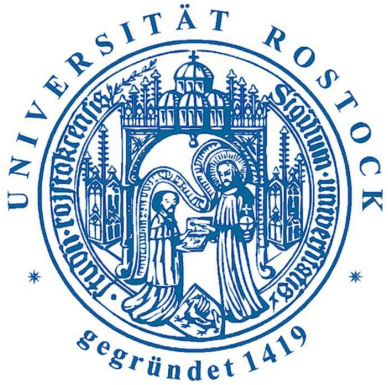

Softwaretechnik WS 2024-25
Projektabgabedokument Sprint 4

Übung-2 Team-1

Dogukan Karakoyun, 223202023
Hüseyin Kayabasi, 223201801
Helin Oguz, 223202103
Cagla Yesildogan, 223201881

1 Einleitung

1.1 Produktbacklog vor und nach dem Sprint

Zu Beginn des Sprints war ich verantwortlich für die Umsetzung der folgenden User Stories aus dem initialen Produktbacklog:

- **US-9.1**: Aufgaben nach Priorität filtern
- **US-9.2**: Aufgaben nach Status filtern
- **US-9.3**: Verantwortliche Person anzeigen
- **US-4.5**: Übersichtliche Darstellung mit Filter- und Sortieroptionen
- **US-2.3**: Aufgabenstatus setzen (z.B. offen, in Bearbeitung, fertig)
- **US-4.4**: Fälligkeitsdatum für User Stories verwalten

Im Laufe des Sprints wurden einige Anforderungen **verfeinert und ergänzt**, basierend auf der tatsächlichen Nutzung und UI-Rückmeldungen:

Änderungen und ihre Herkunft:

1. **UI Feedback für aktive Filter (US-9.1, 9.2, 4.5)**
   Ein Hinweisfeld oberhalb der Aufgabenliste wurde hinzugefügt, um die aktiven Filter (Priorität, Status, Sortierung) klar sichtbar zu machen. Diese Änderung basierte auf dem Wunsch der Benutzer nach besserer Nachvollziehbarkeit.

2. **Neue Sortierfunktion & Kombinierbarkeit (US-4.5)**
   Neben der Sortierung nach Priorität wurde auch eine Sortierung nach Erstellungsdatum (`createdAt`) ergänzt. Die Sortierung funktioniert nun vollständig in Kombination mit den Filtern.

3. **Modernisiertes Layout (US-4.5)**
   Das visuelle Layout der Aufgabenliste wurde mit Bootstrap-Komponenten überarbeitet, um die Übersichtlichkeit und Responsivität zu erhöhen.

4. **Detailansicht implementiert (US-9.3)**
   Jede Aufgabe enthält nun einen Link zur Detailansicht, damit zusätzliche Informationen angezeigt werden können. Diese Erweiterung wurde während des Sprints als zusätzliche Anforderung identifiziert.

5. **Ergänzung im Backlog: Einführung der Kalenderansicht für Tasks (US-4.6)**
   Ein neues Kalender-Modul wurde eingeführt, um eine visuelle Darstellung aller erstellten Aufgaben auf Basis ihres Erstellungsdatums zu ermöglichen. **Grund:** Bessere zeitliche Übersicht über Projektaufgaben und schnellere Identifikation von Aufgabenkonzentrationen in bestimmten Zeiträumen.

6. **Backlog-Erweiterung um Status und Fälligkeitsdatum (US-2.3, US-4.4)**
   Die Datenstruktur der User Story wurde erweitert, sodass auch Statuswerte und Fälligkeitsdaten erfasst, angezeigt und bearbeitet werden können. Dies umfasst Änderungen im Formular, zusätzliche Methoden im Service sowie eine Validierung im Backend.

Reflexion & kontinuierliche Verbesserung: Das Team hat im Sprint nicht nur die ursprünglichen Anforderungen umgesetzt, sondern auch aktiv über die **Benutzerfreundlichkeit und Flexibilität der Filterfunktionen** reflektiert. Mehrere kleine Verbesserungen wurden aus gezieltem Feedback heraus abgeleitet, ins Backlog aufgenommen und umgesetzt.Die Entscheidung, Status und Deadlines in das Backlog-Management zu integrieren, wurde aus dem Wunsch heraus getroffen, den Fortschritt von Aufgaben besser verfolgen und zeitliche Engpässe frühzeitig erkennen zu können.

1.2 Sprint-Planung

Im Sprint 4 waren wir für die Umsetzung folgender User Stories und technischer Aufgaben verantwortlich:

- **US-9.1:** Filterung nach Priorität
  - UI: Dropdown zur Auswahl der Priorität
  - Backend: Filterung in der Controller-Logik
  - UI: Aktive Filteranzeige
- **US-9.2:** Filterung nach Status
  - UI: Dropdown zur Auswahl des Status
  - Backend: Kombinierte Filterung mit Priorität
- **US-9.3:** Anzeige des `assignedUser`
  - Anzeige in der Aufgabenliste inkl. Tooltip (Name, E-Mail)
  - Anzeige in der Detailansicht
- **US-4.5:** Visuelle und technische Überarbeitung
  - Responsive UI mit Bootstrap
  - Sortierung nach `priority` und `createdAt`
  - Dynamische Anzeige der Sortieroptionen je nach Auswahl
  - Reset-Button zur Rücksetzung aller Filter
  - UI: Darstellung aller aktiven Filter (inkl. Sortierung)
- **US-4.6:** Visuelle Kalenderansicht für Tasks
  - FullCalendar-Integration im Frontend zur Darstellung von Aufgaben.
  - Erstellung eines /calendar-Endpoints (Thymeleaf-basiert) zur Anzeige des Kalenders.
  - REST-Endpoint /api/calendar, welcher alle Aufgaben inklusive Erstellungsdatum als JSON zurückgibt.
  - Nutzung des Erstellungsdatums (createdAt) als Startzeit für Kalendereinträge.
  - Mapping von Task-Entitäten zu FullCalendar-Events im Controller.
  - Erweiterung des Services zur Transformation von Task-Daten.
- **US-2.3**: Status setzen für User Stories
  - UI: Dropdown zur Auswahl des Status (offen, in Bearbeitung, fertig)
  - Backend: Verarbeitung des neuen Statusfelds im Update-Prozess
- **US-4.4**: Fälligkeitsdatum hinzufügen für User Stories
  - UI: Eingabefeld für das Fälligkeitsdatum im Story-Formular
  - Backend: Speicherung und Validierung des Datums im Service

  US-4.6: "Als Teammitglied möchte ich eine visuelle Kalenderansicht aller Aufgaben mit ihren Erstellungszeitpunkten sehen können, damit ich den zeitlichen Verlauf und die Dichte der Aufgaben im Projekt besser einschätzen kann."
- **Zusätzliche Leistungen (nicht geplant):**
  - Implementierung der Detailansicht für Aufgaben (`/tasks/{id}`)
  - Integration eines "Details"-Buttons in der Aufgabenübersicht

Aufwandsschätzung

Für die Sprint-Planung wurde die Story-Point-Schätzungsmethode verwendet. Jedes Teammitglied hat eine Schätzung abgegeben. Falls es große Unterschiede gab, musste das höchste geschätzte Teammitglied eine Begründung liefern.

User Story	Task	Dogukan	Helin	Hüseyin	Cagla	Ø Punkte	Begründung
US-2.3	Status setzen (User Story erweitern)	3	3	3	2	2.75	Kleine Erweiterung der bestehenden Update-Methode
US-4.4	Fälligkeitsdatum hinzufügen (User Story)	4	3	4	3	3.5	Ähnlich zu Status-Update, aber mit Validierung
US-9.1	Aufgaben nach Priorität filtern (Dropdown + Backend-Filter)	5	5	6	5	5.25	Kombination mit anderen Filtern notwendig
US-9.2	Aufgaben nach Status filtern (Dropdown + Backend-Filter)	4	5	4	5	4.5	Technisch ähnlich zu US-9.1, aber weniger Komplexität
US-9.1/9.2	Aktive Filter UI-Feedback (Anzeige im Frontend)	3	3	3	3	3	UI only, wenig Backend-Logik
US-4.5	Sortierung + Filterung kombinieren (Backend + Layout)	7	6	7	6	6.5	Komplexe Kombination im Controller/Service
US-4.5	Reset-Button + visuelles Redesign (Bootstrap, Layout)	4	5	4	5	4.5	Visuelle Integration und State-Reset
US-9.3	Zuständiger Nutzer anzeigen (Badge + Detailansicht)	5	5	5	5	5	Datenanreicherung + Darstellung
US-4.6	Aufgaben im Kalender visuell darstellen (FullCalendar + JSON API + Template)	6	6	7	6	6,25	Neue Integration in bestehendes Layout, Mapping von Datenformaten, API-Anbindung und visuelle Anzeige mit Kalender-Plugin

Table 1: Aufwandsschätzung (Story Points) für Sprint 4

2.1 Klassendiagramm

Backlog Management [US-2.1, US-2.2, US-2.3, US-4.4]

In diesem Abschnitt wird die verfeinerte Struktur für die User Stories US-2.1 (User Stories erstellen), US-2.2 (User Stories priorisieren), US-2.3 (Status setzen) und US-4.4 (Fälligkeitsdatum setzen) modelliert. Die Modellierung erfolgt mit einem Klassendiagramm, das die wichtigsten Komponenten, Attribute, Methoden und Relationen enthält. Zusätzlich werden die Verantwortlichkeiten der Klassen erläutert sowie die Erweiterbarkeit der Architektur hervorgehoben. Die im Diagramm dargestellte Klasse `UserStoryForm` wurde ursprünglich konzipiert, um eine saubere Trennung zwischen Formulardaten und Datenmodell zu ermöglichen. In der finalen Implementierung wurde jedoch auf ein separates DTO verzichtet, und die Werte werden direkt an den Controller übergeben. Das Klassendiagramm spiegelt somit auch die frühen Designentscheidungen im Projektverlauf wider.

