestalten, werden **Feature Flags** verwendet. Neue Features werden inaktiv ausgeliefert und durch Schalter aktiviert. Das ermöglicht **Canary Releases**, A/B‑Tests und schnelles Rollback. Flags werden in der Datenbank gespeichert (z. B. Unleash) und können pro User‑Segment aktiviert werden. Experimente werden mit Metriken versehen, um Effekte auf KPIs zu messen. Feature‑Flags sind in der CI/CD‑Pipeline integriert (Konfigurationsmatrix). Ein Abuse‑Case ist das versehentliche Offenlassen veralteter Flags; daher werden Flags mit Ablaufdatum versehen und regelmäßig aufgeräumt.

## 3 Architektur & Design

### 3.1 Architekturüberblick (C4)

**Kontext‑Diagramm (Level 1)** – Die Plattform besteht aus folgenden Hauptkomponenten:

* **Web‑Browser** (Nutzer:innen) greift über HTTPS auf das **Frontend** (React) zu.
* **Frontend** kommuniziert via REST/GraphQL mit dem **Backend** (FastAPI) über eine **API‑Gateway**.
* **Backend** interagiert mit der **Datenbank** (PostgreSQL) und dem **CRM‑System** (CiviCRM) per HTTP/REST.
* **n8n‑Automatisierung** orchestriert Workflows (z. B. Double‑Opt‑In, E‑Mail‑Benachrichtigungen【1†L118-L126】).
* **Deployment Environment** ist Kubernetes in der Cloud; Logging & Monitoring‑Systeme empfangen Metriken und Logs.

**Container‑Diagramm (Level 2)** –

| Container | Technologie | Verantwortung |
| --- | --- | --- |
| **Frontend‑Container** | Node/React 18 | Liefert statische Assets, verarbeitet Interaktionen, validiert Eingaben. |
| **Backend‑Container** | Python 3.11/FastAPI | Bietet REST‑/GraphQL‑Endpoints, authentifiziert Anfragen (JWT), führt Business‑Logik aus. |
| **Database‑Container** | PostgreSQL 15 | Persistiert Nutzer‑ und Inhaltsdaten, führt Migrationen aus. |
| **Cache** | Redis | Beschleunigt Sessions, Rate‑Limiting. |
| **CRM Proxy** | PHP‑Bridge/HTTP Client | Proxyt Anfragen an CiviCRM (falls CiviCRM extern betrieben). |
| **Automation** | n8n | Workflow‑Engine für Prozesse (Double‑Opt‑In, Notifications). |
| **CI/CD Runner** | GitHub Actions Runner | Baut, testet und deployt Containerimages. |
| **Observability Stack** | Prometheus, Grafana, Loki | Sammelt Metriken/Logs, visualisiert Dashboards, versendet Alerts. |

### 3.2 Architekturentscheidungen (ADR‑Log)

| ADR‑ID | Entscheidung | Alternativen | Begründung & Evidence | Risiken |
| --- | --- | --- | --- | --- |
| **ADR‑001** | **Containerisierung via Docker/Kubernetes** | Weiterhin Plesk Shared Hosting | Container erhöhen Isolation und Skalierbarkeit【66†L39-L47】; ermöglichen reproduzierbare Builds und einfache Rollbacks. Plesk ist komplex und bietet geringe Flexibilität【9†L25-L33】. | Mehraufwand für DevOps‑Setup; benötigtes Cloud‑Budget. |
| **ADR‑002** | **Python FastAPI für Backend** | Node.js/Express, Laravel | FastAPI bietet asynchrones IO, automatische OpenAPI‑Docs und hohe Performance; bestehende Codebasis basiert bereits darauf【19†L35-L39】. | Lernkurve für Teammitglieder ohne Python‑Kenntnisse; weniger Ökosystem im Vergleich zu Express. |
| **ADR‑003** | **React 18/TypeScript für Frontend** | Vue.js, Angular | Modernes Ökosystem, starke Community, hoher Wiedererkennungswert; TS erleichtert Typ‑Sicherheit【19†L35-L39】. | Umfangreiche Build‑Toolchain; potenzielle Overhead bei großen Bundles. |
| **ADR‑004** | **CI/CD über GitHub Actions** | GitLab CI, Jenkins | GitHub Actions ist bereits in Nutzung, lässt sich durch Actions für CodeQL, Dependabot, SBOM leicht erweitern【19†L20-L28】. | Vendor Lock‑in; Execution‑Limits bei Self‑Hosted Runnern. |
| **ADR‑005** | **SLSA Level 2 (Ziel Level 3)** | Keine formale Supply‑Chain‑Security | SLSA definiert Reifegrade für Build‑Provenance und schützt vor Tampering【94236474580611†L619-L628】. | Zusätzlicher Aufwand zum Erstellen/Verifizieren von Attestationen; Tools müssen integriert werden. |
| **ADR‑006** | **OAuth 2.1 / OpenID Connect** | Eigenes Auth‑System | Standardisierte Authentifizierung, Delegation; Integration mit Drittanbietern; sichere Token‑Verwaltung【94236474580611†L619-L628】. | Komplexere Konfiguration; Abhängigkeit von IdP. |
| **ADR‑007** | **PostgreSQL** | MySQL, MongoDB | ACID‑Konformität, leistungsfähiges relationales DB‑System mit JSON‑Support; gute Unterstützung in Python ORMs. | Möglicher Overhead bei horizontaler Skalierung; zusätzliche Wartung. |

### 3.3 API‑Design (REST / GraphQL)

* **REST v1** – Klares Versioning in der URL (/api/v1/...), Nutzung von JSON als Standardformat. HTTP‑Methoden entsprechend der Ressource (GET/POST/PUT/DELETE). Pagination über limit/offset. Fehlercodes: 4xx (Client‑Fehler), 5xx (Server‑Fehler). Rate‑Limiting (z. B. 100 Requests/min pro IP). ETag‑Header zur Cache‑Steuerung.
* **GraphQL (Optional)** – Für komplexe Clients; Schema wird mit SDL definiert; Resolver auf Backend‑Seite; Support für Federation.
* **OpenAPI‑Spezifikation** – Alle REST‑Endpunkte werden per OpenAPI 3.1 beschrieben. Dokumentation wird automatisch generiert (FastAPI).
* **Fehlerbehandlung** – Konsistente Fehlerantworten (z. B. { "error": { "code": "NotFound", "message": "Resource not found" } }). Logging und Monitoring bei unerwarteten Exceptions【50†L151-L159】.
* **Ratenbegrenzung & Sicherheit** – API‑Key‑/Bearer‑Auth; JWT‑Verifizierung; CORS‑Konfiguration konfigurierbar über Environment‑Variablen (siehe Code‑Diff im Audit).

### 3.4 Datenmodell & Persistenz

* **Nutzer** (users): id (UUID), email, password\_hash, name, role, created\_at, updated\_at.
* **Artikel** (articles): id, title, slug, content, author\_id (FK), status, created\_at, updated\_at.
* **Kommentare** (comments): id, article\_id (FK), user\_id (FK), body, created\_at.
* **Spenden** (donations): id, user\_id (FK), amount, currency, status, created\_at.
* **Audit‑Logs** (audit\_logs): id, user\_id, action, resource\_type, resource\_id, timestamp.
* **Feature Flags** (feature\_flags): key, value, expires\_at.

