Designbeschreibung

Inhaltsverzeichnis

1. Allgemeines

Die vorliegende Designbeschreibung bildet das fundamentale Dokument zur technischen und konzeptionellen Ausgestaltung des Online-Aufgaben-Planungssystems. Sie dient als Brücke zwischen den im Pflichtenheft definierten Anforderungen und der technischen Implementierung. Das Ziel ist es, beteiligten Softwareentwicklern eine klare Orientierung über die Architektur, die Systemgrenzen und die zentralen Entwurfsentscheidungen zu geben, bevor eine Auseinandersetzung mit der detaillierten Quelltext-Dokumentation erfolgt.

Im Gegensatz zur vorangegangenen Prototyp-Studie handelt es sich hierbei um das Gesamtsystem, welches für den produktiven Einsatz in Unternehmen konzipiert ist. Der Fokus liegt auf einer robusten, skalierbaren und sicheren Architektur. Das System wird als moderne Web-Applikation in einer Frontend-Backend-Trennung (Angular und Spring Boot) realisiert.

Besonders Wert wird auf die Wartbarkeit und Erweiterbarkeit gelegt. Durch den Einsatz von Design Patterns und einer strikten Einhaltung von Coding Conventions (Angular Style Guide & Oracle Java Conventions) wird sichergestellt, dass das System auch bei künftigen Funktionserweiterungen (wie z. B. Aufgabenhierarchien) stabil bleibt. Ein umfassendes Sicherheitskonzept basiert auf dem JWKS Workflow von Zitadel zur Validierung von JWTs, wobei die JWTs primär für die Autorisierung der Nutzer innerhalb eines strikten rollenbasierten Zugriffskonzepts (RBAC) verwendet werden, welches ebenfalls integraler Bestandteil des Entwurfs ist.

2. Produktübersicht

Das System ist eine hochspezialisierte Plattform zur automatisierten Zeit- und Aufgabenplanung. Es ermöglicht Nutzern, komplexe Aufgaben unter Berücksichtigung von realen Rahmenbedingungen wie Deadlines, kognitiver Belastung und bestehenden Terminen effizient zu verwalten. Die Anwendung ist für den Einsatz in professionellen Organisationsstrukturen optimiert und unterstützt die Mehrmandantenfähigkeit durch die Trennung verschiedener Unternehmen.

2.1. Kernfunktionen

Erweitertes Rollen- und Benutzermanagement (RBAC):

- Differenzierung zwischen Super-Admin (globales Unternehmensmanagement), Unternehmens-Admin (Nutzerverwaltung innerhalb einer Firma) und Benutzer (operatives Aufgabenmanagement).
- Sicheres Einladungsverfahren für neue Nutzer zur Gewährleistung kontrollierter Registrierungsprozesse.

Benutzer- und Organisationsverwaltung

- Super-Admin:
  - Unternehmen anlegen/löschen
  - Super-Admins verwalten
- Unternehmens-Admin:
  - Nutzer einladen
  - Rollen zuweisen
  - Nutzer verwalten

Aufgabenmanagement:

- Erstellung von Aufgaben mit folgenden Attributen:
  - Titel, Dauer, Unternehmen
  - optional: Deadline, Beschreibung, Abhängigkeiten, „Nicht planen vor“, Anforderungsgrad
- Pre-Flight-Validierung zur Prüfung der Planbarkeit
- Bearbeiten, Löschen und Aktualisieren von Aufgaben
- Filter- und Suchfunktionen (Deadline, Status, Dauer, Anforderungsgrad)

Arbeitszeiten und Pausenverwaltung

- Definition individueller Arbeitszeiten pro Nutzer
- Verhinderung von Überschneidungen bei mehreren Organisationen
- Erstellung von einmaligen oder wiederkehrenden Pausen
- Berücksichtigung in der automatischen Planung

Automatische Aufgabenplanung

- Ziel & Grundidee
  - Automatische Aufgabenplanung
  - Nutzung eines Constraint-Satisfaction-Modells
  - Es werden mehrere Lösungen berechnet
  - Die Lösung mit der besten Bewertung wird gewählt
- Bewertung
  - Das Brechen bzw. Einhalten von Constraints addiert bzw. subtrahiert Punkte von einem Bewertungs-Score
- Constraints
  - Unterscheidung zwischen:
    - **Hard Constraints**: Deadlines, Arbeitszeiten, Termine, Abhängigkeiten
    - **Soft Constraints**: Pausen, kognitive Belastung
  - Slack-Zeiten berücksichtigen
    - Abhängigkeiten
    - Nicht planen vor
    - Deadlines

Eigenbau: Constraint-Übersicht (Was wird geprüft?)

Score-Hierarchie (lexikographisch):
HARD >> MEDIUM >> SOFT
Ein Plan mit Hard = -1 ist **immer** schlechter als einer mit Hard = 0, egal wie gut Medium/Soft sind.
Beispiel: [-1 / 0 / 0] < [0 / -999 / -999]

HARD Constraints — dürfen NIE verletzt werden

H1: Keine Zeitüberlappung
Zwei Chunks desselben Accounts dürfen nicht gleichzeitig liegen.

Prüfung: Für jedes Chunk-Paar (i, j) im selben Account: überlappen sich die Zeiträume?

Strafe: −(Überlappung in Minuten)

Beispiel:
Chunk A: Mo 10:00-12:00
Chunk B: Mo 11:00-13:00
→ Überlappung: 60 min
→ hardScore -= 60

H2: Slot-Kapazität
Ein Chunk darf nicht länger sein als der Slot, in dem er liegt.

Prüfung:
chunk.duration ≤ slot.duration?

Strafe: −(Überlauf in Minuten)

Beispiel:
Chunk: 180 min
Slot: 120 min (08:00-10:00)
→ Überlauf: 60 min
→ hardScore -= 60

H3: Abhängigkeiten einhalten
Wenn Task B von Task A abhängt: Letzter Chunk von A muss VOR erstem Chunk von B enden.

Prüfung:
endeAllerChunks(A) < startErsterChunk(B)?

Strafe: −1 pro Verletzung

Beispiel:
Task A letzter Chunk: Di 16:00
Task B erster Chunk: Di 14:00
→ B startet vor A-Ende → hardScore -= 1

H4: "Nicht vor"-Datum
Kein Chunk darf vor dem notBefore-Zeitpunkt des Tasks eingeplant werden.

Prüfung:
chunk.slot.start ≥ task.notBefore?

Strafe: −(Differenz in Minuten)

Beispiel:
notBefore: Mi 08:00
Chunk eingeplant: Di 14:00
→ 18h zu früh → hardScore -= 1080

H5: Account-Zuordnung
Ein Chunk darf nur in einem Slot seines eigenen Accounts liegen.

Prüfung:
chunk.task.accountId == slot.accountId?

Strafe: −1 pro Verletzung

H6: Chunk-Reihenfolge
Chunks desselben Tasks müssen chronologisch in der richtigen Reihenfolge liegen.

Prüfung:
chunk[i].slot.start < chunk[i+1].slot.start?

Strafe: −1 pro Vertauschung

Beispiel:
Chunk 0: Mi 14:00
Chunk 1: Di 10:00
→ Chunk 1 liegt vor Chunk 0
→ hardScore -= 1

MEDIUM Constraints — Plan suboptimal, aber nicht ungültig

M1: Deadline einhalten
Der letzte Chunk eines Tasks muss vor der Deadline enden.

Prüfung: Nur auf letztem Chunk:
chunk.effectiveEnd ≤ task.deadline?

Strafe: −(Verspätung in Minuten)

Beispiel:
Deadline: Fr 17:00
Letzter Chunk endet: Mo 10:00 (nächste Woche)
→ 65h zu spät → mediumScore -= 3900

Warum Medium, nicht Hard?
Damit der Solver auch bei unlösbaren Deadlines eine "am wenigsten schlechte" Lösung findet, statt abzubrechen.

M2: Alle Tasks eingeplant
Jeder Chunk muss einem Slot zugewiesen sein.

Prüfung:
chunk.assignedSlot != null?

Strafe: −(chunkDuration in Minuten)

Beispiel:
120-min-Chunk ohne Slot
→ mediumScore -= 120

M3: Manuelle Chunks nicht teilen
Chunks mit customChunkSize (splittable = false) müssen in EINEN zusammenhängenden Slot passen.

Prüfung:
if (!splittable):
chunk liegt komplett in einem Slot?

Strafe: −1 pro geteiltem Chunk

SOFT Constraints — Optimierungsziele

S1: Dringende Tasks früh planen
Tasks mit kleinem Slack sollen frühere Slots bekommen.

Berechnung:
Je kleiner der Slack UND je früher der zugewiesene Slot, desto höher die Belohnung.

Reward: +(totalSlots − slotIndex) × slackGewicht

Beispiel:
Task mit Slack=30min in Slot 2 von 50
→ softScore += 48 × Gewicht

S2: 10 Minuten Pause zwischen Chunks
Nach jedem Chunk (außer dem letzten eines Tasks) sollen mindestens 10 min Pause liegen.

Prüfung:
Abstand zwischen chunk[i].ende und chunk[i+1].start ≥ 10 min?

Strafe: −(10 − tatsächlichePause)

Beispiel:
Chunk 0 endet: 10:00
Chunk 1 startet: 10:05
→ Nur 5 min Pause
→ softScore -= 5

S3: Pausen-Verschmelzung
Wenn eine automatische 10-min-Pause direkt neben einer fixen Pause liegt: automatische kürzen/entfernen.

Prüfung:
Endet ein Chunk ≤10 min vor einer fixen Pause?

Reward: +1 pro vermiedener Leerlauf-Minute

S4: Chunk-Kompaktheit
Chunks desselben Tasks sollen zeitlich nahe beieinander liegen.

Berechnung:
Tage zwischen erstem und letztem Chunk des Tasks.

Strafe: −(Tage Abstand)

Beispiel:
Chunk 0: Montag
Chunk 3: Freitag
→ 4 Tage Abstand
→ softScore -= 4

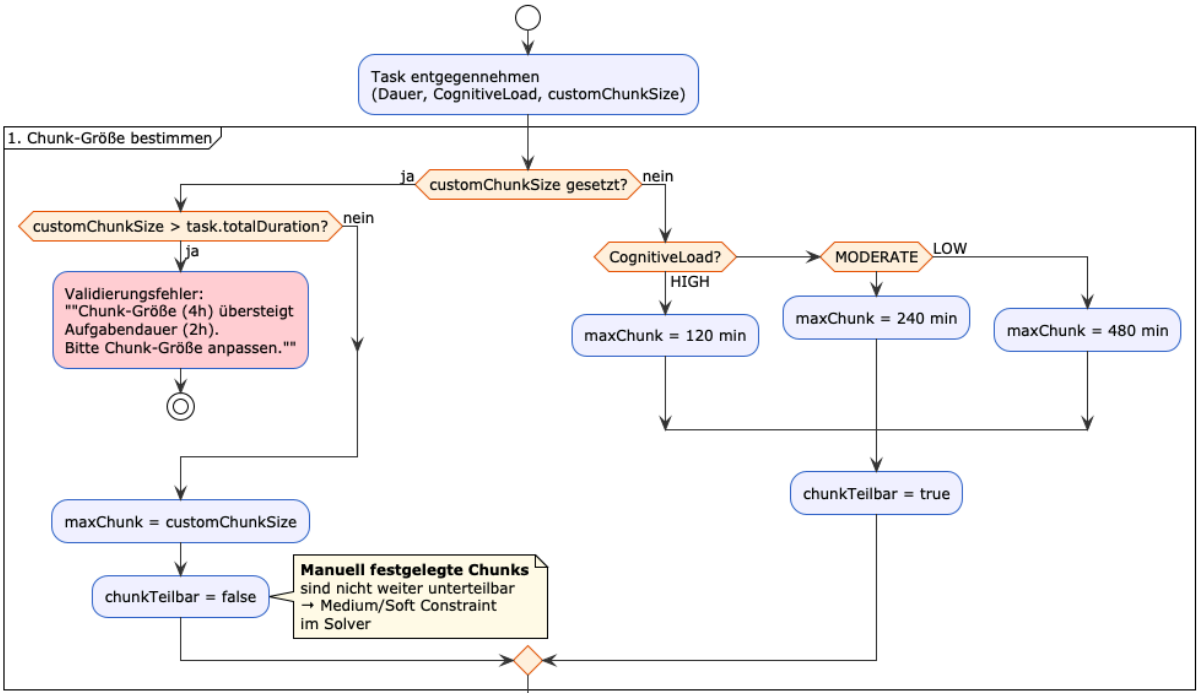
S5: Alternative Task Mode
Wenn 2+ Tasks knapp vor Deadline stehen: bevorzuge abwechselnde Chunk-Verteilung gegenüber sequentieller.

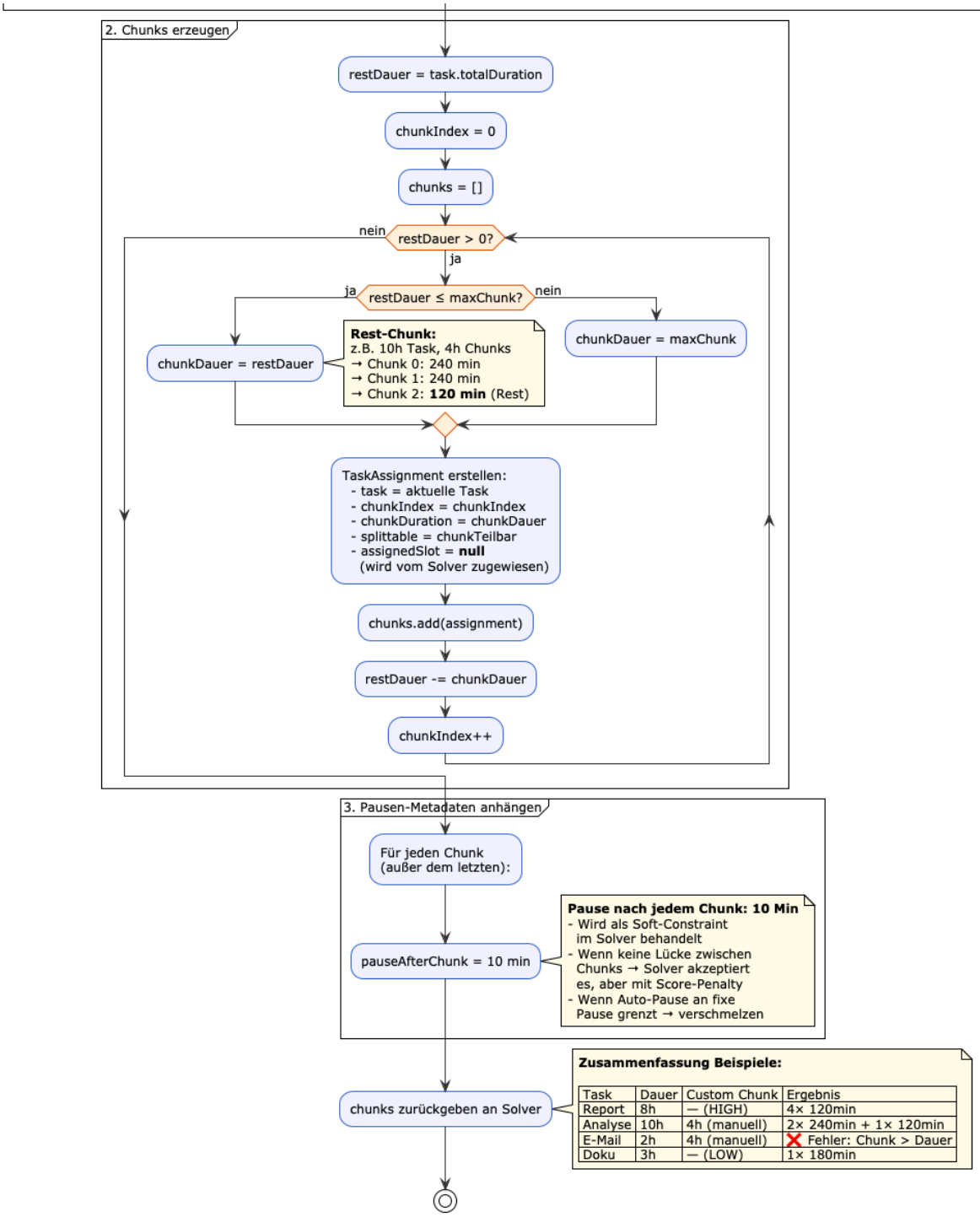
Prüfung:
Mehrere Tasks mit Slack < Schwellwert?
→ Chunks gleichmäßig verteilt?