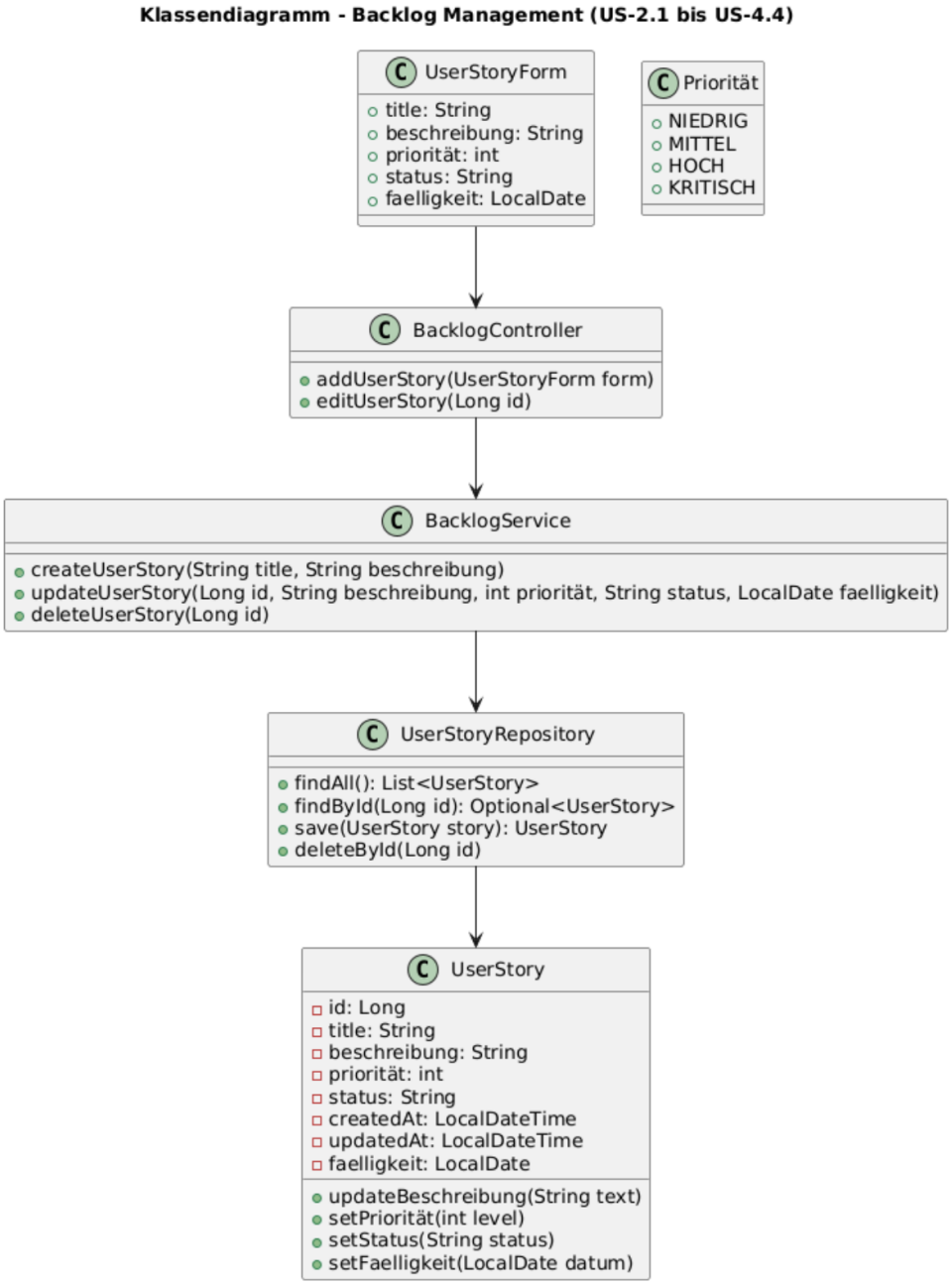

Figure 1: Klassendiagramm zu US-2.1, US-2.2, US-2.3, US-4.4 – Backlog Management

Modellierte Klassen und Verantwortlichkeiten

- **BacklogController**: Entgegennahme von Benutzeranfragen zur Erstellung und Bearbeitung von User Stories sowie Weiterleitung an den Service.
- **BacklogService**: Verwaltet die Geschäftslogik zur Erstellung, Bearbeitung und Löschung von User Stories. Nutzt dazu das Repository. Beinhaltet nun auch Methoden zum Setzen von Status und Fälligkeitsdatum.
- **UserStoryRepository**: Schnittstelle zur Datenbank für das Speichern, Finden und Löschen von User Stories.
- **UserStory**: Modelliert eine einzelne User Story mit Attributen wie `id`, `title`, `beschreibung`, `priorität`, `status`, `createdAt`, `updatedAt` und `faelligkeit`. Enthält Methoden zur Bearbeitung dieser Attribute.
- **Priorität (Enum)**: Definiert die Prioritätsstufen: `NIEDRIG`, `MITTEL`, `HOCH`, `KRITISCH`.
- **UserStoryForm**: Datenträgerklasse zur Formularübertragung zwischen UI und Backend. Enthält Felder für alle notwendigen Eingaben.

Relevante User Stories:

US-9.1, US-9.2, US-9.3, US-4.5

Das folgende Klassendiagramm zeigt die zentralen Entitäten und deren Beziehungen, die im Rahmen des **Sprint 4** umgesetzt wurden. Der Fokus liegt auf der Implementierung von Filter- und Sortierfunktionen, der Anzeige verantwortlicher Benutzer sowie der detaillierten Aufgabenansicht mit Dateiunterstützung.

Ist in Ordnung so, aber es wäre besser, wenn ihr hier gleich an das Pair Programming gedacht hättet, also dass man wenigstens 2 Nutzer zuordnen kann

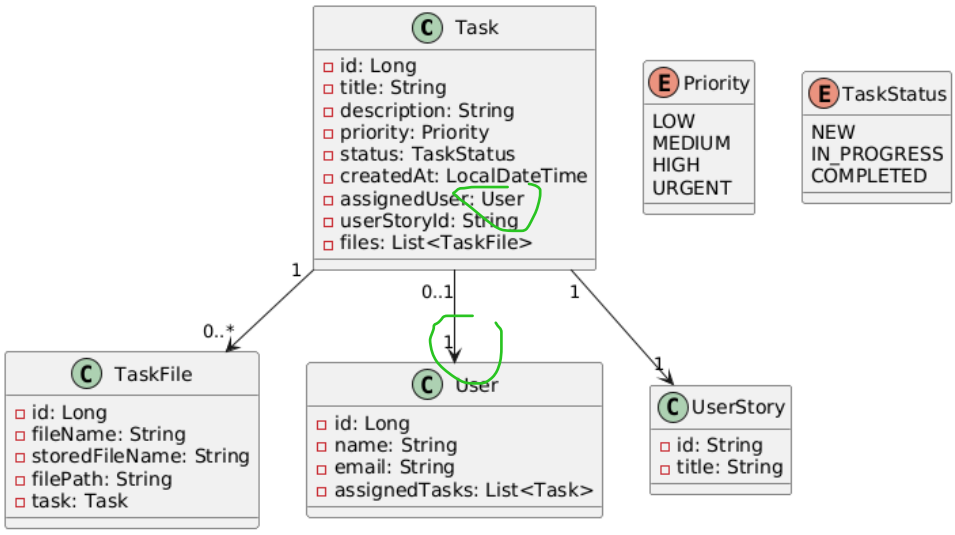

Figure 2: Klassendiagramm der im Sprint 4 erweiterten Komponenten

Modellierte Klassen und Verantwortlichkeiten

- **Task**: Diese Klasse stellt eine Aufgabe dar, inklusive Attributen wie Titel, Beschreibung, Priorität, Status, Erstellungszeitpunkt sowie Relationen zu Benutzern und Dateien. Relevante Erweiterungen in Sprint 4:
  - Filterung und Sortierung nach `priority` und `createdAt`.
  - Zuordnung von Aufgaben zu einem verantwortlichen Benutzer über `assignedUser`.
  - Verlinkung zu einer User Story über `userStoryId`.
  - Unterstützung mehrerer zugehöriger Dateien via `files`.

