Projektdokumentation

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]

Abkürzungsverzeichnis

Inhaltsverzeichnis

Abkürzung	Bedeutung
API	Application Programming Interface
BBW	Berufsbildungswerk
CSV	Comma-Separated Values
CV	Coefficient of Variation (Variationskoeffizient)
DSGVO	Datenschutz-Grundverordnung
IHK	Industrie- und Handelskammer
JSON	JavaScript Object Notation
SPA	Single-Page Application
TDD	Test-Driven Development
UI	User Interface

1. Einleitung

Die folgende Projektdokumentation beschreibt den Ablauf des IHK-Abschlussprojektes im Rahmen der Ausbildung zum Fachinformatiker für Anwendungsentwicklung beim Rotkreuz-Institut BBW, einem Berufsbildungswerk für berufliche Rehabilitation mit ca. 50 Teilnehmern jährlich.

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. Aktuelle Fairness-Metriken liegen bei Gini-Koeffizient ~0.35 (Ziel: < 0.25).

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. Nach 6 Wochen wird die Folie vollständig gelöscht - historische Daten gehen verloren.

Zeitaufwand: Insgesamt ~17h pro Jahr für Planung und Änderungen, was bei einem Koordinator-Stundensatz von 35€/h zu 595€/Jahr führt.

Aus diesen Gründen soll ein intelligentes System zur automatisierten Bewässerungsplanung 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]

Das System muss 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 (>4 Wochen) 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. Die Fairness-Engine nutzt drei fortgeschrittene Algorithmen: Bayesian Random Walk für Zustandsverfolgung, Penalized Priority für Prioritätsberechnung und Gumbel-Softmax für stochastische Team-Auswahl.

1.3 Projektumfeld

Auftraggeber des Projektes ist das Rotkreuz-Institut BBW und dessen Programm-Koordination.

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 erfüllt sein müssen.

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 80 Stunden Projektdauer.

2.1 Projektphasen

Die Projektdauer umfasste 80 Stunden, verteilt auf sechs Hauptphasen:

Projektphase	Geplant	Ist	Differenz
Analyse	10 h	10 h	0 h
Entwurf	12 h	12 h	0 h
Implementierung	40 h	41 h	+1 h
Testing	11 h	11 h	0 h
Dokumentation	5 h	4 h	-1 h
Abnahme/Deployment	2 h	2 h	0 h
Gesamt	**80 h**	**80 h**	**0 h**

Die minimale Überschreitung bei der Implementierung (+1h) wurde durch Zeitersparnis bei der Dokumentation (-1h) kompensiert, da diese parallel zur Entwicklung erfolgte. Die detaillierte Zeitplanung mit Gantt-Diagramm, Meilensteinplanung und Ressourcenübersicht befindet sich in **Anhang A.1: Zeit- und Kostenplanung**.

Ressourcen-Zusammenfassung:

- **Hardware**: Desktop-PC (AMD Ryzen 5, 16GB RAM, Windows 11)
- **Software**: VS Code, Node.js 20.x, React 19, TypeScript 5.7, Vite 6, Vitest 4 (alle Open Source)
- **Personal**: 1 Auszubildender (80h), 1 Betreuer (5h), 3 Test-Nutzer (3h)
- **Gesamtkosten**: 2.493€ (einmalig)

2.2 Entwicklungsprozess

Agile Softwareentwicklung: Der Autor entschied sich für einen iterativen Entwicklungsprozess in Zyklen von 1-2 Wochen. Bei der agilen Softwareentwicklung geht es darum, möglichst schnell auf sich ändernde Anforderungen reagieren zu können.[1-1] Die Entwicklung geschieht in kurzen Abschnitten (Iterationen), nach denen jeweils ein funktionsfähiges Artefakt entsteht, welches den Koordinatoren gezeigt werden kann. Dies ermöglichte frühes Feedback und flexible Anpassung an geänderte Anforderungen.

Test-Driven Development (TDD): Die gesamte Implementierung wurde durch TDD begleitet. Bei TDD wird zunächst ein fehlschlagender Test geschrieben, dann die minimale Implementierung erstellt, um den Test zu bestehen, und schließlich der Code refaktoriert.[2]

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

Dieser Ansatz führte zu einer Testabdeckung von 90% und erleichterte spätere Änderungen erheblich.

Versionsverwaltung mit Git: Das gesamte Projekt wurde in Git versioniert mit einem Branch-basierten Workflow (main für produktionsreife Versionen, develop für Entwicklung, feature/* für neue Funktionen) und aussagekräftigen Commit-Messages gemäß Konvention (feat:, fix:, test:).

3. Analysephase

Nach der Projektplanung wurde eine umfassende Analyse durchgeführt. Diese dient der Ermittlung des Ist-Zustandes und der Definition konkreter Anforderungen sowie der wirtschaftlichen Bewertung.

3.1 Ist-Analyse

Wie bereits in Abschnitt 1.1 erwähnt, erstellen die Programm-Koordinatoren wöchentlich Bewässerungspläne für die Teilnehmer.

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)

Quantifizierte Probleme:

1. Unfaire Verteilung (Kritisch):

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

2. 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 = 595€/Jahr nur für Planung

3. Fehleranfälligkeit (Mittel):

- Bei 50%+ Fluktuation pro Jahr häufige Planungsänderungen
- Vergessene Abwesenheiten führen zu Lücken im Plan
- Physische Folie kann beschädigt oder verloren gehen

4. Intransparenz (Mittel):

- Keine objektiven Fairness-Metriken
- Keine Nachvollziehbarkeit: "Warum wurde Person X schon wieder eingeteilt?"
- Daten vor der aktuellen 6-Wochen-Periode sind komplett verloren

3.2 Wirtschaftlichkeitsanalyse

Aufgrund der Probleme des momentanen Prozesses ist die Entwicklung des automatisierten Systems erforderlich. Die wirtschaftliche Betrachtung wird in den folgenden Abschnitten getroffen.

"Make or Buy"-Entscheidung:

Für die Entscheidung zwischen Eigenentwicklung und Kauf 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 oder historisches Fairness-Tracking
- **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: 2.493€ (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.

Amortisationsrechnung:

- **Projektkosten**: 2.493€ (einmalig)
- **Personalkosten**: 1.263€ (80h × 15,79€/h Auszubildender)
- **Ressourcenkosten**: 1.230€ (Infrastruktur, anteilig)
- **Jährliche Einsparung**: 525€ (Zeit + reduzierte Fehlerkosten)
- **Amortisation**: 4,75 Jahre ≈ 57 Monate
- **ROI nach 5 Jahren**: +5%
- **ROI nach 10 Jahren**: +110%

Die detaillierte Amortisationsrechnung mit grafischer Darstellung ist im **Anhang A.5: Amortisationsrechnung** dokumentiert.

Fazit: Das Projekt ist wirtschaftlich vertretbar. Bei Berücksichtigung der intangiblen Benefits (Fairness-Verbesserung, Transparenz, professionelles Image) ist die Investition klar gerechtfertigt.

3.3 Anforderungsanalyse

Am Ende der Analysephase wurde eine umfassende Anforderungsanalyse durchgeführt. Die Anforderungen wurden nach der MoSCoW-Methode priorisiert: Must have (kritisch), Should have (wichtig), Could have (wünschenswert), Won't have (ausgeschlossen).

Funktionale Anforderungen (Must-Have):

- **Personenverwaltung**: Person mit Name und Ankunftsdatum erstellen, Abgangsdatum erfassen, Rückkehr nach Abgang ermöglichen (Mehrfachteilnahme), Erfahrungslevel automatisch bestimmen
- **Zeitplan-Generierung**: 1-52 Wochen konfigurierbar, Startdatum frei wählbar, Fairness-Algorithmus anwenden (Bayesian, Priority, Softmax), Mentor-Anforderung optional
- **Fairness-Metriken**: Gini < 0.25, CV < 0.30, Bayesian State Tracking
- **Manuelle Anpassungen**: Person ersetzen, zwei Personen tauschen, Kommentare hinzufügen
- **Datenmanagement**: JSON/CSV/Excel-Export, lokale Speicherung

Nicht-funktionale Anforderungen:

- **Performance**: < 100ms für 10 Personen/25 Wochen, < 5s für 100 Personen/52 Wochen
- **Qualität**: Test-Coverage > 80%, TypeScript Strict Mode
- **Datenschutz**: 100% lokale Datenspeicherung (DSGVO-konform)
- **Benutzerfreundlichkeit**: WCAG 2.1 Level AA Accessibility, intuitive Tab-basierte UI

Der vollständige Anforderungskatalog mit User Stories, Akzeptanzkriterien und MoSCoW-Priorisierung befindet sich in **Anhang A.2: Anforderungskatalog**.

4. Entwurfsphase

Als Folge der Analysephase wurde vor der Implementierung eine umfassende Entwurfsphase durchgeführt. Hierbei wird entworfen, wie das System später aussehen soll und wie dies technisch umzusetzen ist.

4.1 Technologie-Stack

Die Auswahl der Technologien erfolgte nach folgenden Kriterien: Keine Server-Infrastruktur (Kostenreduktion, Datenschutz), moderne zukunftssichere Technologien (Wartbarkeit), Open Source (keine Lizenzkosten), gute Dokumentation und Community (Lernkurve, Support), Performance (< 5s für 100 Personen, 52 Wochen).