Migrationen werden mit Alembic gesteuert (Versionstabelle alembic\_version). Datenretention: personenbezogene Daten von inaktiven Nutzer:innen werden nach 24 Monaten anonymisiert. Backups erfolgen täglich, aufbewahrt für 30 Tage.

### 3.5 Integrationen & Schnittstellen

* **CRM / CiviCRM** – REST‑API; Authentifizierung per API‑Key; Integration über dedizierten Service mit Retry‑Mechanismen & Circuit‑Breaker.
* **Payment Provider** – Spenden werden an externes Payment Gateway (z. B. Stripe) delegiert. Webhooks werden verifiziert.
* **Email‑Dienst** – Versand via SMTP oder API‑Provider (z. B. SendGrid). DKIM/SPF‑Einträge werden gesetzt.
* **Analytics** – Datenerfassung mit Matomo oder Plausible; keine personenbezogenen Tracking‑IDs.
* **CI/CD Tools** – GitHub Actions; CodeQL & Dependabot; SBOM‑Generator (Syft)【57†L158-L165】.
* **Third‑Party Libraries** – Strikte Versionierung; keine unmaintained Pakete; regelmässige Updates mittels Dependabot‑Groups【42†L45-L53】.
* **Contract‑Tests** – Für externe Schnittstellen (CRM, Payment) werden Pact‑Tests erstellt, um Versionskompatibilität zu gewährleisten.

## 4 Delivery‑Setup & Governance

### 4.1 Arbeitsmodell (Scrum)

* **Sprintlänge**: 2 Wochen.
* **Zeremonien**: Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospektive.
* **Artefakte**: Product Backlog, Sprint Backlog, Increment.
* **Commitments**: Product Goal, Sprint Goal, Definition of Done.
* Laut dem offiziellen Scrum Guide 2020 (noch 2025 aktuell【606628840459431†L96-L107】) gibt es ein Scrum‑Team mit drei Verantwortlichkeiten: Product Owner (PO), Scrum Master (SM) und Entwicklerteam【606628840459431†L132-L163】. Self‑Management statt nur Self‑Organisation wird betont.

### 4.2 Rollen & RACI

| Rolle | Verantwortlichkeiten (R=Responsible, A=Accountable, C=Consulted, I=Informed) |
| --- | --- |
| **Product Owner** | A für Produktvision, Priorisierung; C bei Architektur; I bei Security. |
| **Tech Lead** | R für Architektur, Code‑Qualität; A für technische Entscheidungen; C bei Requirements. |
| **Scrum Master** | R für Prozess‑Einhalten; A für Removing Impediments; I beim Architektursupport. |
| **Entwickler:innen** | R für Umsetzung der Stories; C bei Schätzung; I über Architekturänderungen. |
| **QA‑Engineer** | R für Teststrategie & Ausführung; A für Testberichte; C bei Definition of Done. |
| **DevOps/SRE** | R für CI/CD, Infrastruktur, Observability; C bei Security & Compliance; I bei Feature‑Entwicklungen. |
| **Security Engineer** | R für Threat Modeling, Security‑Policies & Scans; A für Sicherheitsfreigaben; C bei Architektur & Code‑Reviews. |
| **UX‑Designer:in** | R für UI/UX‑Design, Barrierefreiheit; C bei Story Refinement. |
| **Data Engineer** | R für Datenmodelle, ETL‑Prozesse; C bei Analytics. |
| **Stakeholder** | I über Fortschritt, Risiko; C bei Geschäftszielen. |

### 4.3 Entscheidungsforen & Change‑Management

* **Architekturboard**: Wöchentliches Meeting; Mitglieder: Tech Lead, DevOps, Security Engineer, PO. Ziel: Architekturentscheidungen, ADR‑Review.
* **Change‑Control‑Board (CCB)**: Monatlich; bewertet CRs (Change Requests) in Bezug auf Scope, Budget, Risiken.
* **Backlog‑Refinement**: Jede Woche; Stories werden geschätzt (Story Points), Akzeptanzkriterien überprüft.
* **Eskalationsprozess**: Projekt‑Blocker werden an PO & Sponsor eskaliert; definierte Zeit zur Lösung (max. 48 h).

## 5 DevEx & Code‑Standards

### 5.1 Repo‑Layout

* **Mono‑Repo** mit Workspaces (/frontend, /backend, /infra, /docs).
* **Packages**: Wiederverwendbare Komponenten (z. B. /packages/ui, /packages/shared).
* **Tooling**: scripts/setup-workspace.ps1 für lokale Einrichtung【9†L42-L51】. Devcontainer für Codespace【12†L39-L47】.

### 5.2 Branching & Versionierung

* **Branching‑Modell**: Trunk‑Based Development. Kurzlebige Feature‑Branches (< 1 Woche), Merge via Pull Request.
* **Commit‑Konventionen**: Conventional Commits (feat:, fix:, docs:) & SemVer.
* **Release‑Versionen**: Tagging nach vX.Y.Z. **CHANGELOG.md** automatisch generiert.
* **Release‑Branches** nur für Hotfixes (Backport).

### 5.3 Code‑Qualität & Static Analysis

* **Formatter/Linter**: Prettier & ESLint für JS/TS; Black & Flake8 für Python; PHP CS Fixer für PHP‑Bridges.
* **Pre‑Commit‑Hooks**: Husky & lint‑staged laufen lokal; verhindern das Einchecken von Lint‑Fehlern und Tests【14†L35-L43】.
* **Static Analysis**: CodeQL, SonarQube.
* **Testabdeckung**: Minimale Coverage 80 %; Mutation‑Testing mit Stryker (TS) & Mutmut (Python) in Nightly‑Runs.
* **Review‑Policy**: Mindestens 2 Review‑Freigaben; Security‑Engineer muss Security‑relevante Änderungen absegnen; PR‑Template mit Checkliste.

### 5.4 Dependency‑Management & Lizenz‑Compliance

* **Dependabot Groups**: Updates werden gruppiert (z. B. alle PHP‑Pakete in einem PR)【42†L45-L53】. CVEs werden innerhalb 48 h gepatcht (Zero‑CVE‑Policy【1†L161-L169】).
* **SBOMs**: Für jede Release wird eine CycloneDX‑SBOM generiert【57†L158-L165】.
* **Lizenz‑Check**: Mit FOSSA oder ORT; nur kompatible OSS‑Lizenzen (MIT, Apache 2.0, GPL v3) zulässig.
* **Deny‑List**: Verbietet unmaintained Pakete; Abnahme durch Security‑Engineer.

