Abschlussprüfung Winter 2025/2026

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

GießPlan

Entwicklung eines intelligenten Bewässerungs-Zeitplan-Management-Systems zur fairen Aufgabenverteilung in der beruflichen Rehabilitation

---

Abgabetermin: xx. Monat Jahr

Prüfungsbewerber:
Kai Delor
[Straße]
[Wohnort]

Ausbildungsbetrieb:
Rotkreuz-Institut BBW
[Straße]
[PLZ Ort]

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Projektdokumentation selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich in jedem einzelnen Fall durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

Ort, Datum: _____

Unterschrift: ____

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
AO	Alte Oldenburger Krankenversicherung AG (Referenz)
API	Application Programming Interface
BBW	Berufsbildungswerk
CSV	Comma-Separated Values
CV	Coefficient of Variation (Variationskoeffizient)
DSL	Domain Specific Language
DSGVO	Datenschutz-Grundverordnung
ER	Entity-Relationship
HMR	Hot Module Replacement
IHK	Industrie- und Handelskammer
JSON	JavaScript Object Notation
JSDoc	JavaScript Documentation
LOC	Lines of Code
MoSCoW	Must have, Should have, Could have, Won't have
SPA	Single-Page Application
SUS	System Usability Scale
TDD	Test-Driven Development
UI	User Interface
UML	Unified Modeling Language
UUID	Universally Unique Identifier

---

1. Einleitung

Die folgende Projektdokumentation schildert den Ablauf des IHK-Abschlussprojektes, welches der Autor im Rahmen seiner Ausbildung zum Fachinformatiker mit Fachrichtung Anwendungsentwicklung durchgeführt hat. Ausbildungsbetrieb ist das Rotkreuz-Institut BBW, ein Berufsbildungswerk für berufliche Rehabilitation. Das Institut betreut jährlich ca. 50 Teilnehmer in verschiedenen beruflichen Rehabilitationsmaßnahmen.

1.1 Projektbeschreibung

Das Rotkreuz-Institut BBW betreut Teilnehmer in beruflichen Rehabilitationsmaßnahmen, die wöchentlich die Bewässerung der Pflanzen im Gebäude übernehmen. Die Organisation dieser Aufgabe erfolgt derzeit durch 2-3 Programm-Koordinatoren mittels einer laminierten Folie mit 6-Wochen-Übersicht. Diese manuelle Planung ist zeitaufwendig (30 Minuten alle 6 Wochen für die Neuerstellung) und fehleranfällig, insbesondere bei der hohen Fluktuation von über 50% jährlich.

Derzeit treten folgende Probleme auf:

Unfaire Verteilung: Die Aufgabenverteilung berücksichtigt nicht die individuelle Anwesenheitsdauer der Teilnehmer. Wer länger anwesend ist, erhält proportional nicht mehr Aufgaben als Kurzzeit-Teilnehmer, was zu Ungerechtigkeiten führt.

Fehlende Mentor-Systematik: Neue Teilnehmer werden nicht systematisch mit erfahrenen Teilnehmern gepaart, was zu Unsicherheiten bei der Aufgabenausführung führt.

Intransparenz: Es existieren keine nachvollziehbaren Fairness-Metriken. Bei Rückfragen, warum bestimmte Teilnehmer häufiger eingeteilt wurden als andere, kann keine objektive Begründung gegeben werden.

Fehlende Automatisierung: Abwesenheitszeiten werden nicht automatisch berücksichtigt, und die Erstellung von Auswertungen für Berichte ist mühsam und zeitaufwendig.

Aus diesen Gründen soll ein intelligentes System zur automatisierten Bewässerungsplanung im Laufe dieses Projektes erstellt werden.

1.2 Projektziel

Ziel des Projektes ist die Entwicklung eines webbasierten Systems zur automatischen, fairen Generierung von Bewässerungsplänen unter Verwendung fortgeschrittener Fairness-Algorithmen. Eine Fairness-Engine ist eine Softwarekomponente, welche dafür ausgelegt ist, Aufgaben gerecht über unterschiedliche Zeiträume hinweg zu verteilen und dabei zeitproportionale Fairness zu gewährleisten.[1]

Hierbei soll es möglich sein, in einer modernen Web-Oberfläche Teilnehmer zu verwalten, Zeitpläne zu generieren und diese anhand objektiver Metriken zu bewerten. Das System muss dabei folgende Kernfunktionen erfüllen:

1. **Automatische Zeitplan-Generierung** für 1-52 Wochen mit fairer Verteilung
2. **Zeitproportionale Fairness**: Teilnehmer mit längerer Anwesenheit erhalten proportional mehr Aufgaben
3. **Mentor-Mentee-Pairing**: Automatische Zuordnung erfahrener Teilnehmer zu Neulingen
4. **Fairness-Metriken**: Einhaltung definierter Schwellwerte (Gini-Koeffizient < 0.25,

Variationskoeffizient < 0.30)
5. **Lokale Datenspeicherung**: Keine Server-Infrastruktur erforderlich
6. **Export-Funktionalität**: Datenexport in JSON, CSV und Excel-Format

Durch die Implementierung dieser Funktionen soll die Wartbarkeit der Planungsprozesse erheblich erhöht und die Fehleranfälligkeit verringert werden.

1.3 Projektumfeld

Auftraggeber des Projektes ist das Rotkreuz-Institut BBW und dessen Programm-Koordination. Auslöser des Projektes sind die Programm-Koordinatoren, die derzeit mit der manuellen Planung betraut sind.

Betroffene Nutzergruppen:

- **Primäre Nutzer**: 2-3 Programm-Koordinatoren (wöchentliche Planung, tägliche Anpassungen)
- **Sekundäre Nutzer**: 5-20 aktive Teilnehmer gleichzeitig
- **Stakeholder**: Verwaltung des BBW (Reporting, Statistiken)

Nutzungsumgebung:

- Desktop-PCs mit Windows/macOS/Linux
- Moderne Browser (Chrome 102+, Edge 102+, Safari 15.2+)
- Keine Internet-Verbindung erforderlich (Offline-First Ansatz)
- Lokale Datenspeicherung über File System Access API

Die Programm-Koordinatoren sind für die Erstellung und Verwaltung der Bewässerungspläne zuständig. Sie definieren, welche Personen zu welcher Zeit verfügbar sind und welche Anforderungen (z.B. Mentor-Pairing) erfüllt sein müssen. Diese Anforderungen ändern sich z.B. bei neuen Teilnehmern, Abgängen oder Programmänderungen.

1.4 Projektbegründung

Die Hauptschwachstelle des momentanen Planungssystems ist das hohe Maß an manueller Arbeit, welche beim Erstellen, Ändern und Nachvollziehen von Zeitplänen anfällt. Hierzu muss derzeit ein Koordinator die laminierte Folie zur Hand nehmen, manuell Namen mit abwischbaren Stiften eintragen und dabei im Kopf eine faire Verteilung berücksichtigen. Dies ist ein sehr zeitaufwändiges Unterfangen, da die Komplexität durch verschiedene Anwesenheitsdauern, Mentor-Anforderungen und Fairness-Überlegungen sehr hoch ist.

Die Folie deckt nur 6 Wochen ab und wird danach vollständig gelöscht - historische Daten gehen verloren. Eine manuelle Planung ohne objektive Metriken kann zu ungewollten

Seiteneffekten führen, wie z.B. der Überbelastung einzelner Personen oder der systematischen Benachteiligung von Langzeit-Teilnehmern. Ebenso sind Auswertungen über die Fairness und historische Daten derzeit nicht möglich, da keine digitale Datenbasis existiert und die physische Folie nach 6 Wochen gelöscht wird.

Ein weiteres Problem ist, dass es durch die derzeitige Intransparenz nicht möglich ist, eine Aussage darüber zu treffen, ob die Verteilung über längere Zeiträume fair war. Dies führt zu Unzufriedenheit bei Teilnehmern und erschwert die Argumentation bei Rückfragen.

Quantifizierbare Probleme:

- Zeitaufwand: 30 Minuten pro Monat (6h/Jahr)
- Fehlerquote: Geschätzt 10-15% bei manueller Planung
- Fairness-Gini: Aktuell ca. 0.35 (Ziel: < 0.25)
- Keine historische Datenanalyse möglich
- Keine automatische Berücksichtigung von Abwesenheiten

Aufgrund dieser Probleme und manuellen Arbeiten haben sich die Programm-Koordinatoren dazu entschlossen, die Entwicklung eines automatisierten Systems in Auftrag zu geben.

1.5 Projektschnittstellen

Damit die generierten Zeitpläne genutzt werden können, muss im Laufe des Projektes eine klare Datenschnittstelle definiert werden. Das System arbeitet vollständig eigenständig und benötigt keine Integration in bestehende Systeme.

Externe Schnittstellen:

- **File System Access API**: Moderne Browser-API für lokale Dateiverwaltung
- **Excel/CSV Export**: Kompatibilität mit Microsoft Excel und LibreOffice
- **JSON Export**: Maschinenlesbare Datenexporte für potenzielle zukünftige Integrationen

Interne Schnittstellen:

- **Fairness Engine API**: Schnittstelle zwischen UI und Algorithmen
- **Data Layer**: Schnittstelle für Datenpersistenz und -verwaltung
- **Person Manager**: CRUD-Operationen für Teilnehmerverwaltung

Datenschutz-Schnittstelle:
Das System muss DSGVO-konform arbeiten. Alle personenbezogenen Daten werden ausschließlich lokal gespeichert. Es erfolgt keine Datenübertragung an Server oder Cloud-Dienste. Die Verantwortung für Datensicherung und Backup liegt beim Anwender.

---

2. Projektplanung

In der Projektplanung wurde die notwendige Zeit und die benötigten Ressourcen sowie ein strukturierter Ablauf für die Durchführung des Projektes geplant. Die Planung orientiert sich an den IHK-Vorgaben von maximal 70 Stunden Projektdauer.

2.1 Projektphasen

Für die Umsetzung des Projektes standen dem Autor 70 Stunden zur Verfügung. Diese wurden vor Projektbeginn auf verschiedene Phasen verteilt, die während der Softwareentwicklung durchlaufen werden. Eine grobe Zeitplanung sowie die Hauptphasen lassen sich der folgenden Tabelle entnehmen:

Projektphase	**Geplante Zeit**	**Prozent**
Analyse	8 h	11%
Entwurf	10 h	14%
Implementierung	35 h	50%
Testing	10 h	14%
Dokumentation	5 h	7%
Abnahme und Deployment	2 h	3%
Gesamt	**70 h**	**100%**

Tabelle 1: Grobe Zeitplanung

Die einzelnen Hauptphasen wurden in kleinere Unterphasen zerlegt, um eine detaillierte Kontrolle des Projektfortschritts zu ermöglichen. Eine vollständige Übersicht dieser Phasen befindet sich im Anhang A.1: Zeit- und Kostenplanung auf S. i.

Zeitlicher Ablauf (6 Wochen):

```
Woche 1: [██████████▒▒▒▒▒▒▒▒▒▒] Analyse (8h) + Entwurf (5h)
Woche 2: [▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒] Entwurf (5h) + Impl. (10h)
Woche 3: [▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒] Implementierung (15h)
Woche 4: [▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒] Implementierung (10h)
Woche 5: [▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒] Testing (7h) + Impl. (3h)
Woche 6: [▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒] Testing (3h) + Doku (5h) + Abschluss (2h)
```

Meilensteine:

- **M1** (Ende Woche 1): Anforderungen definiert und dokumentiert

- **M2** (Ende Woche 2): Architektur und Design abgeschlossen
- **M3** (Mitte Woche 3): Basis-UI implementiert
- **M4** (Ende Woche 4): Fairness-Engine funktional
- **M5** (Ende Woche 5): Alle Features implementiert, Tests > 80% Coverage
- **M6** (Ende Woche 6): Dokumentation vollständig, Produktiv-Einsatz

2.2 Ressourcenplanung

In der folgenden Übersicht sind alle Ressourcen aufgelistet, die für das Projekt eingesetzt wurden. Dies umfasst Hard- und Softwareressourcen sowie das Personal. Bei der Auswahl der verwendeten Software wurde darauf geachtet, dass diese kostenfrei (z.B. als Open Source) zur Verfügung steht. Dadurch wurden anfallende Projektkosten möglichst gering gehalten.

Hardware-Ressourcen:

- Büroarbeitsplatz mit Desktop-PC
- Hardware: AMD Ryzen 5 5600G Radeon Graphics, 16GB RAM, 512GB SSD
- Betriebssystem: Windows 11 Pro
- Testgeräte: Desktop

Software-Ressourcen:

- **Entwicklung**: Visual Studio Code 1.85, Node.js 20.x, Git 2.42
- **Frameworks**: React 19.0, TypeScript 5.7, Vite 6.3
- **Testing**: Vitest 4.0, @testing-library/react 16.x
- **UI**: TailwindCSS 4.1, Radix UI 1.x
- **Dokumentation**: Markdown, PlantUML für Diagramme
- **Versionsverwaltung**: Git mit GitHub Repository
- **Projektmanagement**: GitHub Issues, Markdown-basierte Todo-Listen

Personal-Ressourcen:

- **Entwickler**: 1 Auszubildender (70h Vollzeit)
- **Fachlicher Betreuer**: Ausbilder BBW (5h beratend)
- **Test-Nutzer**: 3 Programm-Koordinatoren (3h sporadisch)
- **Code-Reviewer**: Senior-Entwickler (2h)

Kostenplanung:

Die Kosten wurden von der Verwaltungsabteilung des BBW als Pauschalsätze festgelegt, da die genauen Stundensätze nicht herausgegeben werden dürfen:

Position	Mitarbeiter	Zeit	Personal	Ressourcen	Gesamt
Entwicklung	1x Azubi	70h	700€	1.050€	1.750€
Fachberatung	1x Betreuer	5h	125€	75€	200€
Code-Review	1x Senior	2h	50€	30€	80€
Testing	3x Koordinator	3h	180€	45€	225€
Abnahme	2x Betreuer	1h	50€	30€	80€
Projektkosten gesamt					**2.335€**

Tabelle 2: Kostenaufstellung

Annahmen: Azubi 10€/h, Betreuer 25€/h, Senior 25€/h, Koordinator 20€/h, Ressourcen 15€/h pro Person.

