



Softwaretechnik WS 2024-25

Projektabgabedokument Sprint 3

Übung-2 Team-1

Dogukan Karakoyun, 223202023

Hüseyin Kayabasi, 223201801

Helin Oguz, 223202103

Cagla Yesildogan, 223201881

1 Einleitung

1.1 Produkt-Backlog vor dem Sprint

Änderungen im Produktbacklog

Im Vergleich zum initialen Produktbacklog haben sich im Sprint 3 folgende Änderungen ergeben:

- **Einführung des Backlog-Managements:**
Benutzer können nun User Stories im Backlog erstellen, bearbeiten und priorisieren.
Grund: Unterstützung der Planung und Strukturierung zukünftiger Aufgaben im Scrum-Prozess.
(Issue: US-2.1, US-2.2)
- **Erweiterung der Task-Klasse:**
Attribut `assignedUserId` wurde hinzugefügt, damit Aufgaben einem Benutzer zugewiesen werden können.
Grund: Klare Verantwortung pro Task sichtbar machen (Issue: US-6.1).
- **Erweiterung der Task-Klasse:**
Attribut `userId` wurde hinzugefügt, um Tasks mit User Stories zu verlinken.
Grund: Bessere Nachvollziehbarkeit der Anforderungen (Issue: US-3.2).
- **Erweiterung des Task-Erstellungsformulars:**
Dropdown-Menü zur Auswahl des verantwortlichen Benutzers.
Dropdown-Menü zur Auswahl der zugehörigen User Story.
Pflichtfeld-Validierung für User Story und Assigned User im Formular.
Grund: Verbesserung der Benutzerfreundlichkeit und Validierung beim Erstellen neuer Aufgaben.
- **Implementierung der Datei-Upload-Funktionalität:**
Benutzer können beim Erstellen und Bearbeiten von Aufgaben Dateien hochladen.
Unterstützung echter Datei-Uploads mit `MultipartFile`.
Speicherung von Dateiname, Dateityp und Pfad.
Grund: Erweiterung der Aufgaben um relevante Dokumente (Issue: US-3.1).
- **Implementierung der Datei-Download-Funktionalität:**
Dateien können heruntergeladen werden; originaler Dateiname bleibt erhalten.
Richtiger Content-Type wird gesetzt, um Browser-Kompatibilität sicherzustellen.
Grund: Verbesserung der Benutzerfreundlichkeit bei Datei-Downloads.
- **Validierung von User Story IDs im Backend:**
Beim Erstellen einer Aufgabe wird geprüft, ob die angegebene User Story existiert.
Grund: Vermeidung von fehlerhaften Daten (Issue: US-3.2).
- **Notification-System:**
Log-Ausgabe bei neuer Task-Erstellung zur Simulation eines Notification-Systems.
Vorbereitung einer `sendMail()` Schnittstelle für zukünftige E-Mail-Benachrichtigungen.
Grund: Frühzeitige Grundlage für spätere echte Benachrichtigungssysteme (Issue: US-10.1).
- **Fehlerbehebungen und Optimierungen:**
Richtige Pfadbehandlung beim Upload (Vermeidung von `FileNotFoundException`).
Anpassung des Frontends zur Konsistenz mit bestehendem Design (Color Themes, Button Styles).
Behebung von MIME-Type Problemen bei Datei-Downloads.
Grund: Verbesserung der Gesamtqualität und Fehlerresistenz.

Reflexion über die Anforderungen

Während des Sprints wurde nicht nur der bestehende Produktbacklog umgesetzt. Zusätzlich haben wir **proaktiv** folgende Verbesserungen eingeführt:

- **Optimierung des Datei-Upload-Prozesses:**
Ursprünglich sollten nur Dateinamen gespeichert werden; stattdessen wurde ein echter Upload mit vollständigem File-Handling implementiert.

- **Erweiterung der Validierung:**

Zusätzliche Backend-Validierungen wurden ergänzt, um Eingabefehler frühzeitig abzufangen (z.B. User Story Existenzprüfung).

- **Benutzerfreundlichkeit:**

Die Benutzeroberfläche wurde an bestehende Standards angepasst, um eine konsistente User Experience zu gewährleisten.

Download-Links wurden so gestaltet, dass Dateinamen korrekt angezeigt und heruntergeladen werden können.