## 6 CI/CD & Environments

### 6.1 Pipeline (Build → Test → Security → Package → Deploy)

Die Pipeline folgt dem Fail‑Fast‑Prinzip; Stages sind klar getrennt. (Der initiale Audit zeigte instabile Tests【19†L35-L39】, daher wurde die Pipeline neu konzipiert.)

| Stage | Schritte | Gates | Artefakte |
| --- | --- | --- | --- |
| **Checkout & Setup** | Code Checkout, Setup von Node/Python/Go. Caching von npm/pip/Composer. | – | – |
| **Install & Build** | npm ci/pip install/composer install; Build von Frontend & Backend; Docker‑Images bauen. | Build schlägt fehl bei Lint‑ oder Typfehlern (kein „continue‑on‑error“). | Container‑Images, SBOM, Build‑Logs |
| **Tests** | Unit‑Tests (Vitest, PyTest, PHPUnit); Integration‑Tests; E2E‑Tests mit Playwright. | Mindest‑Coverage 80 %, Mutation Score ≥ 50 %. | Test‑Reports, Coverage‑Reports |
| **Security‑Scans** | SAST (CodeQL), SCA (Syft/Trivy), Secrets‑Scan (Gitleaks)【50†L105-L113】. | Build wird gestoppt bei kritischen CVEs; moderate CVEs → Warnung. | Reports, SBOM, Vulnerability‑List |
| **Package & Publish** | Versionieren, Signieren (Cosign), Upload in Container‑Registry (GHCR). | Prüfsumme & Signatur stimmen; SBOM angehängt. | Container‑Image, SBOM, Provenance |
| **Deploy to Staging** | Helm Chart/ArgoCD; Feature‑Flags aktivieren; Smoke‑Tests. | Smoke‑Tests (HTTP 200, DB‑Verbindung) müssen bestehen. | Staging‑Release |
| **Integration & E2E** | Playwright‑Tests gegen Staging; Last‑Tests (k6). | Response‑Zeit & Fehlerquote innerhalb der Budgets. | E2E‑Reports |
| **Deploy to Prod** | Blue/Green oder Canary: neuer Pod wird parallel ausgerollt, Traffic wird schrittweise umgeschwenkt. | Approval durch PO & QA; kein kritischer Incident. | Prod‑Release |
| **Post‑Deploy** | Monitoring der SLOs, automatische Rollback‑Strategie; Tagging & Release‑Notes. | – | Release‑Notes |

### 6.2 Artefakt‑ & Paketverwaltung

* **Registry**: GitHub Container Registry (GHCR) für Docker‑Images.
* **Helm‑Charts**: Versionierte Charts im Repo /infra/helm.
* **Package Registry**: GitHub Packages für npm‑Packages (z. B. Shared UI‑Library).
* **Artefakt‑Retention**: Artefakte (Builds, Reports) werden 90 Tage aufbewahrt.

### 6.3 Environments

| Umgebung | Zweck | Parität | Besonderheiten |
| --- | --- | --- | --- |
| **Dev Local** | Lokale Entwicklung mit Docker Compose/Devcontainer | Hoch (Docker) | Datenbank & Services werden als Container gestartet. |
| **CI** | Testlauf‑Umgebung | Mittel | Build & Tests; keine externen Secrets; isoliert. |
| **Staging** | Vor‑Prod; reale Cloud‑Umgebung | Hoch | Anonymisierte Daten; Feature‑Flags können experimentell aktiviert werden. |
| **Prod** | Live‑System | Hoch | Zugriffskontrolle; Logging anonymisiert; automatischer Failover. |

### 6.4 Deploy‑Strategien

* **Blue/Green**: Zwei identische Produktions‑Environments; das neue Release wird auf „Green“ ausgerollt und nach Tests per DNS/Ingress aktiviert.
* **Canary**: Kleinere Prozentzahl des Traffics wird auf neue Version geleitet; Monitoring von Fehlern & Performance; bei Problemen Rollback.
* **Feature Flags**: In Kombination mit Canary; einzelne Funktionen können für bestimmte Nutzergruppen aktiviert werden.
* **Datenbank‑Migrationen**: DB‑Migrationsskripte werden in der Deploy‑Pipeline vor dem Traffic‑Switch ausgeführt; Migrations sind Idempotent; Downgrades vorhanden.

### 6.5 Release‑Management

* **Release Trains**: Alle 4 Wochen ein Minor‑Release; Hotfix‑Releases bei Bedarf.
* **Go/No‑Go‑Meeting**: am Ende jeder Sprint‑Review; PO, QA & Tech Lead entscheiden.
* **Rollback**: Wenn Fehler nach dem Rollout > 5 % Fehlerquote (p95) betragen, wird automatisches Rollback eingeleitet; alte Version steht im Blue‑Environment bereit.
* **Versionierung**: SemVer; Breaking Changes in Major‑Releases; Migrations und API‑Versionen kommunizieren.

## 7 Sicherheit & Datenschutz

### 7.1 Threat Modeling

* **Methodik**: STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege).
* **Hauptrisiken**:
  + **Tainted Dependencies**: Veraltete Komponenten können Schwachstellen enthalten【52†L51-L57】. SCA & Dependabot Groups werden eingesetzt【42†L45-L53】.
  + **Build‑Manipulation**: Angriffe auf CI/CD Pipeline werden durch isolierte, ephemere Runner und SLSA‑Attestationen verhindert【94236474580611†L619-L628】.
  + **Secrets‑Leakage**: Secrets werden in Vaults gespeichert; in Logs dürfen keine PII erscheinen【50†L151-L159】.
  + **Injection & XSS**: Validierung aller Eingaben, Verwendung vorbereiteter Statements, Aktivierung CSP.
  + **Authentication**: MFA für Admins; Rate‑Limiting; Password‑Hashing (bcrypt, Argon2).

### 7.2 OWASP‑Kontrollen & Best Practices

| Kategorie | Maßnahme | Evidence |
| --- | --- | --- |
| **A01 – Broken Access Control** | RBAC, Least‑Privilege‑Policies; Unit‑Tests & Pen‑Tests. | – |
| **A02 – Cryptographic Failures** | TLS 1.3 überall; HSTS; Secret‑Rotation; Key‑Management. | 【50†L151-L159】 |
| **A03 – Injection** | Prepared Statements, Input‑Sanitization, ORM. | – |
| **A04 – Insecure Design** | Threat Modeling, Security‑Design‑Reviews pro Epic. | – |
| **A05 – Security Misconfiguration** | IaC mit Default‑Denial; Container‑Hardening【806704564751728†L391-L437】; Regelmäßige Patches. | – |
| **A06 – Vulnerable & Outdated Components** | SCA, Dependabot, Zero‑CVE‑Policy【52†L51-L57】. | 【52†L51-L57】 |
| **A07 – Identification & Authentication** | OAuth 2.1, MFA, Brute‑Force‑Protection. | – |
| **A08 – Software & Data Integrity Failures** | SLSA 2+; SBOM; Signed Commits【94236474580611†L619-L628】. | 【94236474580611†L619-L628】 |
| **A09 – Security Logging & Monitoring** | Zentralisierte Logs (Loki), Anonymisierung & Retention【50†L151-L159】. | 【50†L151-L159】 |
| **A10 – Server‑Side Request Forgery** | Whitelist für externe Requests; Timeout & Size‑Limits. | – |

### 7.3 Security‑Tests & Gates

* **SAST**: CodeQL, SonarQube.
* **DAST**: OWASP ZAP automatisiert in Pipeline; wöchentliche Scans.
* **SCA**: Syft/Trivy; Alerts bei neuen CVEs.
* **Secrets‑Scanning**: Gitleaks, GitHub Secret Scanning.
* **Pen‑Testing**: Halbjährlich externer Pen‑Test; Ergebnisberichte werden umgesetzt.
* **Compliance‑Audit**: Jährliches DSGVO‑Audit (inkl. Auftragsverarbeitungs‑Verträge).

