# Designbeschreibung

# Inhaltsverzeichnis

# 1. Allgemeines

Die vorliegende Designbeschreibung dient als zentrales Dokument zur Erklärung der Grundlagen des Online-Aufgaben-Planungssystems. Ziel ist es, Softwareentwicklern, die bisher nicht mit dem Projekt gearbeitet haben, einen verständlichen Einstieg in die wesentlichen Entwurfsprinzipien zu ermöglichen, bevor sie sich mit der quelltextnahen Javadoc-Dokumentation befassen. Dabei liegt der Fokus nicht nur auf der Darstellung, sondern insbesondere auf der nachvollziehbaren Begründung zentraler Designentscheidungen.

Im Rahmen dieser Arbeit handelt es sich um eine softwaretechnische Studie in Form eines Prototyps für das spätere Gesamtsystem. Ziel dieser Studie ist es, erste Ideen zu entwickeln sowie praktische Erfahrungen mit Technologien und deren Einsatz zu sammeln. Der Prototyp dient vor allem dazu, Konzepte zu testen und ist weder vollständig noch für den produktiven Einsatz gedacht. Eine umfassende Testautomation ist daher zu diesem Zeitraum nicht vorgesehen.

Die Umsetzung der Designstudie erfolgt unter Berücksichtigung grundlegender Qualitätsstandards, um insbesondere die Wartbarkeit und Erweiterbarkeit des Systems zu gewährleisten. Dazu gehören die Einhaltung von Coding Conventions sowie ein konsistentes Dokumentationskonzept. Als methodische Grundlage für den Entwurf werden Architekturkonzepte und Design Patterns genutzt, die sowohl eine strukturierte Entwicklung unterstützen als auch eine gute Nachvollziehbarkeit für andere Entwicklern ermöglichen.

# 2. Produktübersicht

Der entwickelte Prototyp stellt eine webbasierte Anwendung zur Aufgabenverwaltung und automatisierten Zeitplanung dar. Ziel des Systems ist es, Nutzer*innen bei der strukturierten Organisation ihrer Aufgaben sowie bei der effizienten Erstellung eines Wochen- oder Tagesplans zu unterstützen. Im Folgenden werden die äußerlich sichtbaren Funktionsmerkmale, der typische Nutzungsablauf sowie bereits berücksichtigte optionale Features beschrieben.

## 2.1. Kernfunktionen

**Aufgabenverwaltung**

- Nutzer*innen können neue Aufgaben über ein Formular erstellen.
- Zu jeder Aufgabe können folgende Attribute definiert werden:
    - Name der Aufgabe
    - Status (z. B. offen, in Bearbeitung)
    - Deadline
    - Geschätzte Dauer  und bereits erfasste Zeit  (in Stunden)
    - Beschreibung
- Aufgaben können gespeichert und später in die Planung integriert werden.

**Kalenderbasierte Planungsansicht**

- Die Anwendung bietet eine Wochen- und Tagesansicht (umschaltbar).
- Aufgaben bzw. Termine werden als Blöcke im Kalender visualisiert.
- Bereits geplante Termine (z. B. Meetings) werden angezeigt.

**Navigation und Benutzeroberfläche**

- Zentrale Navigation über Sidebar (Workspace, Planungsansicht, Einstellungen).
- Übersichtliche UI mit klarer Trennung zwischen Planung und Verwaltung.

**Benutzer- und Rollenverwaltung (Super-Admin)**

- Es existiert eine Super-Admin-Ansicht zur Verwaltung von:
  - Organisationen
  - Super-Admin-Benutzern
- Anzeige von Benutzerinformationen (Name, E-Mail, Erstellungsdatum).

**Accountverwaltung**

- Nutzer*innen können ihre Accountdaten einsehen.
- Passwortänderung ist möglich.
- Abmeldung (Logout) ist implementiert.

**Planerstellung**

- Über die Funktion „Planung starten“ wird eine automatische Planung ausgelöst.