Hat sich hier nicht eigentlich aber was im TaskService geändert?

Weiter unten gibt es dort eine Methode getFilteredTasks(priority, status).Ist diese nicht neu dazu gekommen? Dann zeigt das doch lieber hier, als die Klassen, die sowieso gleich geblieben sind.

- **User**: Repräsentiert eine für Aufgaben verantwortliche Person. Im UI wird der Name angezeigt, bei Hover zusätzlich die E-Mail. Aufgaben können Benutzern zugewiesen und über das Frontend geändert werden.
- **UserStory**: Dient zur Kategorisierung von Aufgaben. Aufgaben können über das Dropdown einer User Story zugeordnet werden (Pflichtfeld im Formular).
- **TaskFile**: Beschreibt eine Datei, die mit einer Aufgabe verknüpft ist. In Sprint 4 wurde das Attribut `storedFileName` eingeführt, um Uploads mit eindeutigem Namen zu speichern und dennoch den Originalnamen beim Download anzuzeigen.

Beziehungen und Struktur

- Eine `Task` ist optional genau einem `User` zugewiesen.
- Eine `Task` kann genau einer `UserStory` zugeordnet sein.
- Eine `Task` kann mehrere `TaskFile`-Objekte enthalten.

Dieses Modell bildet die Grundlage für die in Sprint 4 erfolgreich implementierten Funktionen, darunter kombinierte Filterung, Benutzeranzeige, Dateiuploads sowie die verbesserte Aufgabenübersicht und Detailansicht.

Erweiterungen gegenüber dem ursprünglichen Entwurf

Im Vergleich zum ursprünglichen Klassendiagramm wurden in Sprint 4 keine neuen Klassen oder Beziehungen eingeführt. Jedoch erfolgten mehrere bedeutende funktionale Erweiterungen, die auf dem bestehenden Modell basieren:

- **Dynamische Filter- und Sortierfunktionen**: Die Attribute `priority`, `status` und `createdAt` wurden im Frontend gezielt zur Kombination von Filter- und Sortierlogiken genutzt. Diese Logik baut direkt auf den bereits vorhandenen Attributen des `Task`-Modells auf.
- **Direkte Benutzerzuweisung per Dropdown**: Die `assignedUser`-Beziehung wurde durch eine neue Interaktionsmöglichkeit im UI erweitert. Der verantwortliche Benutzer kann jetzt direkt aus der Aufgabenliste geändert werden.
- **Dateiverwaltung über die Aufgabenübersicht**: Die bereits existierende Beziehung zu `TaskFile` wurde funktional erweitert. Dateien können nun über ein Upload-Formular in der Aufgabenliste hinzugefügt und über einen individuellen Link heruntergeladen werden.
- **Detailansicht**: Für jede Aufgabe wurde eine eigene Detailansicht implementiert, die alle zugehörigen Informationen (inkl. Dateien und zuständige Person) übersichtlich darstellt.

Diese Erweiterungen zeigen, dass das ursprüngliche Modell modular aufgebaut war und sich gut erweitern ließ, ohne die bestehende Struktur anzupassen.

Kalendar [US-4.6]

Das folgende Klassendiagramm zeigt die modellierten Klassen und ihre Beziehungen im Rahmen der Umsetzung der **US-4.6** (visuelle Kalenderansicht der Tasks). Es veranschaulicht die technischen Komponenten, die benötigt werden, um Aufgaben mit Erstellungszeitpunkten im Kalender darzustellen.

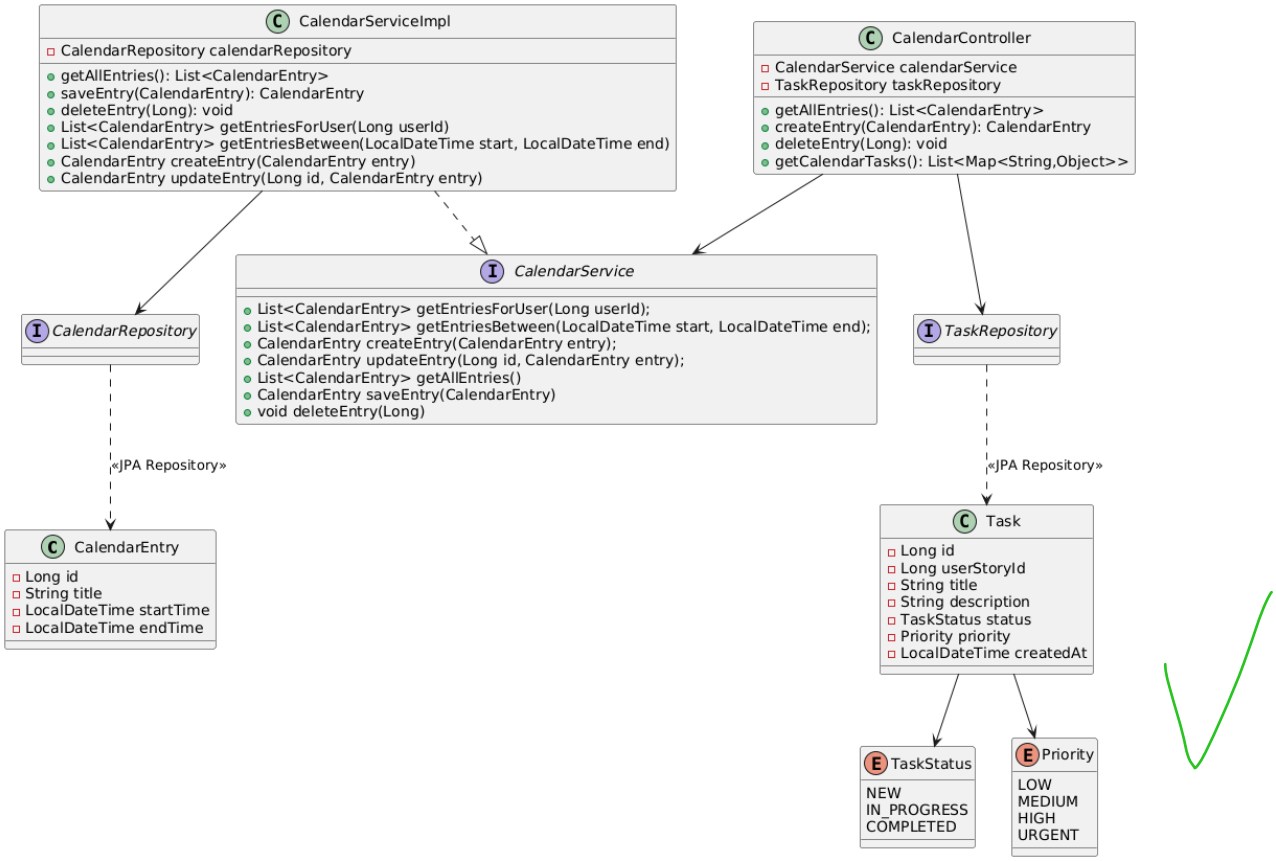

Figure 3: Klassendiagramm zur Umsetzung der Kalenderfunktionalität (US-4.6)

Modellierte Klassen und Verantwortlichkeiten

- **CalendarEntry**: Repräsentiert einen einzelnen Kalendereintrag mit Titel, Start- und Endzeit. Dient als Modellklasse zur Speicherung und Anzeige von Ereignissen im Kalender.
- **CalendarService & CalendarServiceImpl**: Enthält die Geschäftslogik zur Verwaltung der Kalendereinträge. Beinhaltet Methoden wie `getAllEntries()`, `saveEntry()` und `deleteEntry()`.
- **CalendarRepository**: JPA-Interface zur Datenpersistierung der Kalendereinträge.
- **CalendarController**: Verarbeitet alle HTTP-Anfragen bezüglich der Kalenderfunktion. Stellt unter anderem die REST-Schnittstelle `/api/calendar` zur Verfügung, um Aufgaben im Kalender anzuzeigen.
- **Task & TaskRepository**: Die Aufgaben werden hier aus der bestehenden Task-Entität bezogen. Es wird das Attribut `createdAt` als zeitliche Referenz für Kalendereinträge verwendet.
- **TaskStatus & Priority (Enum)**: Status und Priorität der Aufgaben werden ggf. zur visuellen Kodierung (z.B. Farbgebung) verwendet.

User Story – Interface Zuordnung

Modellierung verfeinerter Interfaces und Datenstrukturen

Verfeinerung der Interfaces zwischen den Komponenten

Um die Kommunikation zwischen den Komponenten effizient und erweiterbar zu gestalten, wurden die folgenden Interfaces und Datenstrukturen modelliert und mit neuen Methoden versehen.

UserStoryRepository

```
public interface UserStoryRepository extends JpaRepository<UserStory, Long> {
    List<UserStory> findAll();
    Optional<UserStory> findById(Long id);
    UserStory save(UserStory story);
    void deleteById(Long id);
}
```

Das Interface `UserStoryRepository` stellt Standardmethoden zur Verfügung, um User Stories in der Datenbank zu speichern, zu finden oder zu löschen. Es basiert auf Spring Data JPA.