### 7.4 Datenschutz & Datenverarbeitung

* **Klassifizierung**: Datenklassen (öffentlich, intern, vertraulich, sensibel). Sensible Daten (z. B. E‑Mail‑Adressen) werden verschlüsselt gespeichert; Passwörter werden gehasht.
* **Minimierung & Retention**: Es werden nur notwendige Daten erhoben; Log‑Retention max. 30 Tage【50†L151-L159】.
* **Einwilligungen**: Double‑Opt‑In für Newsletter; Cookie‑Consent‑Banner; Einwilligungen werden auditierbar gespeichert.
* **Betroffenenrechte**: Export & Löschung der Daten innerhalb von 30 Tagen auf Anfrage.
* **Data Processing Agreements**: Verträge mit Dienstleistern (CRM, Payment) werden geprüft (Art. 28 DSGVO).

## 8 Infrastruktur & Betrieb

### 8.1 Infrastructure as Code (IaC), Container & Orchestrierung

* **IaC**: Terraform 1.6 verwaltet Cloud‑Ressourcen (VPC, Subnets, K8s‑Cluster, RDS). Pull‑Request‑Review, automatisierte Plan‑Phase (Plan & Apply).
* **Container**: Docker images, Multistage‑Builds; Hardening (minimalistisches Base‑Image, nur benötigte Tools)【806704564751728†L391-L437】.
* **Kubernetes**: Cluster in Cloud (AKS/EKS/GKE). RBAC, Network Policies, Pod Security Standards; Secrets via Kubernetes Secrets & External Secrets.
* **Service Mesh**: Istio oder Linkerd zur Steuerung von Traffic, Observability & Security (Mutual TLS).
* **Autoscaling**: HPA (Horizontal Pod Autoscaler) basierend auf CPU/Memory & Custom Metrics.

### 8.2 Observability

* **Metriken**: Prometheus sammelt Metriken (HTTP Latency, Error‑Rate, CPU‑/Memory‑Usage).
* **Logs**: Loki/ELK speichert strukturierte Logs; PII wird gehasht oder entfernt【50†L151-L159】.
* **Tracing**: OpenTelemetry + Jaeger; End‑to‑End‑Traces.
* **SLI/SLO**: Für jede Service‑Funktion definierte Service Level Indicators; Alerts (Burn‑Rate Alerts).
* **Dashboards**: Grafana‑Boards für DevOps, Business, Management; KPI‑Visualisierung.

### 8.3 Verfügbarkeit & Resilienz

* **High Availability**: Multi‑AZ Deployment; DB‑Replikation; Pod Disruption Budget.
* **Disaster Recovery**: Backups (DB, Files) täglich; RPO < 1 h, RTO < 4 h. Wiederherstellungsübungen quartalsweise.
* **Chaos‑Engineering**: Gezieltes Abschießen von Pods/Services, um Resilienz zu prüfen.
* **Traffic‑Management**: Circuit Breaker (Hystrix/Resilience4J) & Retry Pattern; Rate Limiter.

### 8.4 Runbooks, On‑Call & Incident‑Management

* **Runbooks**: Dokumentieren Standard‑Operationen (Deployment, Rollback, Scale‑Up/Down).
* **Playbooks**: Für spezifische Incidents (DB‑Ausfall, API‑Down, Security Incident).
* **On‑Call**: 24/7 Rotation; PagerDuty oder Opsgenie; Eskalationswege.
* **Postmortems**: Nach jedem Incident; Root‑Cause‑Analysis, Lesson Learned; Maßnahmen im Backlog.

## 9 Qualitätssicherung & Test

### 9.1 Teststrategie & Pyramide

Die **Testpyramide** besteht aus Unit‑Tests (Basis), Integration‑Tests, End‑to‑End‑Tests und manuelle Explorative Tests.  
- **Unit‑Tests**: Ziel ≥ 80 % Coverage. Tools: Vitest (Frontend), PyTest/pytest‑asyncio (Backend).  
- **Integration‑Tests**: Testen das Zusammenspiel von Komponenten (API + DB + CRM). Stubs/Mocks für externe Services.  
- **Contract‑Tests**: Pact Tests mit CiviCRM & Payment.  
- **End‑to‑End‑Tests**: Playwright‑Tests; simulieren User‑Journeys (Registrierung, Login, Content‑Posting, Donation).  
- **Performance‑Tests**: k6‑Skripte; Ziel, mind. 500 req/s bei Response < 300 ms.  
- **Mutation‑Tests**: Nightly Runs; Ziel Mutations‑Score ≥ 50 %.  
- **Accessibility‑Tests**: axe‑core; Ziel A11y‑Score ≥ 90.

### 9.2 Testfall‑Übersicht (Auszug)

| Test‑ID | Ziel | Schritte | Testdaten | Erwartetes Ergebnis |
| --- | --- | --- | --- | --- |
| **T‑001** | Registrierung validiert E‑Mail & Passwort. | Formular ausfüllen → Absenden → E‑Mail öffnen → Link klicken. | Gültige/ungültige E‑Mails, starke Passwörter. | Erfolgreiche Registrierung → Status 201; ungültige E‑Mail → Fehler 400; E‑Mail versandt. |
| **T‑005** | Zugriffskontrolle Admin vs. Nutzer:in. | Als Admin anmelden → Artikel löschen; Als Nutzer anmelden → Artikel löschen. | Admin‑Creds, Nutzer‑Creds. | Admin kann löschen (HTTP 204); Nutzer erhält 403 Forbidden. |
| **T‑010** | CRM‑Sync. | Registrierung abschließen → CRM API prüfen. | Dummy‑E‑Mail. | Ein neuer Kontakt in CRM; bei Ausfall wird Retry versucht & Logging erfolgt. |
| **T‑020** | Lasttest. | 500 gleichzeitige Registrierungen. | Generierte Testnutzer. | System hält Stand (max. CPU 70 %, Response ≤ 300 ms). |

### 9.3 Leistungs‑/Lasttests & Benchmarks

* **k6**: Skripte definieren Registrierungs‑, Login‑, Artikelaufrufe; Ziel 500 req/s.
* **Lighthouse**: Performance‑Audits; Score ≥ 90【1†L19-L22】.
* **DB‑Profiling**: Nutzung von EXPLAIN ANALYZE; Index‑Optimierung; Überprüfung auf N+1 Queries.

### 9.4 Abnahmekriterien je Deliverable

Für jedes Deliverable werden Abnahmekriterien definiert (z. B. „Feature Story X gilt als erledigt, wenn …“). Kriterien umfassen: - Umsetzung der Akzeptanzkriterien.  
- Erfolgreiche Unit‑/Integration‑Tests.  
- Code‑Review bestanden (keine Blocker).  
- Dokumentation aktualisiert (API‑Docs, ADR‑Log).  
- Kein kritischer Security‑Befund.