Reward: +1 pro ausgewogenem Verteilungsschritt

- **Aufgaben-Chunking**
  - Aufgaben werden vor der Einplanung in kleinere Einheiten (Chunks) aufgeteilt
  - Chunk-Größe wird durch die kognitive Belastung bestimmt:
    - High: max. 120 Minuten
    - Moderate: max. 240 Minuten
    - Low: max. 480 Minuten
  - Manuelle Chunk-Größe (customChunkSize) überschreibt die automatische Berechnung
    - Überschreitet die Chunk-Größe die Aufgabendauer, wird ein Validierungsfehler ausgegeben
    - Manuell festgelegte Chunks sind nicht weiter unterteilbar
  - Der letzte Chunk enthält die verbleibende Restdauer (z. B. bei 10h Aufgabe mit 4h Chunks: 4h + 4h + 2h)
  - Zwischen jedem Chunk (außer dem letzten) wird eine Pause von 10 Minuten als Soft Constraint eingeplant
  - Die fertigen Chunks werden mit zugewiesenem Slot = null an den Solver übergeben

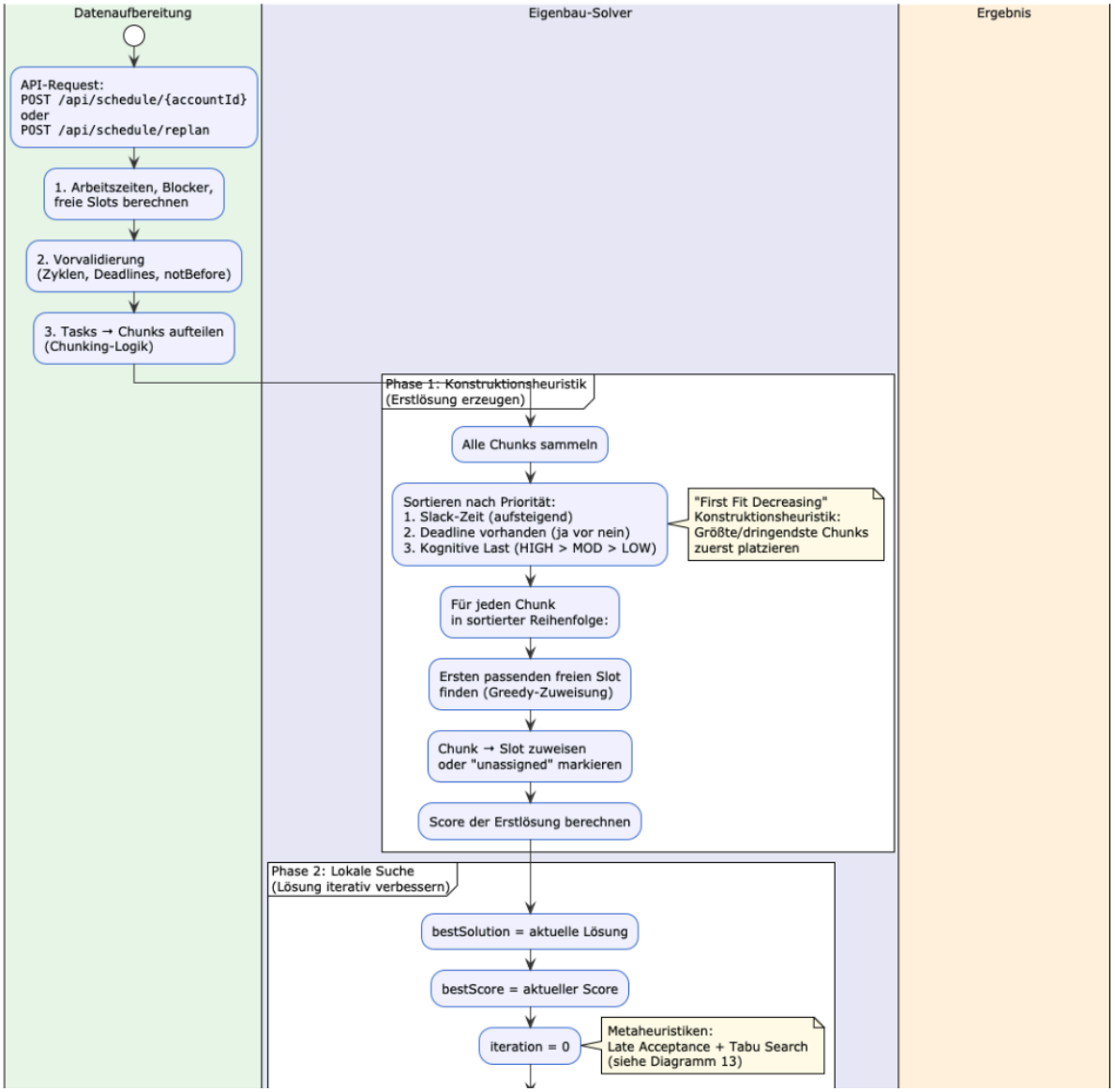
Detail: Chunking-Logik (vor Solver-Übergabe)

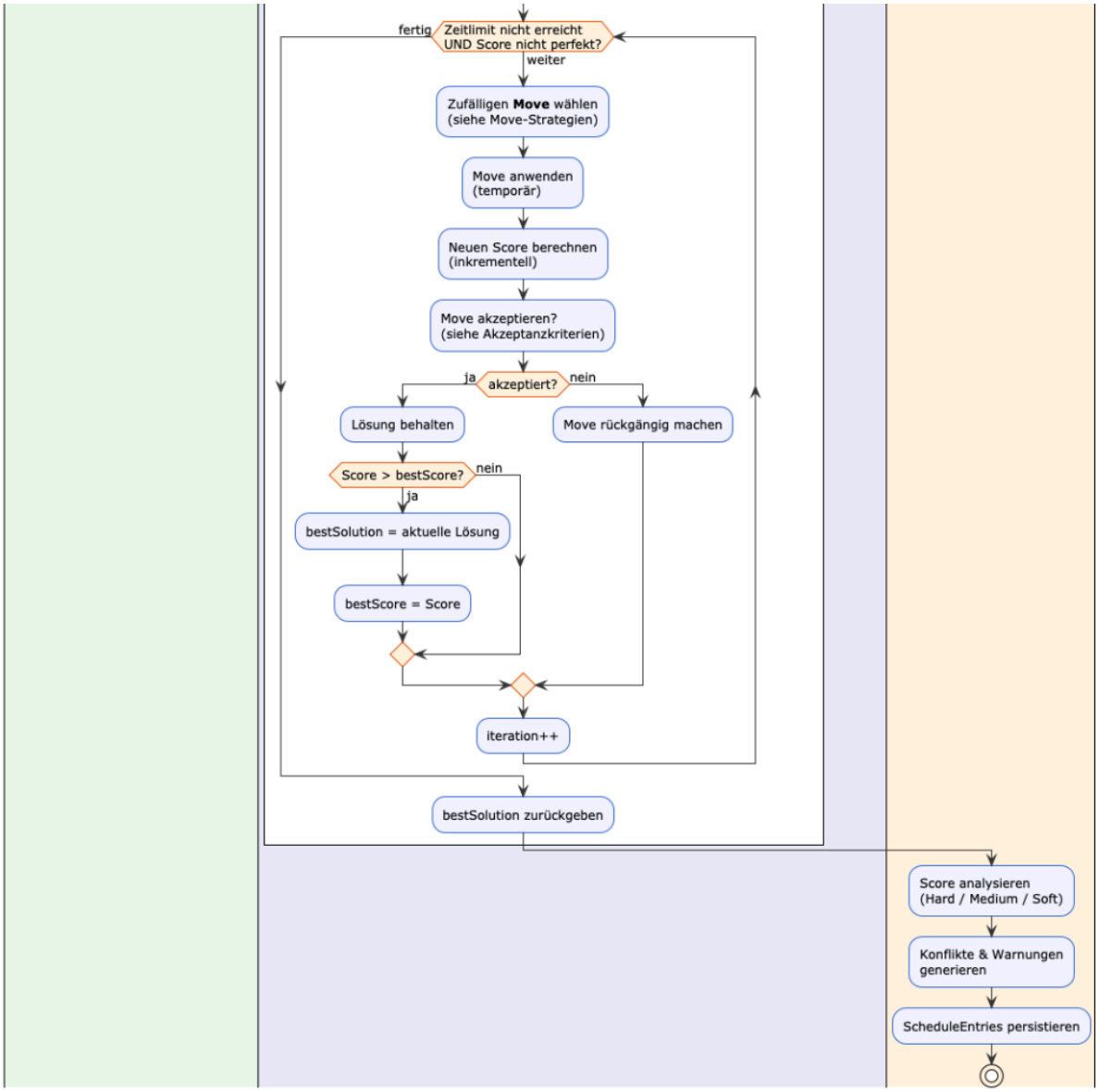



2. Chunks erzeugen
restDauer = task.totalDuration
chunkIndex = 0
chunks = []
nein
restDauer > 0?
ja
ja
restDauer ≤ maxChunk?
nein
chunkDauer = restDauer
Rest-Chunk:
z.B. 10h Task, 4h Chunks
→ Chunk 0: 240 min
→ Chunk 1: 240 min
→ Chunk 2: 120 min (Rest)
chunkDauer = maxChunk
TaskAssignment erstellen:
- task = aktuelle Task
- chunkIndex = chunkIndex
- chunkDuration = chunkDauer
- splittable = chunkTeilbar
- assignedSlot = null
(wird vom Solver zugewiesen)
chunks.add(assignment)
restDauer -= chunkDauer
chunkIndex++
3. Pausen-Metadaten anhängen
Für jeden Chunk
(außer dem letzten):
pauseAfterChunk = 10 min
Pause nach jedem Chunk: 10 Min
- Wird als Soft-Constraint
im Solver behandelt
- Wenn keine Lücke zwischen
Chunks → Solver akzeptiert
es, aber mit Score-Penalty
- Wenn Auto-Pause an fixe
Pause grenzt → verschmelzen
chunks zurückgeben an Solver
Zusammenfassung Beispiele:
Task | Dauer | Custom Chunk | Ergebnis
Report | 8h | — (HIGH) | 4× 120min
Analyse | 10h | 4h (manuell) | 2× 240min + 1× 120min
E-Mail | 2h | 4h (manuell) | ❌ Fehler: Chunk > Dauer
Doku | 3h | — (LOW) | 1× 180min

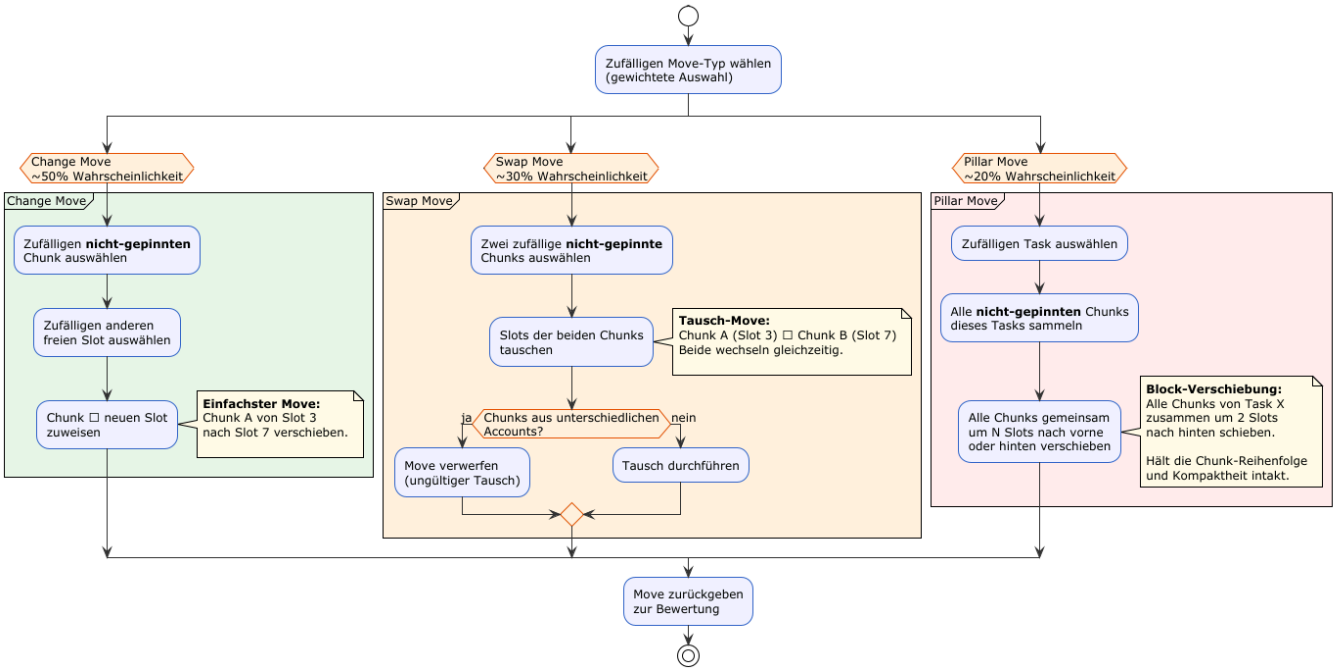
- Initiale Lösung
    - Sortierung der Aufgaben nach Slack-Zeit
    - Einplanung in freie Zeitfenster
    - Berechnung des Scores

Eigenbau-Solver: Hubschrauber-Ansicht (Eigenentwicklung)




fertig
Zeitlimit nicht erreicht UND Score nicht perfekt?
weiter
Zufälligen **Move** wählen (siehe Move-Strategien)
Move anwenden (temporär)
Neuen Score berechnen (inkrementell)
Move akzeptieren? (siehe Akzeptanzkriterien)
ja
akzeptiert?
nein
Lösung behalten
Move rückgängig machen
Score > bestScore?
nein
ja
bestSolution = aktuelle Lösung
bestScore = Score
iteration++
bestSolution zurückgeben
Score analysieren (Hard / Medium / Soft)
Konflikte & Warnungen generieren
ScheduleEntries persistieren

- Generierung von Nachbarlösungen
    - Neue Lösungen werden durch Move-Strategien erzeugt
        - Change: Verschiebung eines Chunks in freie Zeitslots
        - Swap: Tausch zweier Chunks
        - Pillar: Verschiebung aller Chunks einer Aufgabe nach vorne oder hinten
    - Anschließend wird der Score der neuen Lösung berechnet



Accountverwaltung

- Passwort ändern
- Logout
- Account löschen (inkl. Prüfmechanismus bei Abhängigkeiten)

2.2. Typischer Ablauf des Nutzers

Ein typischer Ablauf für die Nutzung der Anwendung gestaltet sich wie folgt:

Login / Zugriff auf das System
Der Nutzer folgt einem Einladungslink des Admins, registriert sich und meldet sich über die verschlüsselte HTTPS-Schnittstelle an. Der Auth Service attestiert im JWT Token die Rechte des Nutzers gegenüber dem Backend.