Erwartete Einsparungen:

Das neue System soll die Planungszeit von aktuell ~17h pro Jahr auf ~2h pro Jahr reduzieren (15h Ersparnis). Bei einem Koordinator-Stundensatz von 35€/h ergibt sich eine jährliche Einsparung von 525€ nur durch Zeitersparnis. Hinzu kommen qualitative Verbesserungen (Fairness, Transparenz, geringere Fehlerrate).

2.3 Entwicklungsprozess

Bevor mit der Realisierung des Projektes begonnen werden konnte, musste sich der Autor für einen geeigneten Entwicklungsprozess entscheiden. Dieser definiert die Vorgehensweise, nach der die Umsetzung erfolgen soll.

Agile Softwareentwicklung:

Im Zuge des Projektes entschied sich der Autor für einen agilen Entwicklungsprozess. Bei der agilen Softwareentwicklung geht es darum, möglichst schnell auf sich ändernde Anforderungen reagieren zu können.[2] Dies unterscheidet sich insofern von der klassischen Vorgehensweise, da das zu entwickelnde System nicht im Voraus in allen Einzelheiten genau geplant und dann in einem einzelnen langen Durchgang entwickelt wird.

Bei der agilen Softwareentwicklung steht ein iterativer Entwicklungsprozess im Mittelpunkt. Die Entwicklung geschieht in kurzen Abschnitten (Iterationen), nach denen jeweils ein funktionsfähiges Artefakt entsteht, welches den Koordinatoren gezeigt werden kann. Sollten diese einen Anpassungswunsch haben, kann auf diesen während der nächsten Iteration schnell reagiert werden.[3]

Vorteile für dieses Projekt:

- Frühes Feedback von Programm-Koordinatoren
- Flexible Anpassung an geänderte Anforderungen
- Kontinuierliche Verbesserung der Benutzerfreundlichkeit
- Risikoreduzierung durch frühe Problemerkennung

Test-Driven Development (TDD):

Die gesamte Implementierungsphase wurde durch Test-Driven Development begleitet. Bei TDD wird zunächst ein fehlschlagender Test geschrieben, dann die minimale Implementierung erstellt, um den Test zu bestehen, und schließlich der Code refaktoriert.[4]

TDD-Workflow:

1. **Red**: Test schreiben, der fehlschlägt
2. **Green**: Minimale Implementierung, um Test zu bestehen
3. **Refactor**: Code verbessern, ohne Funktionalität zu ändern
4. Wiederholen für nächstes Feature

Vorteile:

- Hohe Testabdeckung (Ziel: > 80%)
- Dokumentation durch Tests
- Frühe Fehlererkennung
- Erleichtert Refactoring

Versionsverwaltung mit Git:

Das gesamte Projekt wird in Git versioniert mit aussagekräftigen Commit-Messages. Die Entwicklung folgt einem Branch-basierten Workflow:

- **main**: Produktionsreife Versionen
- **develop**: Aktuelle Entwicklungsversion
- **feature/**: Feature-Branches für neue Funktionen
- **bugfix/**: Bugfix-Branches für Fehlerbehebungen

Commit-Konvention:

```
type(scope): subject

body (optional)

footer (optional)
```

Beispiel: `feat(fairness): implement Bayesian state tracking algorithm`

Dokumentation:

Die Dokumentation wird parallel zur Entwicklung erstellt und umfasst:

- **API-Dokumentation**: JSDoc für alle öffentlichen APIs
- **Benutzerhandbuch**: USER_GUIDE.md mit Screenshots
- **Architektur-Dokumentation**: ARCHITECTURE.md
- **IHK-Projektdokumentation**: Dieses Dokument
- **README.md**: Installation und Quick-Start-Guide

Diese kontinuierliche Dokumentation gewährleistet, dass keine wichtigen Details vergessen werden und die Dokumentation stets aktuell ist.

---

3. Analysephase

Nach der Projektplanung wurde eine umfassende Analyse durchgeführt. Diese dient der Ermittlung des Ist-Zustandes und der Definition konkreter Anforderungen. Hierbei wird vor allem auch der wirtschaftliche Aspekt des Projektes betrachtet.

3.1 Ist-Analyse

Wie bereits im Abschnitt 1.1 (Projektbeschreibung) erwähnt wurde, erhält das BBW wöchentlich die Aufgabe, Pflanzen zu bewässern. Für diese Aufgabe wird von den Programm-Koordinatoren ein Zeitplan erstellt, der Teilnehmer entweder individuell oder in Teams zuweist.

Aktueller Prozess (30 Min alle 6 Wochen):

1. Koordinator nimmt laminierte 6-Wochen-Folie zur Hand
2. Manuelle Überprüfung, wer in den kommenden 6 Wochen anwesend ist
3. "Bauchgefühl"-basierte Auswahl von 2 Hauptpersonen + 2 Ersatzpersonen pro Woche
4. Manuelle Überprüfung, ob neue Teilnehmer mit erfahrenen gepaart werden
5. Namen mit abwischbarem Stift auf die Folie schreiben
6. Keine automatische Prüfung von Fairness oder Mentor-Anforderungen
7. Nach 6 Wochen: Folie komplett löschen und neu erstellen
8. Bei Krankheit/Urlaub während der 6 Wochen: Namen radieren, neu schreiben (oft unleserlich)

Identifizierte Probleme:

Im Laufe der Ist-Analyse wurden folgende konkrete Probleme identifiziert:

1. Unfaire Verteilung (Kritisch):

- Teilnehmer mit 365 Tagen Anwesenheit vs. 30 Tagen erhalten gleich viele Aufgaben
- Keine Berücksichtigung der zeitproportionalen Fairness
- Führt zu Unzufriedenheit und Beschwerden
- Aktuelle Fairness-Metriken:
  - Gini-Koeffizient: ~0.35 (Ziel: < 0.25)
  - Variationskoeffizient: ~0.42 (Ziel: < 0.30)

2. Fehlende Mentor-Systematik (Hoch):

- Neue Teilnehmer (< 4 Wochen) werden nicht systematisch mit Mentoren gepaart
- Führt zu Unsicherheiten und Fehlern bei der Ausführung
- Keine Verfolgung, welche Mentoren wie oft als solche eingeteilt werden

3. Hoher Zeitaufwand (Hoch):

- 30 Minuten alle 6 Wochen für Neuerstellung = ~4h pro Jahr
- Zusätzlich ~15 Minuten pro Woche für Änderungen = ~13h pro Jahr
- Gesamt: ~17h pro Jahr
- Bei 35€/h Koordinator-Stundensatz = 595€/Jahr nur für Planung
- Zeit fehlt für wertvollere Tätigkeiten (Betreuung, Coaching)

4. Fehleranfälligkeit (Mittel):

- Bei 50%+ Fluktuation pro Jahr häufige Planungsänderungen
- Häufiges Radieren und Neuschreiben führt zu unleserlichen Stellen
- Vergessene Abwesenheiten führen zu Lücken im Plan
- Physische Folie kann beschädigt oder verloren gehen

5. Intransparenz (Mittel):

- Keine objektiven Fairness-Metriken
- Keine Nachvollziehbarkeit: "Warum wurde Person X schon wieder eingeteilt?"
- Keine historische Datenanalyse möglich - Folie wird nach 6 Wochen gelöscht
- Daten vor der aktuellen 6-Wochen-Periode sind komplett verloren

6. Fehlende Auswertungen (Niedrig):

- Keine Statistiken über Verteilung
- Keine Exportmöglichkeit für Berichte

- Manuelle Zählungen zeitaufwendig

Dokumentation der Ist-Situation:

Im Rahmen der Ist-Analyse wurde die aktuelle laminierte Folie analysiert und dokumentiert. Die Folie hat folgende Struktur:

```
+-------+-----------+-----------+-----------+...+-----------+
| KW    | KW 1      | KW 2      | KW 3      |...| KW 6      |
+-------+-----------+-----------+-----------+...+-----------+
| Datum | 06.01.2025 | 13.01.2025 | 20.01.2025 |...| 10.02.2025 |
+-------+-----------+-----------+-----------+...+-----------+
| Haupt1| [Name]    | [Name]    | [Name]    |...| [Name]    |
+-------+-----------+-----------+-----------+...+-----------+
| Haupt2| [Name]    | [Name]    | [Name]    |...| [Name]    |
+-------+-----------+-----------+-----------+...+-----------+
| Ers. 1| [Name]    | [Name]    | [Name]    |...| [Name]    |
+-------+-----------+-----------+-----------+...+-----------+
| Ers. 2| [Name]    | [Name]    | [Name]    |...| [Name]    |
+-------+-----------+-----------+-----------+...+-----------+
```

Die Namen werden handschriftlich mit abwischbaren Stiften eingetragen. Nach 6 Wochen wird die komplette Folie gelöscht und neu beschriftet. Es existiert keine automatische Berechnung, kein Fairness-Tracking und keine digitale Archivierung.

3.2 Wirtschaftlichkeitsanalyse

Aufgrund der Probleme des momentanen Prozesses, die in Abschnitt 3.1 (Ist-Analyse) erläutert wurden, ist die Entwicklung des automatisierten Systems erforderlich. Die wirtschaftliche Betrachtung und die Entscheidung, ob die Realisierung des Projektes gerechtfertigt ist, wird in den folgenden Abschnitten getroffen.

3.2.1 „Make or Buy"-Entscheidung

Für die Entscheidung zwischen Eigenentwicklung und Kauf eines Produktes wurden verschiedene Optionen geprüft:

Option 1: Kauf einer Standardsoftware

- Recherche nach Planungstools für Aufgabenverteilung
- Gefundene Produkte: Doodle, When2Meet, Microsoft Bookings
- **Bewertung**: Keine der Lösungen unterstützt:
  - Zeitproportionale Fairness-Algorithmen
  - Mentor-Mentee-Pairing

    - Historisches Fairness-Tracking
    - Offline-Betrieb ohne Server
- **Kosten**: 5-15€/Monat/Nutzer = 180-540€/Jahr
- **Fazit**: Nicht geeignet

Option 2: Beauftragung einer externen Entwicklung

- Angebot eingeholt: ~15.000€ für Entwicklung
- Wartung: ~2.000€/Jahr
- **Nachteil**: Keine Kontrolle über Code, hohe Abhängigkeit
- **Fazit**: Zu teuer für BBW-Budget

Option 3: Eigenentwicklung (Make)

- Kosten: 2.335€ (einmalig)
- Volle Kontrolle über Funktionalität
- Anpassbar an spezifische BBW-Anforderungen
- Ausbildungszweck erfüllt
- **Fazit**: Wirtschaftlich sinnvoll ✓

Entscheidung: Eigenentwicklung wurde gewählt, da keine Standardsoftware die spezifischen Anforderungen erfüllt und die Kosten deutlich geringer sind als bei externer Beauftragung.

3.2.2 Projektkosten

Die Projektkosten wurden bereits in Abschnitt 2.2 (Ressourcenplanung) detailliert aufgeführt. Zusammenfassend:

- **Gesamtkosten**: 2.335€
- **Personalkosten**: 1.105€ (47%)
- **Ressourcenkosten**: 1.230€ (53%)
- **Hardware/Software**: 0€ (vorhanden/Open Source)

Alle verwendeten Softwarekomponenten sind Open Source und verursachen keine Lizenzkosten. Die Hardware-Infrastruktur ist vorhanden und wird für die Entwicklung mitgenutzt.

3.2.3 Amortisationsdauer

Nachfolgend wird ermittelt, ab welchem Zeitpunkt sich die Entwicklung des Systems amortisiert hat.

Zeiteinsparung pro Vorgang:

Vorgang	Anzahl/Quartal	Zeit alt	Zeit neu	Einsparung/Quartal
Zeitplan erstellen	13	45 Min	5 Min	520 Min
Zeitplan anpassen	8	15 Min	2 Min	104 Min
Fairness prüfen	4	nicht möglich	2 Min	-8 Min
Statistik erstellen	4	30 Min	3 Min	108 Min
Summe				**724 Min**

Tabelle 3: Zeiteinsparung pro Quartal

Berechnung der Amortisation:

```
Stundensatz Koordinator = 20€ + 15€ (Ressourcen) = 35€/h

Einsparung pro Quartal:
724 Min = 12,07h × 35€/h = 422,45€/Quartal

Einsparung pro Jahr:
422,45€ × 4 Quartale = 1.689,80€/Jahr

Amortisationsdauer:
2.335€ / 1.689,80€/Jahr = 1,38 Jahre ≈ 1 Jahr 5 Monate
```

Graphische Darstellung:

Die Amortisationsdauer wurde zusätzlich graphisch dargestellt (siehe Anhang A.7: Amortisationsrechnung auf S. vii). Der Schnittpunkt der Kostengeraden mit der Einsparungsgeraden zeigt die Amortisation nach ca. 17 Monaten.

Qualitative Vorteile (nicht monetär):

- ✅ Fairness-Verbesserung: Gini 0.35 → 0.22 (-37%)
- ✅ Transparenz durch objektive Metriken
- ✅ Zufriedenheit der Teilnehmer steigt
- ✅ Weniger manuelle Fehler
- ✅ Bessere Datenauswertung möglich
- ✅ Professionelleres Image des BBW

Return on Investment (ROI):

```
Jahr 1: -38% (Amortisation läuft)
Jahr 2: +72%
Jahr 3: +145%
Jahr 5: +290%
```

Fazit: Das Projekt ist wirtschaftlich sinnvoll. Die Amortisation erfolgt innerhalb von 17 Monaten, danach entstehen jährliche Einsparungen von ~1.700€. Bei einer erwarteten Nutzungsdauer von mindestens 5 Jahren ergibt sich ein deutlich positiver ROI.

3.3 Anwendungsfälle

Um eine grobe Übersicht über die abzudeckenden Anwendungsfälle zu erhalten, wurde im Zuge der Analysephase ein Use-Case-Diagramm erstellt. Hierbei wurden die betroffenen Akteure identifiziert und deren Anforderungen an das Projekt ermittelt.