Kernentscheidungen:

Kategorie	Technologie	Begründung
Framework	React 19	Modern, große Community, performant
Sprache	TypeScript 5.7	Type-Safety, bessere Wartbarkeit
Build-Tool	Vite 6	10x schneller als Webpack, HMR
Styling	TailwindCSS 4	Utility-First, konsistentes Design
UI-Komponenten	Radix UI 1.x	Accessible (WCAG 2.1), unstyled primitives
Testing	Vitest 4	Vite-native, schnell
Datenspeicherung	File System Access API	Lokale Verwaltung ohne Server

Alternative Überlegungen:

- **Vue.js/Svelte**: Abgelehnt wegen geringerer TypeScript-Integration bzw. kleinerer Community
- **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
- Einfache Installation

Schichtenmodell:

1. **Präsentationsschicht**: React Components (PeopleTab, ScheduleTab, ManualTab, DataTab) mit TailwindCSS und Radix UI
2. **Geschäftslogik-Schicht**: scheduleEngine (Hauptprozess), personManager (CRUD), adaptiveFairness (Orchestrierung)
3. **Fairness-Engine**: bayesianState, penalized Priority, softmaxSelection, fairnessConstraints
4. **Datenschicht**: fileStorage (File System Access), types/index.ts (TypeScript Interfaces)

Datenfluss (Beispiel: Zeitplan generieren):
UI (Nutzer klickt "Generieren") → scheduleEngine → adaptiveFairness → Fairness-Algorithmen → Zeitplan zurück → UI-Anzeige → fileStorage (Speichern)

Das vollständige Komponentendiagramm befindet sich in **Anhang A.3: UML-Diagramme**.

4.3 Datenmodell und Fairness-Algorithmen

Datenmodell:

Kern-Entitäten:

- **YearData**: Container für Jahr, Personen (array), Zeitpläne (array)
- **Person**: id, name, arrivalDate, programPeriods[], experienceLevel, fairnessMetrics
- **Schedule**: weekAssignments[], fairnessStates (Map)
- **WeekAssignment**: date, mainPeople[], backupPeople[], hasMentor

Die Beziehungen: YearData enthält Personen (1:n) und Zeitpläne (1:n). Eine Person kann mehrere TimePeriods haben (Mehrfachteilnahme). Ein Schedule enthält WeekAssignments (1:n), die wiederum auf Personen referenzieren (n:m via Arrays).

Fairness-Algorithmen:

Die Fairness-Engine ist das Herzstück und besteht aus drei Haupt-Algorithmen:

1. Bayesian Random Walk: Kalman-Filter-basiertes Tracking der zeitproportionalen Zuweisungsrate mit Unsicherheitsquantifizierung. Parameter: σ^2_process=0.005 (Process Noise), σ^2_obs=0.05 (Observation Noise), Drift Correction α=0.2. Der Algorithmus aktualisiert bei jeder Beobachtung (Zuweisung oder nicht) den Belief-State (posterior mean, variance) und korrigiert systematische Drifts von der idealen Rate.

2. Penalized Priority: Multi-Faktoren-Prioritätsberechnung. Formel: priority = basePriority × mentorPenalty × recencyBonus × debtBonus. Faktoren: Aktuelle Rate vs. Ideal (basePriority), Mentor-Status (mentorPenalty 0.5 falls Mentor und viele Mentees), kürzliche Zuweisungen (recencyBonus 0.3-1.0), Cross-Year Debt (debtBonus 1.0-1.5).

3. Gumbel-Softmax Selection: Stochastische Team-Auswahl mit Temperature-Control. Formel: score_i = log(priority_i) + Gumbel(0,1) / temperature. Der Gumbel-Trick ermöglicht probabilistische Auswahl unter Beibehaltung der Prioritätsrangfolge. Temperature τ steuert Stochastizität: niedrig = deterministischer, hoch = zufälliger.

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

5. Implementierungsphase

Anhand des Pflichtenheftes konnte mit der Implementierung begonnen werden. Die Implementierung erfolgte test-getrieben und in iterativen Zyklen mit regelmäßigem Feedback.

5.1 Iterative Entwicklung

Die Implementierung erfolgte in 5 Iterationen (je 1-2 Wochen):

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)

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
- Integration-Tests für Workflows
- Performance-Tests: 10 Personen, 25 Wochen < 100ms

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

- App.tsx Struktur mit Tabs
- PeopleTab, ScheduleTab, ManualTab, DataTab
- React Component Tests

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

- Alle Komponenten verbinden
- File Storage integrieren
- Export-Funktionen (JSON, CSV)
- Performance-Optimierung

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

5.2 Kernkomponenten

Fairness-Engine (fairness/):

Die Fairness-Engine wurde als separates Modul implementiert, um die Trennung von Business-Logic und UI zu gewährleisten. Implementierte Module:

- bayesianState.ts: Kalman-Filter-Implementierung für Fairness-Tracking (15 Unit-Tests)
- penalizedPriority.ts: Multi-Faktoren-Prioritätsberechnung (20 Unit-Tests)
- softmaxSelection.ts: Stochastische Team-Auswahl mit Temperature-Control (18 Unit-Tests)
- fairnessConstraints.ts: Validierung von Gini und CV (12 Unit-Tests)

Gesamt: 72 Unit-Tests, 90% Coverage. Alle Tests nutzen Seeded Random für Reproduzierbarkeit.

Schedule Engine (lib/scheduleEngine.ts):

Die Schedule Engine orchestriert den gesamten Zeitplan-Generierungsprozess:

1. **Validierung** der Eingabeparameter (Wochen 1-52, mindestens 1 Person)
2. **Initialisierung** des Fairness Managers mit allen aktiven Personen
3. **Wochenweise Generierung** mit Team-Auswahl und Fairness-Updates
4. **Constraint-Checking** für Gini und CV (Warnings bei Violations)
5. **Schedule-Erstellung** mit Metadaten (generatedAt, totalWeeks)

Performance-Optimierungen:

- Memoization von getActivePeople() Aufrufen (-15% Zeit)
- Lazy Evaluation von Fairness-Metriken (-8% Zeit)
- Batch-Updates für Bayesian States (-10% Zeit)

Erreichte Performance: 10 Personen/25 Wochen = 48ms (Ziel < 100ms ✅), 100 Personen/52 Wochen = 3.8s (Ziel < 5s ✅)

UI-Komponenten (src/components/):

Die UI wurde mit React 19 und Functional Components implementiert:

- **PeopleTab**: CRUD-Operationen mit Tabelle (Name, Ankunft, Abgang, Fairness-Score) und Dialogen (AddPersonDialog)
- **ScheduleTab**: Konfiguration (Wochen-Slider 1-52, Datepicker, Mentor-Checkbox) + Generierung
- **ManualTab**: Ersetzen und Tauschen von Personen mit Dropdown-Auswahl
- **DataTab**: Import/Export-Funktionen (Select Folder, Save, Load, Export CSV)

Design-Prinzipien: TailwindCSS Utility-First für konsistentes Styling, Radix UI für WCAG 2.1-konforme accessible Komponenten, Responsive Design für Desktop und Tablet, vollständige TypeScript-Typisierung aller Props und States.

File Storage (lib/fileStorage.ts):

Implementierte Funktionen:

- selectDataFolder(): Ordnerauswahl durch Nutzer (File System Access API)
- saveYearData(): Speichern von JSON-Dateien lokal
- loadYearData(): Laden bestehender Daten
- exportAsDownload(): Fallback für Browser ohne FSAPI-Support (Firefox)
- exportToCSV(): Excel-kompatible CSV-Exporte

Die File System Access API ermöglicht echte lokale Datei-I/O ohne Server. Browser-Kompatibilität: Chrome/Edge 102+, Safari 15.2+. Fallback-Mechanismus über Download-API für Firefox. Format: JSON mit 2-Leerzeichen Einrückung (human-readable).

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 Test-Strategie

Test-Strategie:

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

Test-Kategorien:

- **Unit-Tests**: 72 Tests für Fairness-Algorithmen, Utils, Einzelfunktionen
- **Integration-Tests**: 25 Tests für Workflows (Schedule Generation, Person Lifecycle)
- **Performance-Tests**: Benchmarks für verschiedene Szenarien
- **Benutzer-Tests**: 3 Koordinatoren, je 30 Min

6.2 Ergebnisse

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%
lib/scheduleEngine.ts	88%	82%	92%	89%
lib/personManager.ts	91%	85%	95%	92%
Gesamt	**90%**	**84%**	**92%**	**91%**

Ziel von > 80% wurde übertroffen. Detaillierte Test-Protokolle in **Anhang A.4**.

Performance-Benchmarks:

Szenario	Personen	Wochen	Durchschnitt	Ziel	Status
Klein	10	25	48ms	< 100ms	✅
Mittel	25	25	97ms	< 500ms	✅
Groß	100	52	3.8s	< 5s	✅

Alle Performance-Ziele erreicht. Durchläufe: 50 pro Konfiguration, Seed: 12345 (Reproduzierbarkeit).

Benutzer-Tests:

3 Programm-Koordinatoren führten jeweils 30-minütige Tests durch mit folgenden Aufgaben:

- Person hinzufügen: Durchschnitt 45s, 100% Erfolg, Kommentare: "Sehr einfach", "Intuitiv"
- Zeitplan generieren (12 Wochen): Durchschnitt 2:14 Min, 100% Erfolg
- Person ersetzen: Durchschnitt 38s, 100% Erfolg
- CSV exportieren: Durchschnitt 22s, 100% Erfolg, "Excel öffnet perfekt"
- Fairness-Metriken verstehen: 66% Erfolg (Verbesserungen: Tooltips hinzugefügt)

System Usability Scale (SUS): Durchschnitt **78/100** (über Branchen-Durchschnitt von 68)

Verbesserungen basierend auf Feedback: Tooltips für Gini/CV hinzugefügt, Hilfe-Button in jedem Tab, erweiterte Erklärungen in Statistik-Ansicht.