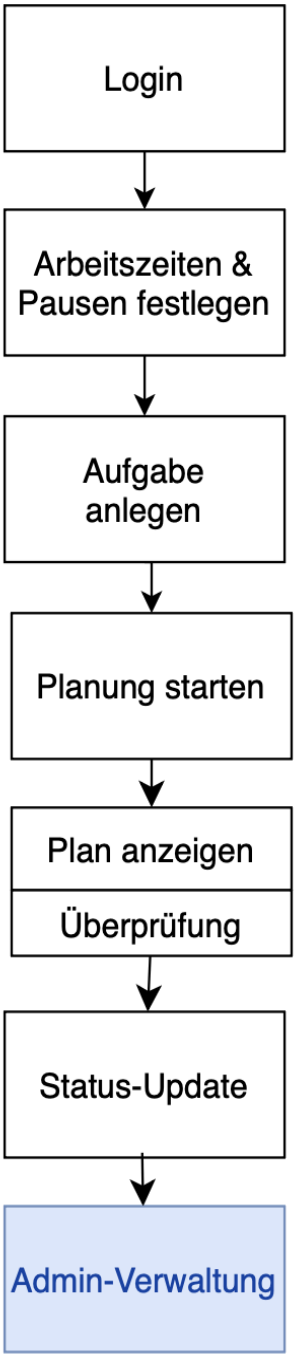
Konfiguration der Basisdaten
Vor der ersten Planung hinterlegt der Nutzer seine individuellen Arbeitszeiten und regelmäßige Pausen im Profil. Oder nutzt die im System hinterlegten Standardzeiten.

Aufgabenerfassung & Validierung
Der Nutzer legt neue Aufgaben an. Das System prüft bereits beim Speichern im Hintergrund, ob die Deadline mit dem aktuellen Arbeitspensum realistisch ist und gibt gegebenfalls Warnhinweise.

Auslösen der automatischen Planung
Der Benutzer startet den Planungsalgorithmus. Dieser läuft asynchron ab, sodass der Nutzer weiterarbeiten kann. Nach Abschluss informiert ein Popup über das Ergebnis.

Review & Interaktion im Kalender
Der generierte Plan wird visuell geprüft. Neue Aufgaben, die nach einer Planung eintreffen, können entweder manuell integriert (mit Überlappungen) oder durch eine komplette Neuplanung eingearbeitet werden.

Bearbeitung & Status-Update
Während des Tages markiert der Nutzer Aufgaben als erledigt. Überfällige Aufgaben werden vom System markiert und dem Nutzer zur Neuplanung angeboten.



Administration (Optional)
Admins verwalten parallel dazu Unternehmensbenutzer, Rollenzuweisungen oder (auf Super-Admin-Ebene) die Unternehmen auf der Plattform.

3. Grundsätzliche Struktur- und Entwurfsentscheidungen

3.1. Architektur & Architekturmuster

Die Anwendung nutzt ein Architekturmuster, um Präsentation, Geschäftslogik und Datenzugriff klar zu trennen und Wartbarkeit sowie Skalierbarkeit zu sichern.

3.1.1. Client-Server-Architektur

Die Anwendung folgt einer klassischen Client-Server-Architektur:

- Das Frontend läuft im Browser und stellt die Benutzeroberfläche bereit
- Das Backend läuft serverseitig und verarbeitet Geschäftslogik sowie Datenzugriffe
- Die Kommunikation erfolgt ausschließlich über HTTP/HTTPS und REST-Endpunkte (JSON)
- Authentifizierung wird ausgelagert durch einen IDP bereitgestellt, welcher über OAuth 2.0 mit dem Frontend kommuniziert und ein JWT für die Autorisierung gegenüber dem Backend ausstellt

3.1.2. Layered Architecture (Schichtenarchitektur im Backend)

Das Backend basiert auf einer mehrschichtigen Architektur:

- Controller-Schicht
  Entgegennahme und Verarbeitung von HTTP-Anfragen sowie Bereitstellung der REST-API
- Service-Schicht
  Enthält die vollständige Geschäftslogik (z. B. Aufgabenplanung, Validierungen, Berechnungen)
- Repository-Schicht
  Abstraktion des Datenzugriffs auf die Datenbank
- Entity-Schicht
  Abbildung der Domänenmodelle (z. B. Nutzer, Aufgaben, Planung)

Diese Struktur sorgt für eine klare Trennung der Verantwortlichkeiten und verbessert Wartbarkeit sowie Testbarkeit.

3.1.3. Frontend-Architektur (SPA)

Das Frontend wird als Single-Page-Application (SPA) umgesetzt.

Vorteile dieser Entscheidung:

- Schnelle Interaktionen ohne Neuladen der Seite
- Flüssige Benutzererfahrung (wichtig für Kalender- und Planungsansichten)
- Reduzierte Serverlast bei UI-Interaktionen

3.1.4. Verwendete Design Patterns

Zur Verbesserung der Struktur und Wartbarkeit werden etablierte Entwurfsmuster eingesetzt:

- Repository Pattern
  Abstraktion und Kapselung des Datenzugriffs
- Dependency Injection
  Verwaltung von Abhängigkeiten durch das Framework (Spring)

3.2. Eingesetzte Technologien und deren Zweck

Die eingesetzten Technologien bilden die Grundlage für eine skalierbare, sichere und wartbare Systemarchitektur im Frontend wie Backend.

3.2.1. Backend

- Java 21 & Spring Boot:
  Grundlage für eine robuste, skalierbare und wartbare Serveranwendung
- Die Authentifizierung wird an den selbst gehosteten Dienst “Zitadel” ausgelagert, wodurch die komplexe Logik standardisiert verwaltet wird (OAuth 2.0), potenzielle Fehlerquellen reduziert und die Sicherheit erhöht wird.
- JPA (Java Persistence API):
  Typsicherer und objektorientierter Datenbankzugriff
- Liquibase:
  Versionierte und reproduzierbare Datenbankmigrationen
- PostgreSQL:
  Relationale Datenbank zur Speicherung aller Systemdaten
- Lombok:
  Reduktion von Boilerplate-Code

3.2.2. Frontend

- Angular & TypeScript:
  Entwicklung einer strukturierten, typisierten Single-Page-Application
- PrimeNG:
  UI-Komponentenbibliothek für konsistente und benutzerfreundliche Oberflächen
- OpenAPI / Swagger:
  Sicherstellung konsistenter Schnittstellen zwischen Frontend und Backend und automatische Generierung der Interfaces im Front- und Backend

3.2.3. Infrastruktur

- Docker:
  Containerisierung der Anwendung für einfache Bereitstellung und Skalierung
- GitHub Actions & Deployment-Tools:
  Automatisierung von Test-, Build- und Deployment-Prozessen

3.3. Trennung von Benutzeroberfläche und Logik

Die Anwendung setzt konsequent auf eine Trennung von Frontend und Backend:

- Das Frontend enthält ausschließlich Präsentationslogik und UI-Komponenten
- Das Backend enthält die vollständige Geschäftslogik und Datenverarbeitung
- Zitadel stellt die Komponenten für die Authentifizierung von Nutzern bereit und wird mittels OAuth 2.0 und Service API angebunden

Eigenschaften dieser Trennung:

- Kommunikation ausschließlich über REST (JSON)
- Kein direkter Zugriff des Frontends auf Datenbank oder Geschäftslogik
- Geschäftslogik (z. B. Planungsalgorithmus) vollständig im Backend gekapselt

Zusätzlich wird ein rollenbasiertes Zugriffskonzept (RBAC) umgesetzt:

- Zugriff auf Funktionen erfolgt abhängig von der Benutzerrolle
- Berechtigungen werden serverseitig geprüft

3.4. Backend-Struktur (Spring Boot)

Die Backend-Struktur orientiert sich an einer klaren Schichtenarchitektur und ist modular aufgebaut.

3.4.1. Controller-Schicht

- Bereitstellung der REST-Endpunkte
- Verarbeitung von HTTP-Anfragen

Typische Controller:

- Aufgabenverwaltung
- Planung
- Benutzer- und Organisationsverwaltung

Entwurfsentscheidung:
Strikte Trennung von API und Geschäftslogik zur Verbesserung von Wartbarkeit und Testbarkeit

3.4.2. Service-Schicht

- Zentrale Geschäftslogik der Anwendung
- Umsetzung komplexer Funktionen wie:
    - automatische Aufgabenplanung
    - Validierungen
    - Berechnungen und Priorisierungen

Entwurfsentscheidung:
Klare Kapselung der fachlichen Logik für bessere Erweiterbarkeit

3.4.3. Datenmodell

Zentrale Domänenobjekte:

- Benutzer
- Unternehmen
- Aufgaben
- Arbeitszeiten
- Planungsdaten
- Pausen

Zusätzlich:

- Verwendung von Enums (z. B. Rollen, Status, Anforderungsgrad)

Entwurfsentscheidung:
Abbildung eines konsistenten Domänenmodells zur Unterstützung der Geschäftslogik

3.4.4. Repository-Schicht

- Zugriff auf die Datenbank über JPA
- Kapselung aller Datenbankoperationen

Entwurfsentscheidung:
Trennung von Persistenz und Logik für bessere Austauschbarkeit der Datenhaltung

3.4.5. Konfiguration

- Systemkonfiguration (z. B. Security, CORS, CSRF, JSON)
- Initialisierung von Systemzuständen

Entwurfsentscheidung:
Zentrale Verwaltung aller Konfigurationen zur besseren Übersichtlichkeit

3.5. Frontend-Struktur (Angular)

Das Frontend basiert auf einer komponentenbasierten Architektur.

3.5.1. Komponentenstruktur

Die Benutzeroberfläche ist in wiederverwendbare Komponenten unterteilt:

- Kalenderansicht (Tag/Woche)
- Aufgabenverwaltung
- Navigation und Sidebar
- Admin- und Einstellungsseiten
- Popups

Entwurfsentscheidung:
Modularisierung zur Wiederverwendbarkeit und klaren Verantwortlichkeiten

3.5.2. Services (Datenzugriff)

- Kommunikation mit dem Backend über HTTP
- Kapselung aller API-Aufrufe

Entwurfsentscheidung:
Trennung von UI und Datenlogik zur Vereinfachung der Komponenten

3.5.3. Routing

- Navigation zwischen Seiten (z. B. Dashboard, Planung, Einstellungen)
- Umsetzung als SPA

Entwurfsentscheidung:
Schnelle Navigation ohne Seitenreload

3.5.4. State-Handling

- Zustand wird in Services und Komponenten verwaltet
- Keine komplexe State-Management-Lösung notwendig

Entwurfsentscheidung:
Reduktion der Komplexität, Erweiterung bei Bedarf möglich

3.6 Vorteile der Architektur

Die gewählte Architektur bietet mehrere entscheidende Vorteile:

Wartbarkeit & Erweiterbarkeit

- Klare Struktur durch Schichtenarchitektur
- Neue Funktionen können gezielt ergänzt werden
- Geringe Abhängigkeiten zwischen Komponenten

Skalierbarkeit

- Stateless Kommunikation (REST / JWT)
- Möglichkeit zur Auslagerung von Services (z. B. Authentifizierung)
- Asynchrone Verarbeitung (Planungsalgorithmus) möglich

Sicherheit

- Rollenbasiertes Zugriffskonzept (RBAC)
- Nutzung eines externen Authentifizierungsdienstes
- Verwendung von OAuth 2 als Industriestandard für sicheres Auth Protokoll
- Sichere Kommunikation über HTTPS

Performance

- SPA sorgt für schnelle UI-Interaktionen

- Backend verarbeitet komplexe Logik effizient
- Asynchrone Verarbeitung entlastet den Nutzerfluss

Testbarkeit

- Jede Schicht kann isoliert getestet werden
- Klare Schnittstellen zwischen Komponenten

Entwicklerfreundlichkeit

- Klare Struktur erleichtert Einarbeitung neuer Entwickler
- Verwendung etablierter Technologien und Standards

4. Statisches Modell

Die statischen UML-Modelle konkretisieren die in Kapitel 1 bis 3 definierten Architektur- und Fachanforderungen: klare Schichten, API-first-Vorgehen, mandantenfähige Datenhaltung und rollenbasierte Zugriffskontrolle. Sie beschreiben, welche Bausteine existieren und wie sie strukturell zusammenhängen, unabhängig von einem konkreten Laufzeitfall.

4.1. Paketdiagramm

Das Paketdiagramm zeigt die logische Schichtung des Backends in Web Layer (controller), Business Layer (service, scheduler), Shared Logic (security, exception, config) und Persistence Layer (repository, entity, entity.enums).
Zusätzlich bildet das Diagramm die API-first-Struktur explizit ab: controller realisiert die generierten Interfaces aus target/generated-sources/api und nutzt DTOs aus target/generated-sources/model.
Der Abhängigkeitsfluss ist gerichtet und konsistent zu Kapitel 1 bis 3: controller -> service -> repository -> entity, mit Querschnittsnutzung von security und exception. Externe Systeme sind ebenfalls modelliert (Angular SPA, Spring-Module, PostgreSQL), wodurch die technische Einbettung der Backend-Architektur nachvollziehbar bleibt. Zur besseren Veranschaulichung der beschriebenen Struktur wird auf das zugehörige Paketdiagramm im Abgabeordner (Verzeichnis „UML-Diagramme“) verwiesen.

4.2. Domänenmodell

Das Domänenmodell beschreibt die zentralen Entitäten und Enums zweier Pakete. Im Hauptpaket sind die Klassen User, Organization, Membership, Task, Appointment (Termine), Break, WorkingHours, ScheduleEntry, Invitation und Session definiert. Das Solver-Paket enthält zusätzlich TaskAssignment, TimeSlot und HardMediumSoftScore. Als Enums sind Role, MembershipRole, MembershipStatus, TaskStatus, CognitiveLoad und ScheduleEntryType modelliert. Ein separater RecurrenceType-Enum existiert nicht – Wiederkehrendes wird stattdessen über ein rruleExpression-Feld im RRule-Format direkt in Appointment und Break abgebildet (z. B. FREQ=WEEKLY;BYDAY=MO,WE,FR). Dieses Format ist ein Standard in vielen anderen Softwareprojekten für wiederkehrende Termine, daher haben wir uns für dieses Format entschieden. Kernentscheidung ist die explizite Membership-Entität als n:m-Auflösung zwischen User und Organization. Damit können Nutzer unterschiedliche Rollen in unterschiedlichen Unternehmen haben, konsistent zu den RBAC-Anforderungen.
Das Diagramm zeigt außerdem die für Aufgaben- und Planungslogik relevanten Beziehungen: reine Task-Abhängigkeiten (explizit ohne Parent-Child-Struktur) sowie die Zuordnung von ScheduleEntry zu genau einem Fachobjekt (Task, Break oder Appointment). Das Solver-Modell bildet dabei eine eigene Schicht: TaskAssignment repräsentiert einen Chunk einer Task und wird nach der Optimierung in einen persistierten ScheduleEntry überführt, während TimeSlot als berechnetes, nicht persistiertes Zeitfenster nur zur Laufzeit des Solvers existiert.
Zur besseren Veranschaulichung siehe Diagramm Domänenmodell im Abgabeordner (UML-Diagramme).