- **Systematische Fehlersuche und -behebung:**

Mehrere nicht geplante, aber dringend notwendige Bugfixes (z.B. korrekter Content-Type für PDF-Downloads) wurden vorgenommen.



1+1/1

1.2 Sprint-Planung

Geplante Aufgaben

Im Sprint 3 wurden folgende Aufgaben erfolgreich geplant und umgesetzt:

- **US-2.1 & US-2.2 User Stories erstellen und priorisieren**

- Implementierung des Backlog-Moduls zur Erstellung und Bearbeitung von User Stories.
- Erweiterung der `UserStory`-Klasse um Felder wie `beschreibung`, `priorität`, `createdAt`, `updatedAt`.
- Einführung eines `Priorität`-Enums zur standardisierten Priorisierung.
- Anlegen eines `BacklogController` zur Steuerung der UI-Logik.
- Nutzung eines DTOs (`UserStoryForm`) zur sicheren Übertragung der Formulardaten.

- **US-6.1 Nutzer zu Task zuordnen**

- Erweiterung der `Task`-Klasse um das Feld `assignedUserId`.
- Anpassung des Task-Erstellungsformulars mit einem Dropdown zur Benutzerauswahl.
- Anzeige des verantwortlichen Benutzers in der Aufgabenübersicht.
- Möglichkeit zur nachträglichen Änderung des zugewiesenen Benutzers.
- Backend-Validierung zur Überprüfung der Benutzer-Existenz.

- **US-3.1 Datei-Upload & Notification System**

- Implementierung eines echten Datei-Uploads mit `MultipartFile`.
- Speicherung von Dateiname, Dateityp und Upload-Datum.
- Erweiterung der `TaskFile`-Entity um neue Felder.
- Anzeige und Download der Dateien in der Aufgabenübersicht mit Original-Dateinamen.
- Implementierung eines simplen `NotificationService` zur Log-Ausgabe bei Task-Erstellung.

- **US-10.1 E-Mail-Benachrichtigung (Stub-Version)**

- Mock-Implementierung einer E-Mail-Benachrichtigung bei Task-Erstellung (Log-Simulation).
- Erstellung einer `sendMail()`-Schnittstelle für zukünftige Erweiterungen.

- **US-3.2 Aufgaben mit User Stories verlinken**

- Erweiterung der `Task`-Klasse um das Feld `userStoryId`.
- Dropdown zur User Story-Auswahl im Erstellungsformular.
- Validierung der Existenz einer User Story im Backend.
- Anzeige der User Story in der Aufgabenübersicht.

• US-5.1 & US-5.2 Nutzerprofile anzeigen und bearbeiten

- Implementierung der Profilansicht über `/profile` für den eingeloggtten Nutzer.
- Anzeige fremder Profile über `/view/{userId}` inklusive Session-Validierung.
- Implementierung der Teamübersicht über `/team` zur Darstellung aller Nutzer.
- Integration eines Profilbild-Uploads mit Speicherung im lokalen Dateisystem.
- Erweiterung der `User`-Entität um das Feld `profilePictureUrl`.
- Umsetzung der Bio-Aktualisierung durch ein Formular mit POST-Verarbeitung.
- Änderung des Passworts mit Sicherheitsprüfung (altes Passwort erforderlich).
- Nutzung von `AccountService` für alle persistenzbezogenen Profiländerungen.

Reflexion

Während des Sprints wurden zusätzlich folgende Verbesserungen vorgenommen:

- Optimierung der Datei-Download-Logik:
 - Unterstützung für den Download von Dateien mit Original-Dateinamen und korrektem Content-Type.
 - Behebung von Problemen beim MIME-Type und Dateinamen bei Downloads.
- Erweiterung der Validierung:
 - Implementierung zusätzlicher Prüfungen, z.B. auf Existenz der User Story beim Erstellen eines Tasks.
- Frontend-Verbesserungen:
 - Anpassung der Formulare und Buttons für ein konsistentes Benutzererlebnis.
 - Hinzufügen eines Zurück-zur-Startseite-Buttons in der Aufgabenübersicht.
- Fehlerbehebungen:
 - Sicherstellung der Ordnerstruktur für Dateiuploads.
 - Korrekte Anzeige und Handhabung von Fehlermeldungen im UI.

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.