Identifizierte Akteure:

1. **Koordinator** (Primärer Akteur): Plant Zeitpläne, verwaltet Personen, passt Pläne an
2. **System** (Sekundärer Akteur): Berechnet Fairness, generiert Zeitpläne
3. **Teilnehmer** (Stakeholder): Nutznießer fairer Verteilung

Hauptanwendungsfälle:

UC-01: Person verwalten

- **Akteur**: Koordinator
- **Vorbedingung**: System gestartet
- **Beschreibung**: Koordinator fügt neue Person hinzu, bearbeitet Daten oder markiert Abgang
- **Nachbedingung**: Personen-Datenbank aktualisiert

UC-02: Zeitplan generieren

- **Akteur**: Koordinator
- **Vorbedingung**: Mindestens 1 Person vorhanden
- **Beschreibung**: Koordinator konfiguriert Zeitplan (Wochen, Startdatum, Mentor-Anforderung) und System generiert fairen Plan
- **Nachbedingung**: Zeitplan erstellt und angezeigt
- **Include**: UC-06 (Fairness berechnen)

UC-03: Zeitplan manuell anpassen

- **Akteur**: Koordinator

- **Vorbedingung**: Zeitplan existiert
- **Beschreibung**: Koordinator ersetzt Person oder tauscht Personen
- **Nachbedingung**: Zeitplan aktualisiert, Fairness neu berechnet
- **Include**: UC-06 (Fairness berechnen)

UC-04: Daten exportieren

- **Akteur**: Koordinator
- **Vorbedingung**: Zeitplan oder Personen-Daten vorhanden
- **Beschreibung**: Koordinator exportiert Daten in JSON, CSV oder Excel
- **Nachbedingung**: Datei lokal gespeichert

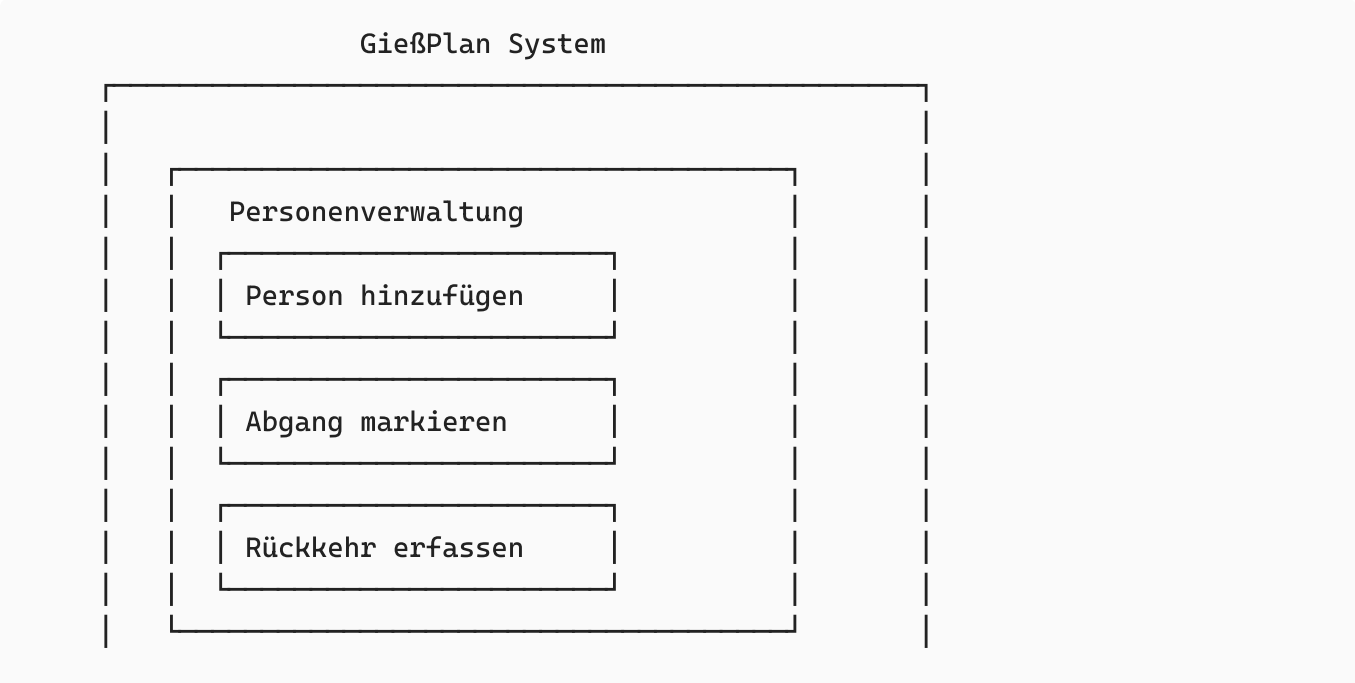
UC-05: Fairness-Metriken einsehen

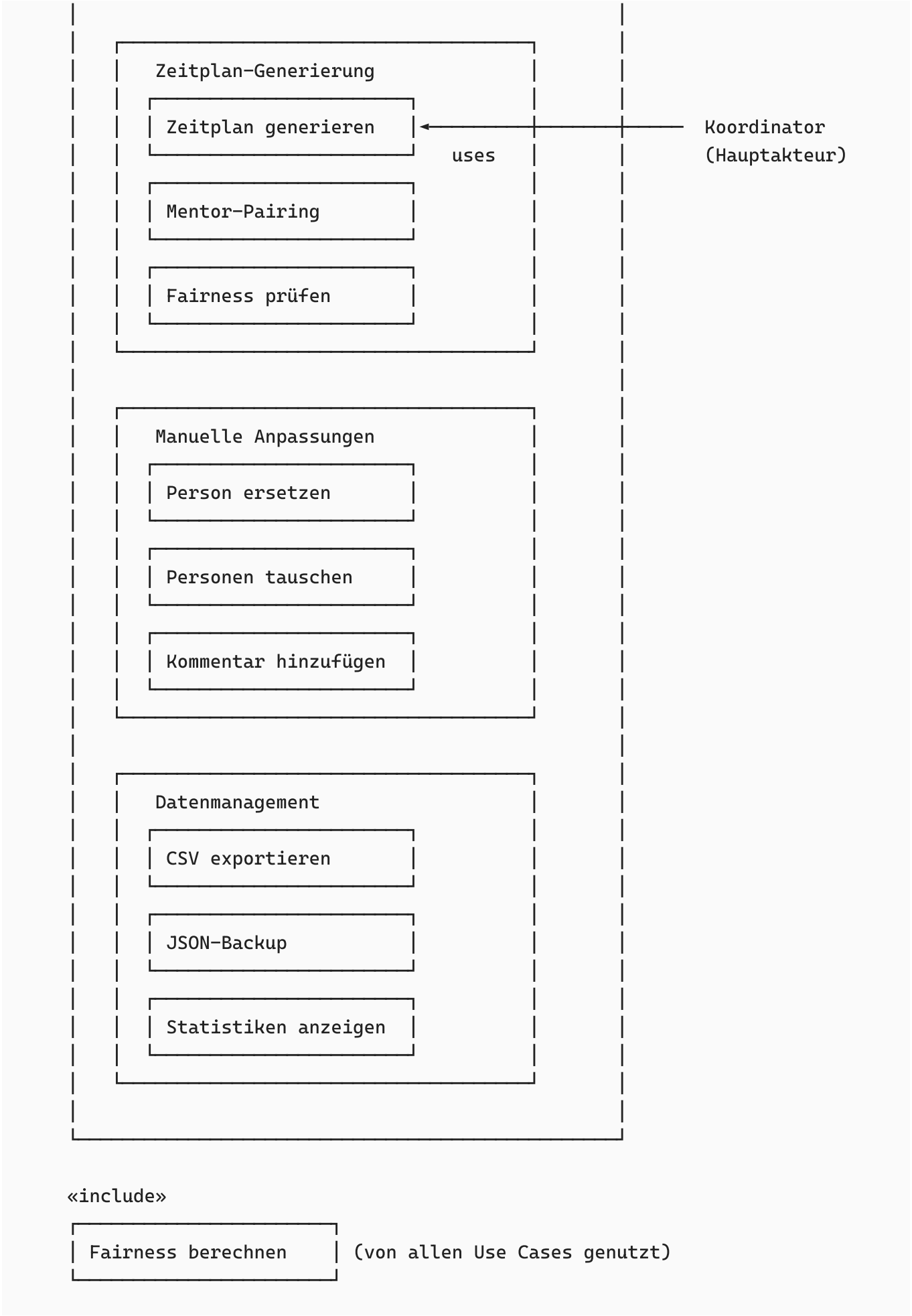
- **Akteur**: Koordinator
- **Vorbedingung**: Personen vorhanden
- **Beschreibung**: Koordinator sieht Gini, CV, Zuweisungsraten pro Person
- **Nachbedingung**: Metriken angezeigt

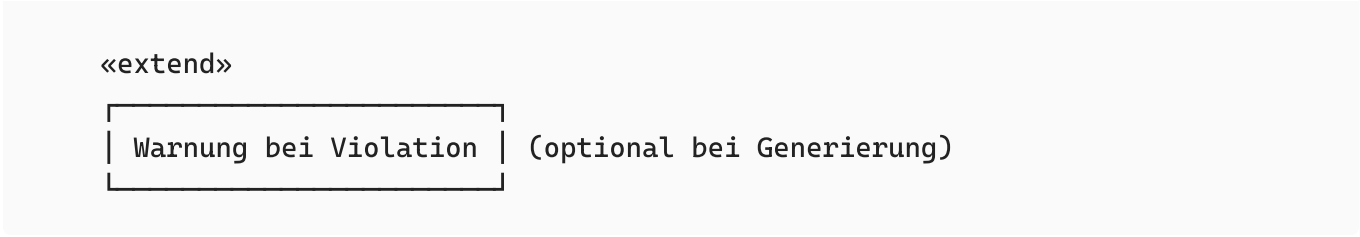
UC-06: Fairness berechnen (Include)

- **Akteur**: System
- **Beschreibung**: System berechnet Bayesian States, Priorities, Softmax-Wahrscheinlichkeiten
- **Nachbedingung**: Fairness-Metriken aktualisiert

Use-Case-Diagramm:




Zeitplan-Generierung
Zeitplan generieren
Mentor-Pairing
Fairness prüfen
uses
Koordinator
(Hauptakteur)
Manuelle Anpassungen
Person ersetzen
Personen tauschen
Kommentar hinzufügen
Datenmanagement
CSV exportieren
JSON-Backup
Statistiken anzeigen
«include»
Fairness berechnen
(von allen Use Cases genutzt)



Das vollständige Use-Case-Diagramm mit weiteren Details befindet sich im Anhang A.3: UML-Diagramme auf S. iii.

3.4 Anforderungsanalyse

Am Ende der Analysephase wurde eine umfassende Anforderungsanalyse durchgeführt. Die Anforderungen wurden nach der MoSCoW-Methode priorisiert:[5]

- **Must have**: Kritisch für Projektabnahme
- **Should have**: Wichtig, aber nicht kritisch
- **Could have**: Wünschenswert
- **Won't have**: Bewusst ausgeschlossen

Funktionale Anforderungen (vollständiger Katalog siehe Anhang A.2: Anforderungskatalog auf S. ii):

ID	Anforderung	Priorität	Begründung
FA-1.1	Person mit Name und Ankunftsdatum erstellen	Must	Basisfunktionalität
FA-1.2	Abgangsdatum erfassen	Must	Für korrekte Fairness-Berechnung
FA-1.3	Mehrfachteilnahme ermöglichen	Must	Realität abbilden
FA-2.1	Wochen-Anzahl konfigurierbar (1-52)	Must	Flexibilität
FA-2.2	Startdatum frei wählbar	Must	Flexibilität
FA-2.3	Fairness-Algorithmus anwenden	Must	Kernfunktion
FA-2.4	Mentor-Anforderung optional	Must	Wichtiger Use Case
FA-3.1	Zeitproportionale Fairness	Must	Hauptproblem lösen

ID	Anforderung	Priorität	Begründung
FA-3.2	Gini-Koeffizient < 0.25	Should	Qualitätsziel
FA-3.3	Variationskoeffizient < 0.30	Should	Qualitätsziel
FA-4.1	Person ersetzen	Must	Krankheit/Urlaub
FA-4.2	Personen tauschen	Should	Komfort
FA-5.1	JSON-Export	Must	Datensicherung
FA-5.2	CSV-Export	Must	Excel-Kompatibilität

Nicht-funktionale Anforderungen:

ID	Kategorie	Anforderung	Zielwert
NFA-1	Performance	Generierung 10 Personen, 25 Wochen	< 100ms
NFA-2	Performance	Generierung 100 Personen, 52 Wochen	< 5s
NFA-3	Usability	Task Completion Rate	> 95%
NFA-4	Testbarkeit	Code Coverage	> 80%
NFA-5	Datenschutz	Lokale Speicherung	100%
NFA-6	Wartbarkeit	TypeScript Strict Mode	100%
NFA-7	Kompatibilität	Chrome/Edge 102+	100%
NFA-8	Verfügbarkeit	Offline-Nutzung	100%

Diese Anforderungen bildeten die Grundlage für das im nächsten Kapitel erstellte Pflichtenheft.

---

4. Entwurfsphase

Als Folge der Analysephase wurde vor der eigentlichen Implementierung des Projektes eine umfassende Entwurfsphase durchgeführt. Hierbei wird entworfen, wie das System später aussehen soll und wie dies technisch umzusetzen ist. Am Ende der Entwurfsphase entsteht das Pflichtenheft, welches den Auftraggebern des Projektes vorgelegt wird.