5. Dynamisches Modell

Das dynamische Modell beschreibt die Laufzeitinteraktion zwischen Akteuren, API-Endpunkten, Services und Persistenz. Die Diagramme referenzieren die Musskriterien (Feature-IDs) und bleiben in Terminologie und Endpunktnamen konsistent zum API-Vertrag. Der Fokus liegt auf den fachlichen Interaktionsschritten; Ausführungsdetails wie synchrone oder asynchrone Verarbeitung werden in Kapitel 2 kontextualisiert.

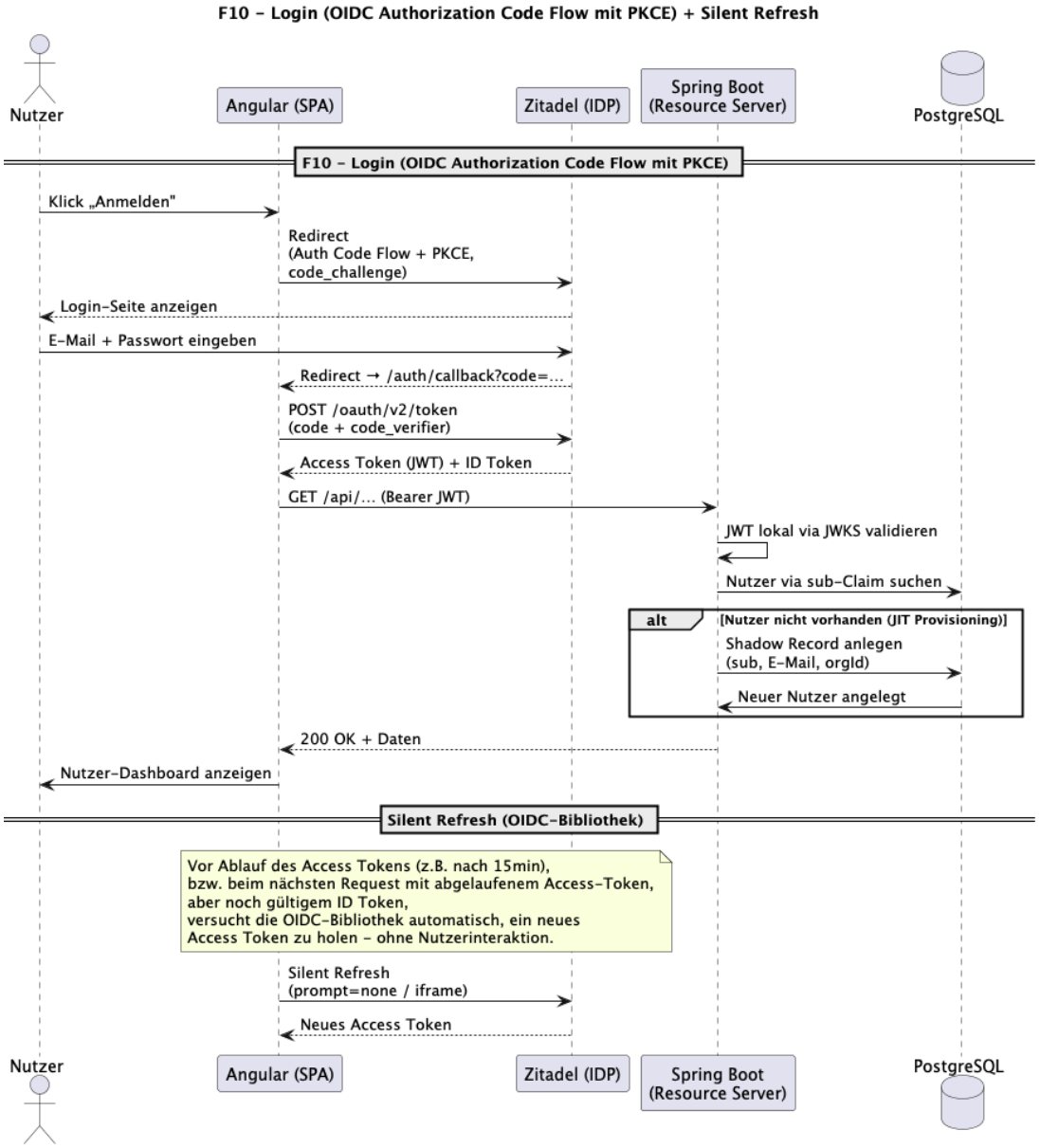
5.1. Sequenzdiagramme

Die Sequenzdiagramme zeigen die Laufzeit Interaktion für zentrale Kernfeatures (Authentifizierung, Einladung/Mitgliedschaft, Aufgabenlebenszyklus, Planung, Super-Admin-Funktionen sowie Passwort-/Kontoverwaltung). Dadurch wird nachvollziehbar, wie API-Aufrufe, Fachlogik und Persistenz zusammenspielen.
In Kombination mit der Nachverfolgbarkeitstabelle entsteht eine direkte Zuordnung von Feature-IDs zu Methoden und Interaktionsabläufen, was die inhaltliche Vollständigkeit der ersten Iteration transparent macht.

5.1.1 Benutzerverwaltung

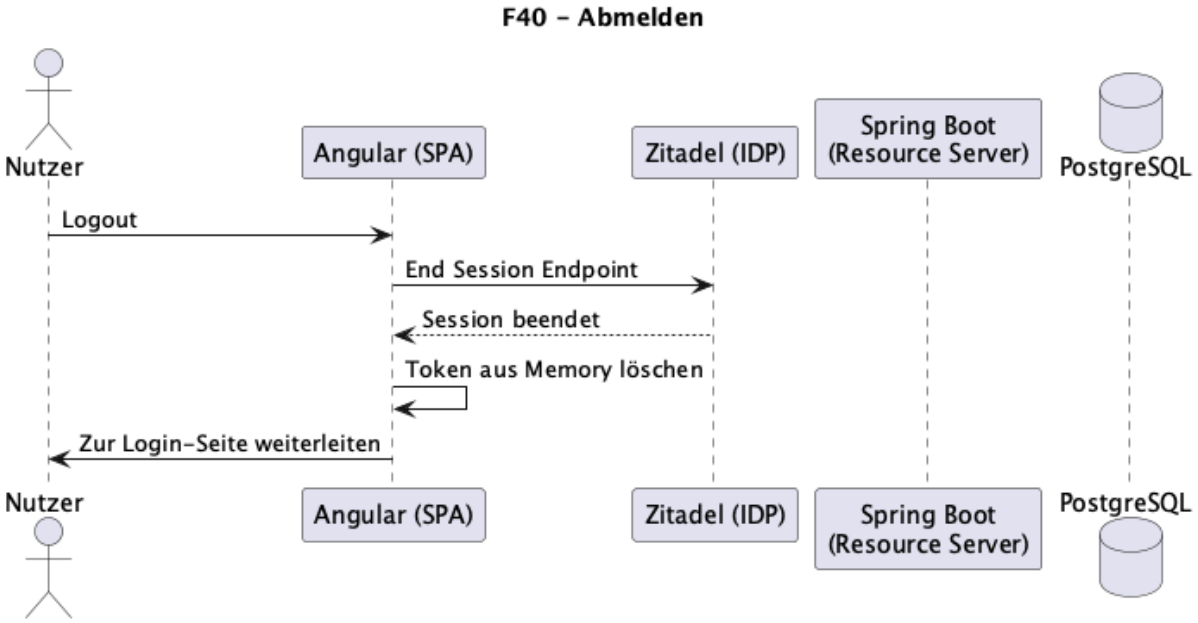
F10 – Login, Refresh

Der Einstiegspunkt für alle Benutzer: F10 zeigt den OIDC Authorization Code Flow mit PKCE zu Zitadel. Der Backend validiert das JWT lokal via JWKS und führt JIT-Provisioning durch (erstellt einen Shadow-Record für neue Benutzer mit sub, email und orgId aus dem Token). Dieser Shadow-Record wird für die spätere Zuordnung von Entitäten zu einem Nutzer in unserem System benötigt. Anschließend zeigt das Diagramm den Silent Refresh: Vor Token-Ablauf (z. B. 15 Min) ruft die OIDC-Bibliothek mit `prompt=none` einen neuen Token ab, ohne dass der Benutzer neu authentifizieren muss.



F40 - Abmelden

Ein Benutzer meldet sich ab. Der Angular-Client ruft Zitadels End Session Endpoint auf, die Sitzung wird beendet, der Token aus dem Memory gelöscht und zur Login-Seite weitergeleitet. Dieser Flow ist einfach und vollständig an Zitadel delegiert.

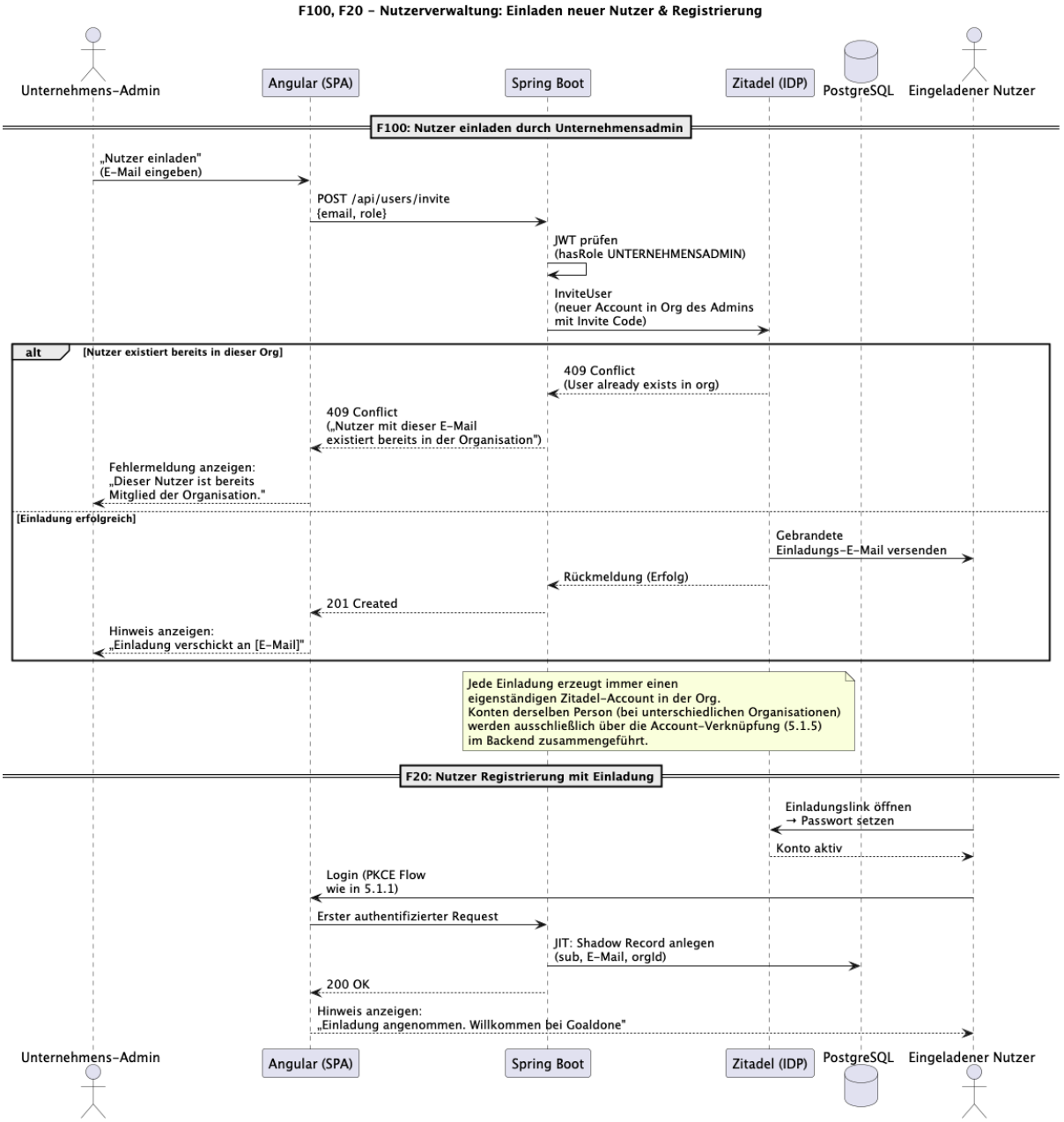


F100, F20 - Einladen neuer Nutzer & Registrierung

F100: Ein Unternehmens-Admin lädt neue Benutzer ein. Über POST `/api/users/invite` werden Einladungscodes generiert und der Benutzer in Zitadel angelegt (oder 409 falls bereits in der Org). Zitadel versendet eine Einladungs-E-Mail.

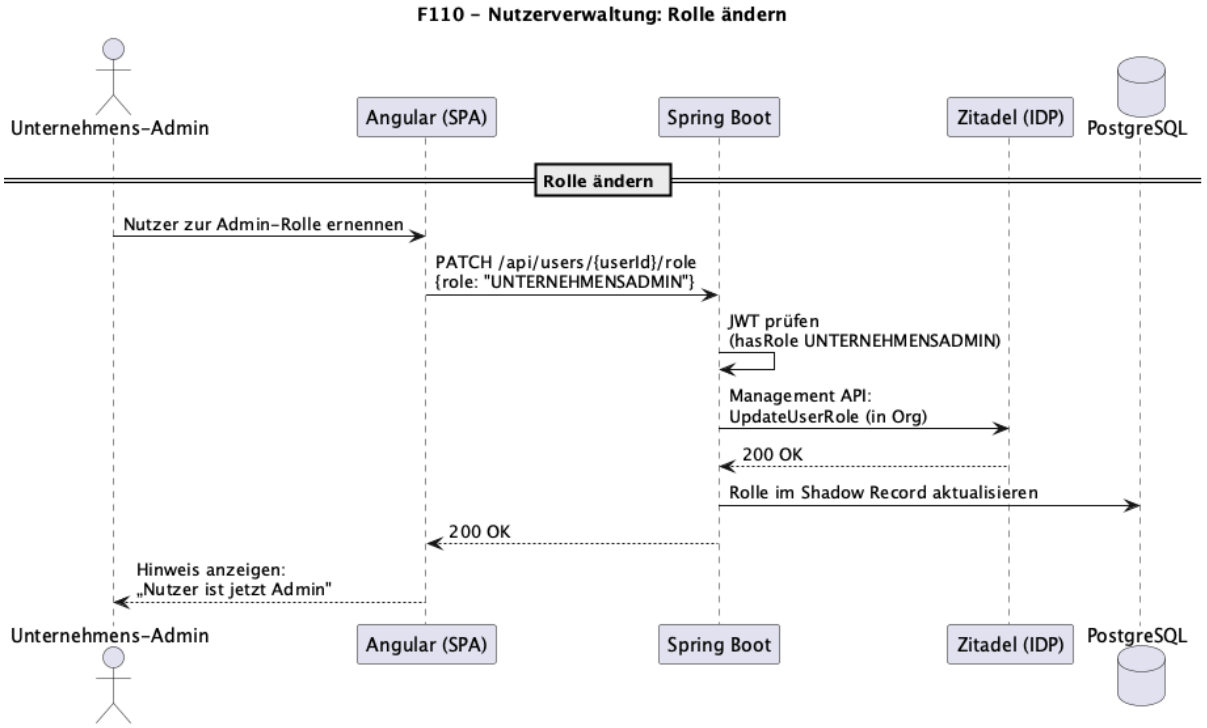
F20: Der eingeladene Benutzer öffnet den Einladungslink, setzt sein Passwort in Zitadel und meldet sich an. Das Backend erstellt den Shadow-Record mit der eingeladenen Rolle.