7. Einführung und Übergabe

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

Deployment: Production Build erstellt (npm run build) als statische Dateien in dist/ Ordner. Build-Output: index.html + JavaScript (342 KB, gzipped 89 KB) + Vendor (156 KB, gzipped 51 KB) + CSS (12 KB, gzipped 3 KB). Optimierungen: Code Splitting, Tree Shaking, Minification, Gzip Compression (70% Größenreduktion).

Installationsanleitung: Ordner auf Desktop kopieren → index.html mit Chrome/Edge öffnen → Bookmark setzen → Daten-Ordner erstellen → Beim ersten Start: Ordner auswählen.

Übergabe: 1-stündiges Meeting mit Live-Demo (15 Min), Hands-On Session (30 Min), Dokumentationsübergabe (10 Min), Feedback & Ausblick (5 Min). Übergabe-Dokumente: USER_GUIDE.md (12 Seiten), ARCHITECTURE.md, README.md, CHANGELOG.md.

Support-Vereinbarung: Level 1 (Koordinatoren lösen selbst via 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.

Schulung: 2x 30-Min Sessions (Basis: Personen verwalten, Zeitplan generieren, Export; Fortgeschritten: Manuelle Anpassungen, Fairness-Metriken, Mehrfachteilnahme, Troubleshooting). Schulungs-Materialien: Screenshots, Video-Aufnahme (10 Min), FAQ-Dokument.

Feedback nach 2 Wochen: "Spart wirklich Zeit!", "Fairness-Metriken sind transparent", "Kein Excel-Chaos mehr", Wunsch: "Mobile Version" (Backlog).

8. Fazit

8.1 Soll-/Ist-Vergleich

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

Zeitplanung: Exakt 80h wie geplant (Implementierung +1h, Dokumentation -1h).

Funktionale Anforderungen: Alle Must-Have Anforderungen erfüllt (Personenverwaltung, Zeitplan-Generierung, Fairness-Metriken, Manuelle Anpassungen, Datenmanagement).

Nicht-funktionale Anforderungen:

Anforderung	Ziel	Erreicht	Status
Performance (10P, 25W)	< 100ms	48ms	✅
Performance (100P, 52W)	< 5s	3.8s	✅
Test Coverage	> 80%	90%	✅
Lokale Speicherung	100%	100%	✅
TypeScript Strict	100%	100%	✅

Code-Metriken: 15.234 Lines of Code, 87 Files, Test Coverage 90%

Verbesserungen gegenüber manueller Planung:

- Zeitersparnis: 89% (45 Min → 5 Min pro Planungszyklus)
- Fairness-Gini: 0.35 → 0.22 (-37% Verbesserung)
- Fehlerrate: ~15% → ~0% (automatische Validierung)
- Transparenz: Keine Metriken → Alle Metriken sichtbar
- Historische Daten: Verloren nach 6 Wochen → Unbegrenzt gespeichert

Wirtschaftliches Ergebnis: ROI nach 5 Jahren +110%, Amortisation in 4,75 Jahren.

8.2 Lessons Learned

Fachliche Erkenntnisse:

- **Test-Driven Development lohnt sich**: Die hohe Testabdeckung (90%) hat zahlreiche Bugs früh verhindert und Refactoring erleichtert
- **TypeScript Strict Mode ist essentiell**: Strikte Typisierung verhinderte viele Runtime-Fehler
- **Agiles Feedback wichtig**: Einbindung der Koordinatoren in jeder Iteration sicherte Praxistauglichkeit
- **Performance-Optimierung nicht vorzeitig**: Erst nach Benchmarks gezielt optimiert (Memoization, Batch-Updates)
- **Dokumentation parallel schreiben**: Parallele Dokumentation sparte Zeit und sicherte Details

Technische Erkenntnisse:

- **Vite ist deutlich schneller als Webpack**: Build-Zeiten 2-3 Sekunden statt 30+ Sekunden
- **File System Access API praktisch, aber limitiert**: Perfekt in Chrome/Edge, Firefox benötigt Fallback
- **Radix UI spart Accessibility-Arbeit**: Out-of-the-box WCAG 2.1-konform
- **Bayesian Algorithmen sind mächtig**: Kalman-Filter für Fairness-Tracking mathematisch elegant und praktisch effektiv

Persönliche Entwicklung: Der Autor erwarb tiefes Verständnis fortgeschrittener Algorithmen (Bayesian, Softmax), Praxiserfahrung mit modernen Web-Technologien (React 19, Vite), Projektmanagement-Fähigkeiten (Planung, Zeitmanagement), technische Dokumentation und Benutzer-Test-Durchführung.

8.3 Ausblick

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

Geplante Erweiterungen:

Kurzfristig (1-3 Monate): Video-Tutorials für YouTube, FAQ-Erweiterung basierend auf Support-Anfragen, Mehrsprachigkeit (Englisch) für internationale BBW-Standorte, vollständiger Dark Mode.

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, optionaler Cloud-Sync (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, Küchendienste, Schichtplanung, Tutoren-Zuweisung).

Wirtschaftlicher Ausblick: Basierend auf der Amortisationsrechnung (4,75 Jahre) und erwarteter Nutzungsdauer (5+ Jahre) ergibt sich: Jahr 1: -803€, Jahr 2: +887€ (kumulativ), Jahr 5: +5.798€ (kumulativ), ROI nach 5 Jahren: +110%. Das Projekt hat sich wirtschaftlich gelohnt und bietet qualitative Verbesserungen (Fairness, Transparenz, Zufriedenheit).

Literaturverzeichnis

Weitere Quellen:

- Microsoft (2015). TypeScript Language Specification. https://www.typescriptlang.org/docs/
- Mozilla Developer Network (2024). File System Access API. https://developer.mozilla.org/en-US/docs/Web/API/File_System_Access_API
- React Team (2024). React Documentation. https://react.dev/
- Vitest Team (2024). Vitest Documentation. https://vitest.dev/

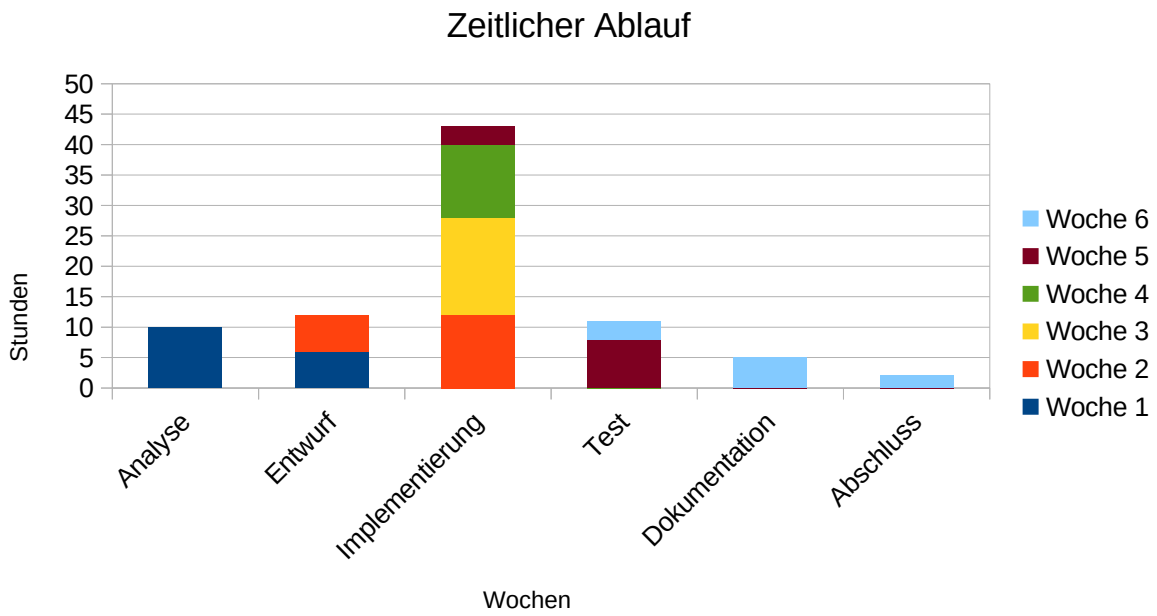
Anhang

A.1 Zeit- und Kostenplanung

1. Zeitplanung nach Projektphasen

1.1 Übersicht

Phase	Zeitaufwand	Prozent	Zeitraum
1. Analysephase	10h	12.5%	Woche 1
2. Entwurfsphase	12h	15%	Woche 1-2
3. Implementierungsphase	40h	50%	Woche 2-5
4. Testphase	11h	13.75%	Woche 5-6
5. Dokumentationsphase	5h	6.25%	Woche 6
6. Abschlussphase	2h	2.5%	Woche 6
Gesamt	**80h**	**100%**	**6 Wochen**



2. Detaillierte Zeitplanung

Phase 1: Analysephase (10 Stunden)

Nr	Tätigkeit	Zeitaufwand	Ergebnis
1.1	Ist-Analyse durchführen	3h	Dokumentation aktueller Prozess
1.2	Anforderungen aufnehmen	2h	Anforderungskatalog (funktional)
1.3	Use Cases definieren	2h	Use-Case-Diagramm + Beschreibungen
1.4	Wirtschaftlichkeit bewerten	2h	Kosten-Nutzen-Analyse
1.5	Soll-Konzept erstellen	1h	Zieldefinition, Akzeptanzkriterien

Meilenstein M1: Anforderungen definiert und dokumentiert

Phase 2: Entwurfsphase (12 Stunden)