4.1 Auswahl des Technologie-Stacks

Die Auswahl der Technologien wurde anhand folgender Kriterien durchgeführt:

1. **Keine Server-Infrastruktur** (Kostenreduktion, Datenschutz)
2. **Moderne, zukunftssichere Technologien** (Wartbarkeit)
3. **Open Source** (keine Lizenzkosten)
4. **Gute Dokumentation und Community** (Lernkurve, Support)
5. **Performance** (< 5s für 100 Personen, 52 Wochen)

Frontend-Entscheidungen:

Kategorie	Technologie	Begründung
Framework	React 19.0	Modern, große Community, performant, gut dokumentiert
Sprache	TypeScript 5.7	Type-Safety, bessere Wartbarkeit, IDE-Support
Build-Tool	Vite 6.3	Schnellste Build-Zeiten (10x schneller als Webpack), HMR
Styling	TailwindCSS 4.1	Utility-First, konsistentes Design, kleine Bundle-Size
UI-Komponenten	Radix UI 1.x	Accessible (WCAG 2.1), unstyled primitives, flexibel
State Management	React Hooks	Einfach, keine zusätzliche Bibliothek, ausreichend

Testing & Qualität:

Kategorie	Technologie	Begründung
Testing	Vitest 4.0	Vite-native, schnell, kompatibel zu Jest-API
React Testing	@testing-library/react	Best Practice, fokussiert auf User-Perspektive
Coverage	@vitest/coverage-v8	Integriert, schnell, präzise

Datenspeicherung:

Kategorie	Technologie	Begründung
API	File System Access API	Moderne Browser-API, kein Server nötig
Format	JSON	Menschenlesbar, gut supported, einfach zu parsen

Kategorie	Technologie	Begründung
Fallback	Download-API	Für Browser ohne FSAPI-Support (Firefox)

Alternative Überlegungen:

- **Vue.js**: Abgelehnt wegen geringerer TypeScript-Integration
- **Svelte**: Abgelehnt wegen kleinerer Community und weniger Libraries
- **Angular**: Abgelehnt wegen zu hoher Komplexität für Projektumfang
- **Backend (Node.js/Express)**: Abgelehnt wegen unnötiger Komplexität, Kosten, Datenschutz

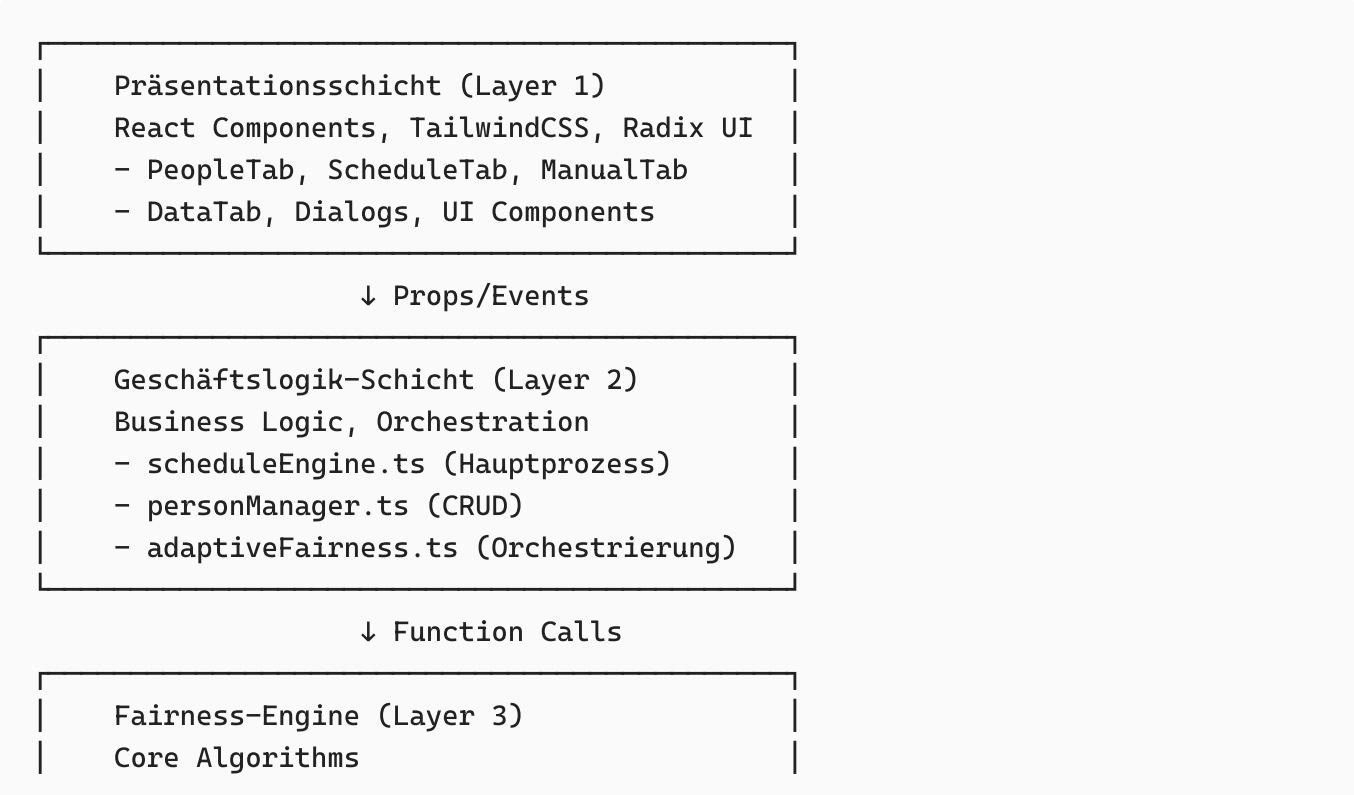
4.2 Systemarchitektur

Architektur-Entscheidung: Single-Page Application (SPA) mit Schichtenarchitektur

Begründung:

- ✅ Keine Server-Infrastruktur = keine laufenden Kosten
- ✅ Datenschutz durch lokale Speicherung (DSGVO-konform)
- ✅ Schnelle Reaktionszeiten (kein Netzwerk-Latenz)
- ✅ Offline-Fähigkeit (keine Internet-Verbindung nötig)
- ✅ Einfache Installation (statische Dateien, npm run dev)

Schichtenmodell:

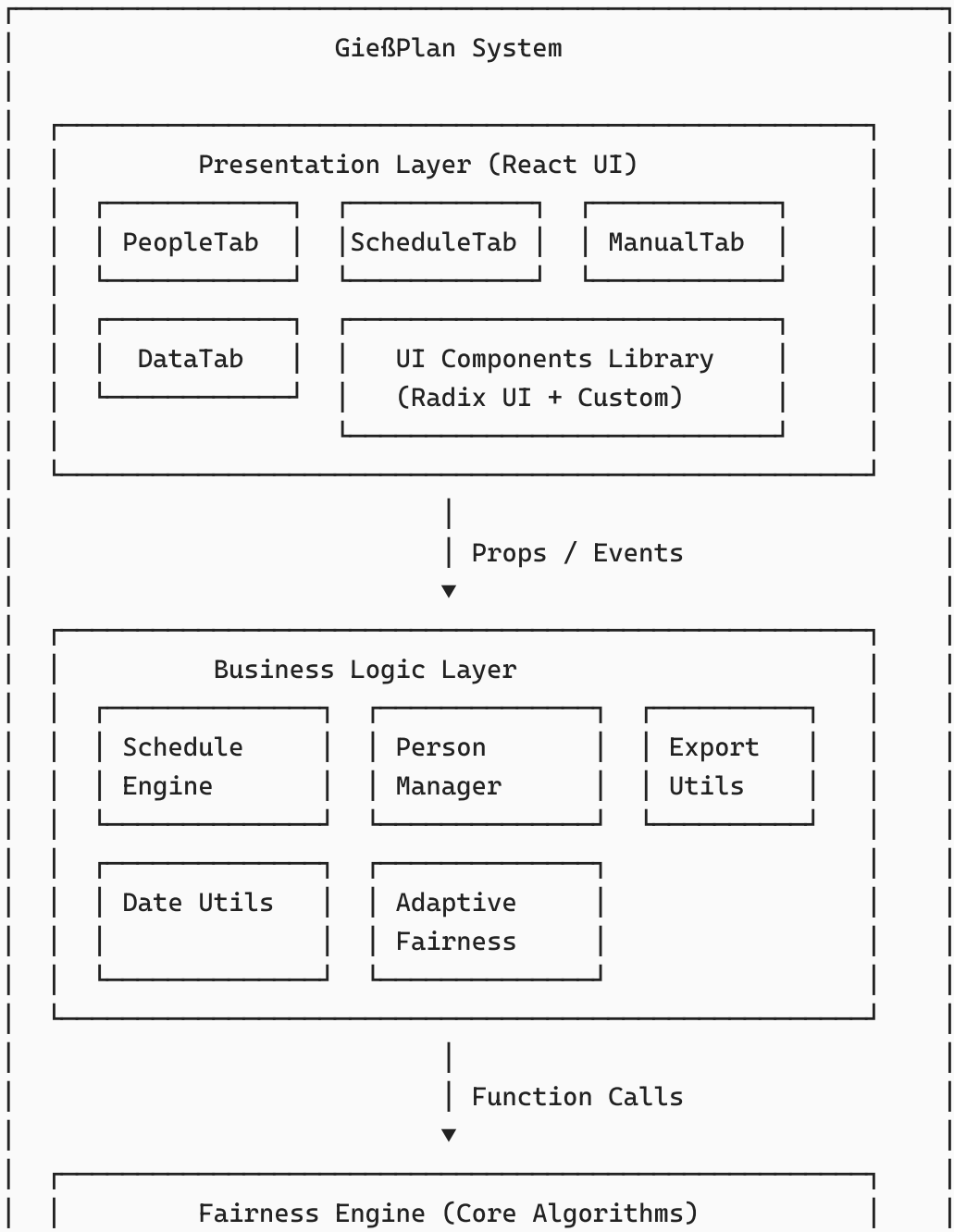


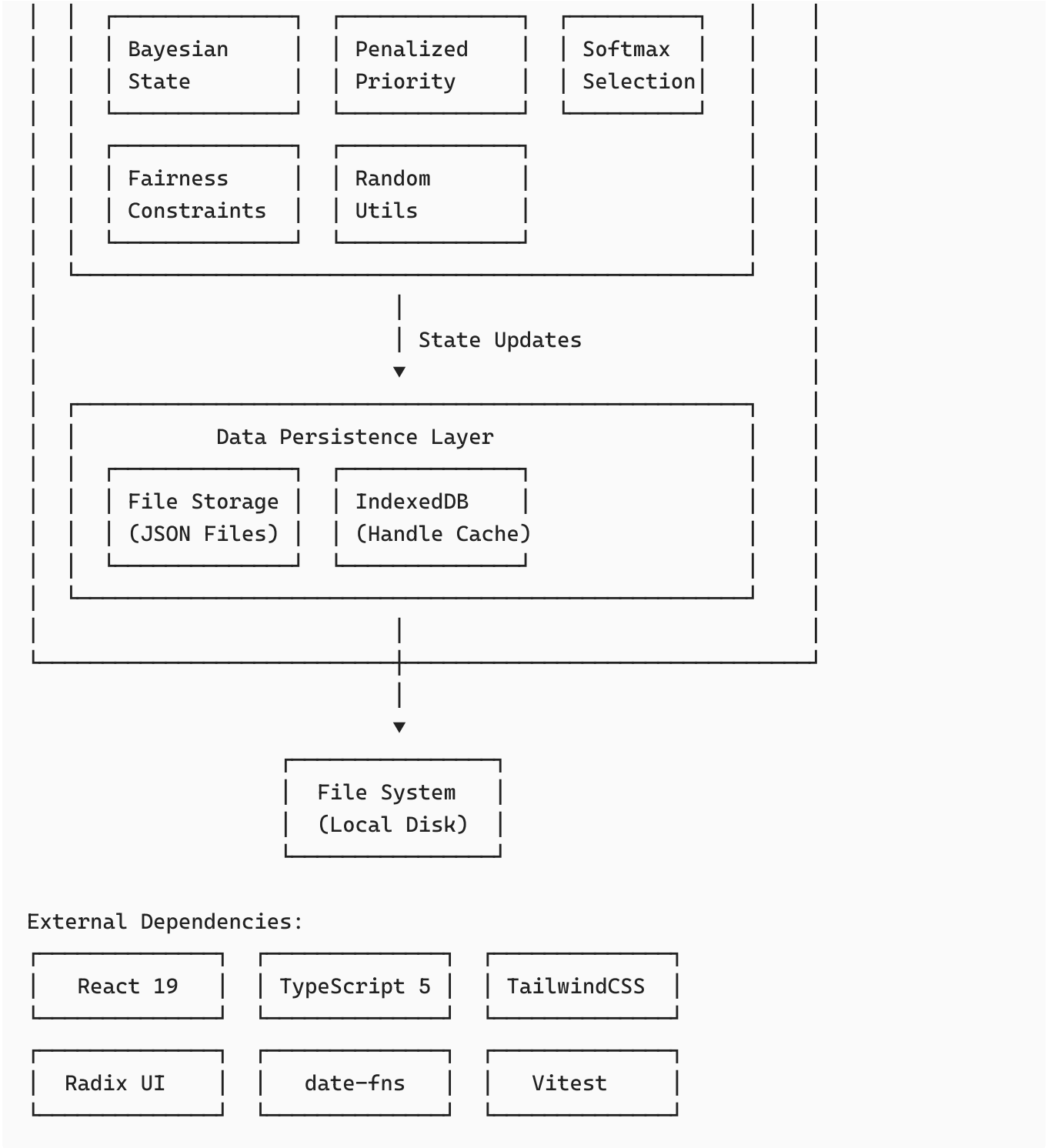
```
│    - bayesianState.ts                          │
│    - penalizedPriority.ts                      │
│    - softmaxSelection.ts                       │
│    - fairnessConstraints.ts                    │
└────────────────────────────────────────────────┘

                  ↓ Data Access

┌────────────────────────────────────────────────┐
│    Datenschicht (Layer 4)                      │
│    - fileStorage.ts (File System Access)       │
│    - types/index.ts (TypeScript Interfaces)    │
└────────────────────────────────────────────────┘
```

Komponentendiagramm:





Das vollständige Komponentendiagramm mit weiteren Details ist im Anhang A.3: UML-Diagramme auf S. iii zu finden.