Wichtig: Jede Einladung erstellt einen neuen Zitadel-Account pro Org; zur organisationsübergreifenden Account-Zusammenführung siehe „Account Verknüpfung".



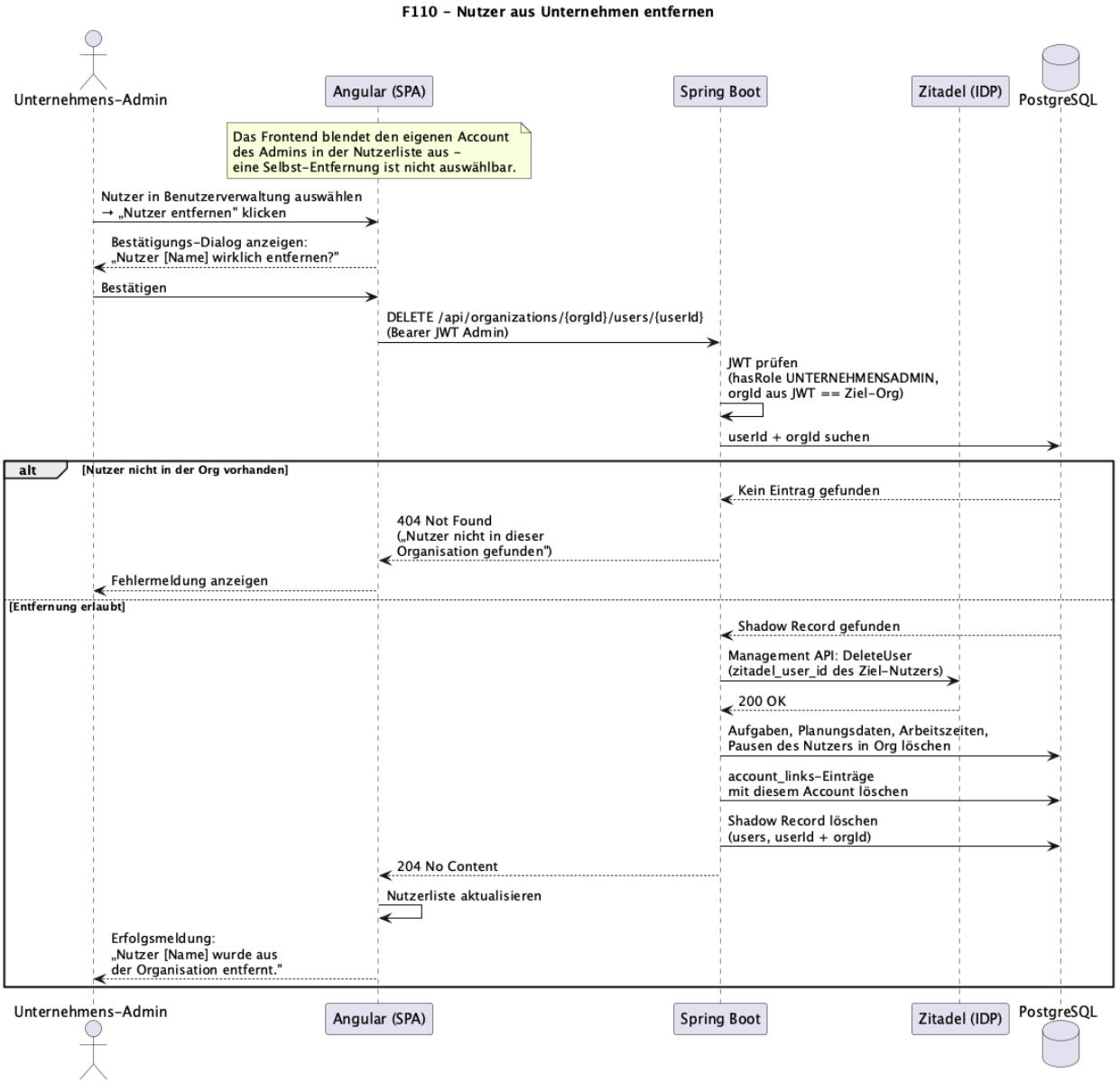
F110 - Benutzerverwaltung: Rolle ändern

Ein Unternehmens-Admin kann die Rolle eines Benutzers ändern (z. B. von Benutzer zu Admin). PATCH `/api/users/{userId}/role` sendet die neue Rolle an die Zitadel Management API und aktualisiert den Shadow-Record in der Datenbank.



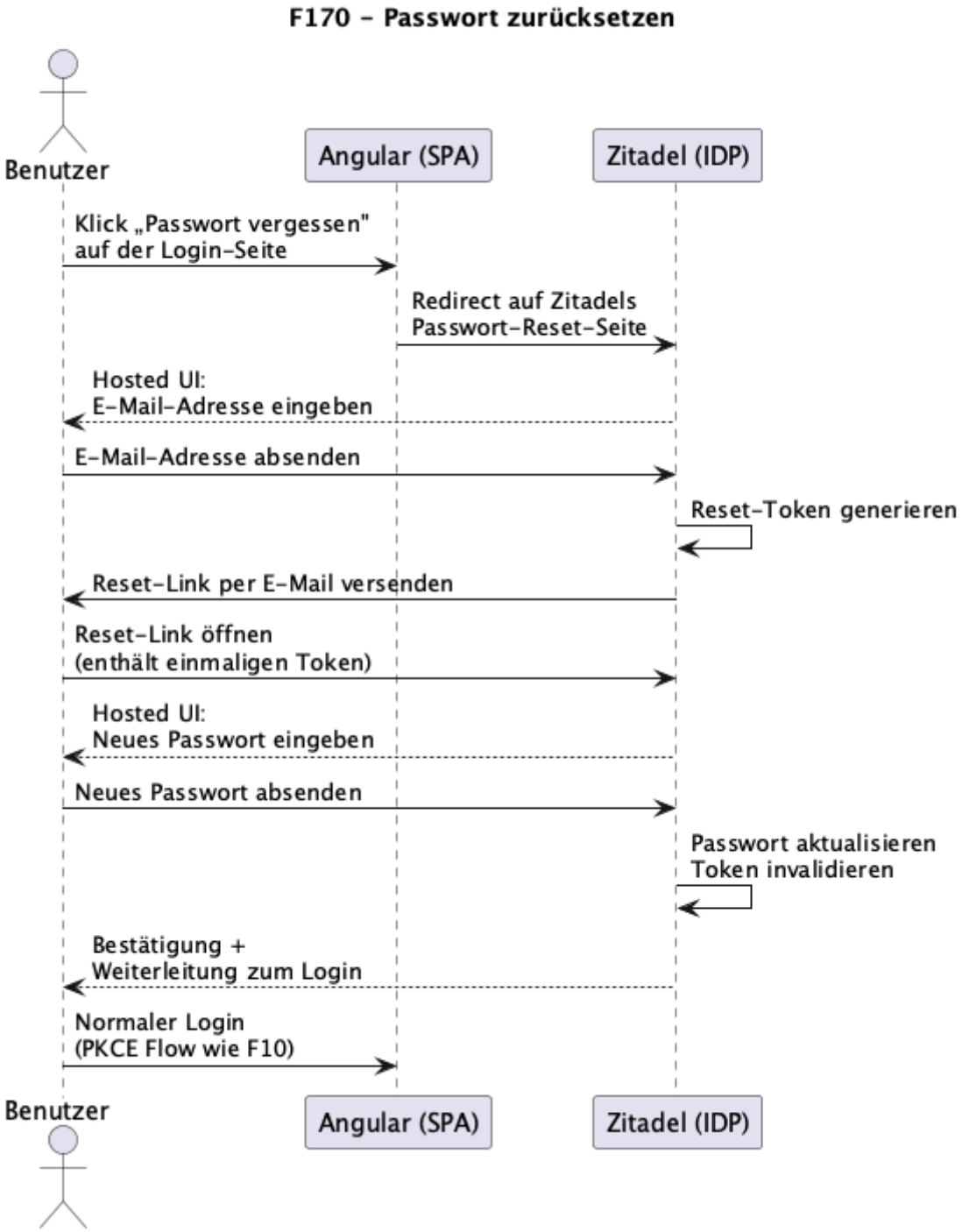
F110 - Nutzer aus Unternehmen entfernen

Ein Unternehmens-Admin kann Benutzer aus der Organisation entfernen. DELETE `/api/organizations/{orgId}/users/{userId}` führt eine Kaskadenlöschung durch: Zitadel-Account wird gelöscht, anschließend alle zugehörigen Tasks, Planungsdaten, Arbeitszeiten, Pausen und Account-Links aus der Datenbank.



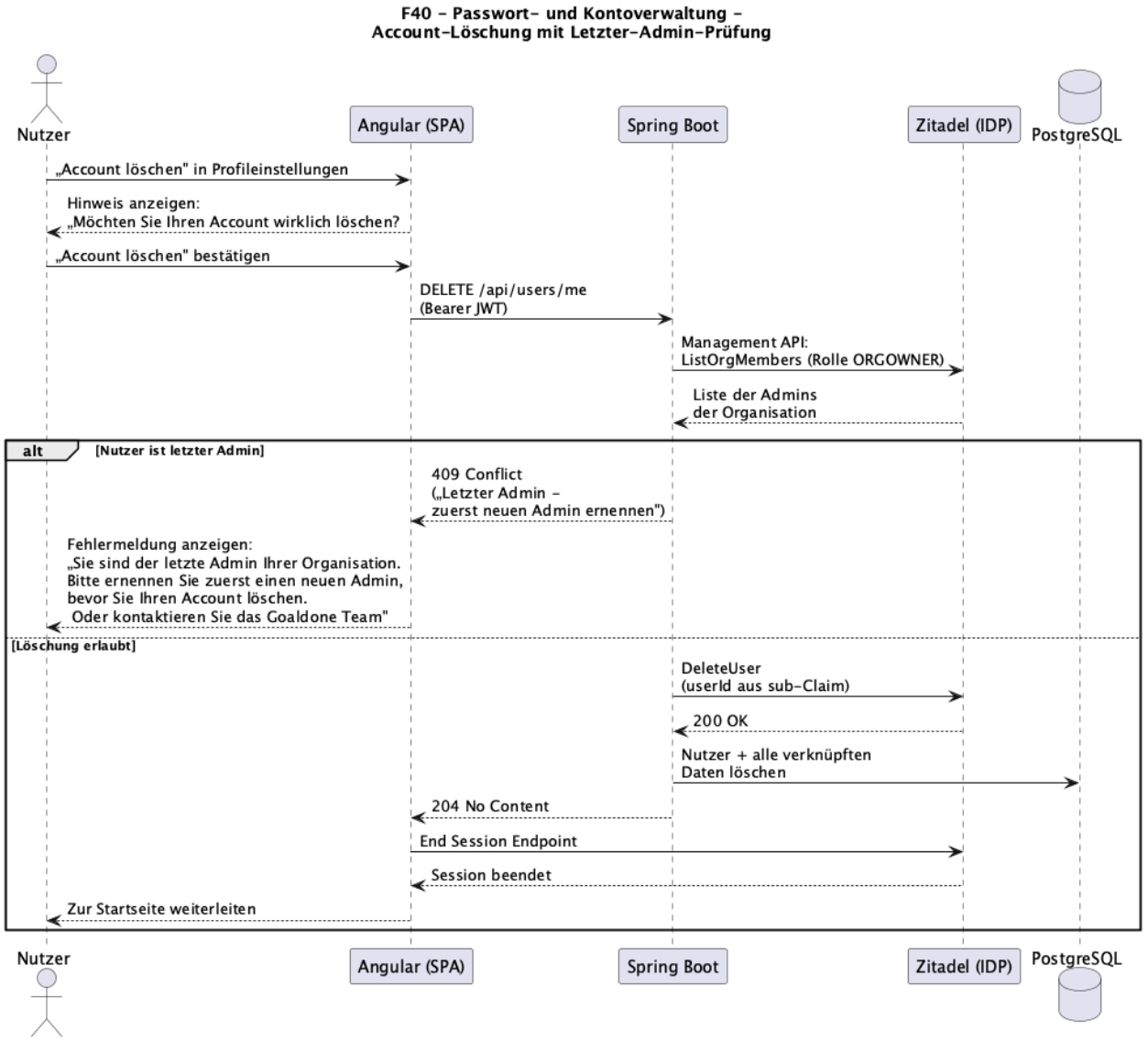
F170 - Passwort vergessen

Passwort-Resets sind vollständig an Zitadel delegiert. Der Benutzer klickt „Passwort vergessen", wird zu Zitadels Hosted UI weitergeleitet, gibt seine E-Mail ein, öffnet den Reset-Link aus der E-Mail, setzt ein neues Passwort und wird dann zur regulären PKCE-Login-Seite weitergeleitet. Der Backend muss hier nicht eingreifen.