Ablauf:

- Prüfung der hinterlegten Arbeitszeiten
- Berücksichtigung von Blockern (z.B. bestehende Termine, Pausen)
- Laden der offenen Aufgaben in einen Aufgabenpool

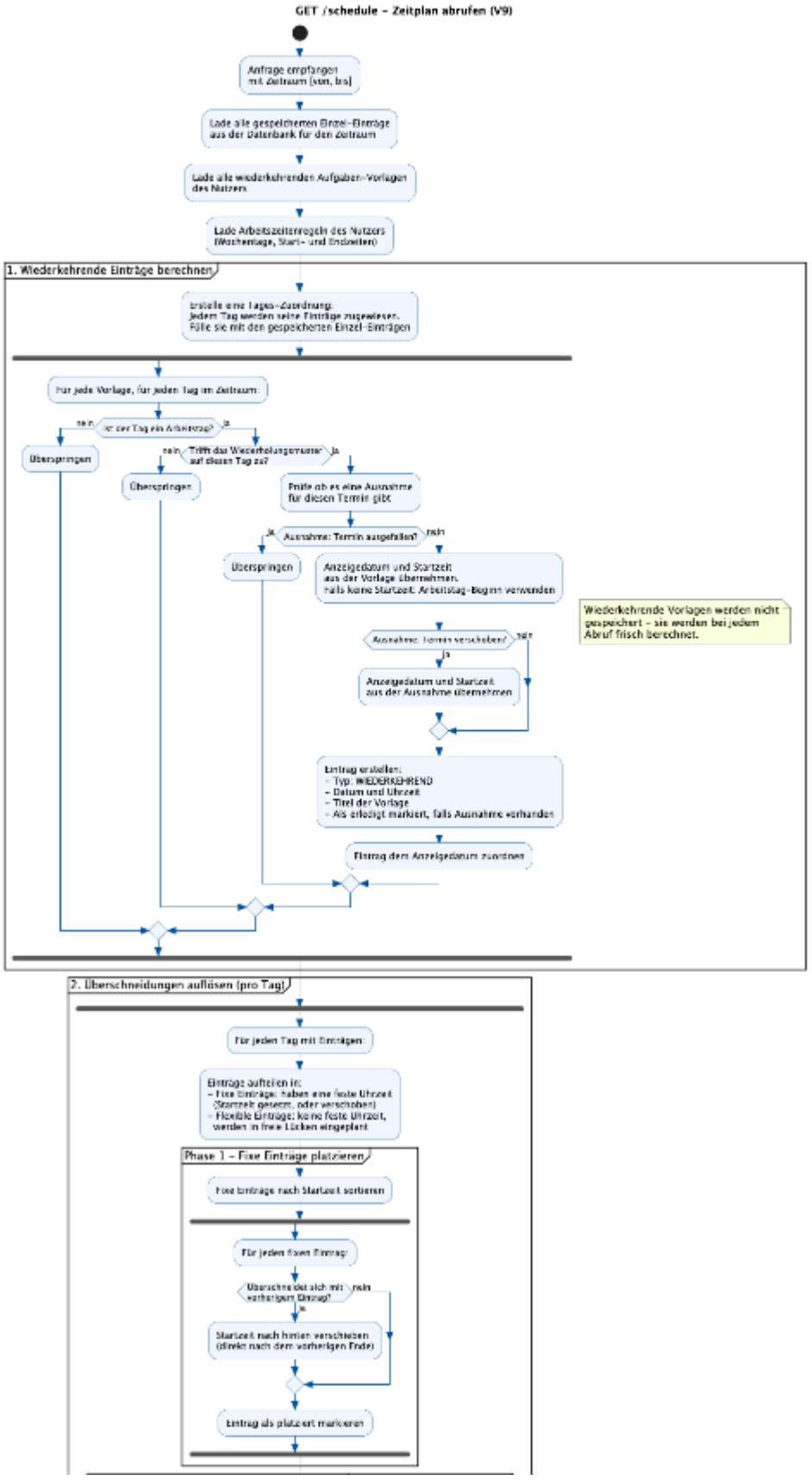
Planung:

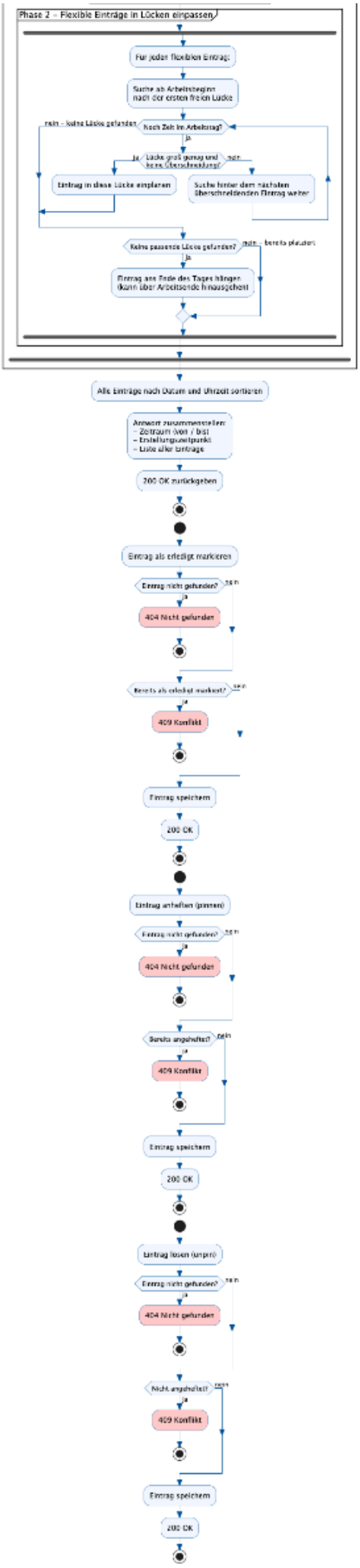
- Tagesweise Einplanung der Aufgaben
- Berechnung des verfügbaren Zeitbudgets pro Tag
- Sortierung nach Dringlichkeit und Anforderungsgrad
- Einplanung in freie Zeitfenster

Besonderheiten:

- Aufgaben können bei Bedarf aufgeteilt werden
- Überfällige Aufgaben werden erkannt und gekennzeichnet

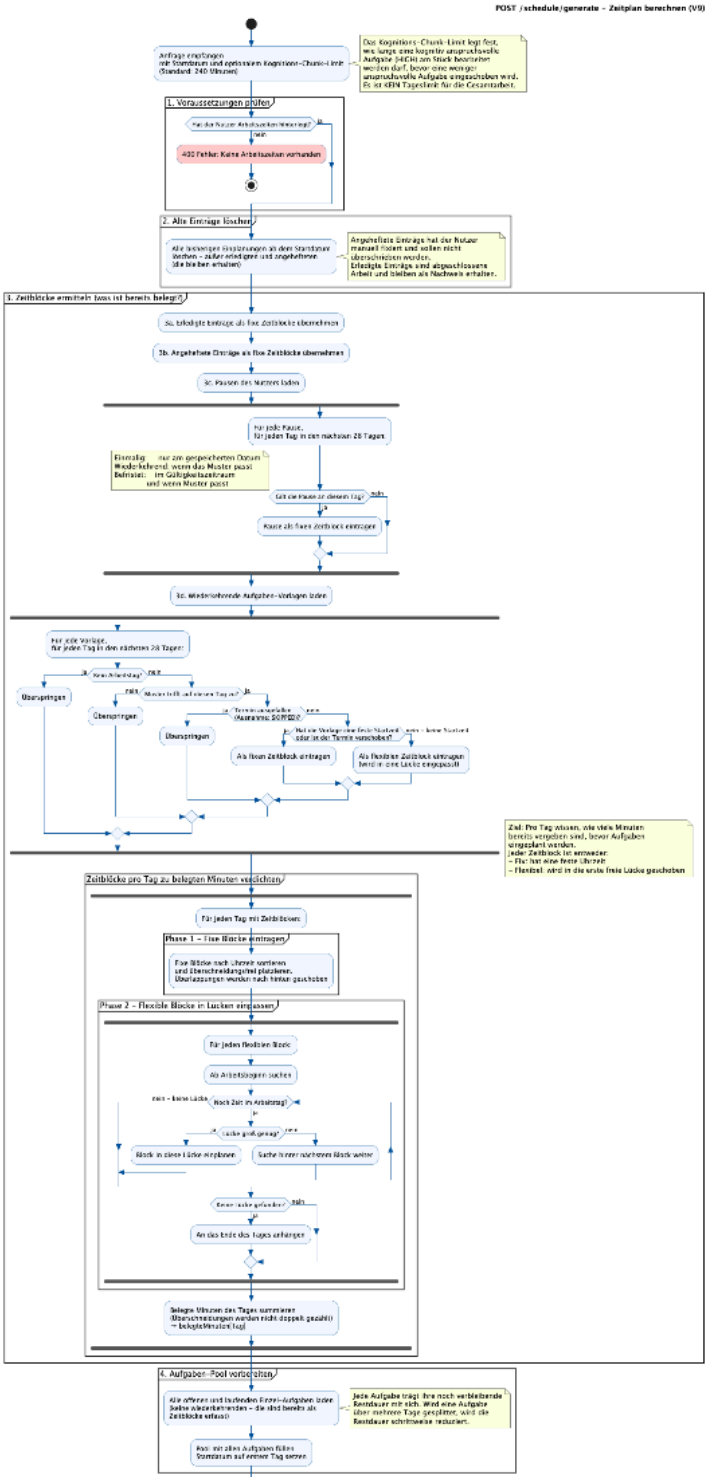
Der genaue Ablauf der Planerstellung ist im folgenden Aktivitätsdiagramm dargestellt und zusätzlich im Abgabeordner in besserer Auflösung enthalten.