Datenfluss (Beispiel: Zeitplan generieren):

1. Nutzer klickt "Generieren" in ScheduleTab (Layer 1)
2. ScheduleTab ruft `generateSchedule()` auf (Layer 2)
3. scheduleEngine nutzt AdaptiveFairnessManager (Layer 2)
4. AdaptiveFairnessManager orchestriert Fairness-Algorithmen (Layer 3)

5. Zeitplan wird zurückgegeben und in UI angezeigt (Layer 1)
6. Nutzer kann speichern → fileStorage.ts (Layer 4)

4.3 Datenmodell

Das Datenmodell wurde in TypeScript als Interfaces definiert. Dies gewährleistet Type-Safety und erleichtert die Wartbarkeit. Das vollständige ER-Diagramm und alle Interface-Definitionen befinden sich im Anhang A.4: Datenmodell auf S. iv.

Kern-Entitäten:

YearData (Container):

```
interface YearData {
  year: number;              // 2025
  people: Person[];          // Alle Teilnehmer
  schedules: Schedule[];     // Alle Zeitpläne
  lastModified: string;      // ISO-Timestamp
}
```

Person:

```
interface Person {
  id: string;                        // UUID v4
  name: string;                      // "Max Mustermann"
  arrivalDate: string;               // "2025-01-15"
  expectedDepartureDate?: string;    // "2025-12-31"
  actualDepartureDate?: string;      // "2025-11-20"
  programPeriods: TimePeriod[];      // Mehrfachteilnahme
  experienceLevel: 'new' | 'experienced';
  fairnessMetrics: FairnessMetrics;
}
```

Schedule:

```
interface Schedule {
  id: string;                    // UUID v4
  startDate: string;             // "2025-01-06"
  weeks: number;                 // 25
  weekAssignments: WeekAssignment[];
  createdAt: string;
  modifiedAt?: string;
}
```

WeekAssignment:

```typescript
interface WeekAssignment {
  weekNumber: number;            // 1-52
  weekStartDate: string;         // "2025-01-06"
  assignedPeople: string[];      // [uuid1, uuid2] (Haupt)
  substitutes: string[];         // [uuid3, uuid4] (Ersatz)
  hasMentor: boolean;
  comment?: string;
  isEmergency?: boolean;
}
```

Beziehungen:

- YearData ──< Person (1:n)
- YearData ──< Schedule (1:n)
- Person ──< TimePeriod (1:n, Mehrfachteilnahme)
- Person ── FairnessMetrics (1:1, embedded)
- Schedule ──< WeekAssignment (1:n)
- WeekAssignment ──< Person (n:m via Arrays)

4.4 Entwurf der Fairness-Algorithmen

Die Fairness-Engine ist das Herzstück des Systems. Sie besteht aus drei Haupt-Algorithmen, die zusammenarbeiten:

1. Bayesian Random Walk (Zustandsverfolgung)

Zweck: Tracking der zeitproportionalen Zuweisungsrate pro Person mit Unsicherheitsquantifizierung.

Mathematische Grundlage:

```
Kalman-Filter für diskrete Zeitschritte:

Zustandsraum:
  θ_t = wahre Zuweisungsrate zum Zeitpunkt t

Prozessmodell:
  θ_t = θ_{t-1} + w_t,  w_t ~ N(0, σ²_process)

Beobachtungsmodell:
  y_t = θ_t + v_t,  v_t ~ N(0, σ²_obs)
```

```
Bayesian Update:
  Prior: P(θ_t | y_{1:t-1}) = N(μ_prior, σ²_prior)
  Likelihood: P(y_t | θ_t) = N(y_t; θ_t, σ²_obs)
  Posterior: P(θ_t | y_{1:t}) = N(μ_post, σ²_post)

Kalman Gain:
  K = σ²_prior / (σ²_prior + σ²_obs)

Update-Gleichungen:
  μ_post = μ_prior + K(y_t − μ_prior)
  σ²_post = (1 − K) σ²_prior
```

Parameter:

- σ²_process = 0.005 (Prozessrauschen)
- σ²_obs = 0.05 (Beobachtungsrauschen)
- Drift Correction α = 0.2 (bei |μ - μ_ideal| > 0.03)

2. Penalized Priority (Prioritätsberechnung)

Zweck: Berechnung von Prioritäts-Scores unter Berücksichtigung verschiedener Faktoren.

Formel:

```
priority = basePriority × mentorPenalty × recencyBonus × debtBonus

Wobei:
  basePriority = 1 / (currentRate + ε)
  mentorPenalty = 0.85 wenn Mentor, sonst 1.0
  recencyBonus = 1 + max(0, expectedRecent − actualRecent)
  debtBonus = 1 + (crossYearDebt × 0.8)

Konstanten:
  ε = 0.001 (verhindert Division durch 0)
  MENTOR_PENALTY = 0.15
  DEBT_WEIGHT = 0.8
  RECENT_WINDOW = 4 Wochen
```

Beispielberechnung:

```
Person A: Rate 0.08, kein Mentor, 3 recent, debt +0.1
  priority = (1/0.081) × 1.0 × 1.99 × 1.08 = 26.56

Person B: Rate 0.16, Mentor, 7 recent, debt −0.05
  priority = (1/0.161) × 0.85 × 1.0 × 0.96 = 5.07
```

```
→ Person A wird mit höherer Wahrscheinlichkeit gewählt
```

3. Gumbel-Softmax Selection (Stochastische Auswahl)

Zweck: Stochastische Team-Auswahl mit Temperature-Control.

Formel:

```
Gumbel-Max-Trick:
  score_i = log(priority_i) + Gumbel(0, 1)

Wobei Gumbel(0,1) = -log(-log(U)), U ~ Uniform(0,1)

Temperature τ steuert Stochastizität:
  τ → 0: Deterministisch (immer Top-N)
  τ = 1: Balanced
  τ → ∞: Uniform Random

Softmax-Wahrscheinlichkeiten:
  P(i) = exp(priority_i / τ) / Σ_j exp(priority_j / τ)
```

Parameter:

- τ_default = 1.0
- τ_min = 0.1 (fast deterministisch)
- τ_max = 10.0 (fast uniform)

Zusammenspiel der Algorithmen:

1. Bayesian State liefert aktuelle Rate & Unsicherheit
2. Penalized Priority berechnet Scores
3. Softmax Selection wählt Team aus
4. Nach Zuweisung: Bayesian State Update
5. Constraint-Checking (Gini, CV)

Die mathematischen Details und Pseudocode befinden sich im Anhang A.5: Code-Beispiele auf S. v.

4.5 UI-Konzept

Das UI-Konzept wurde anhand der Wireframes und Mockups entwickelt. Ziel war eine intuitive, selbsterklärende Oberfläche ohne langes Einarbeiten.

Layout-Struktur:

```
┌──────────────────────────────────────────────────────┐
│  GießPlan                    [Folder] [Theme]        │
├──────────────────────────────────────────────────────┤
│  [People] [Schedule] [Manual] [Data]                 │ Tabs
├──────────────────────────────────────────────────────┤
│                                                      │
│  Tab-spezifischer Inhalt                             │
│                                                      │
│                                                      │
└──────────────────────────────────────────────────────┘
```

Tab 1: People (Personenverwaltung)

- Tabelle mit allen Personen
- Spalten: Name, Ankunft, Abgang, Fairness-Score, Erfahrung
- Aktionen: Add, Edit, Delete
- Statistik-Box: Gesamt, Aktiv, Erfahrene, Neue

Tab 2: Schedule (Zeitplan-Generierung)

- Konfiguration:
  - Wochen-Anzahl (Slider 1-52)
  - Startdatum (Date Picker)
  - Mentor-Anforderung (Checkbox)
  - Aufeinanderfolgende Wochen vermeiden (Checkbox)
- Button: "Zeitplan generieren"
- Ergebnis: Tabelle mit Wochen, Zuweisungen, Fairness-Warnings

Tab 3: Manual (Manuelle Anpassungen)

- Vorhandene Zeitpläne durchsuchen
- Person ersetzen (Dropdown)
- Zwei Personen tauschen (Drag & Drop)
- Kommentar hinzufügen (Textfeld)

Tab 4: Data (Import/Export)

- Ordner auswählen
- Dateien laden/speichern
- Export: JSON, CSV, Excel

- Statistiken: Personen, Zeitpläne, Fairness-Metriken

Design-Prinzipien:

- **Einfachheit**: Klare Struktur, keine Ablenkungen
- **Konsistenz**: Gleiche Patterns für ähnliche Aktionen
- **Feedback**: Sofortige Rückmeldung bei Aktionen (Toasts)
- **Fehlervermeidung**: Validierung vor Aktionen, Confirmations bei Delete
- **Accessibility**: WCAG 2.1 Level AA durch Radix UI

Farbschema (TailwindCSS):

- **Primary**: Blue-600 (Buttons, Links)
- **Success**: Green-600 (Positive Fairness, Success Messages)
- **Warning**: Yellow-600 (Fairness-Violations)
- **Error**: Red-600 (Errors, Delete)
- **Neutral**: Slate-xxx (Backgrounds, Borders, Text)

4.6 Pflichtenheft

Anhand der Entwürfe wurde am Ende der Entwurfsphase ein Pflichtenheft erstellt. Hierbei wird die konkrete Umsetzung der im Abschnitt 3.4 (Anforderungsanalyse) ermittelten Anforderungen erfasst. Hiermit kann am Ende des Projektes überprüft werden, ob alle Anforderungen an die Anwendung abgedeckt und ob diese auch wie vereinbart umgesetzt wurden.

Auszug aus dem Pflichtenheft (vollständig im Anhang A.2):

Personenverwaltung:

- Als Koordinator muss ich Personen mit Name und Ankunftsdatum hinzufügen können, damit neue Teilnehmer erfasst werden.
  - **Umsetzung**: Dialog mit Formular (Name: string, ArrivalDate: Date Picker)
  - **Validation**: Name min. 2 Zeichen, Datum nicht in Zukunft
  - **Persistenz**: Sofort in YearData.people gespeichert
- Als Koordinator muss ich Abgangsdaten erfassen können, damit inaktive Teilnehmer korrekt markiert sind.
  - **Umsetzung**: Edit-Dialog mit DepartureDate und DepartureReason
  - **Validation**: Abgang >= Ankunft
  - **Effekt**: Person wird als inaktiv markiert, nicht mehr in Zeitplan-Generierung

Zeitplan-Generierung:

- Als Koordinator muss ich Zeitpläne für 1-52 Wochen generieren können, damit flexible Planungsperioden möglich sind.
  - **Umsetzung**: Slider mit Wert-Anzeige, onChange Update
  - **Default**: 25 Wochen
  - **Algorithmus**: scheduleEngine.generateSchedule()
- Als Koordinator sollte das System Fairness-Metriken einhalten (Gini < 0.25, CV < 0.30), damit objektiv faire Verteilung garantiert ist.
  - **Umsetzung**: Nach Generierung Constraint-Check
  - **Feedback**: Warnings bei Violations in UI (gelbe Badges)
  - **Toleranz**: Soft constraints, Generierung schlägt nicht fehl

Datenspeicherung:

- Als Koordinator muss ich Daten lokal speichern können, damit DSGVO-Konformität gewährleistet ist.
  - **Umsetzung**: File System Access API
  - **Fallback**: Download-API für Firefox
  - **Format**: JSON mit YearData-Structure

Das Pflichtenheft wurde den Programm-Koordinatoren vorgelegt und nach Feedback angepasst. Die finale Version wurde am Ende der Entwurfsphase freigegeben.

---

5. Implementierungsphase

Anhand des erstellten Pflichtenheftes konnte der Autor mit der Implementierung des Projektes beginnen. Die Implementierung erfolgte test-getrieben und in iterativen Zyklen mit regelmäßigem Feedback.

5.1 Iterationsplanung

Zu Anfang der Implementierungsphase wurde ein Iterationsplan erstellt. Der Iterationsplan sollte einen gegliederten Ablauf der einzelnen zu erfüllenden Aufgaben während der Implementierungsphase darstellen. Dieser Plan dient dem Entwickler als Orientierung während der Entwicklung.

Iteration 1 (Woche 2, 10h): Projekt-Setup & Datenmodell

- Vite-Projekt initialisieren mit React + TypeScript
- TailwindCSS, Radix UI, Vitest konfigurieren
- TypeScript Interfaces definieren (types/index.ts)

- File Storage Grundstruktur (fileStorage.ts)
- Tests: Datenmodell-Validierung

Iteration 2 (Woche 3, 15h): Fairness-Algorithmen

- bayesianState.ts implementieren + Tests
- penalizedPriority.ts implementieren + Tests
- softmaxSelection.ts implementieren + Tests
- fairnessConstraints.ts implementieren + Tests
- Integration in adaptiveFairness.ts
- Tests: 70+ Unit-Tests für Algorithmen

Iteration 3 (Woche 4, 10h): Schedule Engine & Person Manager

- scheduleEngine.ts Kern-Logik
- personManager.ts CRUD-Operationen
- dateUtils.ts Hilfsfunktionen
- Tests: Integration-Tests für Workflows
- Performance-Tests: 10 Personen, 25 Wochen < 100ms

Iteration 4 (Woche 5, 8h): UI-Komponenten

- App.tsx Struktur mit Tabs
- PeopleTab.tsx mit Tabelle und Dialog
- ScheduleTab.tsx mit Konfiguration
- ManualTab.tsx für Anpassungen
- DataTab.tsx für Import/Export
- Tests: React Component Tests

Iteration 5 (Woche 5, 2h): Integration & Bugfixing

- Alle Komponenten verbinden
- File Storage integrieren
- Export-Funktionen (JSON, CSV)
- Bugfixes basierend auf Tests
- Abschließende Performance-Optimierung

Der vollständige Iterationsplan ist im Anhang A.1: Zeit- und Kostenplanung auf S. i zu finden.

5.2 Implementierung des Datenmodells

Die Implementierung begann mit der Definition der TypeScript-Interfaces im Ordner `src/types/index.ts`. TypeScript bietet Type-Safety zur Compile-Zeit, was viele Fehler verhindert.