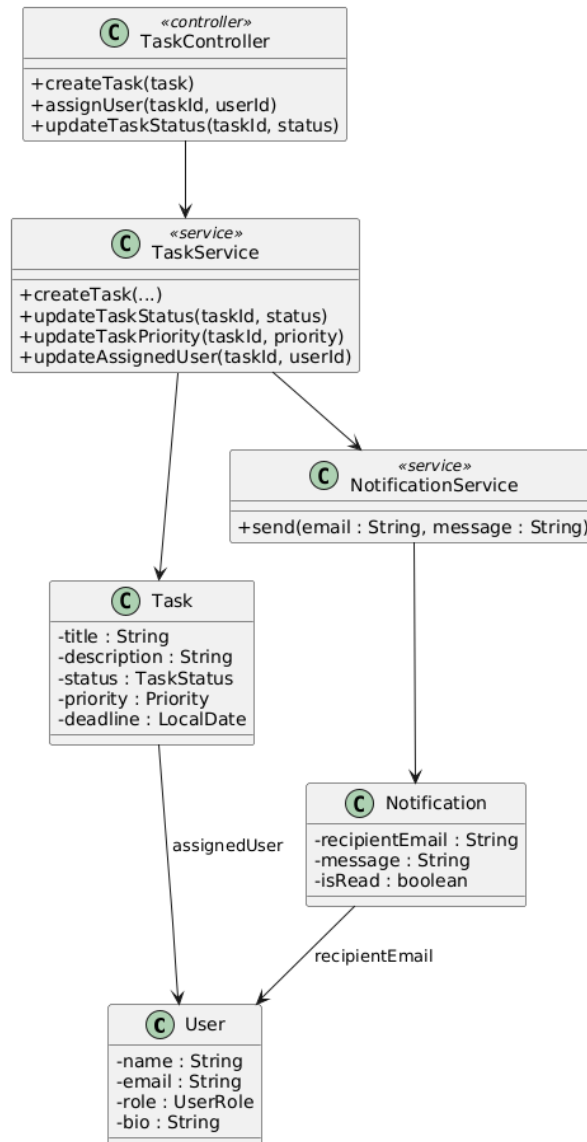
User Story	Task	Dogukan	Helin	Hüseyin	Cagla	Ø Punkte	Begründung
US-2.1, 2.2	Erstellung und Priorisierung von User Stories	3	4	5	3	4	Neue Entität, Enum, Controller, Validierung
US-6.1	Aufgaben Zuweisung an Nutzer (User Assignment)	5	6	5	6	5.5	User-Assignment braucht saubere Validierung
US-3.1	Datei-Upload-Feature implementieren	8	7	7	8	7.5	Dateiverwaltung + Dateiupload mit Speicherpfad
US-3.1	NotificationService (Log Ausgabe)	3	4	3	4	3.5	Mock-Notification, keine externe API
US-10.1	E-Mail Mock Notification implementieren	4	4	5	5	4.5	E-Mail-Struktur definieren, Log-Mock
US-3.2	Aufgaben mit User Story verknüpfen	6	7	6	7	6.5	User Story ID mit Task verlinken, Validierung nötig
US-5.1	Anzeige von Nutzerprofilen und Teamübersicht	4	4	5	4	4.25	Sessionprüfung, Daten aus Datenbank holen, HTML-Ansicht mit Liste
US-5.2	Bearbeitung des eigenen Profils (Bild, Bio, Passwort)	6	5	6	6	5.75	Bild-Upload mit Pfad, Formularverarbeitung, Validierung

Table 1: Aufwandsschätzung (Story Points) für Sprint 3

2.1 Klassendiagramm

Benachrichtigungen [US-10.1]

In diesem Abschnitt wird die strukturierte Modellierung für die User Story US-10.1 *Benachrichtigungen* dargestellt. Die Modellierung erfolgt mit einem Klassendiagramm, das die wichtigsten Komponenten, Attribute, Methoden und Relationen im Kontext des Benachrichtigungssystems enthält. Zusätzlich werden die Verantwortlichkeiten der beteiligten Klassen erläutert, um die Interaktion und den Informationsfluss im System verständlich darzustellen.



Modellierte Klassen und Verantwortlichkeiten

- **TaskController:** Entgegennahme von Benutzeranfragen zur Erstellung und Bearbeitung von Aufgaben. Leitet diese an den TaskService weiter und stößt dabei gegebenenfalls das Versenden von Benachrichtigungen an.
- **TaskService:** Verwaltet die Geschäftslogik zur Erstellung und Aktualisierung von Aufgaben. Löst im Zuge dieser Vorgänge das Versenden von Benachrichtigungen an die betroffenen Benutzer aus.
- **NotificationService:** Verantwortlich für das Erzeugen und den Versand von Benachrichtigungen an Benutzer per E-Mail. Persistiert die versendeten Benachrichtigungen im System.