## 10 Roadmap, Planung & Aufwand

### 10.1 Meilensteine & Gates

| Meilenstein | Beschreibung | Datum | Eintritts‑/Austrittskriterien |
| --- | --- | --- | --- |
| **MS‑0** | Kickoff & Anforderungsworkshop | 15.10.2025 | Projektteam steht; Scope & Ziele abgenommen; Stakeholder kommuniziert. |
| **MS‑1** | Architektur & Backlog fertig | 30.11.2025 | C4‑Diagramme, ADR‑Log, initiales Backlog mit priorisierten Epics, technische Spike‑Ergebnisse. |
| **MS‑2** | MVP‑Release v0.1 auf Staging | 31.01.2026 | Kernfunktionen (Registrierung, Login, Artikel‑Listing) implementiert; CI/CD grün; Tests bestanden. |
| **MS‑3** | Public Beta | 30.04.2026 | Alle Epics E1–E3 abgeschlossen; DSGVO‑Konformität gewährleistet; Testabdeckung ≥ 70 %. |
| **MS‑4** | GA‑Release v1.0 | 31.07.2026 | Alle funktionalen Anforderungen; SLSA Level 2 erfüllt; SLOs stabil; Dokumentation komplett. |
| **MS‑5** | Projektabschluss & Lessons Learned | 30.09.2026 | Übergabe an Betriebs‑Team; Postmortem; Produktziele evaluiert. |

### 10.2 Work Breakdown Structure (WBS)

Die WBS ist dreistufig aufgebaut. Nachfolgend ein Auszug mit den wichtigsten Arbeitspaketen. Alle Pakete enthalten Input‑Evidence‑IDs, konkrete Schritte, Responsible, Dauer (Tage), Aufwand (Personentage/PT), Abhängigkeiten und DoD.

| WBS‑ID | Arbeitspaket | Zweck/Nutzen | Inputs (Evidence‑ID) | Schritte (nummeriert) | Owner | Dauer (Tage) | Aufwand (PT) | Abhängigkeiten | Risiken & Kontrollen | DoD |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1.1** | Projektkickoff & Scope‑Finalisierung | Alignment aller Stakeholder, Risikoerfassung | Projektauftrag, Voranalyse | 1. Kickoff‑Meeting planen; 2. Scope vorstellen & diskutieren; 3. Risiken initial erfassen; 4. Protokoll & Aktionsliste erstellen | PO, PM | 3 | 6 | – | Missverständnisse in Scope → moderated Workshop | Scope‑Doc genehmigt; Aktionsliste verabschiedet |
| **1.2** | Requirements Gathering & Personas | Detaillierte Anforderungen erfassen | User‑Interviews, Analysen | 1. Workshops durchführen; 2. Personas & User‑Journeys dokumentieren; 3. User Stories schreiben; 4. Abnahme durch Stakeholder | BA, PO | 10 | 20 | 1.1 | Zeitplanverzug durch Verfügbarkeit der Stakeholder → Termine früh sichern | Abgenommene User Stories mit Akzeptanzkriterien |
| **2.1** | Architektur‑Design & ADR‑Erstellung | Definiert technische Basis & Richtlinien | Audits【66†L39-L47】【19†L35-L39】, Standards【94236474580611†L619-L628】 | 1. Erarbeiten von C4‑Diagrammen; 2. Alternativen evaluieren & ADR‑Log pflegen; 3. DB‑Modell entwerfen; 4. Schnittstellen & API‑Design spezifizieren | Tech Lead, Architect | 15 | 30 | 1.2 | Falsche Technologie‑Wahl → Architekturboard; PoCs einplanen | ADRs abgenommen; Diagramme im Repo |
| **2.2** | Dev & Repo Setup | Aufbau des Repos, Grundgerüst | Devcontainer【12†L39-L47】, Linter‑Guidelines | 1. Mono‑Repo einrichten; 2. Branch‑Policies konfigurieren; 3. Linting/Husky einrichten; 4. Editorconfig erstellen | DevOps, Dev Lead | 5 | 10 | 1.1 | Tool‑Inkonsistenzen → Standard‑Dokumentation | Repo läuft lokal; CI startet; Pre‑Commit‑Hooks aktiv |
| **2.3** | CI/CD‑Pipeline‑Implementierung | Automatisierung von Builds & Tests | Pipeline‑Best Practices【19†L35-L39】【3†L59-L67】 | 1. YAML‑Workflow erstellen; 2. Build/Test/Deploy‑Steps integrieren; 3. Security‑Scans einbinden; 4. Artefakt‑Verwaltung konfigurieren | DevOps | 10 | 20 | 2.2 | Fehlschlagende Tests【19†L35-L39】 → Testfixing; Build‑Abhängigkeiten aktuell halten | Pipeline grün; Test‑ & Security‑Reports erzeugt |
| **2.4** | Entwicklungsphase (MVP) | Implementiert E1/E2/E3 | Backlog, Architektur | 1. Features implementieren (Registrierung, Login, Artikel); 2. Unit‑/Integration‑Tests schreiben; 3. Code Reviews; 4. E2E Tests | Dev Team | 60 | 120 | 2.3 | Scope‑Creep → PO‑Abstimmung; Leistungsprobleme【19†L35-L39】 | MVP‑Features umgesetzt; Tests bestanden; Deployed auf Staging |
| **2.5** | Security Hardening & Compliance | Erfüllung von DSGVO & OWASP; SLSA Level 2 | Standards【94236474580611†L619-L628】【50†L151-L159】 | 1. Threat Modeling durchführen; 2. OWASP‑Kontrollen implementieren; 3. Logging & Monitoring anonymisieren【50†L151-L159】; 4. SBOM & SLSA‑Attestationen integrieren | Security Engineer, DevOps | 20 | 40 | 2.3 | Komplexität SLSA → Spike einplanen; Tools integrieren | Security‑Checkliste erfüllt; keine kritischen Befunde |
| **2.6** | Staging‑Betrieb & Testing | Qualität & Akzeptanz sicherstellen | Pipeline Reports, Testpläne | 1. Deploy auf Staging; 2. Integration‑, E2E‑Tests ausführen; 3. User Acceptance Test (UAT); 4. Performance‑Tests (k6, Lighthouse)【1†L19-L22】 | QA, DevOps | 15 | 30 | 2.4 | Flaky Tests → Stabilisierung; Verfügbarkeitsprobleme | Tests bestanden; Abnahmeprotokoll vorliegt |
| **2.7** | Deployment Prod (GA) | Produktivbetrieb starten | UAT‑Ergebnisse, Approval | 1. Go/No‑Go Meeting; 2. Blue/Green Deployment; 3. Smoke Tests; 4. Post‑Deploy Monitoring; 5. Release‑Notes & Kommunikation | DevOps, PO | 5 | 10 | 2.6 | Incident beim Rollout → Rollback‑Plan; DNS Switch | Live Version verfügbar; Monitoring OK |
| **3.1** | Dokumentation & Enablement | Nachhaltige Wissensbasis & Onboarding | Audit【12†L39-L47】, Standards | 1. README, CONTRIBUTING, SECURITY.md erstellen; 2. Architektur‑Dokumentation (C4, ERD); 3. User‑Guides & Glossar; 4. ADR‑Log pflegen | Tech Writer, Dev Team | 15 | 30 | 2.2 | Zeitaufwand → Priorisierung; „Docs‑As‑Code“ im Sprint integrieren | Dokumentation abgenommen; Onboarding‑Guide getestet |
| **3.2** | Beta‑Test & Feedback | Produkt reift durch Nutzerfeedback | Beta‑Release | 1. Auswahl von Beta‑Nutzer:innen; 2. Feedback‑Erhebung (Surveys, Analytics); 3. Bugfixing & Optimierungen; 4. A/B‑Tests mit Feature Flags | PO, Dev Team | 20 | 40 | 2.6 | Niedrige Teilnahme → Marketing unterstützen | Feedback implementiert; Fehlerquote reduziert |
| **3.3** | Projektabschluss & Übergabe | Wissen transferieren; Lessons Learned | alle Artefakte | 1. DR‑Test durchführen; 2. Betriebs‑Checkliste vervollständigen; 3. Lessons‑Learned‑Workshop; 4. Übergabe an Support/Betrieb; 5. Abschlussbericht | PM, Tech Lead | 10 | 20 | 2.7 | fehlende Dokumentation → Nachbesserungen | Support kann System betreiben; Projekt formal abgeschlossen |