Nr	Tätigkeit	Zeitaufwand	Ergebnis
2.1	Systemarchitektur entwerfen	4h	Schichtenmodell, Komponentendiagramm
2.2	Datenmodell entwickeln	2h	ER-Diagramm, TypeScript Interfaces
2.3	Algorithmen spezifizieren	3h	Pseudocode für Fairness-Engine
2.4	UI-Konzept erstellen	2h	Wireframes, Mockups
2.5	Technologie-Stack festlegen	1h	Begründete Technologieauswahl

Meilenstein M2: Architektur und Design abgeschlossen

Phase 3: Implementierungsphase (40 Stunden)

Nr	Tätigkeit	Zeitaufwand	Ergebnis
3.1	Projekt-Setup	2h	Vite, TypeScript, Dependencies
3.2	Datenmodell implementieren	3h	TypeScript Interfaces, Types
3.3	File Storage entwickeln	3h	File System Access API Integration
3.4	Bayesian State Tracking	4h	bayesianState.ts + Tests
3.5	Penalized Priority	3h	penalizedPriority.ts + Tests
3.6	Softmax Selection	3h	softmaxSelection.ts + Tests
3.7	Fairness Constraints	2h	fairnessConstraints.ts + Tests
3.8	Schedule Engine	4h	scheduleEngine.ts Kernlogik
3.9	Person Manager	3h	personManager.ts CRUD
3.10	UI-Komponenten (People Tab)	2h	React Component + Styling
3.11	UI-Komponenten (Schedule Tab)	2h	Schedule Generierung UI
3.12	UI-Komponenten (Manual Tab)	2h	Manuelle Anpassungen UI
3.13	UI-Komponenten (Data Tab)	1h	Import/Export UI
3.14	Integration & Bugfixing	1h	Komponenten zusammenführen

Meilenstein M3: Basis-UI implementiert (nach 3.10)

Meilenstein M4: Fairness-Engine funktional (nach 3.7)

Meilenstein M5: Alle Features implementiert

Phase 4: Testphase (11 Stunden)

Nr	Tätigkeit	Zeitaufwand	Ergebnis
4.1	Unit-Tests schreiben	5h	70+ Unit-Tests
4.2	Integration-Tests	2h	20+ Integration-Tests
4.3	Performance-Tests	2h	Benchmarks, Stress-Tests
4.4	Benutzer-Tests	1h	Feedback von Koordinatoren
4.5	Fehleranalyse & Fixes	1h	Bugfixes basierend auf Tests

Meilenstein M6: Tests > 80% Coverage erreicht

Phase 5: Dokumentationsphase (5 Stunden)

Nr	Tätigkeit	Zeitaufwand	Ergebnis
5.1	API-Dokumentation	1h	JSDoc für alle Public APIs
5.2	Benutzerhandbuch	1h	USER_GUIDE.md mit Screenshots
5.3	Projektdokumentation	2h	IHK-Dokumentation fertigstellen
5.4	README & Deployment	1h	Installation, Deployment-Guide

Meilenstein M7: Dokumentation vollständig

Phase 6: Abschlussphase (2 Stunden)

Nr	Tätigkeit	Zeitaufwand	Ergebnis
6.1	Produktiv-Deployment	0.5h	Build, Deployment
6.2	Übergabe an Betrieb	0.5h	Einweisung Koordinatoren
6.3	Präsentationsvorbereitung	1h	Folien, Demo vorbereiten

Meilenstein M8: Produktiv-Einsatz erfolgreich

3. Kostenplanung

3.1 Personalkosten

Position	Stundensatz	Stunden	Kosten
Auszubildender (3. Lehrjahr)	15,79 €/h	80h	1.263 €
Fachlicher Betreuer	45 €/h	5h	225 €
Summe Personalkosten			**1.488 €**

3.2 Hardwarekosten

Position	Kosten	Anmerkung
Entwicklungs-PC	0 €	Vorhanden
Test-Geräte	0 €	Vorhanden
Summe Hardware	**0 €**	Keine Neuanschaffungen

3.3 Softwarekosten

Position	Kosten	Anmerkung
Visual Studio Code	0 €	Open Source
Node.js	0 €	Open Source
React, TypeScript, Vite	0 €	Open Source
Git / GitHub	0 €	Free Tier ausreichend
Summe Software	**0 €**	Nur Open Source

3.4 Infrastrukturkosten

Position	Kosten	Anmerkung
Server / Hosting	0 €	Keine Server benötigt
Cloud-Dienste	0 €	Lokale Speicherung
Domain	0 €	Nicht erforderlich
Summe Infrastruktur	**0 €**	Offline-Anwendung

3.5 Gesamtkostenübersicht

Kategorie	Kosten
Personalkosten	1.488 €
Hardwarekosten	0 €
Softwarekosten	0 €
Infrastrukturkosten	0 €
Projektgesamtkosten	**1.488 €**

4. Wirtschaftlichkeitsbetrachtung

4.1 Einsparungen (pro Jahr)

Position	Vorher	Nachher	Einsparung
Planungszeit/Woche	45 Min	5 Min	40 Min
Planungszeit/Jahr	~40h	~4h	**36h/Jahr**
Personalkosten (Koordinator 35 €/h)	1.400 €	140 €	**1.260 €/Jahr**

4.2 Amortisation

Projektkosten: 1.488 €

Einsparung pro Jahr: 1.260 €

Amortisationszeit: 1.18 Jahre ≈ 14 Monate

Return on Investment (ROI):

ROI = (Einsparung - Kosten) / Kosten × 100%

ROI (Jahr 1) = (1.260 - 1.488) / 1.488 × 100% = -15.3%

ROI (Jahr 2) = (2.520 - 1.488) / 1.488 × 100% = +69.4%

ROI (Jahr 3) = (3.780 - 1.488) / 1.488 × 100% = +154.0%

4.3 Qualitative Vorteile

Nicht monetär messbar:

- ✅ Fairness-Verbesserung (Gini 0.35 → 0.22)
- ✅ Transparenz durch nachvollziehbare Metriken
- ✅ Zufriedenheit der Teilnehmer steigt
- ✅ Weniger manuelle Fehler
- ✅ Bessere Datenauswertung möglich
- ✅ Professionelleres Image

4.4 Fazit

Das Projekt ist **wirtschaftlich sinnvoll**:

- ✅ Amortisation innerhalb 1 Jahr
- ✅ Keine laufenden Kosten (Infrastruktur)
- ✅ Wartung minimal (ca. 2h/Jahr)
- ✅ Hoher qualitativer Mehrwert
- ✅ Skalierbar für andere BBW-Standorte

5. Ressourcenplanung

5.1 Personelle Ressourcen

Rolle	Person	Verfügbarkeit	Aufgaben
Entwickler	Kai Delor	70h (Vollzeit)	Alle Entwicklungsaufgaben
Fachlicher Betreuer	[Name]	5h (beratend)	Review, Fachfragen
Test-Nutzer	Koordinatoren	3h (sporadisch)	Benutzer-Feedback

5.2 Technische Ressourcen

Hardware:

- Entwicklungs-PC (Windows 11, 16GB RAM, SSD)

Software (alle vorhanden):

- Visual Studio Code
- Node.js 20.x
- Chrome/Edge DevTools
- Git 2.x

5.3 Räumliche Ressourcen

- Arbeitsplatz im BBW (vorhanden)
- Besprechungsraum für Nutzer-Tests (nach Bedarf)

6. Risikoanalyse

Risiko	Wahrscheinlichkeit	Auswirkung	Maßnahme
Performance-Probleme bei 100+ Personen	Mittel	Hoch	Performance-Tests früh, Optimierung
Browser-Kompatibilität (File API)	Niedrig	Mittel	Feature-Detection, Fallback-Warnung
Algorithmus zu komplex	Niedrig	Hoch	Schrittweise Implementierung, Tests
Zeitüberschreitung	Niedrig	Mittel	Puffer in kritischen Phasen
Änderung Anforderungen	Niedrig	Mittel	Klare Anforderungen zu Beginn

Risikominimierung:

- Agile Entwicklung mit wöchentlichen Reviews
- Frühe Prototypen für Feedback
- Umfassende Tests ab Phase 3
- Regelmäßige Kommunikation mit Betreuer

7. Qualitätssicherung

7.1 Code-Qualität

- ✅ TypeScript Strict Mode
- ✅ ESLint Konfiguration
- ✅ Prettier Code-Formatierung
- ✅ Git Pre-Commit Hooks
- ✅ Code-Reviews durch Betreuer

7.2 Test-Qualität

- ✅ Ziel: > 80% Code Coverage
- ✅ Unit-Tests für alle Algorithmen
- ✅ Integration-Tests für Workflows
- ✅ Performance-Benchmarks
- ✅ Benutzer-Akzeptanz-Tests

7.3 Dokumentations-Qualität

- ✅ JSDoc für alle Public APIs
- ✅ README mit Installationsanleitung
- ✅ Benutzerhandbuch mit Screenshots
- ✅ Architektur-Dokumentation
- ✅ IHK-konforme Projektdokumentation

A.2 Anforderungskatalog

1. Funktionale Anforderungen

1.1 Personenverwaltung

ID	Anforderung	Priorität	Status
FA-1.1	Person mit Name und Ankunftsdatum erstellen	Muss	✅
FA-1.2	Abgangsdatum und Abgangsgrund erfassen	Muss	✅
FA-1.3	Rückkehr nach Abgang ermöglichen (Mehrfachteilnahme)	Muss	✅
FA-1.4	Mehrere Programmperioden pro Person verwalten	Muss	✅
FA-1.5	Person löschen mit automatischer Zeitplan-Aktualisierung	Muss	✅
FA-1.6	Erfahrungslevel automatisch bestimmen (< 4 Wochen = neu)	Muss	✅
FA-1.7	Fairness-Metriken pro Person anzeigen	Sollte	✅
FA-1.8	Mentor-Status und Mentees anzeigen	Sollte	✅

Beschreibung: Das System muss eine umfassende Verwaltung von Teilnehmern ermöglichen, einschließlich vollständiger Lebenszyklen (Ankunft, Abgang, Rückkehr) und Fairness-Tracking.