- **Notification:** Modelliert eine Benachrichtigung mit Empfängeradresse, Nachrichtentext und Lesestatus. Dient der Speicherung versandter Benachrichtigungen.
- **User:** Repräsentiert die Empfänger der Benachrichtigungen anhand der hinterlegten E-Mail-Adresse.
- **Task:** Dient als Auslöser für Benachrichtigungen, beispielsweise bei der Erstellung neuer Aufgaben oder der Zuweisung von Aufgaben an Benutzer.

Technische Umsetzung der Benachrichtigungen

Das Benachrichtigungssystem wurde als eigenständiger Service (**NotificationService**) implementiert, um systemweit Benachrichtigungen zu ermöglichen.

Neben den im Rahmen von **US-10.1** geforderten Benachrichtigungen im Zusammenhang mit Aufgaben (z.B. bei Erstellung und Zuweisung von Tasks) wird der Service ebenfalls in weiteren Systemprozessen eingesetzt:

- Versand von Verifikations-E-Mails im Rahmen der **Benutzerregistrierung** (US-1.1).
- Versand von E-Mails zum Zurücksetzen von Passwörtern bei **Passwort vergessen**.
- Geplante zukünftige Erweiterung für Benachrichtigungen bei neuen Kommentaren oder Statusänderungen.

Die Architektur des **NotificationService** erlaubt es, weitere Benachrichtigungstypen ohne strukturelle Änderungen am System zu integrieren. Dadurch wird eine modulare und erweiterbare Lösung für alle Benachrichtigungsvorgänge im System geschaffen.

User Profile [US-5.1 und US-5.2]

In diesem Abschnitt wird die verfeinerte Struktur für die User Stories US-5.1 (User profile ansehen) und US-5.2 (User profile bearbeiten) modelliert. Die Modellierung erfolgt mit einem Klassendiagramm, das die wichtigsten Komponenten, Attribute, Methoden und Relationen enthält. Zusätzlich werden die Verantwortlichkeiten der Klassen erläutert.