BacklogService

```
public class BacklogService {
    List<UserStory> getAllUserStories();
    UserStory createUserStory(String title, String beschreibung);
    void updateUserStory(Long id, String beschreibung, int priorit t, String status,
        LocalDate faelligkeit);
    void deleteUserStory(Long id);
}
```

Die Klasse `BacklogService` enthält die Geschäftslogik zur Verwaltung der User Stories. In Sprint 4 wurde sie erweitert, um `Status` und `Fälligkeitsdatum` verarbeiten zu können.

UserStoryForm

```
public class UserStoryForm {
    String title;
    String beschreibung;
    int priorit t;
    String status;
    LocalDate faelligkeit;
}
```

Die Klasse `UserStoryForm` dient als DTO für den sicheren Datentransfer zwischen UI und Controller. Sie schützt die Entität `UserStory` vor direktem Zugriff.

Priorität (Enum)

```
public enum Priorit t {
    NIEDRIG, MITTEL, HOCH, KRITISCH
}
```

Das Enum `Priorität` definiert feste Prioritätsstufen und erlaubt eine konsistente Sortierung und Darstellung im UI.

Status (String)

```
// Beispiele: "offen", "in Bearbeitung", "fertig"
```

Das Feld `status` wird als String gespeichert und erlaubt die Zustandsverwaltung von User Stories. Eine spätere Umstellung auf ein Enum ist möglich.

- Definiert feste Werte für die Priorität einer User Story.
- Unterstützt strukturierte Filterung und UI-Konsistenz.

Status (String)

- Wird als freier String gespeichert.
- Unterstützt typische Statuswerte wie `"offen"`, `"in Bearbeitung"`, `"fertig"`.
- Kann optional in Zukunft durch ein Enum ersetzt werden.

Verantwortlichkeiten der Interfaces

Die in den User Stories US-2.1 bis US-4.4 eingesetzten Interfaces und Klassen sichern eine klare Trennung der Verantwortlichkeiten:

- **UserStoryRepository:** Datenbankschnittstelle mit Standardmethoden (CRUD), implementiert durch Spring Data JPA.
- **BacklogService:** Geschäftslogik zur Erstellung, Bearbeitung und Speicherung von User Stories. Umfasst Erweiterungen wie Status und Deadline-Verwaltung.
- **UserStoryForm:** DTO zur sicheren Datenübergabe vom UI an den Controller, ohne direkten Zugriff auf die Entity-Klasse.
- **Priorität (Enum):** Definiert erlaubte Werte für Prioritätsstufen; verhindert Fehler durch freie Eingaben.
- **Status (String):** Flexibles Attribut zur Verfolgung des Arbeitsfortschritts. Kann zur Validierung erweitert werden.

Relevante User Stories:

US-9.1, US-9.2, US-9.3, US-4.5

Im Rahmen der Umsetzung dieser User Stories wurden die bestehenden Interfaces angepasst und erweitert, um Aufgaben nach Priorität und Status zu filtern, die zuständige Person anzuzeigen sowie Sortierfunktionen zu ermöglichen. Darüber hinaus wurden die Datenstrukturen für die Aufgabenansicht und die Detailanzeige verfeinert.

TaskController

```
@GetMapping("/tasks")
public String showTaskPage(
    @RequestParam(required = false) Priority priority,
    @RequestParam(required = false) TaskStatus status,
    @RequestParam(required = false) String sort,
    @RequestParam(required = false, defaultValue = "asc") String order,
    Model model) {

    List<Task> tasks = taskService.getFilteredTasks(priority, status, sort, order);

    model.addAttribute("tasks", tasks);
    model.addAttribute("selectedPriority", priority);
    model.addAttribute("selectedStatus", status);
    model.addAttribute("selectedSort", sort);
    model.addAttribute("selectedOrder", order);

    return "tasks";
}
```

Dieses Interface übernimmt die zentrale Steuerung der Aufgabenanzeige. Es ermöglicht die dynamische Filterung und Sortierung auf Basis der angegebenen Parameter. Die Daten werden im ‘Model‘ zur Anzeige an das Frontend übergeben.

TaskService

```
public List<Task> getFilteredTasks(Priority priority, TaskStatus status, String sort,
    String order) {
    List<Task> tasks = taskRepository.findAll();

    if (priority != null) {
        tasks = tasks.stream()
                .filter(t -> t.getPriority() == priority)
                .collect(Collectors.toList());
    }

    if (status != null) {
        tasks = tasks.stream()
                .filter(t -> t.getStatus() == status)
                .collect(Collectors.toList());
    }

    if ("priority".equals(sort)) {
        tasks.sort(Comparator.comparing(Task::getPriority));
    } else if ("createdAt".equals(sort)) {
        tasks.sort(Comparator.comparing(Task::getCreatedAt));
    }

    if ("desc".equalsIgnoreCase(order)) {
        Collections.reverse(tasks);
    }

    return tasks;
}
```

Diese Methode bildet das logische Rückgrat der Aufgabenfilterung im System. Sie kombiniert mehrere Filteroptionen und wendet dynamische Sortierung an, basierend auf der Benutzereingabe.

Task

```
public class Task {
    private Long id;
    private String title;
    private String description;
    private Priority priority;
    private TaskStatus status;
    private LocalDateTime createdAt;
    private Long userStoryId;
    private User assignedUser;
}
```

Die Klasse `Task` wurde im Sprint 4 nicht strukturell verändert, jedoch wurden bestimmte Felder im Frontend für Filterung, Sortierung und Anzeige stärker eingebunden.

Priority

```
public enum Priority {
    LOW, MEDIUM, HIGH, URGENT
}
```

TaskStatus

```
public enum TaskStatus {
    NEW, IN_PROGRESS, COMPLETED
}
```

US-4.6

CalendarController

```
@GetMapping("/api/calendar")
public List<Map<String, Object>> getCalendarTasks() {
    List<Map<String, Object>> events = new ArrayList<>();
    List<Task> tasks = taskRepository.findAll();

    for (Task task : tasks) {
        Map<String, Object> event = new HashMap<>();
        event.put("title", task.getTitle());
        event.put("start", task.getCreatedAt());
        events.add(event);
    }

    return events;
}
```

Diese Methode stellt eine REST-Schnittstelle zur Verfügung, die alle im System erfassten Aufgaben mit ihrem Erstellungsdatum in ein für FullCalendar kompatibles JSON-Format umwandelt. Dadurch kann eine visuelle Kalenderansicht generiert werden. Die zugrunde liegende Datenstruktur basiert auf einer Transformation von Task-Objekten zu generischen Key-Value-Paaren.

CalendarServiceImpl

```
@Override
public List<CalendarEntry> getAllEntries() {
    return calendarRepository.findAll();
}

@Override
public CalendarEntry saveEntry(CalendarEntry entry) {
    return calendarRepository.save(entry);
}

@Override
public void deleteEntry(Long id) {
    calendarRepository.deleteById(id);
}
```

Diese Methoden bilden die Geschäftslogik zur Verwaltung der Kalendereinträge ab. Sie ermöglichen das Abrufen, Speichern und Löschen von Einträgen über das zugehörige Repository. Die Methoden werden vom Controller genutzt, um die REST-Endpunkte mit den gespeicherten Kalendereinträgen zu versorgen.

CalendarEntry

```
@Entity
public class CalendarEntry {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private LocalDateTime startTime;
    private LocalDateTime endTime;
}
```

Die Entität `CalendarEntry` dient als Datenmodell für Ereignisse im Kalender. Sie speichert den Titel sowie Start- und Endzeitpunkt eines Eintrags. Diese Struktur wird direkt in der Datenbank persistiert und über das Service-Schicht verwaltet.