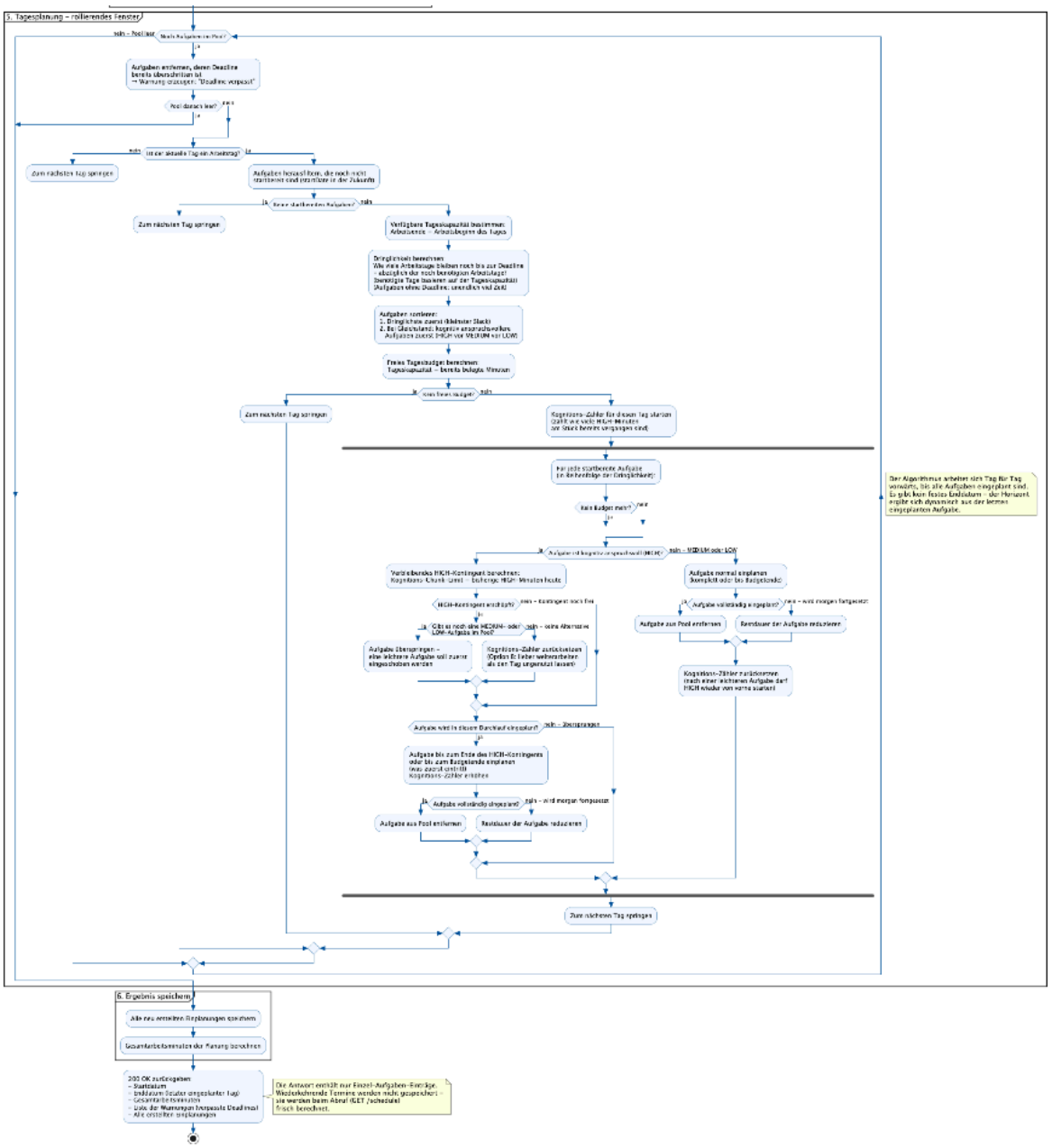

GET /schedule – Zeitplan abrufen (V9)
Anfrage empfangen
mit Zeitraum [von, bis]
Lade alle gespeicherten Einzel-Einträge
aus der Datenbank für den Zeitraum
Lade alle wiederkehrenden Aufgaben-Vorlagen
des Nutzers
Lade Arbeitszeitenregeln des Nutzers
(Wochentage, Start- und Endzeiten)
1. Wiederkehrende Einträge berechnen
Erstelle eine Tages-Zuordnung:
Jedem Tag werden seine Einträge zugewiesen.
Fülle sie mit den gespeicherten Einzel-Einträgen
Für jede Vorlage, für jeden Tag im Zeitraum:
nein
ist der Tag ein Arbeitstag?
ja
Überspringen
nein
Trifft das Wiederholungsmuster
auf diesen Tag zu?
ja
Überspringen
Prüfe ob es eine Ausnahme
für diesen Termin gibt
ja
Ausnahme: Termin ausgefallen?
nein
Überspringen
Anzeigedatum und Startzeit
aus der Vorlage übernehmen.
falls keine Startzeit: Arbeitstag-Beginn verwenden
Wiederkehrende Vorlagen werden nicht
gespeichert – sie werden bei jedem
Abruf frisch berechnet.
Ausnahme: Termin verschoben?
nein
ja
Anzeigedatum und Startzeit
aus der Ausnahme übernehmen
Eintrag erstellen:
- Typ: WIEDERKEHREND
- Datum und Uhrzeit