### 10.3 Gantt‑Plan (tabellarisch)

| Phase | Start | Ende | Dauer (Tage) | Abhängigkeiten |
| --- | --- | --- | --- | --- |
| **Projektkickoff** | 15.10.2025 | 17.10.2025 | 3 | – |
| **Anforderungsanalyse** | 18.10.2025 | 31.10.2025 | 14 | Kickoff |
| **Architektur & Repo Setup** | 01.11.2025 | 15.11.2025 | 15 | Analyse |
| **CI/CD‑Implementierung** | 16.11.2025 | 30.11.2025 | 15 | Setup |
| **Entwicklung MVP** | 01.12.2025 | 30.01.2026 | 60 | CI/CD |
| **Security Hardening** | 01.12.2025 | 20.01.2026 | 20 | CI/CD |
| **Staging & Testing** | 31.01.2026 | 14.02.2026 | 15 | MVP |
| **Beta‑Test** | 15.02.2026 | 15.03.2026 | 20 | Staging |
| **Weiterentwicklung & Fixes** | 16.03.2026 | 30.06.2026 | 75 | Beta |
| **GA‑Release** | 01.07.2026 | 31.07.2026 | 31 | Testing |
| **Abschluss & Übergabe** | 01.08.2026 | 30.09.2026 | 60 | GA |

### 10.4 Tasks je WBS‑Element

Für jedes Arbeitspaket im WBS gibt es detaillierte Tasks. Hier ein exemplarischer Task aus **Arbeitspaket 2.3 (CI/CD‑Implementierung)**:

| Task‑ID | Task | Eingangs‑Voraussetzungen | Schritt‑für‑Schritt‑Anleitung | Verantwortliche Rolle | Dauer (Tage) | Aufwand (PT) | Abhängigkeiten | Risiken & Kontrollen | Akzeptanzkriterien (DoD) | Ergebnis |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2.3‑T1** | Workflow‑Datei ci.yml erstellen | Repo‑Struktur angelegt (WBS 2.2) | 1. Branch setup/ci erstellen; 2. YAML‑Datei anlegen mit Jobs checkout, setup-node, setup-python, install, build, test; 3. Secrets konfigurieren; 4. Pipeline lokal testen; 5. PR stellen & Review durchführen | DevOps | 3 | 6 | 2.2 | YAML‑Syntaxfehler; secret leakage → Code‑Review, Secrets in GitHub Secrets speichern | Workflow läuft erfolgreich in CI; Build & Tests passieren; Code review genehmigt | ci.yml im Repo, Pipeline ausgeführt |
| **2.3‑T2** | Security‑Scans integrieren | Workflow‑Datei existiert | 1. CodeQL Action konfigurieren; 2. Trivy‑Scan für Container integrieren; 3. Gitleaks‑Scan einbinden; 4. continue-on-error bei Info‑Severity konfigurieren【3†L59-L67】 | Security Engineer | 2 | 4 | 2.3‑T1 | Falsch‑Positive → Schwellwerte definieren; Build‑Zeit ↑ | Scans laufen; Reports werden erstellt; Pipeline bricht bei High‑Severity CVEs ab | CodeQL/Trivy/Gitleaks Berichte |
| **2.3‑T3** | Deployment‑Step in Pipeline | Container‑Images gebaut; Staging‑Cluster bereit | 1. Helm Chart bereitstellen; 2. K8s‑Context via GitHub Actions einrichten; 3. Deploy Job definieren (Blue/Green); 4. Smoke Tests nach Deploy; 5. Rollback Script vorbereiten | DevOps | 3 | 6 | 2.3‑T1 | Deploy‑Fehler → Dry‑Run; Helm‑Linting; Cluster‑Zugriff berechtigt | Staging‑Deploy funktioniert; Smoke Tests bestehen; Manifest signiert | Automatisierte Staging‑Deployments |

(Ein vollständiger Aufgaben‑Katalog ist im Anhang; jede Aufgabe beinhaltet definierte Inputs, Schritte, Owner, Risiken, DoD.)

## 11 Budget, Kosten & FinOps

### 11.1 CapEx/OpEx & Kostenprognosen

| Kostenkategorie | Beschreibung | Schätzung (EUR) | Bemerkung |
| --- | --- | --- | --- |
| **Personal** | Entwickler:innen, QA, DevOps, PM, Security, UX. | 700 000 € | 12 Monate, Ø 10 Personen (Tagessatz ≈ 500 €). |
| **Cloud Infrastruktur** | Kubernetes‑Cluster, DB‑Instanzen, Storage, CDN. | 150 000 € | inkl. Dev/Staging/Prod. |
| **Tools & Lizenzen** | CI/CD Runners, Monitoring (Grafana Cloud), Security‑Tools. | 50 000 € | CodeQL (kostenlos in GH), zusätzliche Tools (Snyk). |
| **Third‑Party Services** | Payment Gateway Fees, Newsletter‑Provider. | 30 000 € | abhängig von Transaktionsvolumen. |
| **Schulung & Beratung** | Trainings (SLSA, OWASP), externe Audits. | 40 000 € | Fortbildung, Zertifizierungen. |
| **Puffer (15 %)** | Risiko‑Puffer für unerwartete Kosten. | 180 000 € | +15 %. |
| **Gesamt** |  | **1 150 000 €** | Budgetrahmen lt. Ziel (1,2 Mio. €) eingehalten. |

### 11.2 Kapazitäts‑ & Kostenprognosen

* **Autoscaling & Cloud‑Kosten**: Prognose Tools (AWS Cost Explorer) werden eingerichtet; Alerts bei Budgetüberschreitung.
* **FinOps**: monatliches Kostenmonitoring, Showback pro Team; Optimierung (Spot‑Instances, Reservierungen).
* **Egress‑Kosten**: CDN nutzen (CloudFront/Azure FrontDoor), um Datenübertragung zu reduzieren.