Sequenzdiagramm

US-2.1, US-2.2, US-2.3 und US-4.4 – Backlog Management

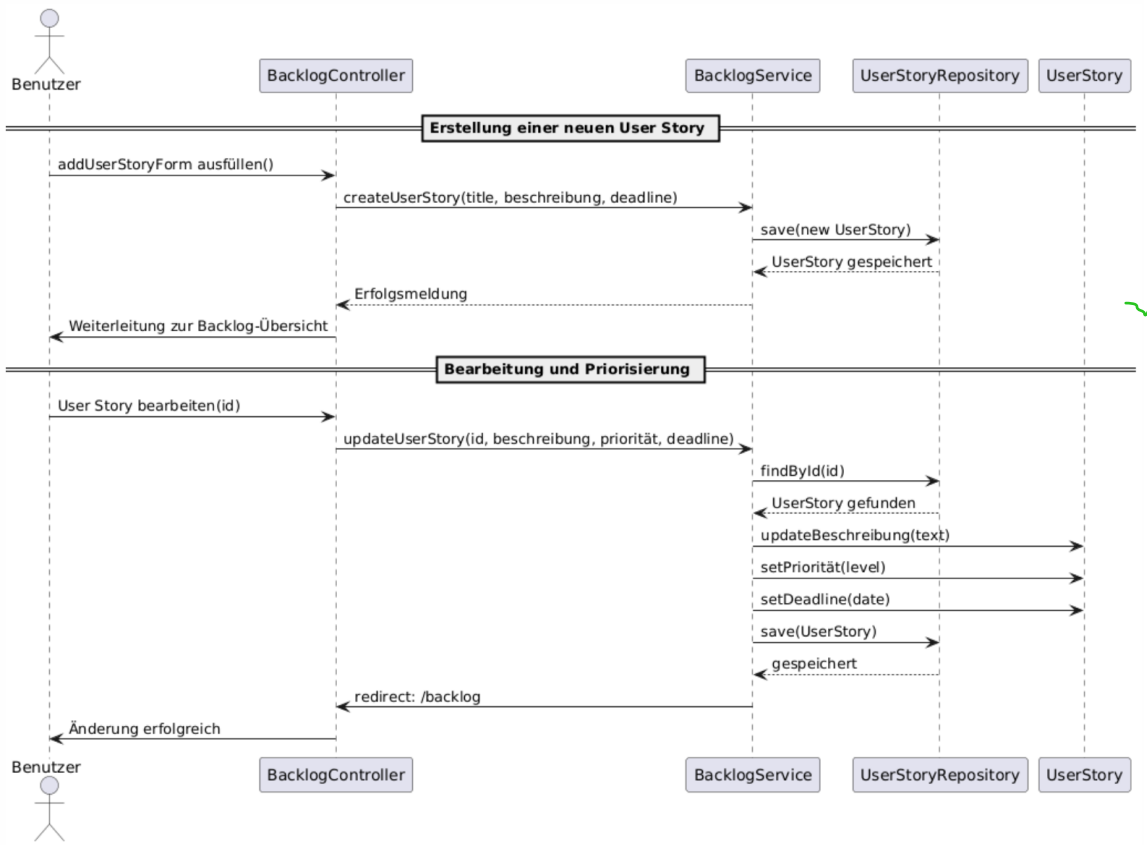

Figure 4: Sequenzdiagramm zur Erstellung, Bearbeitung und Erweiterung von User Stories (US-2.1, US-2.2, US-2.3, US-4.4)

Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde erstellt, um die Benutzerinteraktionen beim Erstellen und Bearbeiten einer User Story im Kontext von **US-2.1, US-2.2, US-2.3 und US-4.4** darzustellen. Es zeigt die Interaktion zwischen dem Benutzer und den Backlog-Komponenten des Systems und veranschaulicht, wie die Anfrage zur Erstellung verarbeitet, gespeichert und anschließend erweitert wird.

Das Diagramm verdeutlicht die saubere Trennung der Verantwortlichkeiten zwischen *Controller*, *Service* und *Repository*. Durch die strukturierte Darstellung wird sichtbar, wie der Benutzer gezielt mit dem System interagiert, ohne direkte Verbindung zur Datenhaltung.

Das kombinierte Sequenzdiagramm deckt nicht nur die Erstellung (US-2.1) und Priorisierung (US-2.2) ab, sondern auch die **Statuspflege** (US-2.3) und die **Verwaltung des Fälligkeitsdatums** (US-4.4). Diese neuen Funktionen werden in den Methoden `setStatus(status)` und `setFaelligkeit(datum)` umgesetzt und erweitern das System um wichtige Planungsfunktionen.

Die Visualisierung zeigt, wie neue Anforderungen systematisch in bestehende Prozesse integriert werden können, ohne die bestehende Architektur zu verletzen. Gleichzeitig macht sie den Datenfluss zwischen den Schichten transparent und nachvollziehbar.

US-4.6 - Kalendar

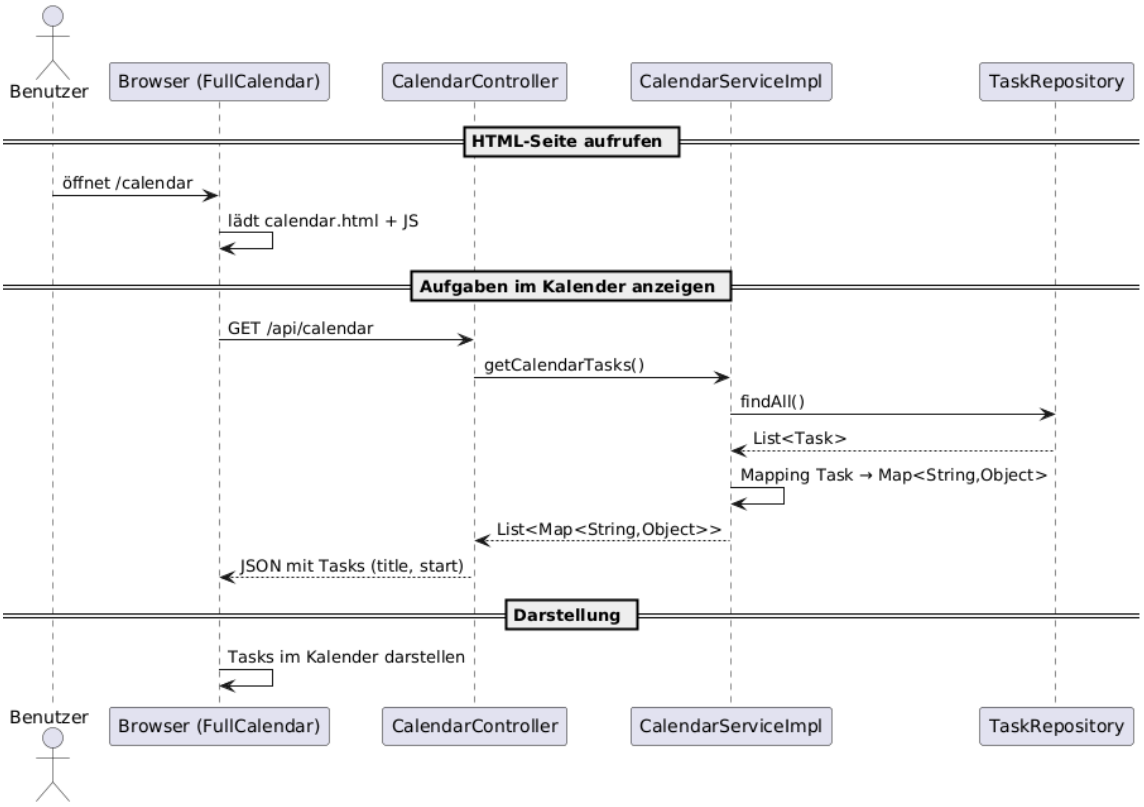

Figure 5: Sequenzdiagramm zur Anzeige der Aufgaben im Kalender (US-4.6)

Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde erstellt, um die Benutzerinteraktion beim Aufrufen der Kalenderansicht sowie den Datenfluss zur Darstellung aller Aufgaben im Kontext der US-4.6 zu veranschaulichen. Es zeigt die zentrale Interaktion zwischen Benutzer, Frontend (FullCalendar), Controller, Service und Repository. Der Ablauf beginnt mit dem Aufruf der Kalenderseite, woraufhin FullCalendar automatisch eine GET-Anfrage an die REST-Schnittstelle `/api/calendar` sendet. Diese wird im Controller entgegengenommen und durch den Service verarbeitet, indem die gespeicherten Tasks über das Repository geladen und in ein JSON-kompatibles Format transformiert werden. Der Benutzer erhält dadurch eine visuelle Darstellung aller Aufgaben im Kalender – basierend auf ihrem Erstellungsdatum.

Das Diagramm illustriert somit die logische Trennung der Schichten (Controller, Service, Repository) und macht deutlich, wie bestehende Datenmodelle ohne strukturelle Änderungen für eine neue, visuelle Funktionalität wiederverwendet werden können.

US-9.1, US-9.2, US-9.3, US-4.5 - Aufgabenfilterung und Detailanzeige

Das folgende Sequenzdiagramm illustriert die wichtigsten Interaktionen zwischen dem Benutzer, dem System und der Datenbank im Rahmen des Sprint 4. Im Fokus stehen die Anforderungen zur Aufgabenfilterung, Anzeige aktiver Filter, Rücksetzung der Filter sowie die Detailansicht einzelner Aufgaben.

Die Filterparameter (z. B. `priority`, `status`, `sort`, `order`) werden über die URL an den Controller übergeben. Dieser ruft im Service entsprechende Logik zur Filterung und Sortierung auf, welche dann die Aufgaben über das Repository abruft. Anschließend werden die gefilterten Aufgaben mit visuellem UI-Feedback dargestellt. Bei einem Klick auf das Detail-Icon wird eine spezifische Aufgabe basierend auf ihrer ID geladen und angezeigt.