- Titel der Vorlage
- Als erledigt markiert, falls Ausnahme vorhanden
Eintrag dem Anzeigedatum zuordnen
2. Überschneidungen auflösen (pro Tag)
Für jeden Tag mit Einträgen:
Einträge aufteilen in:
- Fixe Einträge: haben eine feste Uhrzeit
(Startzeit gesetzt, oder verschoben)
- Flexible Einträge: keine feste Uhrzeit,
werden in freie Lücken eingeplant
Phase 1 – Fixe Einträge platzieren
Fixe Einträge nach Startzeit sortieren
Für jeden fixen Eintrag:
Überschneidet sich mit
vorherigem Eintrag?
nein
ja
Startzeit nach hinten verschieben
(direkt nach dem vorherigen Ende)
Eintrag als platziert markieren




team@goaldone.de

16. März 2026


POST /schedule/generate – Zeitplan berechnen (V3)
1. Voraussetzungen prüfen
400 Fehler: Keine Arbeitszeiten vorhanden
2. Alte Einträge löschen
3. Zeitblöcke ermitteln (was ist bereits belegt?)
Ziel: Pro Tag wissen, wie viele Minuten bereits vergeben sind, bevor Aufgaben eingeplant werden.
Zeitblöcke pro Tag zu belegten Minuten verdichten
Phase 1 – Fixe Blöcke eintragen
Phase 2 – Flexible Blöcke in Lücken einpassen
4. Aufgaben-Pool vorbereiten