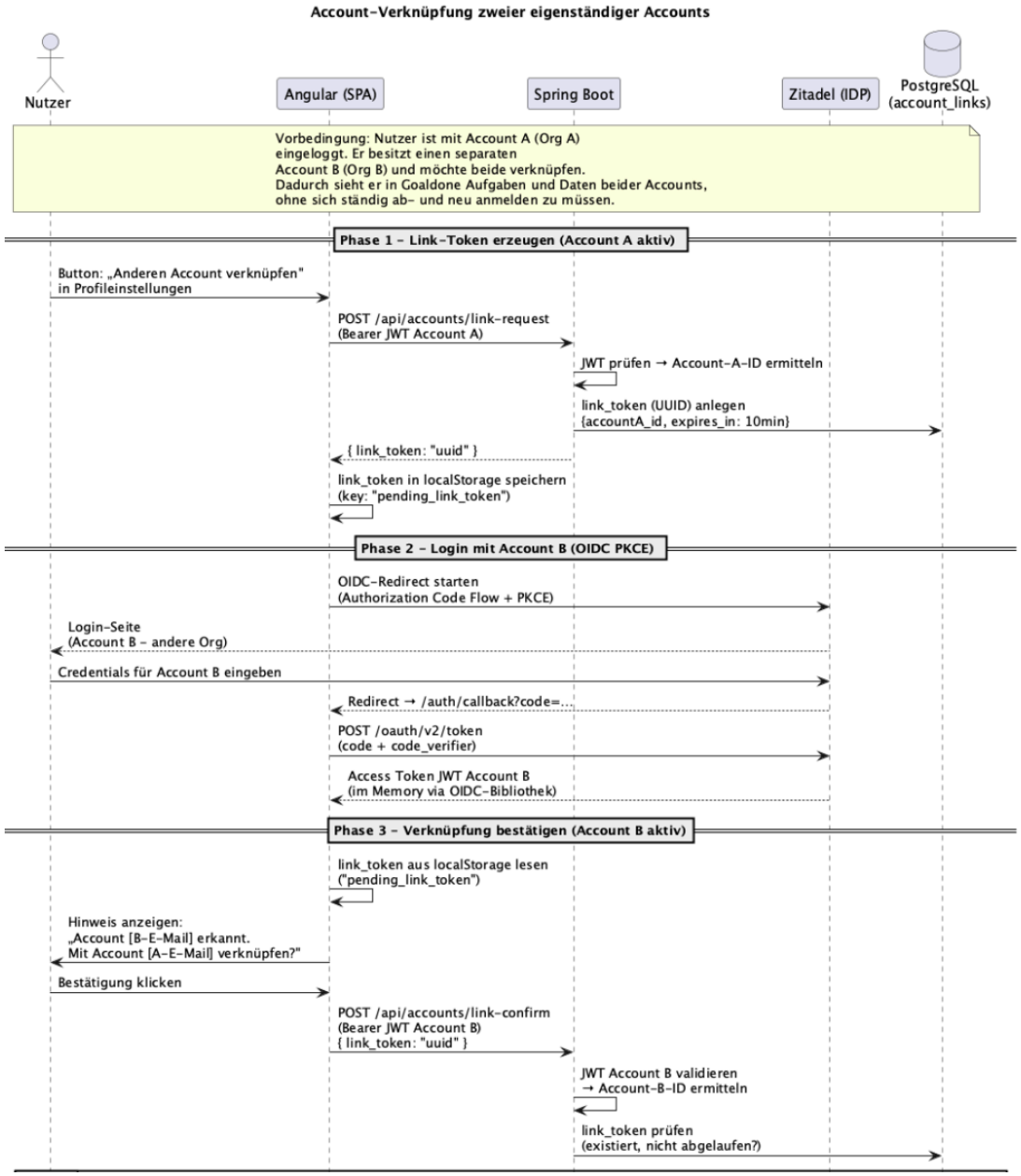
F40 - Account löschen

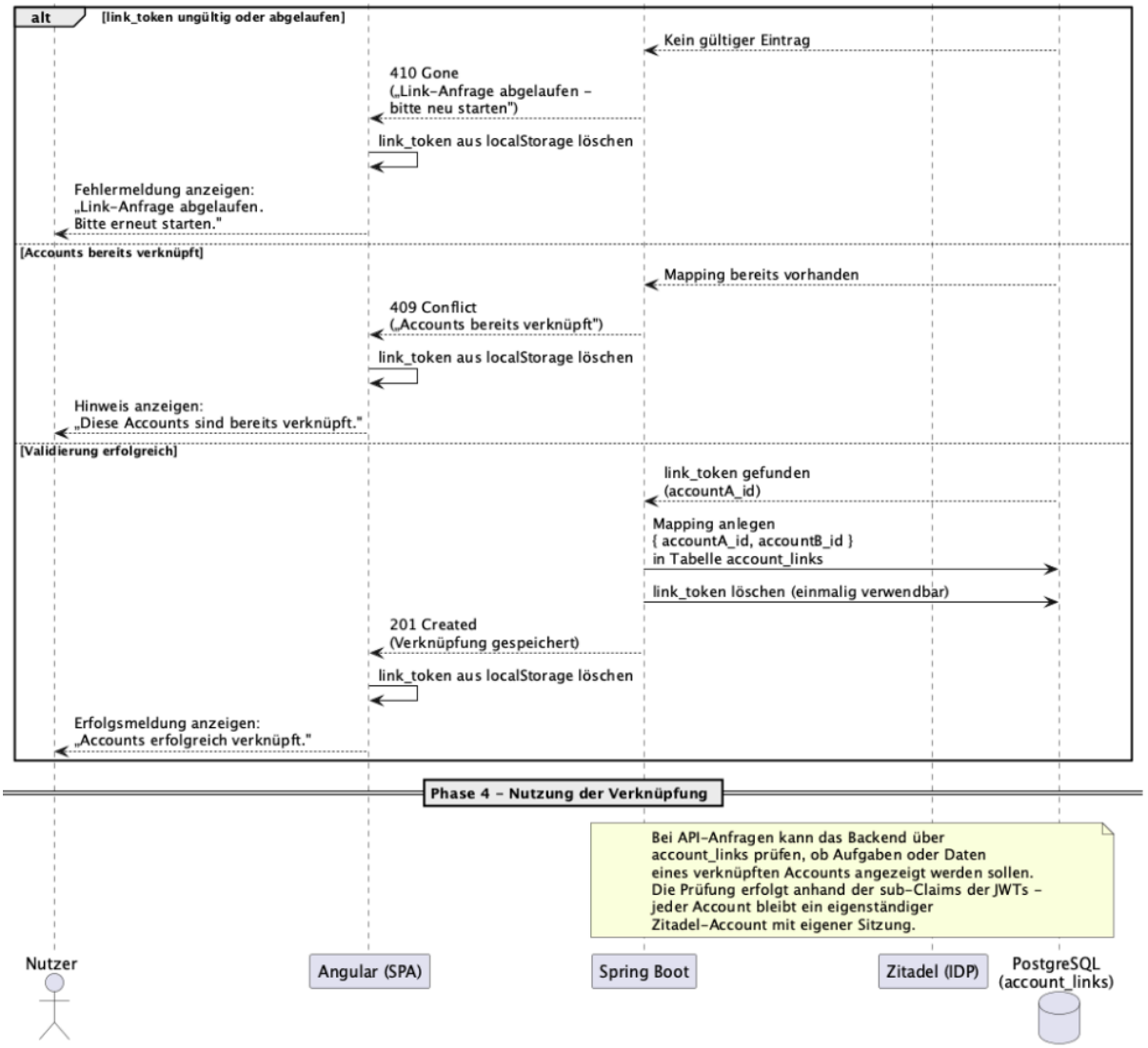
Ein Benutzer kann seinen Account löschen. Das System verhindert, dass der letzte Admin eines Unternehmens gelöscht wird: DELETE `/api/users/me` prüft via Zitadel Management API, ob der Benutzer der letzte ORGOWNER ist. Falls ja, wird 409 Conflict mit Fehlermeldung zurückgegeben. Andernfalls werden der Zitadel-Account und alle zugehörigen Daten in internen Systemen kaskadierend gelöscht.



F190 - Account-Verknüpfung

Dieser fortgeschrittene Flow ermöglicht es einem Benutzer, zwei separate Accounts zu verknüpfen (z. B. Account A in Org A und Account B in Org B).
Der Prozess hat vier Phasen:
1. **Link-Request (Account A aktiv):** POST `/api/accounts/link-request` generiert einen link_token (UUID, 10 Min gültig), der lokal gespeichert wird.
2. **Login mit Account B:** Standardmäßiger OIDC PKCE Authorization Code Flow.
3. **Link-Confirm (Account B aktiv):** POST `/api/accounts/link-confirm` mit Bearer JWT von Account B und link_token erstellt einen Mapping-Eintrag in der `account_links`-Tabelle.
4. **Nutzung:** Beim Backend-Lookup werden verknüpfte Accounts basierend auf JWT-Claims aufgelöst.

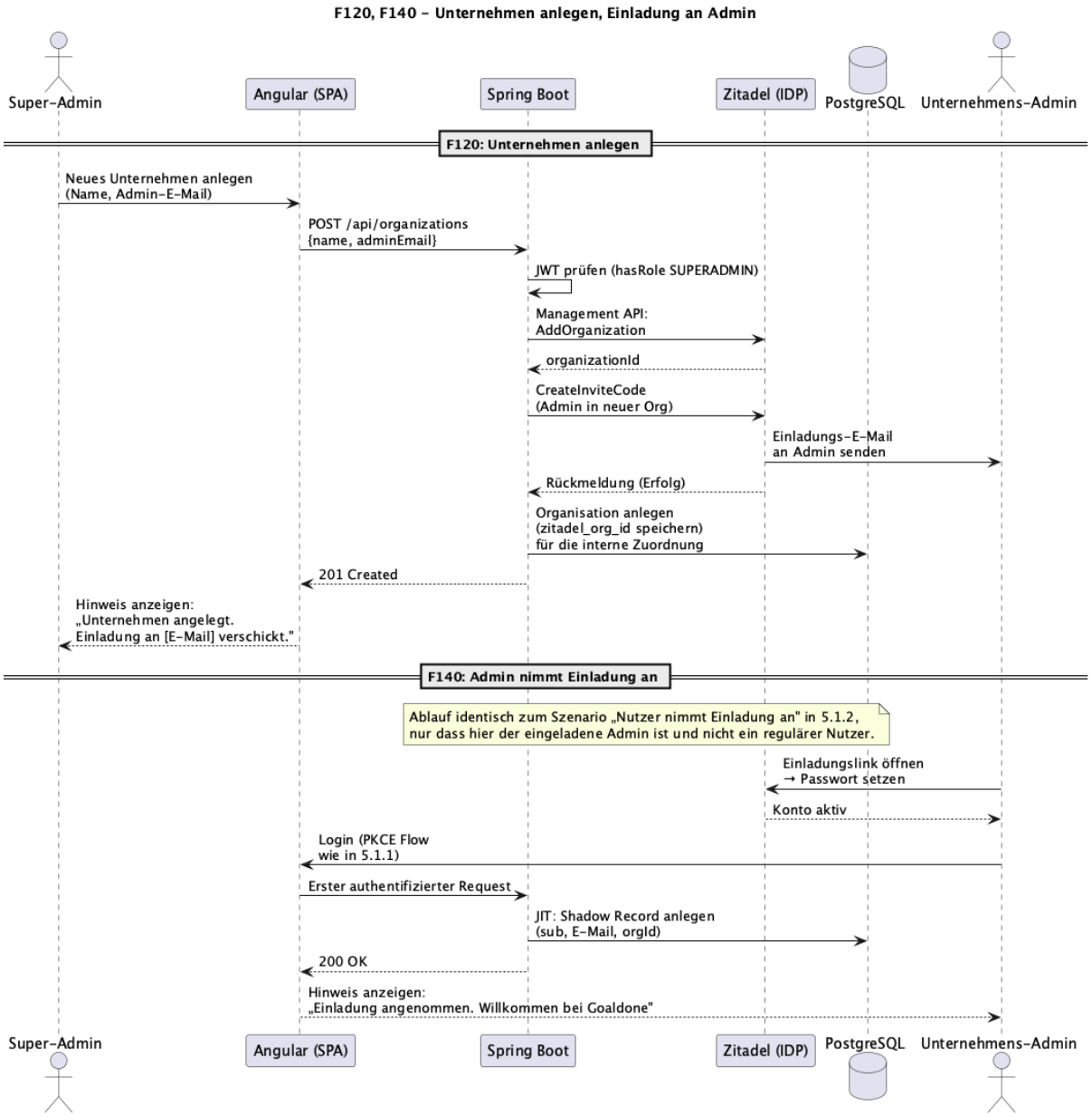



alt
[link_token ungültig oder abgelaufen]
Kein gültiger Eintrag
410 Gone
(„Link-Anfrage abgelaufen –
bitte neu starten")
link_token aus localStorage löschen
Fehlermeldung anzeigen:
„Link-Anfrage abgelaufen.
Bitte erneut starten."
[Accounts bereits verknüpft]
Mapping bereits vorhanden
409 Conflict
(„Accounts bereits verknüpft")
link_token aus localStorage löschen
Hinweis anzeigen:
„Diese Accounts sind bereits verknüpft."
[Validierung erfolgreich]
link_token gefunden
(accountA_id)
Mapping anlegen
{ accountA_id, accountB_id }
in Tabelle account_links
link_token löschen (einmalig verwendbar)
201 Created
(Verknüpfung gespeichert)
link_token aus localStorage löschen
Erfolgsmeldung anzeigen:
„Accounts erfolgreich verknüpft."
Phase 4 – Nutzung der Verknüpfung
Bei API-Anfragen kann das Backend über
account_links prüfen, ob Aufgaben oder Daten
eines verknüpften Accounts angezeigt werden sollen.
Die Prüfung erfolgt anhand der sub-Claims der JWTs –
jeder Account bleibt ein eigenständiger
Zitadel-Account mit eigener Sitzung.
Nutzer
Angular (SPA)
Spring Boot
Zitadel (IDP)
PostgreSQL
(account_links)

5.1.2 Super-Admin Verwaltung

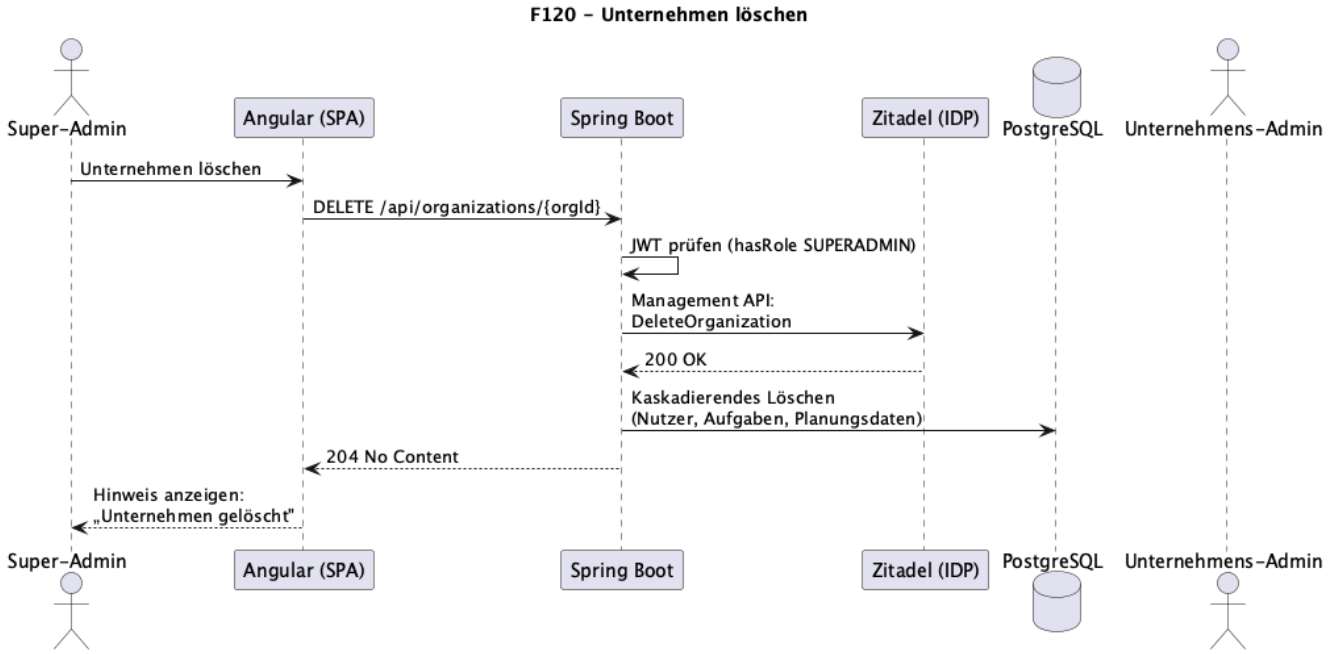
F120, F140 - Unternehmen anlegen und Einladung an Admin verschicken

Ein Super-Admin erstellt ein neues Unternehmen und lädt den ersten Unternehmens-Admin ein. Der Super-Admin sendet POST `/api/organizations` mit Name und Admin-E-Mail. Zitadel erstellt die neue Organisation und generiert einen Einladungscode für den Admin. Der Admin erhält eine Einladungs-E-Mail und akzeptiert diese (F140), woraufhin die Registrierung im neuen Unternehmen abgeschlossen ist.