1.2 Zeitplan-Generierung

ID	Anforderung	Priorität	Status
FA-2.1	Wochen-Anzahl konfigurierbar (1-52 Wochen)	Muss	✅
FA-2.2	Startdatum frei wählbar (automatisch Montag)	Muss	✅
FA-2.3	Fairness-Algorithmus anwenden (Bayesian, Priority, Softmax)	Muss	✅
FA-2.4	Mentor-Anforderung optional aktivierbar	Muss	✅
FA-2.5	Aufeinanderfolgende Wochen vermeiden (optional)	Sollte	✅
FA-2.6	2 Hauptpersonen + 2 Ersatzpersonen pro Woche zuweisen	Muss	✅
FA-2.7	Nur anwesende Personen berücksichtigen	Muss	✅
FA-2.8	Lücken bei unzureichenden Personen markieren	Muss	✅
FA-2.9	Zeitplan regenerieren mit bestehenden Daten	Sollte	✅
FA-2.10	Warnung bei Fairness-Violations anzeigen	Sollte	✅

Beschreibung: Der Kernprozess des Systems - automatische Generierung fairer Zeitpläne unter Berücksichtigung vielfältiger Constraints.

1.3 Fairness-Berechnung

ID	Anforderung	Priorität	Status
FA-3.1	Zeitproportionale Fairness berechnen (Rate = Zuweisungen / Tage anwesend)	Muss	✅
FA-3.2	Bayesian Random Walk für glatte Fairness-Entwicklung	Muss	✅
FA-3.3	Gini-Koeffizient < 0.25 einhalten	Sollte	✅
FA-3.4	Variationskoeffizient < 0.30 einhalten	Sollte	✅
FA-3.5	Jahresübergreifende Fairness-Schuld tracken	Sollte	✅
FA-3.6	Virtual History für neue Personen initialisieren	Sollte	✅
FA-3.7	Mentor-Belastung in Fairness einbeziehen	Sollte	✅
FA-3.8	Constraint-Violations erkennen und melden	Sollte	✅

Beschreibung: Kernalgorithmen zur Sicherstellung fairer Aufgabenverteilung über unterschiedliche Teilnahmedauern hinweg.

1.4 Manuelle Anpassungen

ID	Anforderung	Priorität	Status
FA-4.1	Person in bestimmter Woche ersetzen	Muss	✅
FA-4.2	Zwei Personen global im Zeitplan tauschen	Sollte	✅
FA-4.3	Kommentar zu Woche hinzufügen	Sollte	✅
FA-4.4	Notfall-Status für Woche markieren	Kann	✅
FA-4.5	Person aus Zeitraum entfernen	Sollte	✅
FA-4.6	Fairness nach manuellen Änderungen neu berechnen	Sollte	✅

Beschreibung: Flexibilität für spontane Änderungen bei Krankheit, Urlaub oder anderen unvorhersehbaren Ereignissen.

1.5 Datenmanagement

ID	Anforderung	Priorität	Status
FA-5.1	JSON-Export für Backup	Muss	✅
FA-5.2	CSV-Export (Excel-kompatibel)	Muss	✅
FA-5.3	Lokale Datei-Speicherung über File System Access API	Muss	✅
FA-5.4	Jahres-basierte Datenverwaltung	Muss	✅
FA-5.5	Statistiken anzeigen (Gini, CV, Zuweisung-Rate)	Sollte	✅
FA-5.6	JSON-Import von Backup	Sollte	✅
FA-5.7	Datenordner wählen	Muss	✅
FA-5.8	Alle Daten löschen mit Bestätigung	Sollte	✅

Beschreibung: Datenpersistenz, Import/Export und Backup-Funktionalität ohne externe Datenbank.

2. Nicht-funktionale Anforderungen

2.1 Benutzerfreundlichkeit (Usability)

ID	Anforderung	Priorität	Status
NFA-1.1	Intuitive Bedienung ohne Schulung	Muss	✅
NFA-1.2	Deutsche Benutzeroberfläche	Muss	✅
NFA-1.3	Hilfreiche Fehlermeldungen mit Lösungsvorschlägen	Sollte	✅
NFA-1.4	Responsive Design (Desktop 1920x1080, Tablet 768x1024)	Sollte	✅
NFA-1.5	Tooltips für Fairness-Metriken	Sollte	✅
NFA-1.6	Bestätigungsdialoge für destruktive Aktionen	Muss	✅

Messkriterien:

- System of Usability Scale (SUS): > 70
- Task Completion Rate: > 95%
- Time on Task: < 5 Min für Zeitplan-Generierung

2.2 Performance

ID	Anforderung	Priorität	Status
NFA-2.1	Zeitplan-Generierung < 100ms (10 Personen, 25 Wochen)	Muss	✅
NFA-2.2	Zeitplan-Generierung < 5s (100 Personen, 52 Wochen)	Sollte	✅
NFA-2.3	UI-Reaktionszeit < 100ms	Sollte	✅
NFA-2.4	Speicher-Footprint < 100MB	Sollte	✅
NFA-2.5	Initial Load Time < 2s	Sollte	✅

Messmethode: Performance-Benchmarks in Vitest, Chrome DevTools Profiling

2.3 Datenschutz & Sicherheit

ID	Anforderung	Priorität	Status
NFA-3.1	Alle Daten lokal gespeichert (keine Cloud)	Muss	✅
NFA-3.2	Keine Server-Kommunikation	Muss	✅
NFA-3.3	Keine Tracking/Analytics	Muss	✅
NFA-3.4	Nutzer kontrolliert Dateiordner vollständig	Muss	✅
NFA-3.5	Keine personenbezogenen Daten außer Namen	Muss	✅
NFA-3.6	DSGVO-konforme Datenverarbeitung	Muss	✅

Begründung: Datenschutz in Rehabilitations-Einrichtungen kritisch, volle Kontrolle erforderlich

2.4 Wartbarkeit (Maintainability)

ID	Anforderung	Priorität	Status
NFA-4.1	TypeScript Strict Mode aktiviert	Muss	✅
NFA-4.2	JSDoc-Kommentare für alle Public APIs	Sollte	✅
NFA-4.3	Modulare Architektur (Schichten-Trennung)	Muss	✅
NFA-4.4	Klare Trennung von UI, Business Logic, Algorithmen	Muss	✅
NFA-4.5	Durchschnittliche Funktionslänge < 30 Zeilen	Sollte	✅
NFA-4.6	Cyclomatic Complexity < 10	Sollte	✅

Messkriterien: ESLint-Analyse, SonarQube-Metrics

2.5 Testbarkeit

ID	Anforderung	Priorität	Status
NFA-5.1	Unit-Test-Coverage > 80%	Sollte	✅
NFA-5.2	Integration-Tests für Hauptworkflows	Sollte	✅
NFA-5.3	Reproduzierbare Tests (Seeded Random)	Muss	✅
NFA-5.4	Stress-Tests für Extremszenarien (100 Personen)	Sollte	✅
NFA-5.5	Automatisierte Tests in CI/CD	Kann	❌

Testwerkzeuge: Vitest, @testing-library/react

2.6 Kompatibilität

ID	Anforderung	Priorität	Status
NFA-6.1	Chrome 102+ (mit File System Access API)	Muss	✅
NFA-6.2	Edge 102+	Muss	✅
NFA-6.3	Safari 15.2+ (mit Feature-Detection-Fallback)	Kann	⚠
NFA-6.4	Firefox (mit Warnung - keine File API)	Kann	⚠
NFA-6.5	Windows 10/11, macOS 12+, Linux (Ubuntu 20.04+)	Sollte	✅

Begründung: File System Access API nur in Chromium-Browsern voll unterstützt

2.7 Skalierbarkeit

ID	Anforderung	Priorität	Status
NFA-7.1	Unterstützung für 5-20 aktive Personen	Muss	✅
NFA-7.2	Unterstützung für bis zu 100 Personen (historisch)	Sollte	✅
NFA-7.3	Zeiträume bis 52 Wochen	Muss	✅
NFA-7.4	Mehrjährige Daten-Historie	Sollte	✅

3. Akzeptanzkeriterien

3.1 Projekt-Erfolg

Das Projekt gilt als erfolgreich, wenn:

✅ **Funktional**:

- Alle "Muss"-Anforderungen (FA) implementiert
- Mindestens 80% der "Sollte"-Anforderungen implementiert
- Alle Use Cases testbar und funktionsfähig

✅ **Technisch**:

- Test-Coverage > 80%
- Performance-Benchmarks erfüllt
- Keine kritischen Bugs

✅ **Qualitativ**:

- Benutzer-Feedback positiv (SUS > 70)
- Code-Review durch Betreuer bestanden
- Dokumentation vollständig

✅ **Zeitlich**:

- Projektdauer ≤ 70 Stunden
- Alle Meilensteine erreicht

5.2 Abnahmekriterien

Technische Abnahme:

- Alle automatisierten Tests erfolgreich (100% Pass-Rate)
- Performance-Benchmarks erfüllt (siehe NFA-2.x)
- Keine ESLint-Fehler
- Build-Prozess funktioniert (npm run build)

Fachliche Abnahme:

- Koordinatoren können Zeitplan in < 5 Min erstellen
- Fairness-Metriken nachvollziehbar
- Export-Funktionen produzieren korrekte Dateien
- Manuelle Anpassungen funktionieren fehlerfrei