5. Tagesplanung – rollierendes Fenster
Aufgaben entfernen, deren Deadline bereits überschritten ist
→ Warnung ausgeben: "Deadline verpasst"
Zum nächsten Tag springen
Aufgaben herausfiltern, die noch nicht startbereit sind (erst in der Zukunft)
Zum nächsten Tag springen
Verfügbare Tageskapazität bestimmen: Arbeitsende − Arbeitsbeginn des Tages
Dringlichkeit berechnen
Wie viele Arbeitstage bleiben noch bis zur Deadline – abzüglich der noch benötigten Arbeitstage?
Überzählige Tage basieren auf der Tageskapazität
Aufgaben ohne Deadline: unendlich viel Zeit
Aufgaben sortieren:
1. Dringlichste zuerst (kleinster Slack)
2. Bei Gleichstand: kognitiv anspruchsvollere
Aufgaben zuerst: HIGH vor MEDIUM vor LOW
Freies Tagesbudget berechnen
Tageskapazität − bereits belegte Minuten
Zum nächsten Tag springen
Kognitions-Zähler für diesen Tag starten
(Zählt wie viele HIGH-Minuten
am Stück bereits vorgegeben sind)
Für jede startbereite Aufgabe
(in der Reihenfolge der Dringlichkeit):
Kein Budget mehr?
Verbleibendes HIGH-Kontingent berechnen:
Kognitions-Limit − bisherige HIGH-Minuten heute
HIGH-Kontingent erschöpft?
Gibt es noch eine MEDIUM- oder LOW-Aufgabe im Pool?
Aufgabe überspringen – eine leichtere Aufgabe soll zuerst eingeschoben werden
Kognitions-Zähler zurücksetzen
(Gedanken können weiterarbeiten
als ein Tag ungenutzt lassen)
Aufgabe wird in diesem Durchlauf eingeplant?
Aufgabe bis zum Ende des HIGH-Kontingents oder bis zum Budgetende einplanen
Neue zuletzt eingeplante Kognitions-Zähler erhöhen
Aufgabe vollständig eingeplant?
Aufgabe aus Pool entfernen
Restdauer der Aufgabe reduzieren
Aufgabe normal einplanen
(komplett oder bis Budgetende)
Aufgabe vollständig eingeplant?
Aufgabe zum Pool entfernen
Restdauer der Aufgabe reduzieren
Kognitions-Zähler zurücksetzen
(nach einer leichteren Aufgabe darf HIGH wieder von vorne starten)
Zum nächsten Tag springen
Der Algorithmus arbeitet sich Tag für Tag vorwärts, bis alle Aufgaben eingeplant sind. Es gibt kein festes Endedatum – der Horizont ergibt sich dynamisch aus der letzten eingeplanten Aufgabe.
6. Ergebnis speichern
Alle neu erstellten Planungen speichern
Gesamtarbeitsminuten der Planung berechnen
200 OK zurückgeben:
- Startdatum
- Enddatum (letzter eingeplanter Tag)
- Gesamtarbeitsminuten
- Liste der Warnungen (verpasste Deadlines)
- Alle erstellten Einplanungen
Die Antwort enthält nur Einzel-Aufgaben-Einträge. Wiederkehrende Termine werden nicht gespeichert – sie werden beim Abruf (GET /schedule) frisch berechnet.

## 2.2. Typischer Ablauf des Nutzers

Ein typischer Ablauf für die Nutzung des Prototyps gestaltet sich wie folgt:

**Login / Zugriff auf das System**
Der/die Nutzer*in meldet sich im System an
(implizit vorhanden durch Accountansicht).

**Navigation zur Planungsansicht**
Über die Sidebar wird die Planungsansicht geöffnet, die den Kalender darstellt.

**Anlegen von Aufgaben**

- Über „Aufgabe hinzufügen“ wird das Aufgabenformular geöffnet.
- Der/die Nutzer*in trägt alle relevanten Informationen ein und speichert die Aufgabe.

**Start der automatischen Planung**

- Durch Klick auf „Planung starten“ wird der Planungsalgorithmus ausgelöst.
- Aufgaben werden automatisch auf freie Zeitslots verteilt.

**Überprüfung und Anpassung**

- Der generierte Plan wird im Kalender angezeigt.
- Nutzer*innen können die Planung visuell prüfen (und perspektivisch anpassen).

**Wechsel zwischen Ansichten**

- Nutzer*innen können zwischen Tages- und Wochenansicht wechseln, um Details oder Gesamtübersichten zu sehen.

**Verwaltung**

- Super-Admins können Organisationen und Benutzer verwalten.

# 3. Grundsätzliche Struktur- und Entwurfsentscheidungen

Der Prototyp bietet eine klare Trennung zwischen Benutzeroberfläche und Geschäftslogik. Das Frontend wird als Single-Page-Application (SPA) mit Angular 21 und TypeScript umgesetzt und kommuniziert über definierte REST-Endpunkte mit dem Backend. Sicherheitsmechanismen wie JWT-basierte Authentifizierung und rollenbasiertes Zugriffskontrollmodell sorgen für den Schutz der Anwendung.