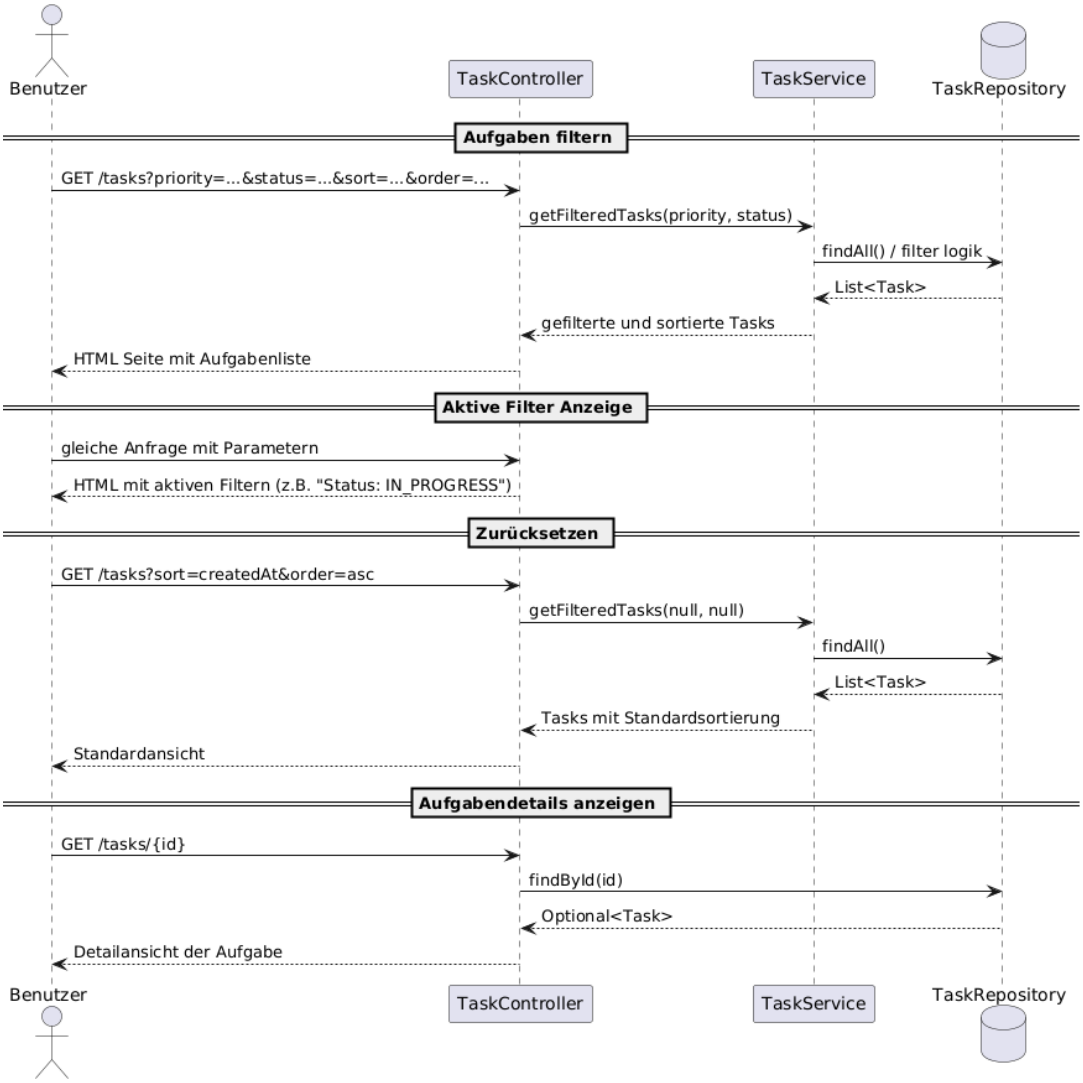

Figure 6: Sequenzdiagramm der implementierten Funktionen im Sprint 4

2.2 Tracing der User Stories und relevante Anforderungen

Table 2: Tracing der User Stories und relevanter Klassen in Sprint 4

Klasse	**Verantwortlichkeit / Funktion**	**Relevante User Story**
BacklogController	Entgegennahme und Weiterleitung von UserStory-Formulardaten, inkl. Status und Fälligkeitsdatum	US-2.3, US-4.4
BacklogService	Verarbeitung von Beschreibung, Priorität, Status und Fälligkeitsdatum bei Erstellung/Bearbeitung der Story	US-2.3, US-4.4
UserStory	Repräsentiert eine User Story mit neuen Attributen: Status, Fälligkeitsdatum	US-2.3, US-4.4
Task	Speicherung von Titel, Beschreibung, Priorität, Status, Erstellungsdatum und zugehöriger User Story	US-9.1, US-9.2, US-4.5, US-4.6
User	Repräsentation des verantwortlichen Benutzers, der einer Aufgabe zugewiesen ist	US-9.3
UserStory	Repräsentiert User Stories zur Zuordnung von Aufgaben (Dropdown-Feld in UI)	US-4.5
TaskController	Verarbeitet Filter- und Sortieranfragen, Detailsicht und Rücksetzlogik	US-9.1, US-9.2, US-9.3, US-4.5, US-4.6
TaskService	Implementiert die Filterlogik für Status, Priorität und Sortierung der Aufgaben	US-9.1, US-9.2, US-4.5
CalendarEntry	Repräsentiert ein Kalendereintrag mit Titel, Start- und Endzeitpunkt	US-4.6
CalendarController	Stellt die REST-Schnittstelle `/api/calendar` zur Verfügung, die Aufgaben als JSON zurückliefert	US-4.6
CalendarService	Koordiniert den Zugriff auf Kalendereinträge (Lesen, Speichern, Löschen)	US-4.6

Detaillierte Beschreibung der Implementierung

Die oben dargestellten Klassen bilden die Grundlage für die in Sprint 4 implementierten Funktionalitäten. Die Klasse `Task` wurde verwendet, um Aufgaben mit verschiedenen Attributen wie `Priority` und `Status` abzubilden. Durch die Klasse `User` kann eine Aufgabe einem bestimmten Benutzer zugewiesen werden, was durch ein entsprechendes Dropdown-Menü in der Übersicht sichtbar ist. Die Klasse `UserStory` dient zur strukturierten Verknüpfung von Aufgaben mit User Stories über ein Auswahlfeld.

Im Rahmen von `US-2.3` und `US-4.4` wurde die Klasse `UserStory` um zwei neue Attribute erweitert: `status` und `faelligkeit`. Diese Felder ermöglichen die Verwaltung des Bearbeitungsstands sowie die zeitliche Planung der User Stories. Entsprechende Eingabefelder wurden im UI ergänzt, während die `BacklogService`-Klasse für deren Validierung und Speicherung zuständig ist.

Der `TaskController` koordiniert alle eingehenden Anfragen zur Filterung, Sortierung, Rücksetzung und Detailansicht. Die gesamte Filterlogik wird im `TaskService` ausgeführt, wo mehrere kombinierte Parameter wie `priority`, `status` und `sort` verarbeitet werden.

Dieses Modell ermöglicht eine klare Zuordnung der User Stories zu den umgesetzten Komponenten und stellt die Verbindung zwischen den Anforderungen und der tatsächlichen Implementierung im System sicher.

Im Rahmen der User Story US-4.6 wurde das bestehende Task-Modell um eine visuelle Kalenderansicht ergänzt. Dabei wurde die Klasse `CalendarController` eingeführt, die eine REST-Schnittstelle zur Verfügung stellt. Diese liefert alle Aufgaben im JSON-Format zurück, wobei das `createdAt`-Attribut als zeitliche Referenz dient. Die Klasse `CalendarService` übernimmt die Geschäftslogik und ruft über das `TaskRepository` die gespeicherten Aufgaben ab. Die Darstellung erfolgt über die HTML-Seite `calendar.html` mit Hilfe der JavaScript-Bibliothek FullCalendar.

Durch diese Erweiterung wurde die bestehende Datenbasis sinnvoll wiederverwendet, um eine zusätzliche visuelle Funktionalität bereitzustellen. Die klare Trennung der Schichten zwischen Controller, Service

und Repository bleibt dabei erhalten.

3.1 Updates zu genutzten Technologien

Zusätzlich zu den im dritten Sprint verwendeten Technologien wurde im Rahmen des vierten Sprints die JavaScript-Bibliothek **FullCalendar** integriert. Diese ermöglicht eine dynamische und visuell ansprechende Darstellung der Aufgaben in Kalenderform im Frontend. FullCalendar wurde clientseitig über ein CDN eingebunden und mit einer eigens implementierten REST-Schnittstelle (`/api/calendar`) im Spring Boot Backend verbunden. Dadurch konnte die Kalenderansicht nahtlos in das bestehende System integriert werden, ohne tiefgreifende Änderungen an der Architektur vorzunehmen.Zusätzlich wurden im Zuge der Erweiterung des Backlog-Moduls neue Formularfelder für `status` und `fälligkeit` im UI ergänzt sowie entsprechende Validierungsmechanismen im Backend eingeführt. Diese Erweiterung konnte nahtlos in das bestehende Spring Boot Setup integriert werden.

3.2 Dokumentation der Codequalität

Abweichungen des Codes zur geplanten Architektur

In diesem Sprint wurden keine Abweichungen zur geplanten Architektur festgestellt.