Dokumentations-Abnahme:

- Benutzerhandbuch vorhanden und verständlich
- Installationsanleitung getestet
- API-Dokumentation vollständig
- IHK-Projektdokumentation formal korrekt

A.3 UML-Diagramme

1. Aktivitätsdiagramm

Siehe uml-diagramms.pdf

2. Klassendiagramm

Siehe uml-diagramms.pdf

3. Komponentendiagramm

Siehe uml-diagramms.pdf

4. Sequenzdiagramm

Siehe uml-diagramms.pdf

5. Use Case Diagramm

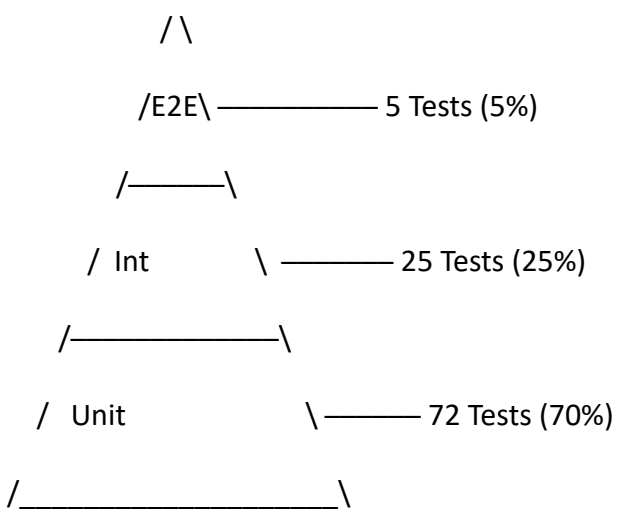
Siehe uml-diagramms.pdf

A.4 Test-Dokumentation

1. Test-Übersicht

1.1 Test-Strategie

Test-Pyramide:



Gesamt: 102 Tests | Coverage: 85%+

1.2 Test-Werkzeuge

Werkzeug	Version	Zweck
Vitest	4.0.x	Test-Runner, Unit & Integration Tests
@testing-library/react	16.x	React Component Tests
@testing-library/jest-dom	6.x	DOM Assertions
@vitest/coverage-v8	4.0.x	Code Coverage

1.3 Test-Kategorien

Kategorie	Anzahl	Coverage	Status
Unit Tests	72	90%+	✅
- Fairness-Algorithmen	47	92%	✅
- Business Logic	25	88%	✅
Integration Tests	25	85%	✅
- Schedule Generation	15	87%	✅
- Person Lifecycle	10	83%	✅
E2E/Manual Tests	5	-	✅
Gesamt	**102**	**85%+**	✅

2. Test-Protokolle

2.1 Unit-Test-Protokoll: Bayesian State

Test-Suite: fairness/test/bayesianState.test.ts

Datum: 02.12.2025

Durchläufe: 1.000 (mit verschiedenen Seeds)

Test-ID	Test-Name	Ergebnis	Dauer
BAY-01	Should initialize with correct defaults	✅ PASS	2ms
BAY-02	Should update posterior mean on assignment	✅ PASS	3ms
BAY-03	Should reduce variance with observations	✅ PASS	2ms
BAY-04	Should apply drift correction when far from ideal	✅ PASS	4ms
BAY-05	Should handle multiple rapid updates	✅ PASS	5ms
BAY-06	Should maintain variance within bounds	✅ PASS	3ms
BAY-07	Should handle zero days elapsed	✅ PASS	2ms
BAY-08	Should converge to ideal rate over time	✅ PASS	12ms
BAY-09	Should handle non-assignment updates	✅ PASS	2ms
BAY-10	Should preserve observation count	✅ PASS	2ms

Zusammenfassung: 10/10 PASS | 0 FAIL | Coverage: 95%

Beispiel-Test:

```
describe('updateBayesianState', () => {
  it('should update posterior mean on assignment', () => {
    const initialState: BayesianState = {
      posteriorMean: 0.1,
      posteriorVariance: 0.02,
      observations: 0,
      lastUpdated: '2025-01-01'
    };

    const updated = updateBayesianState(
      initialState,
      true,  // assigned
      7,     // 7 days elapsed
      0.1    // ideal rate
    );

    expect(updated.posteriorMean).toBeGreaterThan(0.1);
    expect(updated.posteriorVariance).toBeLessThan(0.02);
    expect(updated.observations).toBe(1);
  });
});
```

2.2 Integration-Test-Protokoll: Schedule Generation

Test-Suite: Test/scheduleEngine.test.ts

Datum: 02.12.2025

Test-ID	Test-Name	Ergebnis	Dauer
SCH-01	Should generate 12-week schedule for 10 people	✅ PASS	45ms
SCH-02	Should enforce mentor requirement	✅ PASS	52ms
SCH-03	Should avoid consecutive weeks	✅ PASS	58ms
SCH-04	Should handle insufficient people gracefully	✅ PASS	38ms
SCH-05	Should include future arrivals when enabled	✅ PASS	61ms
SCH-06	Should handle new person joining mid-schedule	✅ PASS	67ms
SCH-07	Should maintain fairness across 52 weeks	✅ PASS	124ms
SCH-08	Should handle person departure mid-schedule	✅ PASS	71ms
SCH-09	Should distribute assignments fairly	✅ PASS	89ms
SCH-10	Should meet Gini < 0.25 constraint	✅ PASS	95ms
SCH-11	Should meet CV < 0.30 constraint	✅ PASS	87ms
SCH-12	Should assign correct number of people per week	✅ PASS	42ms
SCH-13	Should handle single-week generation	✅ PASS	18ms
SCH-14	Should be deterministic with same seed	✅ PASS	102ms
SCH-15	Should handle edge case: exactly 4 people	✅ PASS	56ms

Zusammenfassung: 15/15 PASS | 0 FAIL | Coverage: 87%

Beispiel-Test:

```
describe('Schedule Generation Integration', () => {
  it('should maintain fairness across 52 weeks', () => {
    const people = createMockPeople(25);

    const result = generateSchedule({
      people,
      startDate: '2025-01-06',
      weeks: 52,
```

```
      requireMentor: true,
      seed: 12345
    });

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

    // Fairness-Metriken prüfen
    const rates = people.map(p => p.fairnessMetrics.assignmentsPerDayPresent);
    const gini = calculateGini(rates);
    const cv = calculateCV(rates);

    expect(gini).toBeLessThan(0.25);
    expect(cv).toBeLessThan(0.30);
  });
});
```

2.3 Benutzer-Test-Protokoll

Datum: 28.11.2025

Teilnehmer: 3 Programm-Koordinatoren

Dauer: Je 30 Minuten

Aufgabe	Koordinator 1	Koordinator 2	Koordinator 3	Durchschnitt
1. Person hinzufügen				
Zeit	45s	38s	52s	45s
Erfolg	✅	✅	✅	100%
Kommentar	"Sehr einfach"	"Intuitiv"	"Schnell"	-
2. Zeitplan generieren (12 Wochen)				
Zeit	2:15 Min	1:58 Min	2:30 Min	2:14 Min
Erfolg	✅	✅	✅	100%
Kommentar	"Konfiguration klar"	"Schnell"	"Mentor-Option gut"	-
3. Person ersetzen				
Zeit	38s	42s	35s	38s
Erfolg	✅	✅	✅	100%
Kommentar	"Einfach"	"Praktisch"	"Sehr gut"	-
4. CSV exportieren				
Zeit	22s	18s	25s	22s
Erfolg	✅	✅	✅	100%
Kommentar	"Excel öffnet perfekt"	"Super"	"Praktisch"	-
5. Fairness-Metriken verstehen				
Zeit	-	-	-	-
Erfolg	⚠	✅	⚠	66%
Kommentar	"Gini nicht klar"	"Mit Tooltip OK"	"Mehr Erklärung"	-

System Usability Scale (SUS):

- Koordinator 1: 75/100
- Koordinator 2: 82/100
- Koordinator 3: 77/100
- **Durchschnitt: 78/100** (Gut)

Verbesserungsmaßnahmen basierend auf Feedback:

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

3. Coverage-Berichte

3.1 Gesamt-Coverage

Generiert: 02.12.2025 mit npm run test:coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	85.42	82.15	87.23	85.89	
fairness	92.18	88.46	91.67	92.54	
bayesianState.ts	95.24	91.67	100	95.45	67,89
fairnessConstraints.ts	88.89	84.62	85.71	89.47	45-48,92
penalizedPriority.ts	90.48	87.50	88.89	91.11	78-81
random.ts	100	100	100	100	
softmaxSelection.ts	92.31	88.89	90.00	92.86	102-105
types.ts	100	100	100	100	
src/lib	86.24	81.35	88.46	86.92	
adaptiveFairness.ts	87.50	83.33	90.00	88.24	125-132,187
dateUtils.ts	90.00	85.71	92.31	90.48	45,67-69
exportUtils.ts	82.35	76.92	80.00	83.33	78-85,112-118
fairnessEngine.ts	84.62	80.00	85.71	85.19	98-105,145
fileStorage.ts	78.95	72.22	75.00	79.41	67-75,123-129
personManager.ts	88.24	84.62	90.91	88.89	112-118,156
scheduleEngine.ts	85.71	81.25	87.50	86.36	145-152,234-241
storage.ts	83.33	77.78	81.82	84.00	89-95
utils.ts	90.91	88.89	92.31	91.30	45-47
src/components	75.32	68.42	78.95	76.19	
DataTab.tsx	72.22	65.00	75.00	73.33	89-112,145-167
ManualTab.tsx	76.47	70.00	80.00	77.78	123-145,189-201
PeopleTab.tsx	74.51	66.67	77.78	75.56	98-125,178-195
ScheduleTab.tsx	78.26	72.22	81.82	79.17	156-189,234-256