## 3.1. Architektur & Architekturmuster

- **Client-Server-Architektur**:
  Das Frontend läuft im Browser, das Backend auf dem Server. Die Kommunikation erfolgt ausschließlich über REST-Endpunkte.
- Layered Architecture (Schichtenarchitektur) im Backend:
  - **Controller**: Entgegennahme von HTTP-Requests.
  - **Service**: Kapselt die Geschäftslogik (z. B. Algorithmen zur Aufgabenplanung).
  - **Repository**: Verantwortlich für den Datenzugriff auf die Datenbank.
  - **Entity**: Repräsentiert die Datenmodelle.
- **Frontend**: Single-Page-Application (SPA), bei der alle Benutzerinteraktionen über die REST-API verarbeitet werden. Die Entscheidung für eine SPA wurde getroffen, da sie im Vergleich zu klassischen Multi-Page-Anwendungen eine bessere Benutzererfahrung ermöglicht. Insbesondere erlaubt sie schnelle Interaktionen ohne vollständiges Neuladen der Seite, was für eine interaktive Planungsoberfläche entscheidend ist. Alternativ wäre eine serverseitig gerenderte Anwendung (z. B. klassische Webanwendung) möglich gewesen. Diese wurde jedoch verworfen, da sie bei häufigen UI-Interaktionen zu höheren Ladezeiten und einer weniger flüssigen Nutzung führen würde.
- Zusätzlich werden etablierte Design Patterns eingesetzt, um die Struktur und Wartbarkeit des Systems zu verbessern. Dazu zählen insbesondere:
  - Repository Pattern: Abstraktion des Datenzugriffs über JPA-Repositories.
  - Dependency Injection: Verwaltung von Abhängigkeiten durch das Spring-Framework.

  Diese Muster unterstützen eine klare Strukturierung und erleichtern die Erweiterbarkeit sowie die Testbarkeit der Anwendung.

## 3.2. Eingesetzte Technologien und deren Zweck

**Backend**

- **Java 21 & Spring Boot 3.4**: Robuste, wartbare Enterprise-Architektur.
- **Spring Security**: Authentifizierung und Autorisierung basierend auf RBAC (Role-Based Access Control).
- **JPA (Java Persistence API)** + **Repositories**: Effizienter und typisierter Datenbankzugriff.
- **Liquibase**: Versionierte und sichere Datenbankmigrationen.
- **JWT (JSON Web Token)**: Zustandslose Authentifizierung, hohe Skalierbarkeit.
- **Lombok**: Reduziert Boilerplate-Code für Getter, Setter, Konstruktoren etc.

**Frontend**

- **Angular 21 & TypeScript**: Typisierte, wartbare SPA.
- **PrimeNG**: Umfangreiche UI-Komponentenbibliothek für konsistente Benutzeroberflächen.
- **OpenAPI/Swagger**: Contract-First-Entwicklung für konsistente Schnittstellen zwischen Frontend und Backend.

## 3.3. Trennung von Benutzeroberfläche und Logik

- Das Frontend kennt nur die REST-Endpunkte und hat keinen direkten Zugriff auf Datenbank oder Backend-Logik.
- Geschäftslogiken (z. B. der Planungsalgorithmus) sind im Backend gekapselt.
- Kommunikation erfolgt ausschließlich über JSON via REST.
- Zugriffsrechte werden über das Rollenmodell gesteuert, sodass Nutzer nur auf Endpunkte zugreifen können, die für ihre Rolle vorgesehen sind.

## 3.4. Vorteile der Architektur

- Unabhängige Entwicklung und Wartung von Frontend und Backend, solange die REST-API stabil bleibt.
- Typsicherheit durch TypeScript im Frontend und generierte Java-API-Modelle minimiert Fehler.
- Gezielte Erweiterbarkeit: Neue Funktionalitäten können in der Service-Schicht implementiert werden, ohne bestehende APIs zu verändern.

- Testbarkeit: Jede Schicht kann einzeln per Unit-Test geprüft werden.
- Sichere Datenbankmigrationen dank Liquibase.
- Skalierbarkeit & Sicherheit durch stateless JWT-Authentifizierung und rollenbasiertes Zugriffskontrollmodell.

.

# 4. Struktur- und Entwurfsentscheidungen der einzelnen Pakete/Komponenten der Anwendung

In diesem Abschnitt werden die wichtigsten Struktur- und Entwurfsentscheidungen auf Komponentenebene beschrieben. Ziel ist es, einer neuen Entwicklerperson einen schnellen Überblick über die zentralen Bausteine des Systems zu geben, ohne auf Detailimplementierungen einzugehen.