## 12 Risiken & Chancen

### 12.1 Risikoregister

| Risiko‑ID | Beschreibung | Wahrscheinlichkeit | Auswirkung | Präventiv‑Maßnahme | Reaktiv‑Maßnahme | Owner |
| --- | --- | --- | --- | --- | --- | --- |
| **R1** | **Veraltete Komponenten & CVEs** führen zu Sicherheitslücken【52†L51-L57】. | Mittel | Hoch | Dependabot Groups & regelmäßige Updates【42†L45-L53】; SBOM; SCA. | Notfall‑Patch innerhalb 24 h; Rollback. | Security Engineer |
| **R2** | **CI‑Fehlschläge** blockieren Releases【19†L35-L39】. | Hoch | Mittel | Tests stabilisieren; Build‑Pipeline fail‑fast; Stage‑Isolation. | Pipeline‑Hotfix; Fehleranalyse; temporarily skip non‑critical gates. | DevOps |
| **R3** | **Scope Creep** durch unklare Anforderungen. | Mittel | Mittel | Product Vision dokumentieren; Change‑Board; Backlog‑Priorisierung. | Neue Anforderungen in separates Release verschieben. | PO |
| **R4** | **Compliance‑Verstoß** (DSGVO/ISO). | Niedrig | Hoch | Datenschutz‑Folgenabschätzung; regelmäßige Audits【50†L151-L159】. | Incident‑Response‐Plan; Meldung binnen 72 h. | Datenschutzbeauftragte:r |
| **R5** | **Teamfluktuation** führt zu Wissensverlust. | Mittel | Mittel | Onboarding‑Guide, Dokumentation & Pair‑Programming. | Wissens‑Transfer; Redundanz bei Rollen. | PM |
| **R6** | **Budgetüberschreitung** durch unerwartete Cloud‑Kosten. | Mittel | Mittel | FinOps‑Monitoring; Reservierungen; Limit‑Alerts. | Reduzieren von Non‑Prod‑Umgebungen; Optimierung. | PO, DevOps |
| **R7** | **Externe API‑Änderungen** (CRM/Payment) brechen Integration. | Niedrig | Mittel | Contract‑Tests; Versionierung; SLA‑Überwachung. | Hotfix, Mock‑Server. | Backend Team |
| **R8** | **Container‑Security Vorfälle** (Malware in Images). | Niedrig | Hoch | Hardening & Scanning【806704564751728†L391-L437】; Signierung. | Notary revoke & redeploy; Forensik. | DevOps |

### 12.2 Chancen

* **Continuous Improvement**: Modernisierung von Deployments (GitOps, ArgoCD).
* **Innovation**: Nutzung von AI‑gestützten Features (Recommendation, Chatbots) in späteren Phasen.
* **Open‑Source‑Engagement**: Veröffentlichung von Teilen als Open‑Source; Community‑Beiträge.

## 13 Kommunikation & Enablement

### 13.1 Stakeholder‑Matrix & Kommunikationsplan

| Stakeholder | Rolle | Informationsbedarf | Kommunikationskanal | Frequenz |
| --- | --- | --- | --- | --- |
| **Sponsor** | Budget & strategische Richtung | Status, Risiken, Budget | Steering Committee Meeting; Berichte | Monatlich |
| **PO & Team** | Produktumsetzung | Daily Scrum, Sprint Review | Jira, Slack, Sprint‑Meetings | Täglich/Wöchentlich |
| **Redaktion** | Inhaltserstellung | Feature‑Releases, CMS‑Änderungen | Newsletter, Confluence | Monatlich |
| **Datenschutzbeauftragte:r** | Compliance | Design‑Änderungen, Audit‑Reports | E‑Mail, Audit‑Meetings | Quartal |
| **Endnutzer:innen (Beta)** | Feedback | Produktänderungen, Surveys | E‑Mail, In‑App‑Banner | Bei Releases |
| **IT‑Betrieb** | Betrieb & Support | Incident‑Reports, Deploy‑Pläne | PagerDuty, Runbooks | Laufend |

### 13.2 Entwickler‑Onboarding, Schulung & Wissenssicherung

* **Onboarding‑Guide**: Schritt‑für‑Schritt‑Anleitung zur lokalen Einrichtung, Code‑Style, Branching, CI/CD【12†L39-L47】.
* **Schulungen**: Workshops zu OWASP, SLSA, Kubernetes & DevOps Tools; Pair‑Programming.
* **Knowledge Base**: Confluence/Wiki mit ADR‑Log, Runbooks, FAQs; regelmässige Updates.
* **Communities of Practice**: Kapitel über Security, DevOps, UX; wöchentliche Brown Bags.

## 14 Betrieb & Übergabe

### 14.1 Betriebs‑Checkliste & SLA/OLA

| Bereich | Checkliste | SLA/OLA |
| --- | --- | --- |
| **Monitoring** | Ist Prometheus / Grafana aktiv? Sind Alerts konfiguriert? | SLO Error Rate < 1 %, Alert Reaktionszeit < 15 min. |
| **Backup & Restore** | Werden DB‑Backups täglich erstellt und getestet? | RPO < 1 h; RTO < 4 h. |
| **Security** | Sind alle Secrets gültig und rotationiert? Führen wir periodische Pen‑Tests durch? | CVEs in Prod = 0; Secret‑Rotation alle 90 Tage. |
| **Compliance** | DSGVO‑Audit bestanden? Consent‑Logs vorhanden? | Einhaltung der Löschfristen; Dokumentation aktuell. |
| **Capacity** | Ist ausreichend Ressourcen‑Headroom vorhanden? | CPU & Memory Nutzung < 70 % (p95). |

### 14.2 Kontinuierliche Verbesserung (KVP) & Tech‑Debt

* **Incident Reviews**: Nach jedem Incident Postmortem; Aufgaben im Tech‑Debt Backlog.
* **Retrospektiven**: Jede Sprint‑Retro identifiziert Verbesserungen (Prozess, Technik).
* **Tech‑Debt Backlog**: Sichtbar im Jira; priorisiert nach Business‑Impact & Risiko.
* **Innovation Days**: Alle 8 Wochen; Team arbeitet an Prototypen oder refactored Legacy‑Teile.

## 15 Recherche & Evidence

### 15.1 Executive Summary der Rechercheergebnisse

* **Audit‑Highlights**: Der initiale Audit des bestehenden Repos zeigte instabile CI‑Pipelines, fehlende Tests und 28 offene Abhängigkeits‑PRs【19†L20-L28】【19†L35-L39】. Es gab gute Ansätze bei Security‑Scans und DSGVO‑Bewusstsein【50†L105-L113】【1†L118-L126】.
* **Moderne Best Practices**: Supply‑Chain‑Security wird nach SLSA Levels strukturiert【94236474580611†L619-L628】; Container‑Security Best Practices umfassen Hardening von Images, gesicherte Registries, TLS, RBAC und Netzpolitik【806704564751728†L391-L437】【806704564751728†L455-L468】.
* **Scrum‑Guidelines**: 2020 – 2025 gibt es keine neue Version; die 2020er‑Version bleibt gültig【606628840459431†L96-L107】.
* **Compliance**: ISO 27001 (2022) bietet Rahmen für Informationssicherheits‑Management【663759207839439†L140-L152】; DSGVO verlangt strikte Protokollierung & Minimierung【50†L151-L159】.