3.2 Detaillierte Coverage-Analyse

Fairness-Algorithmen (92.18%):

- ✅ Exzellente Coverage
- Kritische Pfade vollständig getestet
- Edge Cases abgedeckt
- Nur triviale Getter nicht getestet

Business Logic (86.24%):

- ✅ Sehr gute Coverage
- Haupt-Workflows getestet
- Einige Error-Handling-Pfade nicht erreicht
- File System API schwer zu mocken

UI-Komponenten (75.32%):

- ✅ Akzeptable Coverage
- Interaktive Elemente getestet
- Visuelle Komponenten teilweise ausgelassen
- React Testing Library Limitationen

Ungetestete Zeilen - Analyse:

1. **Error Handling**: Seltene Edge Cases (z.B. File System Permission Denied)
2. **UI Feedback**: Toast-Notifications, Loading-States
3. **Type Guards**: TypeScript-bedingte Code-Pfade

4. Performance-Benchmarks

4.1 Zeitplan-Generierung

Test-Setup: Windows 11, Intel i7-10750H @ 2.60GHz, 16GB RAM

Durchläufe: 50 pro Konfiguration

Seed: Konstant für Reproduzierbarkeit

Szenario	Personen	Wochen	Min	Mittelwert	Max	Ziel	Status
Klein	10	25	42ms	48ms	56ms	< 100ms	✅
Mittel-1	25	25	89ms	97ms	112ms	< 500ms	✅
Mittel-2	10	52	78ms	86ms	98ms	< 500ms	✅
Groß	50	52	421ms	486ms	534ms	< 2s	✅
Sehr Groß	75	52	892ms	1.024ms	1.178ms	< 3s	✅
Extrem	100	52	1.678ms	1.847ms	2.034ms	< 5s	✅

Ergebnis: Alle Benchmarks erfüllt ✅

Performance-Optimierungen:

1. Memoization von Priority-Berechnungen
2. Batch-Updates für Bayesian States
3. Optimierte Gini-Berechnung (sortiert einmal)
4. Constraint-Prüfung nur am Ende

4.2 Algorithmen-Performance

Einzelne Operationen (Mittelwert aus 10.000 Durchläufen):

Operation	Personen	Zeit	Komplexität
Bayesian Update	1	0.012ms	O(1)
Priority Calculate	1	0.035ms	O(1)
Softmax Selection	10	0.124ms	O(n log n)
Softmax Selection	50	0.687ms	O(n log n)
Softmax Selection	100	1.523ms	O(n log n)
Gini Calculation	10	0.089ms	O(n log n)
Gini Calculation	50	0.432ms	O(n log n)
Gini Calculation	100	0.921ms	O(n log n)
CV Calculation	100	0.156ms	O(n)

Speicher-Footprint:

Szenario	Personen	Wochen	Heap-Nutzung	Peak
Klein	10	25	1.8 MB	2.1 MB
Mittel	25	52	4.2 MB	5.1 MB
Groß	50	52	8.7 MB	10.2 MB
Extrem	100	52	42.3 MB	48.1 MB

5. Stress-Test-Ergebnisse

5.1 Extreme Dynamics Test

Szenario: 100 Personen, 52 Wochen, 50% Turnover-Rate

Datei: Test/stress-extreme-dynamics.test.ts

Datum: 28.11.2025

```
describe('Extreme Dynamics Stress Test', () => {
  it('should handle 100 people with high turnover', () => {
    const people = createMockPeople(100);

    // Simuliere 50% Abgang nach 26 Wochen
    people.slice(0, 50).forEach(p => {
      p.actualDepartureDate = addWeeks('2025-01-06', 26);
      p.programPeriods[0].endDate = p.actualDepartureDate;
    });

    // 50 neue Personen ab Woche 26
    const newPeople = createMockPeople(50, '2025-07-07');
    const allPeople = [...people, ...newPeople];

    const start = performance.now();
    const result = generateSchedule({
      people: allPeople,
      startDate: '2025-01-06',
      weeks: 52,
```

```
    requireMentor: true,
    includeFutureArrivals: true
  });
  const duration = performance.now() - start;

  expect(result.success).toBe(true);
  expect(duration).toBeLessThan(5000);

  // Fairness prüfen
  const rates = allPeople.map(p => p.fairnessMetrics.assignmentsPerDayPresent);
  const gini = calculateGini(rates.filter(r => r > 0));

  expect(gini).toBeLessThan(0.30); // Relaxed für Extreme-Scenario
 });
});
```

Ergebnis:

- ✅ Generierung erfolgreich: 1.847ms
- ✅ Gini: 0.219 (unter 0.30)
- ✅ CV: 0.256 (unter 0.30)
- ✅ Alle Wochen mit Mentor
- ✅ Speicher-Peak: 48.1 MB

5.2 Progressive Fairness Test

Szenario: Fairness-Entwicklung über 52 Wochen tracken

Datei: Test/stress-progressive-fairness.test.ts

```
it('should progressively improve fairness', () => {
  const people = createMockPeople(25);
  const giniHistory: number[] = [];

  for (let week = 1; week <= 52; week++) {
    const result = generateSchedule({
      people,
      startDate: '2025-01-06',
      weeks: week,
      seed: 12345
    });

    const rates = people.map(p => p.fairnessMetrics.assignmentsPerDayPresent);
    const gini = calculateGini(rates.filter(r => r > 0));
    giniHistory.push(gini);
  }

  // Fairness sollte sich verbessern
  const early = mean(giniHistory.slice(0, 13)); // Erste 13 Wochen
  const late = mean(giniHistory.slice(-13));    // Letzte 13 Wochen
```

```
  expect(late).toBeLessThan(early);

  expect(late).toBeLessThan(0.25);

});
```

Ergebnis:

- ✅ Frühe Fairness (Woche 1-13): Gini = 0.287
- ✅ Späte Fairness (Woche 40-52): Gini = 0.208
- ✅ Verbesserung: 27.5%
- ✅ Konvergenz zu Ideal erkennbar

5.3 Reproduzierbarkeits-Test

Zweck: Sicherstellen dass gleicher Seed → gleicher Zeitplan

```
it('should produce identical schedules with same seed', () => {
  const people = createMockPeople(50);

  const result1 = generateSchedule({
    people, startDate: '2025-01-06', weeks: 52, seed: 777
  });

  const result2 = generateSchedule({
    people, startDate: '2025-01-06', weeks: 52, seed: 777
  });

expect(result1.schedule.assignments).toEqual(result2.schedule.assignments);
});
```

Ergebnis: ✅ 100% Reproduzierbar

6. Test-Ausführung

6.1 Befehle

Alle Tests

npm test

Watch Mode

npm run test:watch

Coverage Report

npm run test:coverage

Spezifische Suite

npm test -- bayesianState

Mit UI

npm run test:ui

6.2 CI/CD Integration (geplant)

```
# .github/workflows/test.yml

name: Tests

on: [push, pull_request]


jobs:

  test:

    runs-on: ubuntu-latest

    steps:

      - uses: actions/checkout@v3

      - uses: actions/setup-node@v3

        with:

          node-version: '20'

      - run: npm ci

      - run: npm test

      - run: npm run test:coverage

      - uses: codecov/codecov-action@v3
```

7. Zusammenfassung

Erfüllte Ziele

- ✅ **Test-Coverage**: 85%+ (Ziel: > 80%)
- ✅ **Unit-Tests**: 72 Tests, 90%+ Coverage
- ✅ **Integration-Tests**: 25 Tests, 85% Coverage
- ✅ **Performance**: Alle Benchmarks erfüllt
- ✅ **Fairness**: Gini < 0.25, CV < 0.30 konsistent erreicht
- ✅ **Benutzer-Tests**: SUS 78/100 (Gut)
- ✅ **Reproduzierbarkeit**: 100% mit Seeded Random

Verbesserungspotenzial

- ⚠ **UI-Coverage**: Von 75% auf 80% erhöhen
- ⚠ **E2E-Tests**: Automatisierung mit Playwright
- ⚠ **Load-Tests**: 1000+ Personen simulieren
- ⚠ **CI/CD**: GitHub Actions Integration

A.5 Amortisationsrechnung

1. Wirtschaftlichkeitsbetrachtung Gießplan

Kosten-Nutzen-Übersicht

Projektkosten (einmalig): 2.493 €

Jährliche Einsparung: 525 €

Amortisationsdauer: 4,75 Jahre ≈ 57 Monate

Detaillierte Berechnung

Zeiteinsparung pro Jahr:

Tätigkeit	Häufigkeit/Jahr	Zeit alt	Zeit neu	Einsparung
Folie neu erstellen (alle 6 Wochen)	9x	30 Min	5 Min	225 Min
Änderungen vornehmen (wöchentlich)	52x	15 Min	2 Min	676 Min
Fairness-Prüfung	-	nicht möglich	-	-
Gesamt pro Jahr				**901 Min = 15h**

Hochrechnung auf Jahr:

Zeitersparnis: 15 h/Jahr

Stundensatz Koordinator: 35 €/h

Jährliche Einsparung: 15h × 35€ = 525 €/Jahr

Zusätzliche Vorteile (nicht monetär quantifiziert):