## 4.1. Backend-Struktur (Spring Boot)

**Controller-Schicht (controller)**

- Verantwortlich für die Bereitstellung der REST-API.
- Nimmt HTTP-Anfragen entgegen und gibt entsprechende Antworten zurück.
- Beispiele:
    - TasksController ⟶ Verwaltung von Aufgaben
    - ScheduleController ⟶ Planung und Zeitverteilung
    - AuthController ⟶ Authentifizierung
    - AdminController, OrganizationsController ⟶ Admin-Funktionalitäten

Entwurfsentscheidung:
Klare Trennung zwischen API-Schicht und Geschäftslogik zur besseren Wartbarkeit und Testbarkeit.

**Service-/Logik-Schicht (implizit bzw. verteilt)**

- Enthält die Geschäftslogik, z. B. Planungslogik oder Validierungen.
- Wird von den Controllern genutzt.
- Teilweise direkt in Controllern oder separaten Klassen implementiert.

Entwurfsentscheidung:
Die Service-Schicht enthält die zentrale Geschäftslogik der Anwendung, wie beispielsweise die Planungslogik und fachliche Regeln. Im Rahmen des Prototyps wurde die Logik teilweise vereinfacht umgesetzt  und ist noch nicht vollständig optimiert. Die vorgesehene Aufteilung der Geschäftslogik wird in der weiteren Implementierung konsequent umgesetzt.

### Datenmodell (entity)

- Enthält die zentralen Domänenobjekte:
  - Task, User, Organization, Break, etc.
- Zusätzlich werden Enums genutzt, z. B.:
  - Role, RecurrenceType, CognitiveLoad

Entwurfsentscheidung:
Verwendung eines klaren Domänenmodells zur Abbildung der Geschäftslogik und Vorbereitung für persistente Speicherung.

### Konfiguration (config)

- Verwaltung von Systemeinstellungen (z. B. CORS, JSON-Konfiguration).
- Initialisierung von Systemzuständen (z. B. Super-Admin).
- Validierung beim Start (StartupValidator).

Entwurfsentscheidung:
Zentrale Bündelung aller Konfigurationen, um Systemverhalten transparent und anpassbar zu machen.

### Fehlerbehandlung

- Zentrale Fehlerbehandlung über GlobalExceptionHandler.

Entwurfsentscheidung:
Einheitliche API-Fehlermeldungen zur Verbesserung der Robustheit und Benutzerfreundlichkeit.

## 4.2. Frontend-Struktur (Angular)

Das Frontend basiert auf einer komponentenbasierten Architektur mit Angular.

**Komponentenstruktur**

- UI ist in wiederverwendbare Komponenten unterteilt, z. B.:
  - Kalenderansicht (Woche/Tag)
  - Aufgabenformular (Modal)
  - Sidebar / Navigation
  - Admin- und Einstellungsseiten

Entwurfsentscheidung:
Modularisierung der UI für bessere Wiederverwendbarkeit und klare Trennung von Verantwortlichkeiten.

**Services (Datenzugriff)**

- Kommunikation mit dem Backend erfolgt über Angular Services (HTTP).
- Kapselung von API-Aufrufen (z. B. Tasks, Schedule, Users).

Entwurfsentscheidung:
Trennung von UI und Datenlogik, um Komponenten schlank zu halten.

**Routing**

- Navigation zwischen Seiten (Workspace, Planung, Einstellungen, Admin) erfolgt über Angular Routing.

Entwurfsentscheidung:
Single-Page-Application (SPA) für schnelle Navigation ohne vollständiges Neuladen.

**State-Handling (einfach gehalten)**

- Zustand wird hauptsächlich lokal in Komponenten oder Services gehalten.
- Keine komplexe State-Management-Library (z. B. NgRx) im Prototyp.

Entwurfsentscheidung:
Reduzierung der Komplexität für den Prototyp, mit Option zur späteren Erweiterung.

## 4.3 Zusammenspiel von Frontend und Backend

- Kommunikation erfolgt über eine REST-Schnittstelle (HTTP + JSON).
- Frontend sendet Anfragen (z. B. Aufgaben anlegen, Planung starten).
- Backend verarbeitet diese und liefert strukturierte Daten zurück.

Entwurfsentscheidung:
Lose Kopplung zwischen Frontend und Backend, wodurch beide unabhängig weiterentwickelt werden können.