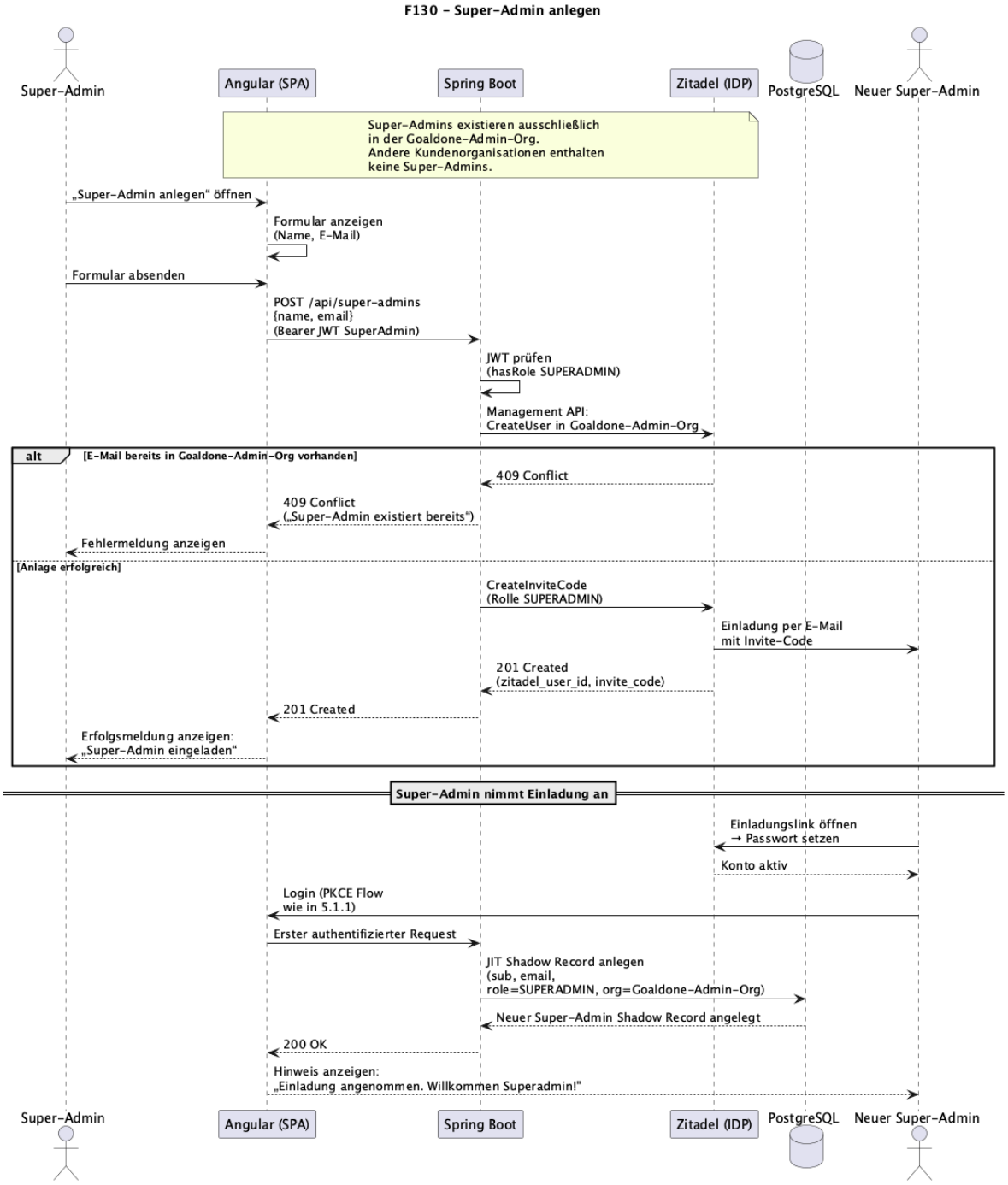
F120 - Unternehmen löschen

Ein Super-Admin kann ein bestehendes Unternehmen löschen. Die Anfrage DELETE `/api/organizations/{orgId}` wird an Zitadel gesendet, die Organisation wird gelöscht, und anschließend erfolgt eine Kaskadenlöschung aller zugehörigen Daten in der Datenbank (Benutzer, Tasks, Planungsdaten).



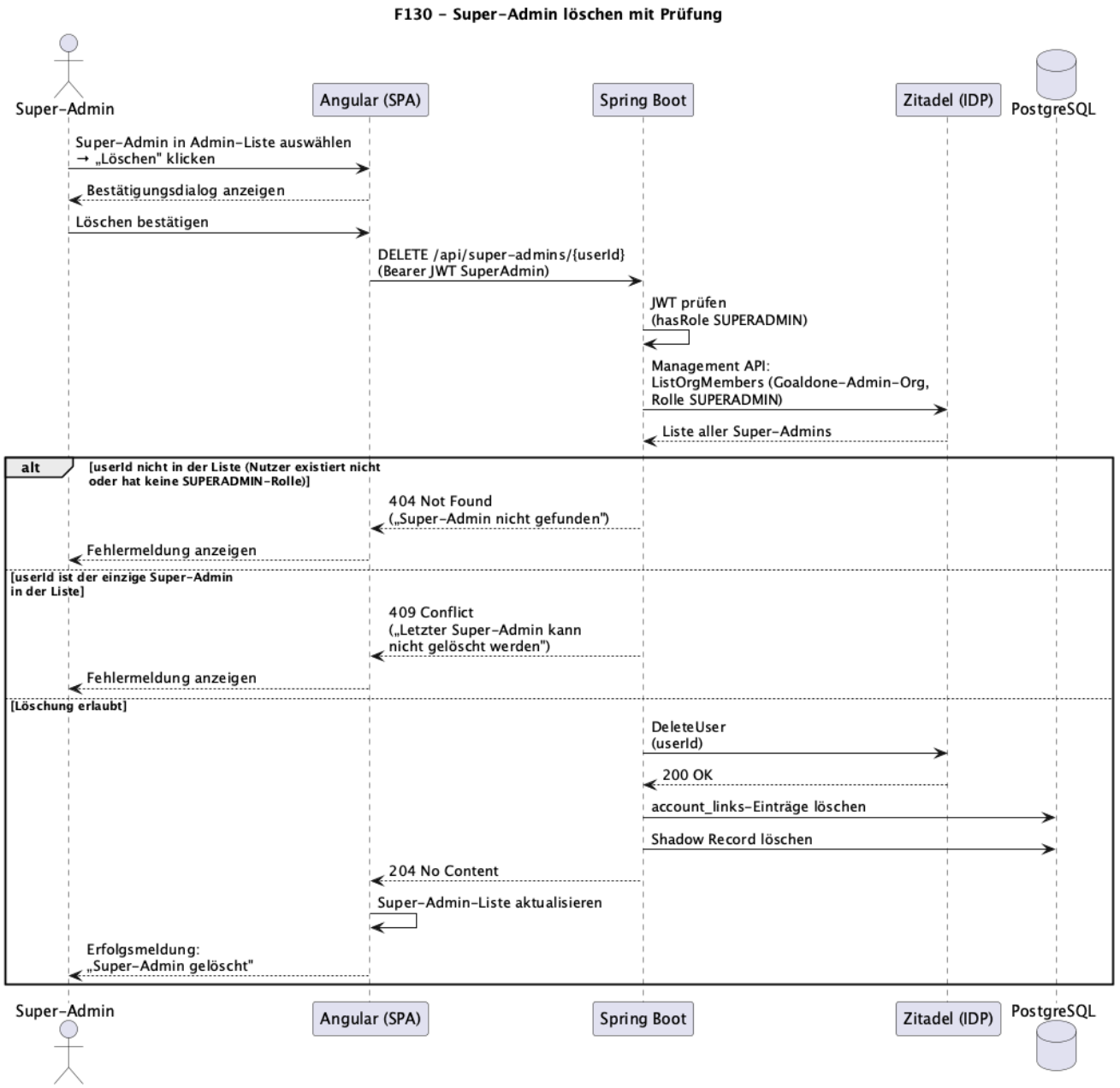
F130 - Neue Super-Admins anlegen

Ein Super-Admin erstellt einen neuen Super-Admin-Account. Super-Admins existieren ausschließlich in einer extra dafür in Zitadel angelegten Goaldone-Admin-Org. Der neue Super-Admin wird über das Frontend angelegt, über Admin-API erstellt und erhält eine Einladungs-E-Mail mit Einladungscode. Nachdem er sein Passwort setzt und sich anmeldet, wird sein Account aktiviert mit der Rolle SUPERADMIN.



F130 - Super-Admin löschen

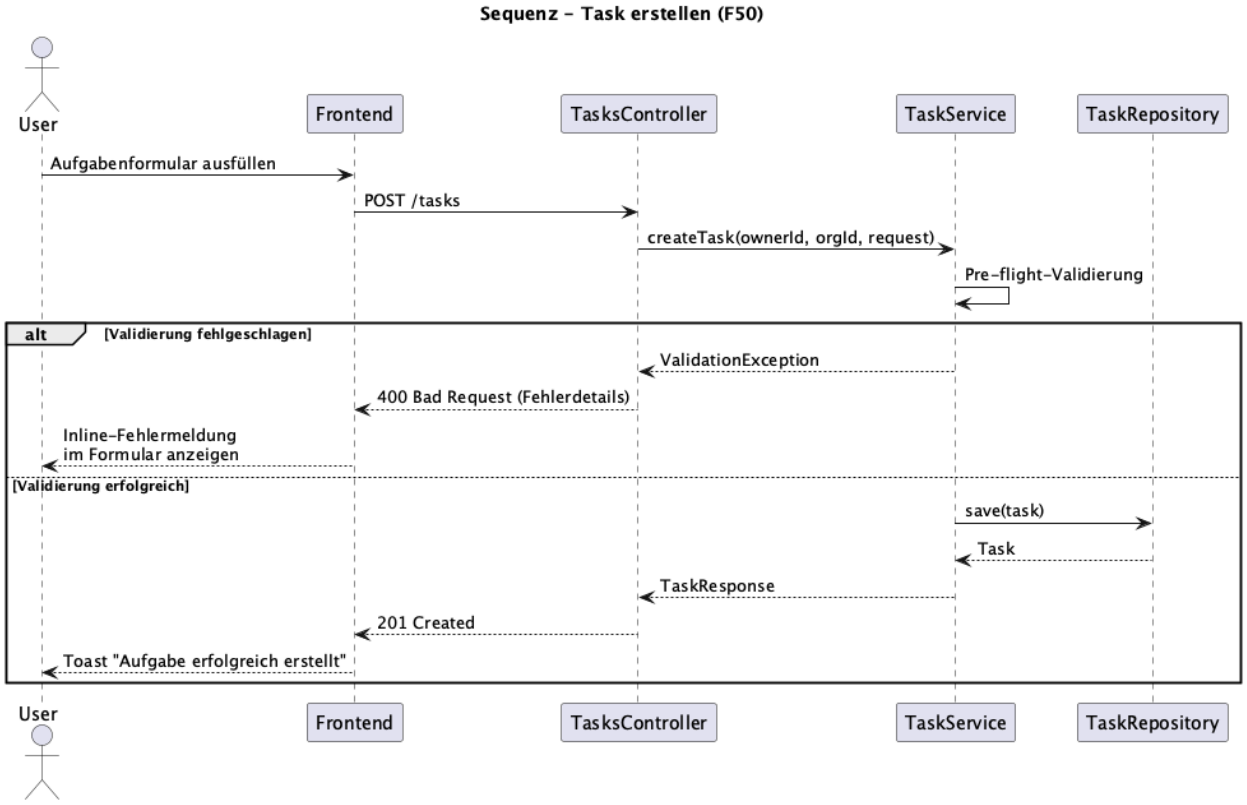
Ein Super-Admin kann einen anderen Super-Admin löschen. Das System verhindert, dass der letzte Super-Admin gelöscht wird (409 Conflict). Wenn die Bedingung erfüllt ist, werden der Zitadel-Account und alle zugehörigen Daten (Shadow Record, Account-Links) gelöscht.



5.1.5 Task-Lebenszyklus und Filterung

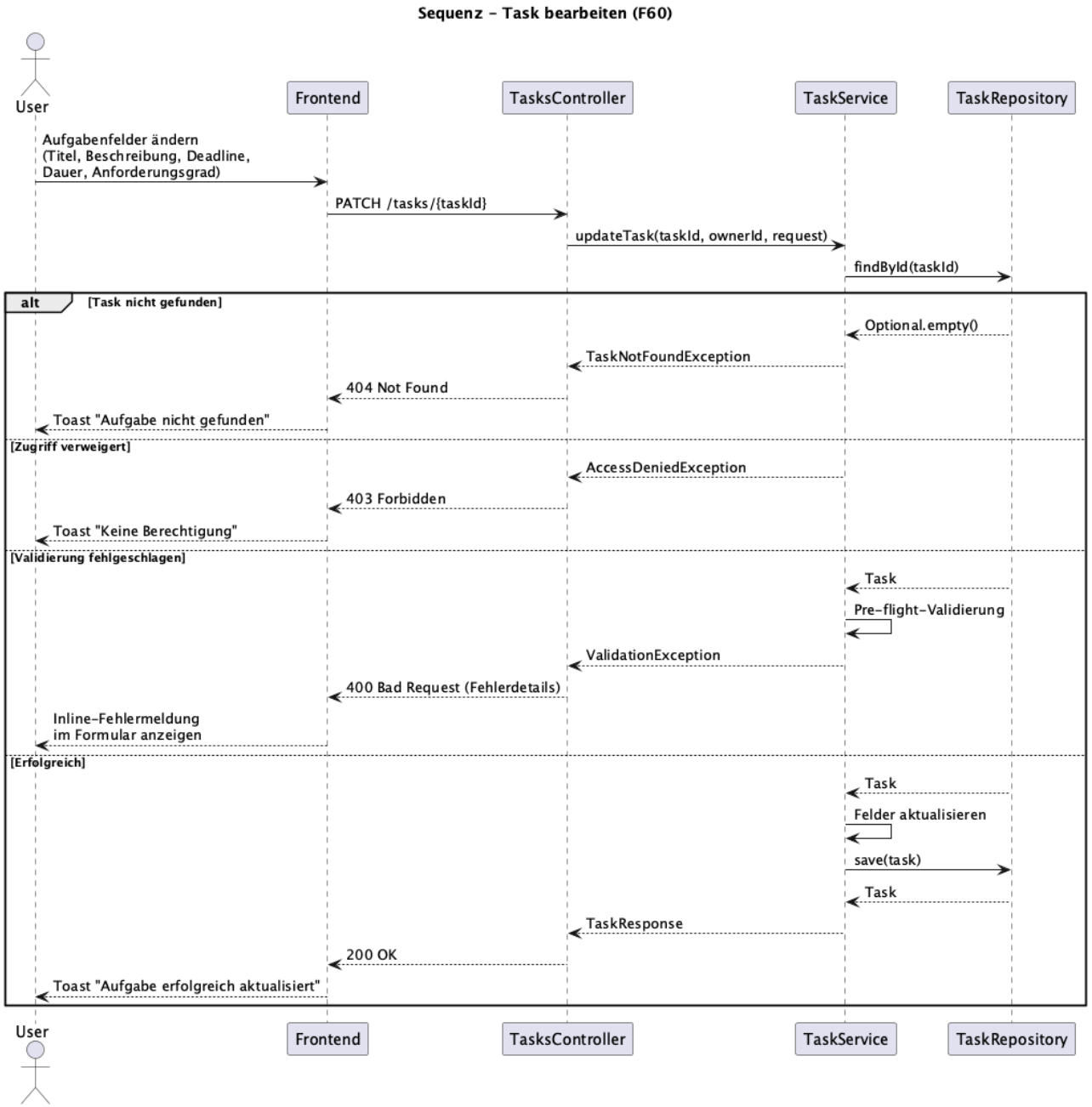
F50 - Task erstellen

Der Flow zeigt, wie eine neue Aufgabe über das Formular angelegt wird. Bevor etwas gespeichert wird, validiert der TaskService die Eingaben – bei Fehlern landet eine inline Meldung direkt im Formular (400), andernfalls wird die Aufgabe persistiert und mit 201 Created quittiert.