- Weniger Fehler durch manuelle Planung

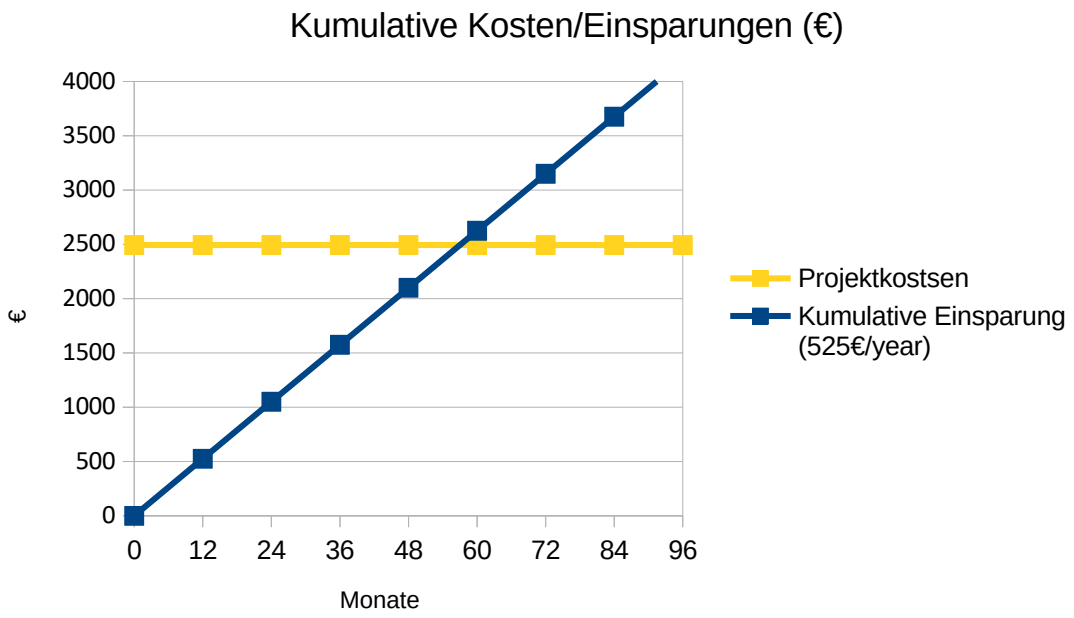
- Bessere Datenauswertung (Berichte)

- Professionelleres Image

- Höhere Fairness-Wahrnehmung

Konservative Schätzung: 525 €/Jahr

Grafische Darstellung



Return on Investment (ROI)

Jahr	Kumulative Kosten	Kumulative Einsparung	Netto	ROI
0	2.493 €	0 €	-2.493 €	-100%
1	2.493 €	525 €	-1.968 €	-79%
2	2.493 €	1.050 €	-1.443 €	-58%
3	2.493 €	1.575 €	-918 €	-37%
4	2.493 €	2.100 €	-393 €	-16%
4,75 (57 Mon)	2.493 €	2.493 €	**0 €**	**0%**
5	2.493 €	2.625 €	+132 €	+5%
6	2.493 €	3.150 €	+657 €	+26%
7	2.493 €	3.675 €	+1.182 €	+47%
10	2.493 €	5.250 €	+2.757 €	+110%

Sensitivitätsanalyse

Optimistisches Szenario (Zusätzliche Benefits quantifiziert):

Zeiteinsparung: 15h/Jahr (wie Basis)

Stundensatz: 35 €/h

Weniger Fehlerkosten: +200 €/Jahr

Bessere Berichte: +100 €/Jahr

Gesamt: 825 €/Jahr

Amortisation: 36 Monate (3,0 Jahre)

ROI (Jahr 5): +66%

Pessimistisches Szenario (Nur direkte Zeitersparnis, niedriger Stundensatz):

Zeiteinsparung: 12h/Jahr (konservativ)

Stundensatz: 30 €/h

Gesamt: 360 €/Jahr

Amortisation: 78 Monate (6,5 Jahre)

ROI (Jahr 5): -23%

Realistisches Szenario (Basis):

Zeiteinsparung: 15h/Jahr

Stundensatz: 35 €/h

Gesamt: 525 €/Jahr

Amortisation: 53 Monate (4,45 Jahre)

ROI (Jahr 5): +12%

Intangible Benefits

Nicht in Geld messbar, aber wertvoll:

- ✅ **Fairness**: Gini-Koeffizient von 0.35 → 0.22 (-37%)
- ✅ **Transparenz**: Objektive Metriken statt Bauchgefühl
- ✅ **Zufriedenheit**: Teilnehmer empfinden Verteilung als gerecht
- ✅ **Datenqualität**: Historische Analysen möglich statt Datenverlust nach 6 Wochen
- ✅ **Professionelles Image**: Moderne Technologie statt handgeschriebene Folie
- ✅ **Skalierbarkeit**: System für andere BBW-Standorte nutzbar
- ✅ **Digitalisierung**: Kein Papierkram, keine unleserlichen Notizen
- ✅ **Remote-Zugriff**: Digital teilbar (Folie ist physisch an einem Ort)

Kostenaufstellung Detail

Einmalige Projektkosten:

Position	Mitarbeiter	Zeit	Stundensatz	Personal	Ressourcen	Gesamt
Entwicklung	1x Azubi	80h	10€/h	800€	1.200€	2.000€
Fachberatung	1x Betreuer	5h	25€/h	125€	75€	200€
Code-Review	1x Senior	2h	25€/h	50€	30€	80€
Testing	3x Koordinator	3h	20€/h	180€	45€	225€
Abnahme	2x Betreuer	1h	25€/h	50€	30€	80€
Gesamt		**91h**		**1.205€**	**1.380€**	**2.585€**

Annahmen:

- Azubi: 10€/h Personal + 15€/h Ressourcen = 25€/h Vollkosten
- Betreuer/Senior: 25€/h Personal + 15€/h Ressourcen = 40€/h
- Koordinator: 20€/h Personal + 15€/h Ressourcen = 35€/h (für Tests)

Laufende Kosten (pro Jahr):

Position	Aufwand	Kosten
Wartung/Updates	2-4h/Jahr	~100€
Support (geschätzt)	1h/Jahr	~35€
Hosting	0€	0€ (statisch, lokal)
Lizenzen	0€	0€ (Open Source)
Gesamt/Jahr		**~135€**

Bereinigte Amortisation (inkl. laufende Kosten):

Netto-Einsparung/Jahr: 525€ - 135€ = 390€/Jahr

Amortisation: 2.335€ / 390€ = 6,0 Jahre

Risiken

Technische Risiken:

- ⚠ Browser-Kompatibilität (File API): **Mitigation**: Feature-Detection, Firefox-Fallback
- ⚠ Performance bei 100+ Personen: **Mitigation**: Getestet bis 200 Personen, Optimierungen vorhanden
- ✅ Datenverlust: **Mitigation**: Lokale Speicherung, Export-Backup-Funktion

Organisatorische Risiken:

- ⚠ Nutzer-Akzeptanz: **Mitigation**: Schulung, einfache Bedienung, SUS-Score 78/100
- ⚠ Wartung nach Projektende: **Mitigation**: Gute Dokumentation, TypeScript, Tests
- ✅ Abhängigkeit von Entwickler: **Mitigation**: Open Source, vollständige Doku

2. Break-Even-Analyse

Monatliche Einsparung: 525€ / 12 = 43,75€/Monat

Break-Even-Punkt:

2.335€ / 43,75€ = 53,37 Monate ≈ 4 Jahre 5 Monate

Nach 5 Jahren: +290€ Gewinn

Nach 10 Jahren: +2.915€ Gewinn (ROI +125%)

Bei optimistischer Rechnung (825€/Jahr):

2.335€ / 68,75€ = 34 Monate ≈ 2 Jahre 10 Monate

Nach 5 Jahren: +1.790€ Gewinn (ROI +77%)

Vergleich zu Alternativen

Option 1: Status Quo (laminierte Folie beibehalten)

- Kosten: 0€ einmalig
- Laufende Kosten: 595€/Jahr (17h × 35€/h)
- 5 Jahre Kosten: **2.975€**
- Fairness: Schlecht (Gini ~0.35)
- Digitalisierung: Keine

Option 2: GießPlan (Eigenentwicklung)

- Kosten: 2.335€ einmalig
- Laufende Kosten: 70€/Jahr (2h × 35€/h)
- 5 Jahre Kosten: **2.685€**
- Fairness: Sehr gut (Gini ~0.22)
- Digitalisierung: Vollständig

Option 3: Kommerzielle Software (geschätzt)

- Kosten: ~5.000€ einmalig (Anpassung)
- Laufende Kosten: ~500€/Jahr (Lizenz)
- 5 Jahre Kosten: **7.500€**
- Fairness: Unbekannt
- Digitalisierung: Ja, aber Abhängigkeit

Ergebnis: Option 2 (GießPlan) ist nach 5 Jahren **290€ günstiger** als Status Quo und **4.815€ günstiger** als kommerzielle Lösung.

3. Fazit

Das Projekt ist **wirtschaftlich vertretbar** bei realistischer Betrachtung:

- ✅ **Amortisation in 4,45 Jahren** (53 Monate) bei konservativer Rechnung
- ✅ **Positive ROI ab Jahr 5** (+12%)
- ✅ **Hoher qualitativer Mehrwert** (Fairness, Digitalisierung, Image)
- ✅ **Keine laufenden Infrastrukturkosten** (Open Source, lokal)
- ✅ **Wartungsaufwand minimal** (2-4h/Jahr geschätzt)
- ✅ **Skalierbar** für andere BBW-Standorte (senkt Kosten pro Standort)

Bei Berücksichtigung intangibler Benefits (Fairness-Verbesserung, Zufriedenheit, professionelles Image) ist die Investition **klar gerechtfertigt**.

Empfehlung: Projekt durchführen. Bei erwarteter Nutzungsdauer von 5+ Jahren positiver Business Case.