Durchgeführte automatische Tests und Testergebnisse

Teststrategie: Auch in diesem Sprint war es unser Ziel, sicherzustellen, dass die Kernfunktionen sowohl auf Service-Ebene als auch auf API-Ebene korrekt funktionieren. Wir haben daher Unit-Tests und REST-Tests für die folgenden neuen Module geschrieben:

- **White-box-Tests:** direkt im Service (`CalendarServiceImpl`, `NotificationService`, `UserStoryService`), um die Logik intern zu prüfen.
- **Black-box-Tests:** über MockMvc für die REST-Controller (`CalendarRestController`, `NotificationRestContro` `UserStoryRestController`), um die API aus Sicht eines Benutzers zu testen.

Testübersicht:

Test-ID	Zeitpunkt	Herkunft / Beschreibung	Ergebnis
TC18	15.07.2025	Unit-Test: CalendarServiceImpl.getAllEntries() gibt alle Kalender-Einträge zurück	Pass
TC19	15.07.2025	Unit-Test: CalendarServiceImpl.saveEntry() speichert neuen Eintrag korrekt	Pass
TC20	15.07.2025	Unit-Test: CalendarServiceImpl.deleteEntry() löscht Kalender-Eintrag nach ID	Pass
TC21	15.07.2025	REST-Test: GET /api/calendar gibt alle Kalender-Einträge zurück	Pass
TC22	15.07.2025	REST-Test: POST /api/calendar erstellt neuen Kalender-Eintrag	Pass
TC23	15.07.2025	REST-Test: DELETE /api/calendar/id löscht Kalender-Eintrag	Pass
TC24	15.07.2025	Unit-Test: NotificationService.send() speichert Nachricht im Repository	Pass
TC25	15.07.2025	Unit-Test: NotificationService.sendTaskCreationNotification() gibt Konsole-Ausgabe zurück	Pass
TC26	15.07.2025	Unit-Test: NotificationService.send(null, null) speichert leere Nachricht	Pass
TC27	15.07.2025	REST-Test: POST /api/notifications/id/markAsRead mit gültiger ID	Pass
TC28	15.07.2025	REST-Test: POST /api/notifications/id/markAsRead mit bereits gelesener Nachricht	Pass
TC29	15.07.2025	REST-Test: POST /api/notifications/id/markAsRead mit ungültiger ID	Pass
TC30	15.07.2025	REST-Test: GET /api/notifications/unread mit gültiger E-Mail	Pass
TC31	15.07.2025	Unit-Test: UserStoryService.getAllStories() gibt alle Stories zurück	Pass
TC32	15.07.2025	Unit-Test: UserStoryService.getStoryById() bei existierender ID	Pass
TC33	15.07.2025	Unit-Test: UserStoryService.getStoryById() bei nicht existierender ID	Pass
TC34	15.07.2025	Unit-Test: UserStoryService.saveStory() speichert korrekt	Pass
TC35	15.07.2025	Unit-Test: UserStoryService.deleteStory() löscht nach ID	Pass
TC36	15.07.2025	REST-Test: GET /api/userstories gibt alle Stories zurück	Pass
TC37	15.07.2025	REST-Test: GET /api/userstories/id bei gültiger ID	Pass
TC38	15.07.2025	REST-Test: GET /api/userstories/id bei ungültiger ID	Pass
TC39	15.07.2025	REST-Test: POST /api/userstories erstellt neue Story	Pass
TC40	15.07.2025	REST-Test: PUT /api/userstories/id aktualisiert existierende Story	Pass
TC41	15.07.2025	REST-Test: PUT /api/userstories/id bei ungültiger ID gibt 404 zurück	Pass
TC42	15.07.2025	REST-Test: DELETE /api/userstories/id löscht erfolgreich	Pass
TC43	15.07.2025	REST-Test: DELETE /api/userstories/id bei ungültiger ID gibt 404 zurück	Pass

Table 3: Testübersicht: Gesamtübersicht Calendar-, Notification- und UserStory-Tests

Ablageort der Tests:

- `src/test/java/com/example/demo/service/CalendarServiceImplTest.java`
- `src/test/java/com/example/demo/controller/CalendarRestControllerTest.java`
- `src/test/java/com/example/demo/service/NotificationServiceTest.java`
- `src/test/java/com/example/demo/controller/NotificationRestControllerTest.java`
- `src/test/java/com/example/demo/service/UserStoryServiceTest.java`
- `src/test/java/com/example/demo/controller/UserStoryRestControllerTest.java`

Reflexion zur Teststrategie:

Automatisierte Tests geben uns Sicherheit bei Änderungen am Code. Durch die Kombination aus Unit-Tests für die Service-Ebene (`CalendarServiceImpl`, `NotificationService`, `UserStoryService`) und REST-Tests für die API-Ebene (`CalendarRestController`, `NotificationRestController`, `UserStoryRestController`) konnten wir sowohl die interne Logik als auch die externen Schnittstellen zuverlässig testen.

Sobald etwas bricht, erkennen wir es direkt im Testlauf. Zusätzlich sparen wir langfristig Zeit bei Refactorings, da wir nicht manuell alles überprüfen müssen.

Für unser Projekt ist diese Mischung aus White- und Black-box-Tests optimal und ausreichend.

Durchgeführte manuelle Tests

Testdurchführung: Zwei Teammitglieder haben die neu implementierten Funktionen des Kalenders, der Benachrichtigungen und der User Stories manuell getestet. Dabei wurde überprüft, ob Aufgaben korrekt erstellt, einem Benutzer zugewiesen und im Sprint-Board dargestellt werden können. Auch die Dateianhänge, Filterfunktionen sowie der automatische Mailversand bei Aufgabenbenachrichtigungen wurden getestet. Dabei wurden sowohl Fehlerfälle als auch typische Nutzungsszenarien abgedeckt.

Testergebnisse:

Rolle	Ziel des Tests	Version	Ergebnis
Teammitglied	UserStory wird erfolgreich über die UI erstellt und erscheint korrekt in der Tabelle	UI vom 15.07.2025	Pass
Teammitglied	Neue Aufgabe kann erstellt, im Sprint-Board angezeigt und einem Benutzer zugewiesen werden	UI vom 15.07.2025	Pass
Teammitglied	Kalendereintrag wird erstellt und im Kalender-Modul gespeichert	UI vom 15.07.2025	Pass
Teammitglied	Benachrichtigungsmail wird nach Task-Erstellung automatisch an den Empfänger gesendet	UI vom 15.07.2025	Pass
Teammitglied	Aufgaben können erfolgreich mit Dateien versehen und später heruntergeladen werden	UI vom 15.07.2025	Pass
Teammitglied	Filter und Sortierfunktionen in der Aufgabenübersicht funktionieren wie erwartet	UI vom 15.07.2025	Pass

Table 4: Testergebnisse: Manuelle Tests

3.3 Tracing

Auch im vierten Sprint wurde ein systematisches Tracing zwischen den modellierten UML-Diagrammen und der tatsächlichen Code-Implementierung durchgeführt. Dazu wurde eine eigene Tracing-Tabelle erstellt, die die Verknüpfung zwischen den relevanten Klassen, Interfaces sowie User Stories und deren Implementierung zeigt.

Diese Dokumentation unterstützt die Nachvollziehbarkeit der Umsetzung und gewährleistet Konsistenz zwischen Modellierung und Code.

GitLab-Link zur Tracing-Dokumentation: Link

3.4 Laufender Prototyp

GitLab-Link zum Prototyp: Link

Im Rahmen des vierten Sprints wurden zentrale Funktionalitäten zur Aufgabenfilterung und Sortierung implementiert. Die Anwendung erlaubt es nun, Aufgaben gezielt nach Kriterien wie Priorität, Status und Erstellungsdatum zu durchsuchen und übersichtlich anzuzeigen.

Implementierte User Stories

- **US-9.1**: Aufgaben nach Priorität filtern
- **US-9.2**: Aufgaben nach Status filtern
- **US-9.3**: Anzeige des zugewiesenen Nutzers
- **US-4.5**: Kombination von Filter- und Sortieroptionen mit responsivem Layout
- **US-2.3**: Aufgabenstatus setzen (offen, in Bearbeitung, fertig)
- **US-4.4**: Fälligkeitsdatum für User Stories festlegen
- **US-4.6:** Visuelle Darstellung der Aufgaben in Kalenderform mithilfe von FullCalendar

Benutzerperspektive

Benutzer haben nun die Möglichkeit, Aufgaben nach verschiedenen Kriterien zu filtern und zu sortieren. Zusätzlich kann der Bearbeitungsstatus sowie ein Fälligkeitsdatum direkt bei der Erstellung oder Bearbeitung einer User Story angegeben werden. Dabei werden die gewählten Filter in der Benutzeroberfläche deutlich angezeigt. Ein Zurücksetzen-Button ermöglicht das Entfernen aller aktiven Filter. Zusätzlich kann über ein Symbol die Detailansicht einer Aufgabe geöffnet werden. Darüber hinaus können Aufgaben nun auch in einer Kalenderansicht dargestellt werden, die alle Aufgaben basierend auf ihrem Erstellungsdatum visuell aufbereitet. Dies ermöglicht es den Benutzer*innen, den zeitlichen Verlauf und die Verteilung der Aufgaben im Projekt übersichtlich nachzuvollziehen. Die Kalenderdarstellung wurde mithilfe der Bibliothek FullCalendar realisiert und ist über einen eigenen Navigationspunkt erreichbar.

rasit

Beispielhafte Screenshots

Screenshots zur Veranschaulichung der implementierten Funktionalitäten befinden sich im Anhang:

- Aufgabenübersicht mit Filter- und Sortieroptionen
- Anzeige aktiver Filter
- Reset-Funktion
- Detailansicht einer Aufgabe

Installation und Kompilation

Die Installationsanleitung bleibt unverändert. Eine detaillierte Anleitung zur Ausführung befindet sich in der README-Datei: README.md.

