Planify

Webbasierte Anwendung zur Planung und Organisation von Veranstaltungen

Autoren: Marius Gutschalk, Nils Beck, Julian Blönnigen, Lasse Schillinger

---

Ziel

Ziel des Projekts **Planify** ist die Entwicklung einer webbasierten Anwendung, die die **Planung, Organisation und Koordination von Events** (z. B. Partys, Vereinsveranstaltungen, Meetings) unterstützt.
Das System ermöglicht mehreren Benutzern, Events gemeinsam zu verwalten, Aufgaben zu verteilen und die Kommunikation innerhalb eines Teams zu strukturieren.

Hauptziele:

- Effiziente Planung und Verwaltung von Events
- Zentrale Übersicht über Aufgaben, Teilnehmer und Termine
- Einfache Kommunikation zwischen Organisatoren und Teilnehmern
- Nachvollziehbare Abläufe durch Benutzerrollen und Statusanzeigen

---

Vorteil

Planify bietet im Gegensatz zu klassischen Tabellen oder Messenger-Gruppen eine **strukturierte, benutzerfreundliche Plattform**, die alle relevanten Informationen zu einem Event zentralisiert.

Vorteile für Nutzer:

- Keine chaotischen WhatsApp-Nachrichten oder E-Mail-Ketten mehr
- Klare Aufgabenverteilung und Verantwortlichkeiten
- Jederzeit abrufbarer Eventstatus
- Bessere Nachvollziehbarkeit von Änderungen und Zuständigkeiten

Vorteile für das Projektteam:

- Klare Architektur mit Trennung von Frontend, Backend und Datenbank
- Dokumentierbare Prozesse für Analyse, Design, Implementierung und Tests
- Gut skalierbare Basis für spätere Erweiterungen (z. B. Kalender-API, Benachrichtigungen)

Metriken

Zur Erfolgsmessung werden folgende **Metriken** definiert:

Kategorie	Metrik	Zielwert
Funktionalität	Anzahl implementierter Kernfunktionen (Event, Aufgaben, Benutzer, Kommentare)	≥ 90 % der spezifizierten Anforderungen
Benutzerfreundlichkeit	Durchschnittliche Navigationsschritte bis zur Zielaktion	≤ 3 Schritte
Performance	Durchschnittliche Antwortzeit des Servers	≤ 500 ms
Zuverlässigkeit	Fehlerrate bei Standard-Use-Cases	≤ 2 %
Teamkoordination	Erfüllte Meilensteine im Zeitplan	≥ 95 %
Codequalität	Linting-Fehler pro 1.000 Zeilen Code	≤ 5

Rahmenbedingungen

Standards für Produkte und Systeme

- **Softwarearchitektur:** Client-Server-Modell (REST-API)
- **Programmiersprachen:**
  - Frontend: React (JavaScript/TypeScript)
  - Backend: Node.js (Express) oder Java (Spring Boot)
- **Datenbank:** PostgreSQL oder MySQL
- **Versionsverwaltung:** Git / GitHub
- **Dokumentation:** Markdown, UML-Diagramme (PlantUML), PDF-Berichte
- **Teststandards:** Unit-Tests (Jest/JUnit), Integrationstests mit Postman
- **Stakeholder:** Hochschule(DHBW), Universitäten, Unternehemn, Privatpersonen

---

Rechtliche Bestimmungen

- **Datenschutz:** DSGVO-konforme Speicherung von Benutzerdaten
  - Passwörter werden gehasht (z. B. bcrypt)
  - Keine unnötige Speicherung personenbezogener Daten
- **Lizenzierung:** Open-Source-Lizenzen (z. B. MIT oder Apache 2.0) für externe Bibliotheken
- **Urheberrecht:** Quellcode und Inhalte sind Eigenleistung des Projektteams

---

Projekt- und Produktgegner

- Konkurrenzprodukte wie Google Calendar oder Trello

Produktbudget

Da es sich um ein Hochschulprojekt handelt, wird **kein externes Budget** benötigt.
Kosten fallen nur für:

- Entwicklungsumgebung (lokal, kostenlos)
- Optionale Hosting-Kosten (z. B. Render, Railway, Vercel – meist mit Free-Tier)

Gesamtkosten: < **50 €** (nur bei optionalem Hosting oder Domain)

Zeitliche Rahmenbedingungen

- **Projektlaufzeit:** 2 Semester / ca. **6 Monate**
- **Phasen:**
    1. Anforderungsanalyse (2 Wochen)
    2. Systementwurf & Architekturplanung (3 Wochen)
    3. Implementierung (8 Wochen)
    4. Testphase & Fehlerbehebung (4 Wochen)
    5. Dokumentation & Präsentation (3 Wochen)

Puffer: 2–3 Wochen für unerwartete Bugs, Gruppenausfälle oder Realitätszusammenbrüche

---

Risiken

| Risiko | Wahrscheinlichkeit | Beschreibung | Gegenmaßnahme | |--------|---------------|--------------| | **Teamkoordination** | 90% | Unklare Aufgabenverteilung oder Kommunikationsprobleme | Wöchentliche Meetings, Aufgabenverwaltung in GitHub Projects | | **Technische Komplexität** | 60% | Schwierigkeiten mit Frameworks oder Datenbankmodell | Frühzeitiger Prototyp, regelmäßige Code-Reviews | | **Zeitmangel** | 10% | Überschneidung mit Prüfungsphasen oder anderen Projekten | Fester Zeitplan mit Meilensteinen | | **Versionskonflikte** | 100% | Merge-Konflikte im Git-Repository | Branch-Strategie (z. B. GitFlow) | | **Datenverlust** | 70% | Fehlerhafte Migration oder Drop der DB | Regelmäßige Backups | | **Motivationsverlust** | 100% | Teammitglieder verlieren Interesse | Aufgabenrotation, Fortschrittspräsentationen | | **Mitgliederverlust** | 50% | Exmatrikulation eines Mitglieds | Mehr Lernen, Aufgaben gleich verteilen |

ToGo / NotToGo

ToGo:

- Anforderungen sind realistisch und klar definiert

- Team besitzt notwendige technische Kenntnisse
- Entwicklungsumgebung und Tools stehen fest
- Zeitrahmen und Dokumentationsplan vorhanden

NotToGo:

- Kein funktionsfähiges Grundsystem nach der Hälfte der Projektzeit
- Teamkommunikation zusammengebrochen
- Anforderungen ändern sich fundamental
- Technische Kernkomponenten (z. B. Datenbank oder Auth) scheitern dauerhaft