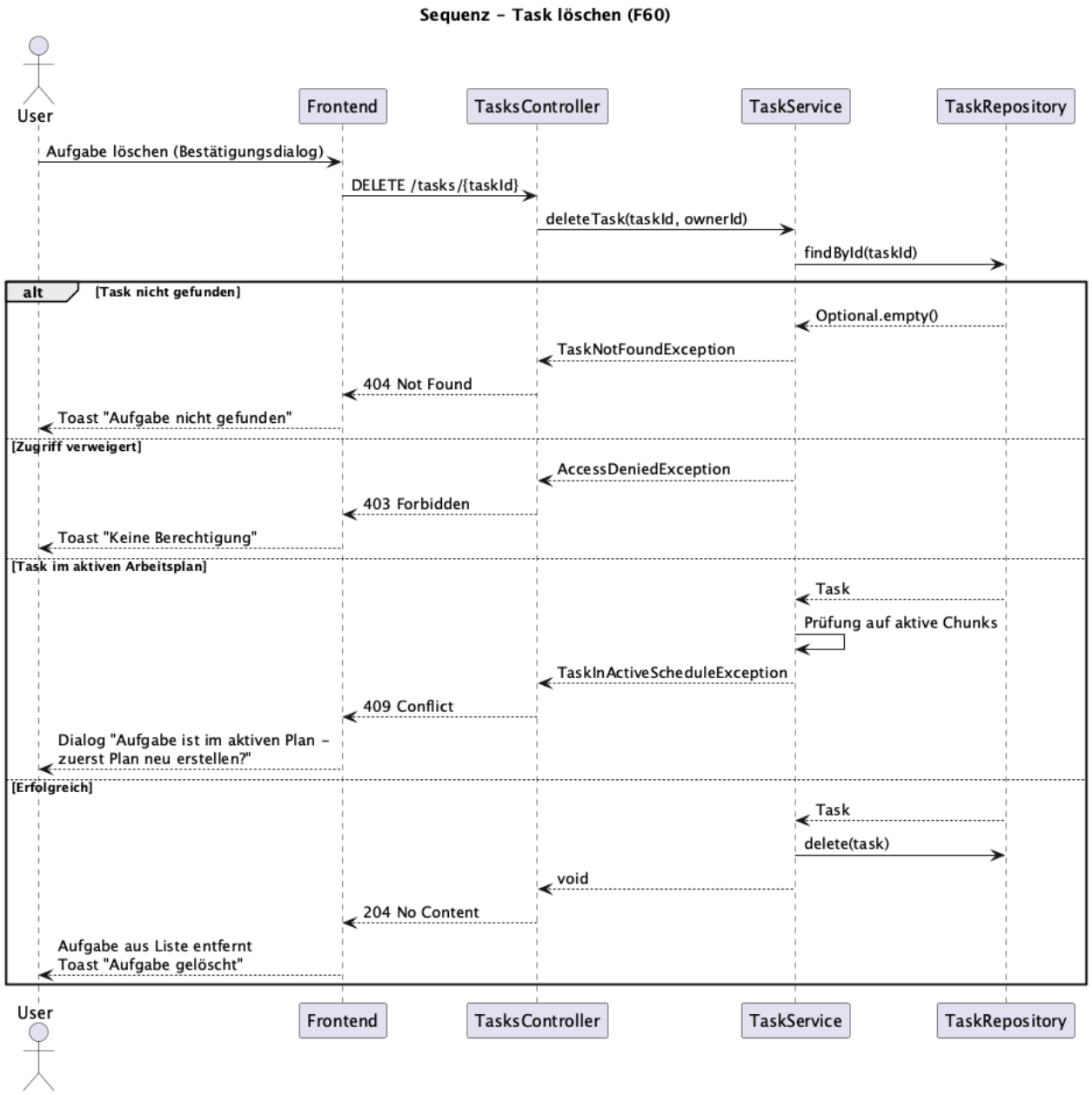
F60 - Task bearbeiten

Das Diagramm zeigt das Aktualisieren von Aufgabenfeldern (Titel, Beschreibung, Deadline, Dauer, Anforderungsgrad) via PATCH /tasks/{taskId}. Nach dem Lookup werden 404 Not Found, 403 Forbidden sowie 400 Bad Request bei Validierungsfehlern behandelt. Im Erfolgsfall persistiert der TaskService die Änderungen und antwortet mit 200 OK.



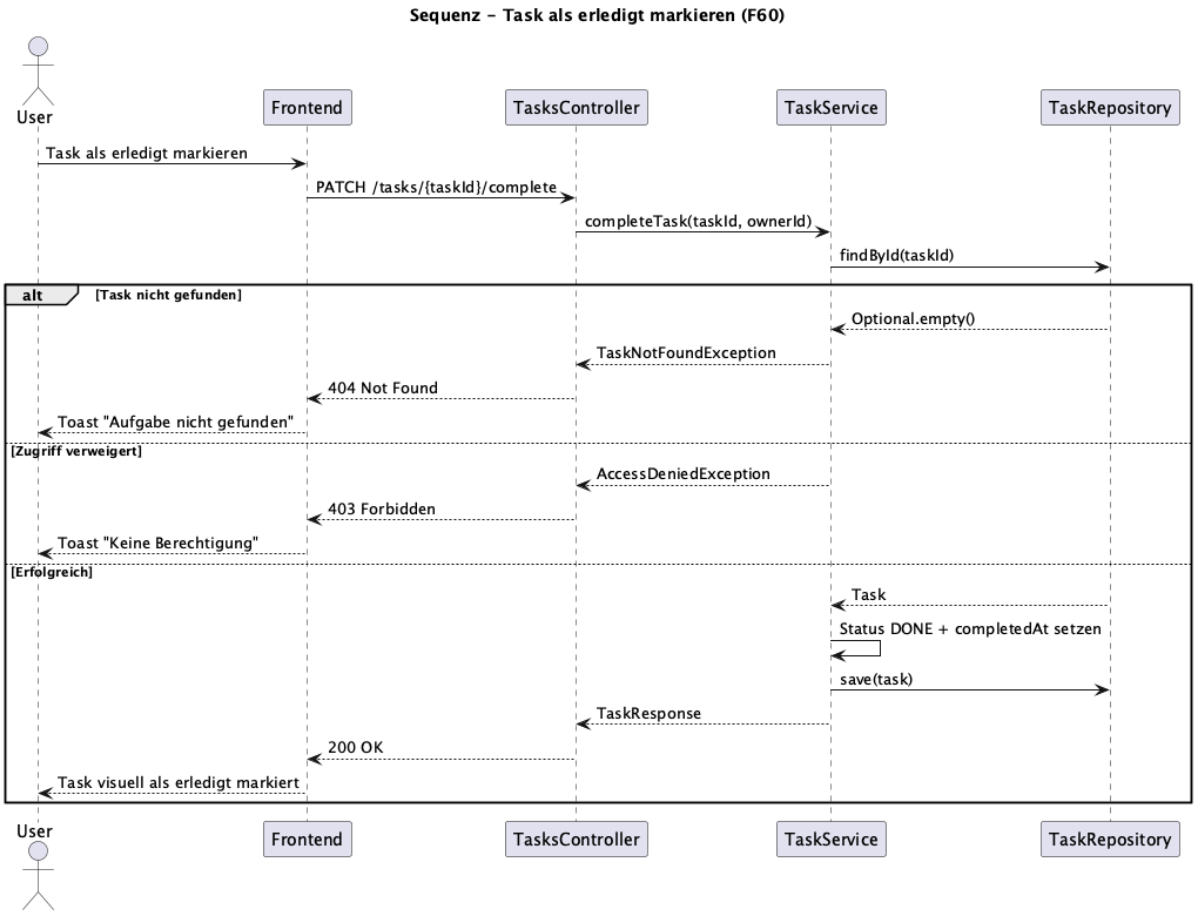
F60 - Task löschen

Zeigt das Löschen einer Aufgabe, das der Nutzer erst per Bestätigungsdialog auslöst. Interessant ist der Sonderfall: Ist die Aufgabe noch in einem aktiven Arbeitsplan verplant, blockiert der TaskService die Aktion mit 409 Conflict und informiert den Nutzer. Ansonsten folgt 204 No Content.



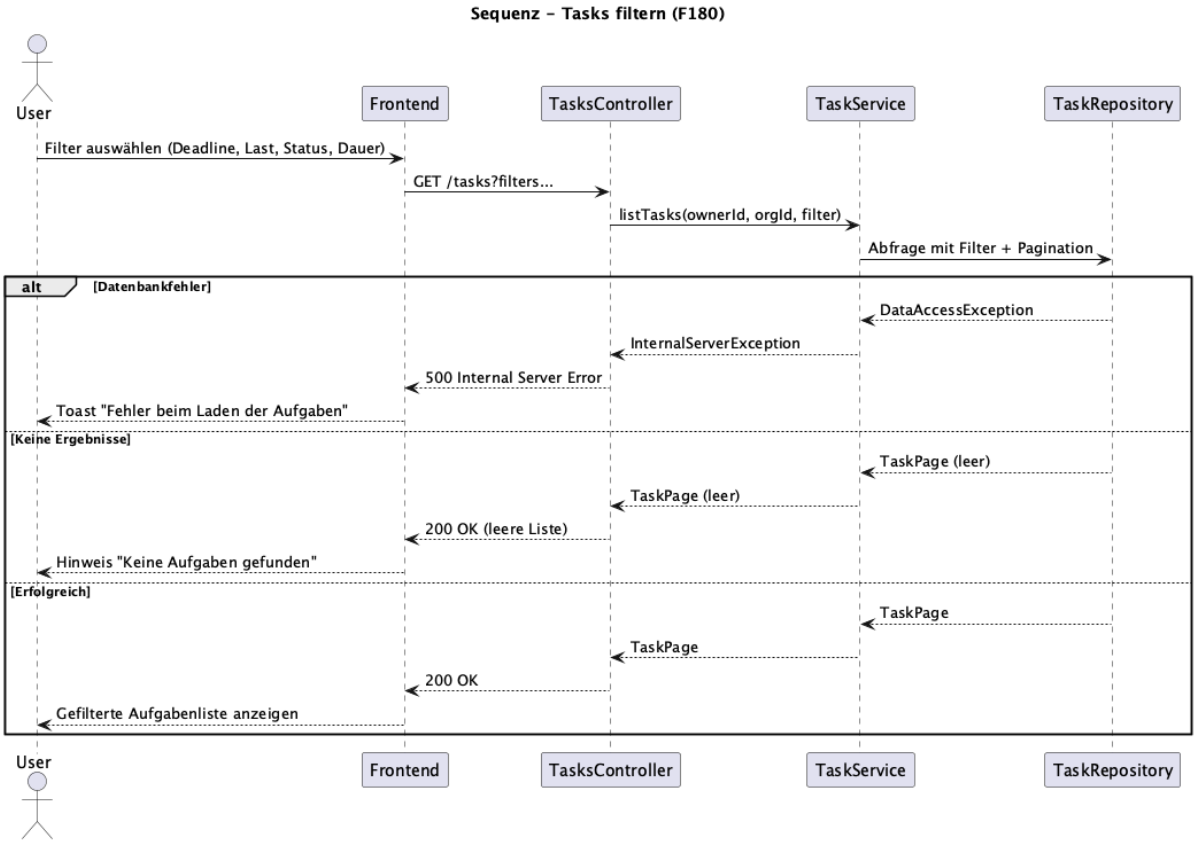
F60 - Task als erledigt markieren

Zeigt den Übergang OPEN → DONE. Der TaskService setzt den Status und schreibt completedAt– beides in einem Schritt. Standardfehlerpfade (404, 403) sind vorhanden, der Flow ist aber bewusst schlank gehalten.



F60 - Task filtern

Zeigt, wie Aufgaben nach Deadline, Last, Status und Dauer gefiltert und paginiert zurückgegeben werden. Neben dem Normalfall sind ein Datenbankfehler (500) und eine leere Treffermenge abgedeckt – letztere ist kein Fehler, sondern ein eigener gültiger Zustand mit entsprechendem Hinweis im Frontend.



F70 - Pausen verwalten

Zeigt die Verwaltung von Pausen, die der Nutzer über das Frontend anlegt. Es gibt hierbei mehrere Validierungsstufen: Zunächst werden Basisfelder geprüft, anschließend je nach Typ entweder einmalige oder wiederkehrende Pausenfelder, inklusive Abgleich mit den Arbeitszeiten und Überschneidungsprüfung bestehender Pausen. Bei Konflikten antwortet der BreakService mit 409 Conflict, andernfalls folgt 201 Created. Das Diagramm befindet sich aufgrund seiner Größe auch separat im Abgabeordner (UML-Diagramme).

5.1.6 Schedule-Generierung (F70, F80, F90, F150, F160)

Das Sequenzdiagramm *Schedule-Generierung* zeigt den fachlichen Kernablauf der Planerzeugung über POST /schedule/generate: Laden der relevanten Planungsdaten (Tasks, Breaks, Recurrence-Informationen, Working Hours), Anwenden von Hard- und Soft-Constraints sowie Persistierung neuer Schedule-Einträge.

Aufgrund der Größe des Diagramms zur Schedule-Generierung befindet sich dieses separat als Abbildung im Abgabeordner (Verzeichnis „UML-Diagramme“).