3.5 Abweichung von der Planung

Im Verlauf des Sprints 4 ergaben sich einige Abweichungen von der ursprünglichen Planung, die jedoch zur funktionalen Verbesserung und Usability der Anwendung beitrugen. Besonders hervorzuheben sind folgende Erweiterungen, die über die initiale Planung hinausgingen:

- **UI-Feedback für aktive Filter:** Zusätzlich zu den funktionalen Filteroptionen wurde eine dynamische Anzeige der aktiven Filter im UI integriert, um die Transparenz für die Nutzer zu erhöhen.
- **Reset-Funktion mit Default-Sortierung:** Ein Zurücksetzen-Button wurde implementiert, der nicht nur alle aktiven Filter löscht, sondern gleichzeitig die Sortierung automatisch auf `createdAt` (ascending) zurücksetzt.
- **Kombinierte Filter– und Sortierlogik:** Während die Stories nur einfache Filterung oder Sortierung vorsahen, wurde die Logik so erweitert, dass beliebige Kombinationen von `priority`, `status`, `sort` und `order` gemeinsam verarbeitet und korrekt dargestellt werden können.
- **Feinoptimierung des Frontends:** Mehrere kleinere Korrekturen in der `tasks.html` Datei wurden vorgenommen, um Edge Cases wie leere oder null-Werte korrekt darzustellen (z. B. für `selected`, `default values` oder `filePath == null`).
- **Backlog-Erweiterung um Status und Fälligkeitsdatum:** Ursprünglich war nur die Erstellung einfacher User Stories geplant. Im Sprintverlauf wurde jedoch beschlossen, den Bearbeitungsstatus sowie ein Fälligkeitsdatum zu integrieren. Dadurch konnte der Fortschritt einzelner Stories besser verfolgt und die Zeitplanung im Backlog verbessert werden.
- **Integration einer Kalenderansicht:** Ursprünglich war keine visuelle Kalenderdarstellung der Aufgaben vorgesehen. Im Verlauf des Sprints wurde jedoch entschieden, eine separate Kalenderseite mithilfe der JavaScript-Bibliothek *FullCalendar* zu implementieren. Diese zeigt alle Aufgaben basierend auf ihrem Erstellungsdatum in einem Monatsüberblick an und verbessert dadurch die zeitliche Übersicht über den Projektverlauf erheblich.

Diese Ergänzungen steigerten nicht nur die Benutzerfreundlichkeit und visuelle Konsistenz der Anwendung, sondern verdeutlichen auch das flexible und lösungsorientierte Arbeiten im Rahmen des agilen Entwicklungsprozesses.

4. Dokumentation individuelle Beiträge

Die folgende Tabelle dokumentiert den Beitrag der Teammitglieder zur Erstellung dieses Projektabgabedokuments für den Sprint.

Sprint	**Verantwortliche Teammitglieder**	**Anwesende während der Meetings**	**(Online-) Beiträge zum Inhalt durch**	**Korrektur-gelesen durch**
Sprint 4	Dogukan, Helin, Hüseyin, Cagla	Dogukan, Helin, Hüseyin, Cagla	Dogukan, Helin, Hüseyin, Cagla	Dogukan, Helin, Hüseyin, Cagla

Table 5: Beitrag der Teammitglieder zum Projektabgabedokument

Die nächste Tabelle dokumentiert die Beiträge der Teammitglieder zu den im Sprintbacklog durchgeführten Tasks.

Task ID	Task Beschreibung / Name	Teammitglieder	Beitrag in %
US-9.1-T1	Implementierung des Priority-Filters (Backend + Frontend)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-9.2-T1	Implementierung des Status-Filters (Backend + Frontend)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-9.1/9.2-T2	Anzeige der aktiven Filter in der UI (Thymeleaf + Logik)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-4.5-T1	Integration der kombinierten Sortierung und Filterung im Backend	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-4.5-T2	Dropdown-Layout für Filteroptionen (Sortierung: asc/desc, Priority, Status)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-4.5-T3	Reset-Button zur Zurücksetzung aller Filterparameter	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-4.5-T4	Responsive Design der Aufgabenübersicht (Bootstrap)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-9.3-T1	Anzeige des zuständigen Nutzers in der Aufgabenübersicht (Badge + Tooltip)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-2.3-T1	Erweiterung der User Story um Bearbeitungsstatus (UI + Backend-Validierung)	Dogukan, Helin, Hüseyin, Çağla	25% jeweils
US-4.4-T1	Hinzufügen eines Fälligkeitsdatums zur User Story inkl. Speicherung und Anzeige	Doğukan, Helin, Hüseyin, Çağla	25% jeweils
US-4.6-T1	Integration einer Kalenderansicht zur visuellen Darstellung der Aufgaben mit FullCalendar	Dogukan, Helin, Hüseyin, Çağla	25% jeweils

Table 6: Beitrag der Teammitglieder zu den im Sprint 4 erledigten Tasks

Sieht super aus soweit!

Guckt bitte nochmal auf meine Kommentare und wenn nötig, macht die nötigen Änderungen. Wenn ihr meint, dass dort nichts mehr zu tun ist, dann begründet mir das bitte nochmal.

Anhang – Screenshots

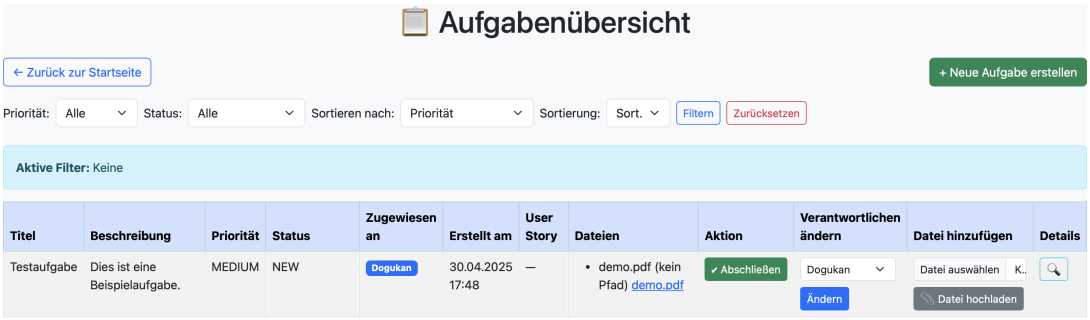

Figure 7: Aufgabenübersicht mit Filter- und Sortieroptionen

Figure 8: UI-Anzeige der aktiven Filterkombinationen

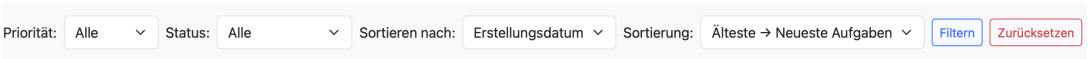

Figure 9: Zurücksetzen-Button zur Standardansicht

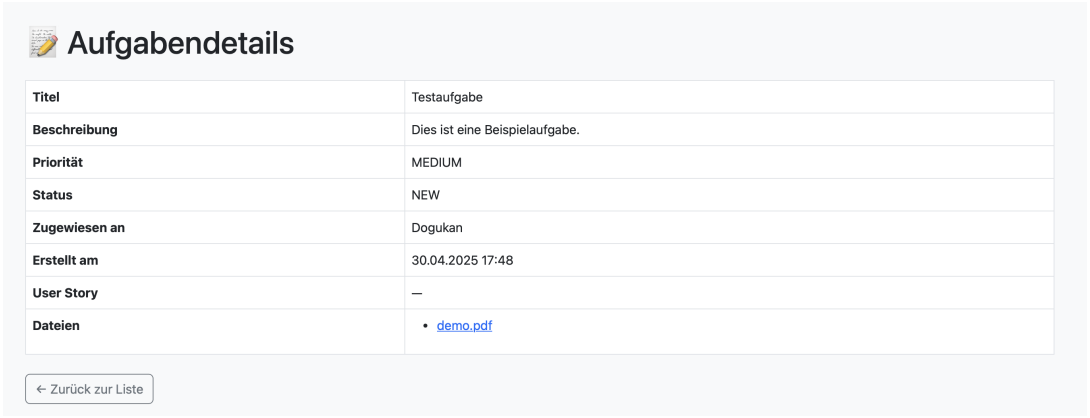

Figure 10: Detailansicht einer Aufgabe