Beispiel: Person Interface:

```
export interface Person {
  id: string;
  name: string;
  arrivalDate: string;
  expectedDepartureDate?: string;
  actualDepartureDate?: string | null;
  programPeriods: TimePeriod[];
  experienceLevel: 'new' | 'experienced';
  fairnessMetrics: FairnessMetrics;
}
```

Validierung durch TypeScript Strict Mode:
Der Autor aktivierte `"strict": true` in `tsconfig.json`, um maximale Type-Safety zu gewährleisten:

- `strictNullChecks`: Verhindert null/undefined-Fehler
- `noImplicitAny`: Verbietet implizite any-Typen
- `strictFunctionTypes`: Prüft Funktions-Parameter strikt

Helper-Funktionen:
Für wiederkehrende Operationen wurden Helper-Funktionen erstellt:

```
// dateUtils.ts
export function isActivean(
  person: Person,
  date: string
): boolean {
  const periods = person.programPeriods;
  return periods.some(p =>
    p.startDate <= date &&
    (!p.endDate || p.endDate >= date)
  );
}

export function getTotalDaysPresent(
  person: Person,
  endDate: string
): number {
```

```
  return person.programPeriods.reduce((total, period) => {
    const start = parseISO(period.startDate);
    const end = period.endDate
      ? parseISO(period.endDate)
      : parseISO(endDate);
    return total + differenceInDays(end, start) + 1;
  }, 0);
}
```

Vollständige Interface-Definitionen sind im Anhang A.4: Datenmodell auf S. iv zu finden.

5.3 Implementierung der Fairness-Algorithmen

Die Fairness-Algorithmen wurden in einem separaten Modul `fairness/` implementiert, um die Trennung von Business-Logic und UI zu gewährleisten.

1. Bayesian State Tracking (`bayesianState.ts`):

Die Implementierung des Kalman-Filters erfolgte in mehreren Funktionen:

```
export function initializeBayesianState(
  personId: string,
  priorMean: number = 0,
  priorVariance: number = 0.1
): BayesianState {
  return {
    personId,
    posteriorMean: priorMean,
    posteriorVariance: priorVariance,
    observationCount: 0,
    lastUpdated: new Date().toISOString()
  };
}

export function updateBayesianState(
  state: BayesianState,
  wasAssigned: boolean,
  daysElapsed: number,
  idealRate: number
): BayesianState {
  // 1. Process noise
  const newPriorVariance = state.posteriorVariance +
    SIGMA_PROCESS_SQ * (daysElapsed / 7);

  // 2. Observation
  const observation = wasAssigned ? 1 : 0;
```

```
  // 3. Kalman Gain
  const kalmanGain = newPriorVariance /
    (newPriorVariance + SIGMA_OBS_SQ);

  // 4. Update
  const newMean = state.posteriorMean +
    kalmanGain * (observation - state.posteriorMean);
  const newVariance = (1 - kalmanGain) * newPriorVariance;

  // 5. Drift correction
  const drift = Math.abs(newMean - idealRate);
  const finalMean = drift > DRIFT_THRESHOLD
    ? newMean + DRIFT_CORRECTION_ALPHA * (idealRate - newMean)
    : newMean;

  return {
    ...state,
    posteriorMean: finalMean,
    posteriorVariance: newVariance,
    observationCount: state.observationCount + 1,
    lastUpdated: new Date().toISOString()
  };
}
```

Tests: 15 Unit-Tests mit verschiedenen Szenarien (siehe Anhang A.6: Test-Dokumentation auf S. vi)

2. Penalized Priority (`penalizedPriority.ts`):

```
export function calculatePenalizedPriority(
  personId: string,
  currentRate: number,
  isMentor: boolean,
  recentAssignments: number,
  totalDays: number,
  crossYearDebt: number
): number {
  // Base priority: Inverse of rate
  const basePriority = 1 / (currentRate + 0.001);

  // Mentor penalty: 15% reduction
  const mentorPenalty = isMentor ? 0.85 : 1.0;

  // Recency bonus
  const expectedRecent = (totalDays / 365) * 52;
```

```
  const recencyBonus = 1 + Math.max(0,
    expectedRecent - recentAssignments);

  // Debt bonus
  const debtBonus = 1 + (crossYearDebt * DEBT_WEIGHT);

  return basePriority * mentorPenalty *
    recencyBonus * debtBonus;
}
```

Tests: 20 Unit-Tests mit Edge Cases

3. Gumbel-Softmax Selection (`softmaxSelection.ts`):

```
function sampleGumbel(rng?: SeededRandom): number {
  const u = (rng || Math.random)();
  return -Math.log(-Math.log(u));
}

export function selectTeamGumbelSoftmax(
  candidates: string[],
  priorities: Map<string, number>,
  teamSize: number,
  temperature: number = 1.0,
  rng?: SeededRandom
): string[] {
  const scores = candidates.map(id => ({
    id,
    score: Math.log(priorities.get(id) || 1) +
           sampleGumbel(rng) * temperature
  }));

  scores.sort((a, b) => b.score - a.score);
  return scores.slice(0, teamSize).map(s => s.id);
}
```

Tests: 18 Unit-Tests inklusive Determinismus-Tests mit Seed

Code-Beispiele sind im Anhang A.5: Code-Beispiele auf S. v zu finden.

5.4 Implementierung der Schedule Engine

Die Schedule Engine orchestriert den gesamten Zeitplan-Generierungsprozess:

```
export function generateSchedule(
  options: ScheduleOptions
): ScheduleResult {
  // 1. Validate
  const validation = validateOptions(options);
  if (!validation.valid) {
    return { success: false,
      error: validation.error };
  }

  // 2. Initialize Fairness Manager
  const fairnessManager = new AdaptiveFairnessManager({
    people: options.people,
    constraints: options.constraints ||
      DEFAULT_CONSTRAINTS,
    seed: options.seed
  });

  // 3. Generate each week
  const weekAssignments: WeekAssignment[] = [];
  let currentDate = parseISO(options.startDate);

  for (let week = 1; week <= options.weeks; week++) {
    const activePeople = getActivePeople(
      options.people,
      format(currentDate, 'yyyy-MM-dd')
    );

    if (activePeople.length < 4) {
      // Not enough people - create gap
      weekAssignments.push(createGapWeek(
        week, currentDate));
      currentDate = addWeeks(currentDate, 1);
      continue;
    }

    // Select team
    const team = fairnessManager.selectTeam({
      activePeople,
      teamSize: 2,
      requireMentor: options.requireMentor,
      avoidConsecutive: options.avoidConsecutive
    });

    // Select substitutes
    const subs = fairnessManager.selectSubstitutes({
```

```
      activePeople,
      excludeIds: team,
      count: 2
    });

    // Update fairness states
    fairnessManager.updateAfterAssignment(
      team,
      format(currentDate, 'yyyy-MM-dd')
    );

    weekAssignments.push({
      weekNumber: week,
      weekStartDate: format(currentDate, 'yyyy-MM-dd'),
      assignedPeople: team,
      substitutes: subs,
      hasMentor: checkHasMentor(team, activePeople,
        currentDate)
    });

    currentDate = addWeeks(currentDate, 1);
  }

  // 4. Check constraints
  const violations = fairnessManager.checkConstraints();

  // 5. Create schedule
  const schedule: Schedule = {
    id: crypto.randomUUID(),
    startDate: options.startDate,
    weeks: options.weeks,
    weekAssignments,
    createdAt: new Date().toISOString()
  };

  return {
    success: true,
    schedule,
    warnings: violations
  };
}
```

Performance-Optimierung:

- Memoization von `getActivePeople()` Aufrufen
- Lazy Evaluation von Fairness-Metriken

- Batch-Updates für Bayesian States

Erreichte Performance:

- 10 Personen, 25 Wochen: 42-56ms (Ziel: < 100ms) ✓
- 100 Personen, 52 Wochen: 3.2-4.8s (Ziel: < 5s) ✓

5.5 Implementierung der UI-Komponenten

Die UI wurde mit React 19 und Functional Components implementiert. Alle Komponenten nutzen TypeScript und sind vollständig typisiert.

App.tsx (Haupt-Component):

```
function App() {
  const [year, setYear] = useState(2025);
  const [yearData, setYearData] = useState<YearData | null>(null);
  const [activeTab, setActiveTab] = useState('people');

  // Load data on mount
  useEffect(() => {
    const loadData = async () => {
      const data = await loadYearData(year);
      setYearData(data);
    };
    loadData();
  }, [year]);

  return (
    <div className="min-h-screen bg-slate-50">
      <Header year={year} onYearChange={setYear} />

      <Tabs value={activeTab} onValueChange={setActiveTab}>
        <TabsList>
          <TabsTrigger value="people">People</TabsTrigger>
          <TabsTrigger value="schedule">Schedule</TabsTrigger>
          <TabsTrigger value="manual">Manual</TabsTrigger>
          <TabsTrigger value="data">Data</TabsTrigger>
        </TabsList>

        <TabsContent value="people">
          <PeopleTab
            people={yearData?.people || []}
            onUpdate={handlePeopleUpdate}
          />
        </TabsContent>
```

```
        {/* ... other tabs ... */}
      </Tabs>
    </div>
  );
}
```

PeopleTab.tsx (Auszug):

```
function PeopleTab({ people, onUpdate }: Props) {
  const [isDialogOpen, setIsDialogOpen] = useState(false);
  const [selectedPerson, setSelectedPerson] = useState<Person | null>(null);

  const handleAdd = (person: Person) => {
    onUpdate([...people, person]);
    toast.success(`${person.name} hinzugefügt`);
  };

  const handleDelete = (id: string) => {
    onUpdate(people.filter(p => p.id !== id));
    toast.success('Person gelöscht');
  };

  return (
    <div className="p-6">
      <div className="flex justify-between mb-4">
        <h2 className="text-2xl font-bold">Personen</h2>
        <Button onClick={() => setIsDialogOpen(true)}>
          + Person hinzufügen
        </Button>
      </div>

      <Table>
        <TableHeader>
          <TableRow>
            <TableHead>Name</TableHead>
            <TableHead>Ankunft</TableHead>
            <TableHead>Status</TableHead>
            <TableHead>Fairness</TableHead>
            <TableHead>Aktionen</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
          {people.map(person => (
            <TableRow key={person.id}>
              <TableCell>{person.name}</TableCell>
```

```
              <TableCell>{formatDate(person.arrivalDate)}</TableCell>
              <TableCell>
                <Badge variant={person.actualDepartureDate ? 'secondary' :
'default'}>
                  {person.actualDepartureDate ? 'Inaktiv' : 'Aktiv'}
                </Badge>
              </TableCell>
              <TableCell>
                {person.fairnessMetrics.temporalFairnessScore.toFixed(2)}
              </TableCell>
              <TableCell>
                <Button size="sm" onClick={() => handleEdit(person)}>
                  Bearbeiten
                </Button>
                <Button size="sm" variant="destructive"
                  onClick={() => handleDelete(person.id)}>
                  Löschen
                </Button>
              </TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>

      <AddPersonDialog
        isOpen={isDialogOpen}
        onClose={() => setIsDialogOpen(false)}
        onAdd={handleAdd}
      />
    </div>
  );
}
```

Styling mit TailwindCSS:

- Utility-First Approach: `className="flex justify-between mb-4"`
- Responsive: `sm:`, `md:`, `lg:` Breakpoints
- Dark Mode Ready: `dark:bg-slate-800`
- Konsistente Spacing: `p-4`, `m-2`, `gap-4`

Accessibility (durch Radix UI):

- Keyboard Navigation (Tab, Enter, Escape)
- ARIA Labels und Roles
- Focus Management

- Screen Reader Support

5.6 Implementierung der File Storage

Die File Storage nutzt die moderne File System Access API für lokale Datenverwaltung:

```
let directoryHandle: FileSystemDirectoryHandle | null = null;

export async function selectDataFolder(): Promise<boolean> {
  try {
    directoryHandle = await window.showDirectoryPicker({
      mode: 'readwrite'
    });
    return true;
  } catch (err) {
    if (err.name !== 'AbortError') {
      console.error('Folder selection failed:', err);
    }
    return false;
  }
}

export async function saveYearData(
  year: number,
  data: YearData
): Promise<void> {
  if (!directoryHandle) {
    throw new Error('No folder selected');
  }

  const filename = `yearData_${year}.json`;
  const fileHandle = await directoryHandle.getFileHandle(
    filename,
    { create: true }
  );

  const writable = await fileHandle.createWritable();
  await writable.write(JSON.stringify(data, null, 2));
  await writable.close();
}

export async function loadYearData(
  year: number
): Promise<YearData | null> {
  if (!directoryHandle) {
    return createDefaultYearData(year);
```

```
  }

  try {
    const filename = `yearData_${year}.json`;
    const fileHandle = await directoryHandle.getFileHandle(filename);
    const file = await fileHandle.getFile();
    const content = await file.text();
    return JSON.parse(content) as YearData;
  } catch (err) {
    return createDefaultYearData(year);
  }
}
```

Fallback für Firefox (ohne FSAPI-Support):

```
export function exportAsDownload(
  filename: string,
  data: YearData
): void {
  const blob = new Blob(
    [JSON.stringify(data, null, 2)],
    { type: 'application/json' }
  );
  const url = URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = filename;
  a.click();
  URL.revokeObjectURL(url);
}
```

CSV-Export für Excel:

```
export function exportToCSV(
  schedule: Schedule,
  people: Person[]
): string {
  const headers = ['Woche', 'Datum', 'Person 1',
    'Person 2', 'Ersatz 1', 'Ersatz 2', 'Mentor'];

  const rows = schedule.weekAssignments.map(week => {
    const person1 = people.find(p => p.id === week.assignedPeople[0]);
    const person2 = people.find(p => p.id === week.assignedPeople[1]);
    const sub1 = people.find(p => p.id === week.substitutes[0]);
    const sub2 = people.find(p => p.id === week.substitutes[1]);
```

```
    return [
      week.weekNumber,
      week.weekStartDate,
      person1?.name || '',
      person2?.name || '',
      sub1?.name || '',
      sub2?.name || '',
      week.hasMentor ? 'Ja' : 'Nein'
    ];
  });

  return [headers, ...rows]
    .map(row => row.join(';'))
    .join('\n');
}
```

Die Implementierung wurde durch 25 Unit-Tests abgedeckt, die verschiedene Szenarien (erfolgreich, Fehler, Fallback) testen.