### 15.2 Evidence Log

| Evidence‑ID | Quelle | Datum | Scoring (0–5) | Confidence | Extrahierte Fakten |
| --- | --- | --- | --- | --- | --- |
| **E‑19a** | Audit Repo – Build fails & Dependabot PRs【19†L35-L39】【19†L20-L28】 | 2024 | 5 | High | Instabile Tests (TS‑Compile, PHP Tests) blockieren Pipeline; 28 Dependabot PRs offen. |
| **E‑50a** | DSGVO & Logging【50†L151-L159】【50†L105-L113】 | 2024 | 4 | High | Logs dürfen keine PII enthalten; Logs max. 30 Tage; Consent‑Mechanismen notwendig; Scans (Gitleaks) laufen. |
| **E‑42a** | Dependabot Groups【42†L45-L53】 | 2025‑06 | 4 | High | GitHub Dependabot unterstützt Gruppenupdates zur Reduktion der PR‑Flut. |
| **E‑57a** | SBOM & Supply‑Chain【57†L158-L165】 | 2024 | 4 | Medium | SBOMs sollen jede Release begleiten; transitive Abhängigkeiten sichtbar; unterstützt Compliance. |
| **E‑66a** | Containerisierung Vorteile【66†L39-L47】 | 2024 | 4 | Medium | Container bieten Isolation, reproduzierbare Deployments und Skalierbarkeit. |
| **E‑14a** | DevEx Verbesserungen【14†L35-L43】 | 2024 | 4 | High | Einführung von Husky & Pre‑Commit Hooks verbessert Code‑Qualität & Developer Experience. |
| **E‑606a** | Scrum Guide 2020 – 2025【606628840459431†L96-L107】 | 2025 | 5 | High | Offizieller Scrum Guide wurde 2020 aktualisiert; seitdem unverändert – gültig bis 2025. |
| **E‑942a** | SLSA Framework【94236474580611†L619-L628】 | 2025‑06 | 5 | High | SLSA definiert 4 Reifestufen; verifizierbare Build‑Provenance verhindert Tampering. |
| **E‑942b** | Pipeline Hardening & Logging【94236474580611†L660-L702】 | 2025‑06 | 5 | High | Pipeline‑Hardening: isolierte Build‑Umgebungen, secrets in Vault, Logging aller Build‑Schritte; scanning & alerts. |
| **E‑663a** | ISO 27001 Standard【663759207839439†L140-L152】 | 2025 | 3 | Medium | ISO 27001 definiert Anforderungen an ein Informationssicherheits‑Managementsystem. |
| **E‑806a** | Container Security Best Practices【806704564751728†L391-L437】【806704564751728†L455-L468】 | 2025 | 5 | High | Sicherung von Images (trustworthy images, minimal base), gesicherte Registries, TLS, RBAC, Netzwerk‑Isolation; runtime security. |
| **E‑50b** | Security Logging & Monitoring【50†L151-L159】 | 2024 | 4 | High | Logging anonymisieren, Retention < 30 Tage, Alerts definieren. |

**Quality‑Bewertung**: Quellen mit Score ≥ 4 gelten als **High Confidence**; 3 – 4 als **Medium**. Größere Entscheidungen (Architecture, Security) basieren nur auf High‑/Medium‑Confidence Quellen. Widersprüche: Keiner der Quellen widerspricht einer anderen; sie ergänzen sich. Ältere Quellen (> 2023) wurden nicht verwendet.

### 15.3 „Nicht übernommen & warum“

* **Plesk‑Hosting**: Obwohl im Audit als aktueller Zustand erwähnt, wurde Plesk nicht in die Zielarchitektur übernommen. Grund: Shared‑Hosting erschwert CI/CD, Containerisierung & Skalierung【9†L25-L33】.
* **Legacy‑PHP‑Bridges**: Alte PHP‑Dateien werden im Zielsystem ersetzt durch gesicherte API‑Proxy oder Microservice in Python.
* **Monorepo Beibehaltung**: Trotz Kopplungsproblemen im Audit wird zunächst das Monorepo beibehalten; Trennung in eigenständige Repos wird als optionaler Schritt für spätere Phasen betrachtet, um Komplexität zu vermeiden.
* **GraphQL‑Only API**: Aufgrund Team‑Kenntnissen und Einfachheit wurde ein REST‑First‑Ansatz gewählt; GraphQL ist optional.

## 16 Anhänge

### 16.1 Glossar

* **ADR (Architectural Decision Record)** – Dokumentiert eine Architekturentscheidung mit Kontext, Alternativen, Begründung und Konsequenzen.
* **CI/CD** – Continuous Integration / Continuous Delivery. Automatisierung der Softwarebereitstellung.
* **SBOM (Software Bill of Materials)** – Liste aller Komponenten (inkl. transitive Abhängigkeiten) eines Software‑Artefakts【57†L158-L165】.
* **SLA/SLO/SLI** – Service Level Agreement / Objective / Indicator: definierte Zielwerte für Servicequalität.
* **SLSA (Supply Chain Levels for Software Artifacts)** – Framework zur Absicherung der Software‑Lieferkette【94236474580611†L619-L628】.
* **STRIDE** – Threat‑Modeling‑Methode (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege).
* **KVP** – Kontinuierlicher Verbesserungsprozess.

### 16.2 Vorlagen & Checklisten

– siehe separate Dateien (Deployment Runbook, ADR‑Template, Testplan‑Template).

### 16.3 ADR‑Log (vollständig)

– Referenziert im Kapitel 3.2.

### 16.4 Revisionshistorie

| Version | Datum | Änderungen | Autor:in |
| --- | --- | --- | --- |
| **v1.0** | 30.09.2025 | Erstversion des Umsetzungsplans. | Projektteam |

### 16.5 Literaturverzeichnis / Quellen

Die Quellen sind inline zitiert; siehe Evidence‑Log (Kapitel 15).

## Action Sheet – Top 10 Prioritäten für die nächsten 14 Tage

| Nr. | Aufgabe | Begründung | Owner | Deadline |
| --- | --- | --- | --- | --- |
| **1** | Kickoff‑Meeting vorbereiten und durchführen | Projektbeginn, Scope klären | PO, PM | 15.10.2025 |
| **2** | Stakeholder‑Interviews planen | Anforderungen erfassen & Personas erstellen | BA, UX | 20.10.2025 |
| **3** | Repo‑Grundgerüst anlegen & Devcontainer bereitstellen | Entwickler:innen können produktiv starten【12†L39-L47】 | DevOps | 22.10.2025 |
| **4** | CI‑Pipeline initial erstellen (Lint & Test) | Verhindert instabile Builds【19†L35-L39】 | DevOps | 25.10.2025 |
| **5** | ADR‑Workshop ansetzen | Technische Leitplanken definieren; Security Anforderungen integrieren【94236474580611†L619-L628】 | Architect, Sec Eng | 28.10.2025 |
| **6** | Abhängigkeits‑Audit & Dependabot Groups konfigurieren | Sicherheitslücken früh schließen【42†L45-L53】 | Dev Lead, Sec Eng | 24.10.2025 |
| **7** | Datenschutz‑Folgenabschätzung starten | DSGVO‑Compliance sicherstellen【50†L151-L159】 | Datenschutzbeauftragte:r | 30.10.2025 |
| **8** | Onboarding‑Guide & Contributing Draft erstellen | Schnellere Einarbeitung neuer Teammitglieder【12†L39-L47】 | Tech Writer | 27.10.2025 |
| **9** | Setup Observability‑Stack (Prometheus, Grafana, Loki) im Dev‑Cluster | Frühzeitiges Monitoring ermöglicht schnelle Fehlererkennung【50†L151-L159】 | DevOps/SRE | 31.10.2025 |
| **10** | Threat‑Modeling Workshop durchführen | Risiken identifizieren & Controls planen【94236474580611†L619-L628】【50†L151-L159】 | Sec Eng, Dev Team | 01.11.2025 |