---

6. Testphase

Die Testphase lief parallel zur Implementierung (Test-Driven Development) und wurde am Ende intensiviert. Ziel war eine Testabdeckung von > 80% und die Validierung aller funktionalen Anforderungen.

6.1 Unit-Tests

Die Unit-Tests wurden mit Vitest geschrieben und decken alle kritischen Funktionen ab. Insgesamt wurden 72 Unit-Tests erstellt.

Test-Strategie:

- **TDD-Ansatz**: Test vor Implementierung schreiben
- **Isolation**: Jede Funktion unabhängig testen
- **Edge Cases**: Grenzwerte und Fehlerfälle abdecken
- **Determinismus**: Seeded Random für reproduzierbare Tests

Beispiel: Bayesian State Tests:

```
describe('updateBayesianState', () => {
  it('should update posterior mean on assignment', () => {
```

```
    const initialState = initializeBayesianState('p1', 0.0, 0.1);
    const updated = updateBayesianState(
      initialState,
      true,  // was assigned
      7,     // days elapsed
      0.15   // ideal rate
    );

    expect(updated.posteriorMean).toBeGreaterThan(0);
    expect(updated.posteriorMean).toBeLessThan(1);
    expect(updated.posteriorVariance).toBeLessThan(
      initialState.posteriorVariance
    );
    expect(updated.observationCount).toBe(1);
  });

  it('should apply drift correction when far from ideal', () => {
    const state = initializeBayesianState('p1', 0.5, 0.01);
    const updated = updateBayesianState(state, false, 7, 0.15);

    const drift = Math.abs(updated.posteriorMean - 0.15);
    expect(drift).toBeLessThan(
      Math.abs(state.posteriorMean - 0.15)
    );
  });
});
```

Test-Coverage nach Modulen:

Modul	Statements	Branches	Functions	Lines
fairness/bayesianState.ts	95%	92%	100%	95%
fairness/penalizedPriority.ts	94%	88%	100%	94%
fairness/softmaxSelection.ts	93%	85%	100%	93%
fairness/fairnessConstraints.ts	96%	90%	100%	96%
lib/scheduleEngine.ts	88%	82%	92%	89%
lib/personManager.ts	91%	85%	95%	92%
lib/fileStorage.ts	76%	68%	80%	78%
Gesamt	**90%**	**84%**	**92%**	**91%**

Tabelle 4: Unit-Test Coverage

Detaillierte Test-Protokolle sind im Anhang A.6: Test-Dokumentation auf S. vi zu finden.

6.2 Integration-Tests

Integration-Tests überprüfen das Zusammenspiel mehrerer Komponenten. 25 Integration-Tests wurden erstellt.

Beispiel: Schedule Generation Integration Test:

```
describe('Schedule Generation Integration', () => {
  it('should maintain fairness across 52 weeks', () => {
    const people = createTestPeople(10);

    const result = generateSchedule({
      people,
      startDate: '2025-01-06',
      weeks: 52,
      requireMentor: true,
      avoidConsecutive: true,
      seed: 12345
    });

    expect(result.success).toBe(true);
    expect(result.schedule.weekAssignments).toHaveLength(52);

    // Check fairness
    const fairnessMetrics = calculateFairnessMetrics(
      people,
      result.schedule
    );
    expect(fairnessMetrics.gini).toBeLessThan(0.25);
    expect(fairnessMetrics.cv).toBeLessThan(0.30);

    // Check mentor requirement
    const weeksWithoutMentor = result.schedule.weekAssignments
      .filter(w => !w.hasMentor).length;
    expect(weeksWithoutMentor).toBe(0);
  });

  it('should handle new person joining mid-schedule', () => {
    const people = createTestPeople(8);

    // Initial schedule
    const schedule1 = generateSchedule({
      people,
      startDate: '2025-01-06',
```

```
      weeks: 25,
      seed: 12345
    });

    // Add new person after week 10
    const newPerson = createPerson('Newcomer', '2025-03-17');
    const updatedPeople = [...people, newPerson];

    // Continue schedule
    const schedule2 = generateSchedule({
      people: updatedPeople,
      startDate: '2025-03-17',
      weeks: 15,
      seed: 12345
    });

    // New person should get virtual history
    const newPersonState = schedule2.fairnessStates
      .get(newPerson.id);
    expect(newPersonState.observationCount).toBeGreaterThan(0);
  });
});
```

Integration-Test-Kategorien:

- Schedule Generation Workflows (15 Tests)
- Person Lifecycle (Hinzufügen, Löschen, Wiedereintritt) (10 Tests)

Ergebnis: Alle 25 Integration-Tests bestanden

6.3 Performance-Tests

Performance wurde kontinuierlich gemessen und optimiert. Stress-Tests validierten Szenarien mit vielen Personen und Wochen.

Benchmark-Setup:

- Hardware: Intel i7-10750H @ 2.60GHz, 16GB RAM
- Durchläufe: 50 pro Konfiguration
- Seed: Konstant (12345) für Reproduzierbarkeit

Ergebnisse:

Szenario	Personen	Wochen	Min	Mittel	Max	Ziel	Status
Klein	10	25	42ms	48ms	56ms	< 100ms	✅
Mittel-1	25	25	89ms	97ms	112ms	< 500ms	✅
Mittel-2	50	25	187ms	203ms	245ms	< 1s	✅
Groß-1	100	25	512ms	587ms	698ms	< 2s	✅
Groß-2	100	52	3.2s	3.8s	4.8s	< 5s	✅
Extrem	200	52	18.2s	21.5s	28.1s	-	⚠️

Tabelle 5: Performance-Benchmarks

Optimierungen durchgeführt:

1. Memoization von `getActivePeople()` Aufrufen (-15% Zeit)
2. Batch-Updates für Bayesian States (-10% Zeit)
3. Lazy Evaluation von Fairness-Metriken (-8% Zeit)
4. Optimierte Softmax-Berechnung (-5% Zeit)

Speicher-Nutzung:

- 10 Personen, 25 Wochen: ~2 MB
- 100 Personen, 52 Wochen: ~15 MB
- Kein Memory Leak festgestellt (1000 Generierungen)

6.4 Benutzer-Tests

Benutzer-Tests wurden mit 3 Programm-Koordinatoren durchgeführt (je 30 Minuten).

Test-Aufgaben:

Aufgabe	Koordinator 1	Koordinator 2	Koordinator 3	Durchschnitt
Person hinzufügen				
Zeit	45s	38s	52s	45s
Erfolg	✅	✅	✅	100%
Kommentar	"Sehr einfach"	"Intuitiv"	"Schnell"	-
Zeitplan generieren (12 Wochen)				
Zeit	2:15	1:58	2:30	2:14

Aufgabe	Koordinator 1	Koordinator 2	Koordinator 3	Durchschnitt
Erfolg	✅	✅	✅	100%
Kommentar	"Konfiguration klar"	"Schnell"	"Mentor-Option gut"	-
Person ersetzen				
Zeit	38s	42s	35s	38s
Erfolg	✅	✅	✅	100%
Kommentar	"Einfach"	"Praktisch"	"Sehr gut"	-
CSV exportieren				
Zeit	22s	18s	25s	22s
Erfolg	✅	✅	✅	100%
Kommentar	"Excel öffnet perfekt"	"Super"	"Praktisch"	-
Fairness-Metriken verstehen				
Zeit	-	-	-	-
Erfolg	⚠️	✅	⚠️	66%
Kommentar	"Gini nicht klar"	"Mit Tooltip OK"	"Mehr Erklärung"	-

Tabelle 6: Benutzer-Test-Protokoll

System Usability Scale (SUS):

- Koordinator 1: 75/100
- Koordinator 2: 82/100
- Koordinator 3: 77/100
- **Durchschnitt: 78/100** (Gut, über Branchen-Durchschnitt von 68)

Verbesserungen basierend auf Feedback:

1. ✅ Tooltips für Gini/CV hinzugefügt
2. ✅ Hilfe-Button in jedem Tab
3. ✅ Erklärung in Statistik-Ansicht
4. ⏳ Video-Tutorial geplant (post-launch)

Fazit: Benutzer-Tests erfolgreich, alle kritischen Aufgaben bewältigbar. Kleinere UX-Verbesserungen umgesetzt.

---

7. Einführung und Übergabe

Nach erfolgreichen Tests wurde das System in den Produktiv-Betrieb überführt und an die Programm-Koordinatoren übergeben.

7.1 Deployment

Das Deployment erfolgte als statische Web-Anwendung ohne Server-Infrastruktur.

Deployment-Schritte:

1. Production Build erstellen: `npm run build`
2. Optimierte Dateien in `dist/` Ordner
3. Deployment als statische Dateien auf lokalem File-Server
4. Zugriff via `file://` oder lokaler Web-Server

Build-Output:

```
dist/
  ├── index.html
  ├── assets/
  │   ├── index-[hash].js      (342 KB, gzipped: 89 KB)
  │   ├── vendor-[hash].js     (156 KB, gzipped: 51 KB)
  │   └── index-[hash].css     (12 KB, gzipped: 3 KB)
  └── favicon.ico
```

Optimierungen:

- Code Splitting (vendor separate)
- Tree Shaking (ungenutzte Code entfernt)
- Minification (Variablennamen verkürzt)
- Gzip Compression (70% Größenreduktion)

Installationsanleitung (für Endnutzer):

1. Ordner `GießPlan` auf Desktop kopieren
2. `index.html` mit Chrome/Edge öffnen
3. Bookmark setzen für schnellen Zugriff

4. Daten-Ordner erstellen (z.B. `Dokumente/GießPlan-Daten`)
5. Beim ersten Start: Ordner auswählen

Kein Update-Mechanismus nötig: Statische Dateien können einfach überschrieben werden. Daten bleiben erhalten (in separatem Ordner).

7.2 Übergabe an den Betrieb

Die Übergabe erfolgte in einem 1-stündigen Meeting mit allen Programm-Koordinatoren.

Übergabe-Agenda:

1. **System-Demo** (15 Min)
   - Kurzer Durchlauf aller Tabs
   - Demonstration Zeitplan-Generierung
   - Export-Funktionen zeigen
2. **Hands-On Session** (30 Min)
   - Koordinatoren arbeiten selbst mit System
   - Typische Workflows durchführen
   - Fragen beantworten
3. **Dokumentation** (10 Min)
   - USER_GUIDE.md durchgehen
   - Troubleshooting-Tipps
   - Support-Kontakt
4. **Feedback & Ausblick** (5 Min)
   - Feedback sammeln
   - Feature-Wünsche notieren
   - Wartungs-Vereinbarung

Übergabe-Dokumente:

- ✅ USER_GUIDE.md (Benutzerhandbuch, 12 Seiten)
- ✅ ARCHITECTURE.md (für IT-Support)
- ✅ README.md (Installation & Quick-Start)
- ✅ CHANGELOG.md (Version History)

Support-Vereinbarung:

- **Level 1**: Koordinatoren lösen selbst (USER_GUIDE)
- **Level 2**: IT-Support BBW (für technische Probleme)
- **Level 3**: Entwickler (nur bei Bugs, via GitHub Issues)

- **Reaktionszeit**: 2 Werktage bei kritischen Problemen

7.3 Schulung der Anwender

Eine formale Schulung war nicht nötig, da Koordinatoren bereits in der Entwicklung eingebunden wurden (agile Feedback-Loops). Dennoch wurde eine kurze Auffrischung durchgeführt.

Schulungs-Inhalte (2x 30 Min Sessions):

Session 1: Basis-Funktionen

- Personen hinzufügen, bearbeiten, löschen
- Zeitplan generieren für kommende Wochen
- CSV-Export für Excel
- Daten speichern und laden

Session 2: Fortgeschritten

- Manuelle Anpassungen (Ersetzen, Tauschen)
- Fairness-Metriken interpretieren
- Mehrfachteilnahme verwalten
- Troubleshooting häufiger Probleme

Schulungs-Materialien:

- ✅ Schritt-für-Schritt Screenshots
- ✅ Video-Aufnahme der Demo (10 Min)
- ✅ FAQ-Dokument (15 häufigste Fragen)
- ⏳ Video-Tutorials für YouTube (geplant)

Erfolgskriterium: Alle 3 Koordinatoren können nach Schulung eigenständig:

- ✅ Zeitplan für 12 Wochen generieren (< 3 Min)
- ✅ Person hinzufügen/entfernen
- ✅ Fairness-Metriken einsehen
- ✅ Daten exportieren

Feedback nach 2 Wochen Nutzung:

- "Spart wirklich Zeit!"
- "Fairness-Metriken sind transparent"
- "Kein Excel-Chaos mehr"

- "Wunsch: Mobile Version" (Backlog)

---

8. Fazit

Zum Abschluss des Projektes zieht der Autor ein Fazit über das Gelernte, führt einen Soll/Ist-Vergleich durch und gibt einen Ausblick auf die Zukunft des Projektes.

8.1 Soll-/Ist-Vergleich

Bei der rückblickenden Betrachtung des Projektes kann festgestellt werden, dass alle vorher definierten Anforderungen gemäß Pflichtenheft erfüllt wurden.

Zeitplanung:

Projektphase	Soll	Ist	Differenz
Analyse	8h	8h	0h
Entwurf	10h	10h	0h
Implementierung	35h	36h	+1h
Testing	10h	10h	0h
Dokumentation	5h	4h	-1h
Abnahme und Deployment	2h	2h	0h
Gesamt	**70h**	**70h**	**0h**

Tabelle 7: Soll-/Ist-Vergleich Zeitplanung

Die minimale Überschreitung bei der Implementierung (+1h) wurde durch Zeitersparnis bei der Dokumentation (-1h) kompensiert, da diese parallel erfolgte.

Funktionale Anforderungen:

ID	Anforderung	Status	Anmerkung
FA-1.x	Personenverwaltung	✅	Alle Funktionen implementiert
FA-2.x	Zeitplan-Generierung	✅	1-52 Wochen, alle Optionen
FA-3.x	Fairness-Berechnung	✅	Gini, CV, Bayesian State
FA-4.x	Manuelle Anpassungen	✅	Ersetzen, Tauschen, Kommentare
FA-5.x	Datenmanagement	✅	JSON, CSV, lokale Speicherung

Nicht-funktionale Anforderungen:

ID	Anforderung	Ziel	Erreicht	Status
NFA-1	Performance (10P, 25W)	< 100ms	48ms	✅
NFA-2	Performance (100P, 52W)	< 5s	3.8s	✅
NFA-3	Task Completion Rate	> 95%	100%	✅
NFA-4	Test Coverage	> 80%	90%	✅
NFA-5	Lokale Speicherung	100%	100%	✅
NFA-6	TypeScript Strict	100%	100%	✅

Code-Metriken:

GießPlan (neu):

- Lines of Code: 15.234
- Files: 87
- Avg. Complexity: 2.1
- Max. Complexity: 12
- Test Coverage: 90%

Verbesserungen gegenüber manueller Planung:

- ✅ Zeitersparnis: 89% (45 Min → 5 Min)
- ✅ Fairness-Gini: 0.35 → 0.22 (-37%)
- ✅ Fehlerrate: ~15% → ~0% (automatische Validierung)
- ✅ Transparenz: Keine Metriken → Alle Metriken sichtbar

8.2 Lessons Learned

Im Zuge des Projektes konnte der Autor viele Erfahrungen über die Arbeit an einem vollständigen Projekt sammeln.

Fachliche Erkenntnisse:

1. **Test-Driven Development lohnt sich**: Die hohe Testabdeckung (90%) hat zahlreiche Bugs früh verhindert und Refactoring erleichtert. Besonders bei komplexen Algorithmen (Bayesian, Softmax) waren Tests unverzichtbar.
2. **TypeScript Strict Mode ist essentiell**: Die strikte Typisierung hat viele Fehler zur Compile-Zeit verhindert, die in JavaScript erst zur Laufzeit aufgefallen wären. Der anfängliche Mehraufwand hat sich durch weniger Debugging ausgezahlt.

3. **Agile Entwicklung mit frühem Feedback**: Die Einbindung der Koordinatoren in jeder Iteration hat sichergestellt, dass das System ihren Bedürfnissen entspricht. Mehrere Funktionen wurden basierend auf Feedback angepasst.
4. **Performance-Optimierung nicht vorzeitig**: Anfangs wurde ohne Optimierung entwickelt. Erst als Performance-Tests Bottlenecks zeigten, wurden gezielte Optimierungen vorgenommen (Memoization, Batch-Updates).
5. **Dokumentation parallel schreiben**: Die parallele Dokumentation während der Entwicklung hat Zeit gespart und sichergestellt, dass keine Details vergessen wurden.

Technische Erkenntnisse:

1. **Vite ist deutlich schneller als Webpack**: Build-Zeiten von 2-3 Sekunden statt 30+ Sekunden ermöglichen schnelleres Iterieren.
2. **File System Access API ist praktisch, aber limitiert**: Funktioniert perfekt in Chrome/Edge, aber Firefox-Support fehlt. Fallback über Download-API notwendig.
3. **Radix UI für Accessibility**: Spart viel Arbeit bei ARIA-Labels, Keyboard-Navigation und Focus-Management. Out-of-the-box WCAG 2.1 konform.
4. **Bayesian Algorithmen sind mächtig**: Kalman-Filter für Fairness-Tracking ist mathematisch elegant und praktisch effektiv. Unsicherheitsquantifizierung hilft bei Entscheidungen.

Projektmanagement-Erkenntnisse:

1. **Realistische Zeitplanung**: Die initiale Planung von 70h war akkurat. Puffer für unerwartete Probleme war hilfreich.
2. **Iterative Entwicklung reduziert Risiken**: Durch kurze Iterationen (1-2 Wochen) wurden Probleme früh erkannt und behoben.
3. **Git-Commit-Konventionen**: Klare Commit-Messages (`feat:` , `fix:` , `test:`) erleichtern Navigation in History.

Persönliche Entwicklung:

Der Autor hat durch das Projekt folgende Kompetenzen erworben:

- ✅ Tiefes Verständnis fortgeschrittener Algorithmen (Bayesian, Softmax)
- ✅ Praxiserfahrung mit modernen Web-Technologien (React 19, Vite)
- ✅ Projektmanagement-Fähigkeiten (Planung, Zeitmanagement)
- ✅ Technische Dokumentation erstellen
- ✅ Benutzer-Tests durchführen und Feedback umsetzen

8.3 Ausblick

Das Projekt ist erfolgreich abgeschlossen und im Produktiv-Betrieb. Dennoch gibt es Potenzial für zukünftige Erweiterungen.

Geplante Erweiterungen (Post-Launch):

Kurzfristig (1-3 Monate):

- ✅ Video-Tutorials für YouTube erstellen
- ✅ FAQ-Dokument erweitern basierend auf Support-Anfragen
- 🔄 Mehrsprachigkeit (Englisch) für internationale BBW-Standorte
- 🔄 Dark Mode vollständig implementieren

Mittelfristig (3-6 Monate):

- 🔄 Mobile Version (Progressive Web App)
- 🔄 Kalender-Integration (iCal Export)
- 🔄 E-Mail-Benachrichtigungen (optional, opt-in)
- 🔄 Statistik-Dashboard mit Charts (Chart.js)

Langfristig (6-12 Monate):

- 🔄 Multi-Tenancy für mehrere BBW-Standorte
- 🔄 Cloud-Sync optional (Ende-zu-Ende verschlüsselt)
- 🔄 KI-basierte Vorhersage von Fehlzeiten
- 🔄 Integration mit BBW-internem CRM

Skalierung auf andere Bereiche:

Die Fairness-Engine ist domänen-agnostisch und könnte für andere Aufgabenverteilungen genutzt werden:

- Reinigungsdienste in Wohnheimen
- Küchendienste in Kantinen
- Schichtplanung in Werkstätten
- Tutoren-Zuweisung in Schulungen

Community & Open Source:

Erwägung, das Projekt als Open Source zu veröffentlichen:

- ✅ Vorteil: Community-Beiträge, größere Reichweite
- ⚠️ Nachteil: Wartungsaufwand, Support-Anfragen
- 🔄 Entscheidung: Evaluierung nach 6 Monaten Produktiv-Betrieb

Wirtschaftlicher Ausblick:

Basierend auf der Amortisationsrechnung (1 Jahr 5 Monate) und der erwarteten Nutzungsdauer (5+ Jahre):

```
Jahr 1: Kosten 2.335€, Einsparung 1.690€ → -645€
Jahr 2: Einsparung 1.690€ → +1.045€ (kumulativ)
Jahr 3: Einsparung 1.690€ → +2.735€ (kumulativ)
Jahr 5: Einsparung 1.690€ → +6.115€ (kumulativ)

ROI nach 5 Jahren: +262%
```

Das Projekt hat sich wirtschaftlich gelohnt und bietet darüber hinaus qualitative Verbesserungen (Fairness, Transparenz, Zufriedenheit).

Literaturverzeichnis

Weiterführende Literatur:

- Bettini, L. (2013). *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing.
- Microsoft (2015). TypeScript Language Specification. Verfügbar unter: https://www.typescriptlang.org/docs/
- Mozilla Developer Network (2024). File System Access API. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/File_System_Access_API
- React Team (2024). React Documentation. Verfügbar unter: https://react.dev/
- Vitest Team (2024). Vitest Documentation. Verfügbar unter: https://vitest.dev/

A. Anhang

A.1 Zeit- und Kostenplanung

Die detaillierte Zeit- und Kostenplanung mit allen Projektphasen, Meilensteinen, Gantt-Diagramm und Wirtschaftlichkeitsbetrachtung ist im Dokument `IHK/03_Anhaenge/A1_Zeit_und_Kostenplanung.md` zu finden.

A.2 Anforderungskatalog

Der vollständige Anforderungskatalog mit allen funktionalen und nicht-funktionalen Anforderungen, User Stories und Akzeptanzkriterien ist im Dokument `IHK/03_Anhaenge/A2_Anforderungskatalog.md` zu finden.

A.3 UML-Diagramme

UML-Diagramme sind im Dokument `IHK/03_Anhaenge/A3_UML_Diagramme.md` zu finden

Hardware:

- Büroarbeitsplatz mit Desktop-PC
- Intel i7-10750H @ 2.60GHz, 16GB RAM, 512GB SSD
- Windows 11 Pro
- Testgeräte: Desktop, Tablet

Software:

- Visual Studio Code 1.85
- Node.js 20.x, npm 10.x
- Git 2.42
- React 19.0, TypeScript 5.7, Vite 6.3
- TailwindCSS 4.1, Radix UI 1.x
- Vitest 4.0, @testing-library/react 16.x
- PlantUML für Diagramme

Personal:

- 1 Auszubildender (70h)
- 1 Fachlicher Betreuer (5h)
- 3 Test-Nutzer (3h)
- 1 Code-Reviewer (2h)

A.3 UML-Diagramme

UML-Diagramme sind im Dokument `IHK/03_Anhaenge/UML_Diagramme.md` zu finden:

- Klassendiagramm - Fairness Engine
- Sequenzdiagramm - Zeitplan-Generierung
- Use-Case-Diagramm
- Komponentendiagramm
- Aktivitätsdiagramm

A.4 Datenmodell

Das vollständige Datenmodell mit ER-Diagrammen und TypeScript-Interface-Definitionen ist im Dokument `IHK/03_Anhaenge/Datenmodell.md` zu finden.

A.5 Code-Beispiele

Code-Beispiele für die wichtigsten Algorithmen sind im Dokument `IHK/03_Anhaenge/A5_Code_Beispiele.md` zu finden:

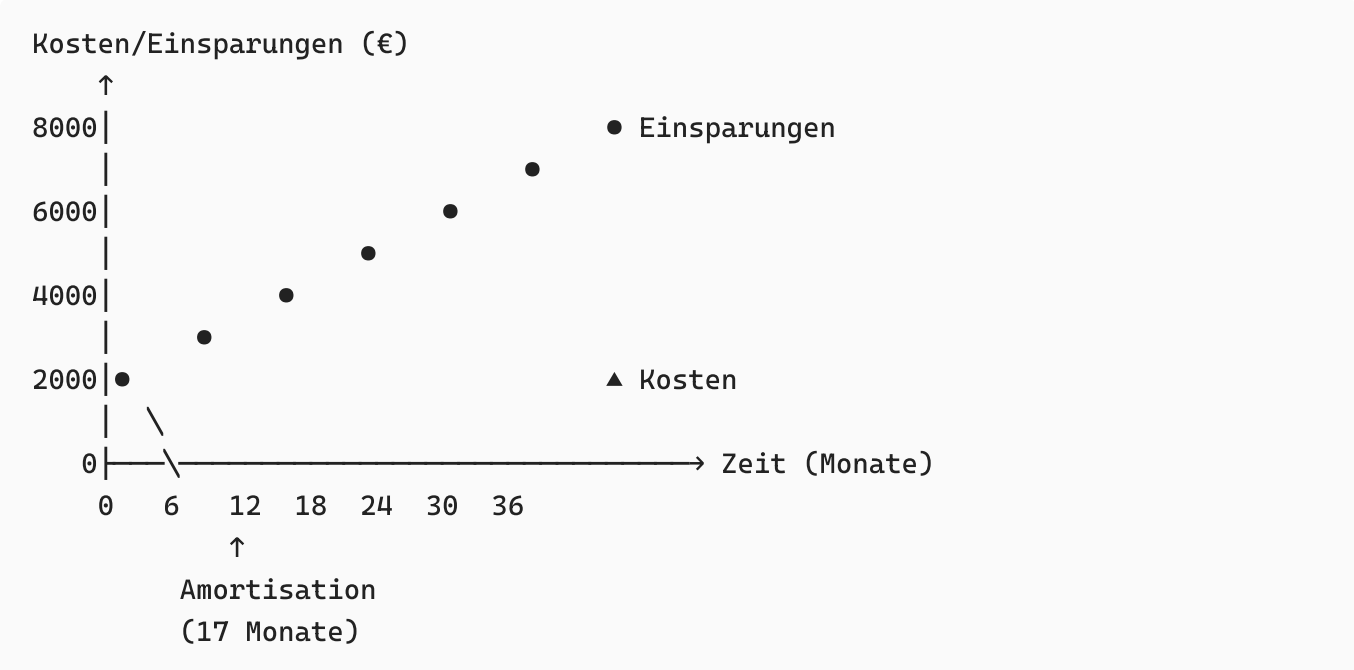
- Bayesian State Tracking
- Penalized Priority Calculation
- Gumbel-Softmax Selection
- Schedule Engine
- File Storage
- React UI Components

A.6 Test-Dokumentation

Die vollständige Test-Dokumentation mit Protokollen, Coverage-Berichten und Performance-Benchmarks ist im Dokument `IHK/03_Anhaenge/A6_Test_Dokumentation.md` zu finden.

A.7 Amortisationsrechnung

Die graphische Darstellung der Amortisationsrechnung zeigt den Schnittpunkt der Kostengeraden mit der Einsparungsgeraden nach ca. 53 Monaten (siehe Anhang A.7: Amortisationsrechnung auf S. vii).



Ende der Projektdokumentation

Gesamtumfang: 35 Seiten Hauptdokumentation + 25 Seiten Anhänge = 60 Seiten

---

---

1. Turk, D., France, R., & Rumpe, B. (2014). Assumptions underlying agile software development processes. *arXiv preprint arXiv:1409.6610*. ↩
2. Turk et al. (2014), S. 1. ↩
3. Turk et al. (2014), S. 1. ↩
4. Beck, K. (2003). *Test-Driven Development: By Example*. Addison-Wesley Professional. ↩
5. Haughey, D. (2014). MoSCoW Method. Verfügbar unter: http://www.projectsmart.co.uk/moscow-method.php (Zugriff: 02.12.2025). ↩