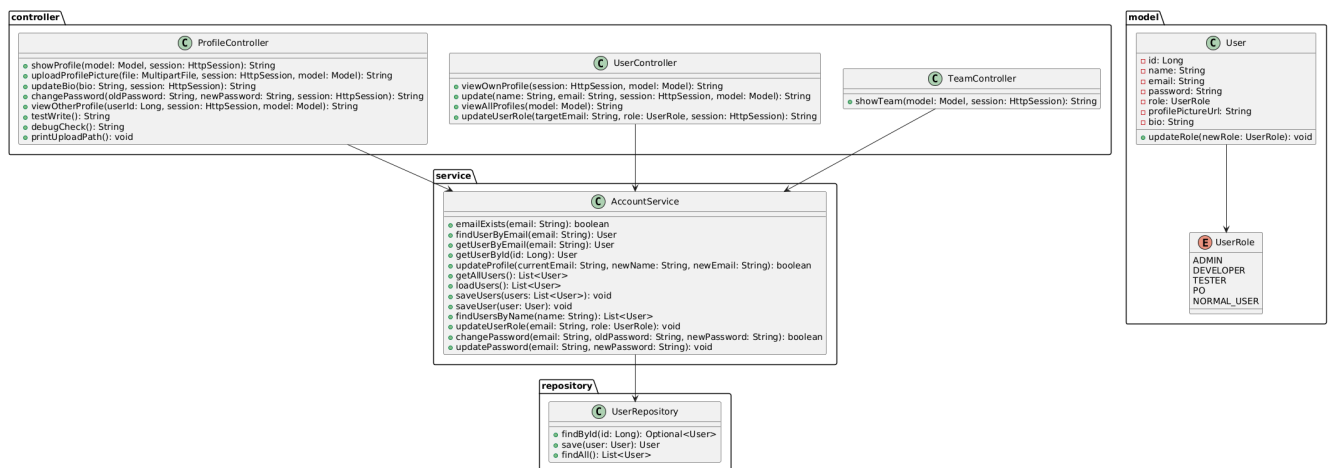


Figure 1: Klassendiagramm zu US-5.1 und US-5.2 – User Profile

Modellierte Klassen und Verantwortlichkeiten

- **User:** Repräsentiert registrierte Nutzer im System. Speichert personenbezogene Informationen wie Name, E-Mail, Passwort, Rolle, Profilbild und Biografie. Dient als zentrale Entität für Authentifikation und Profilverwaltung.
- **UserRole:** Enum zur Definition von Benutzerrollen (z.B. ADMIN, DEVELOPER, TESTER). Wird verwendet, um Rollenrechte im System zu unterscheiden.
- **Account Service:** Kapselt die Geschäftslogik rund um Nutzerverwaltung. Verantwortlich für Registrierung, Login, Rollenverwaltung, Profilbearbeitung und Passwortänderungen. Kommuniziert mit der Datenhaltungsschicht (UserRepository bzw. DatabaseManager).
- **UserRepository:** Abstrakte Schnittstelle zur Datenbank. Dient dem Zugriff auf persistente Nutzerdaten (z.B. laden, speichern, alle abrufen).
- **ProfileController:** Verarbeitet HTTP-Anfragen zur Anzeige und Änderung des eigenen Nutzerprofils. Beinhaltet Funktionen wie Profilanzeige, Foto-Upload, Bio-Aktualisierung und Passwortänderung.
- **UserController:** Zuständig für Benutzerinteraktionen jenseits des eigenen Profils – z.B. andere Profile anzeigen, alle Nutzer listen oder Rollen zuweisen (durch Admins).
- **TeamController:** Zeigt eine Übersicht über alle Teammitglieder an. Dient zur Transparenz und Zusammenarbeit innerhalb eines Entwicklerteams.

Backlog Management [US-2.1 und US-2.2]

In diesem Abschnitt wird die verfeinerte Struktur für die User Stories US-2.1 (User Stories erstellen) und US-2.2 (User Stories priorisieren) 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 Modularität und Erweiterbarkeit der Architektur hervorgehoben.

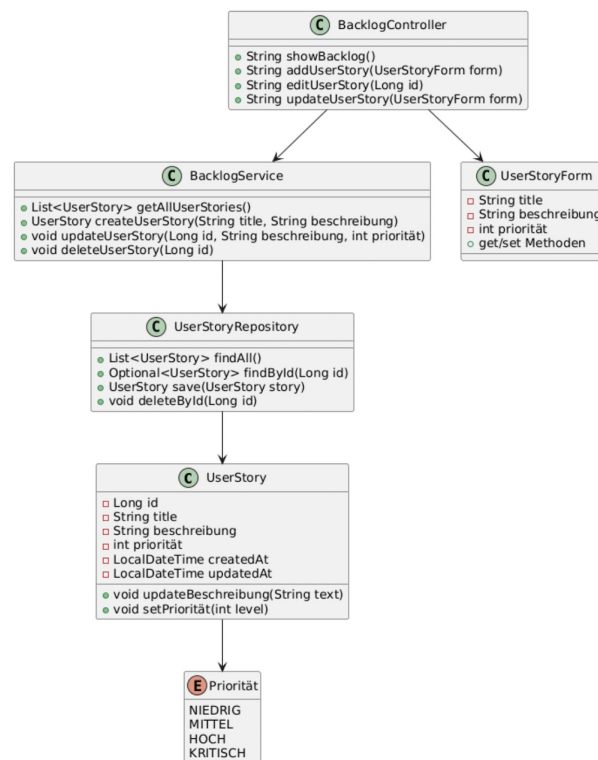


Figure 2: Klassendiagramm zu US-2.1 und US-2.2 – Backlog Management

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.
- **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`, `createdAt`, `updatedAt`. Enthält Methoden zur Bearbeitung der Beschreibung und Priorität.
- **Priorität (Enum):** Definiert die Prioritätsstufen: NIEDRIG, MITTEL, HOCH, KRITISCH.
- **UserStoryForm:** Datenträgerklasse zur Formularübertragung zwischen UI und Backend.

Relevante User Stories:

US-6.1, US-3.1, US-3.2, US-10.1:

Das folgende Klassendiagramm zeigt die Hauptentitäten und deren Beziehungen, die im Rahmen des Sprint 3 implementiert wurden. Es stellt die Implementierung der Benutzerzuweisung, der Aufgabenverlinkung mit User Stories und des Datei-Uploads dar.

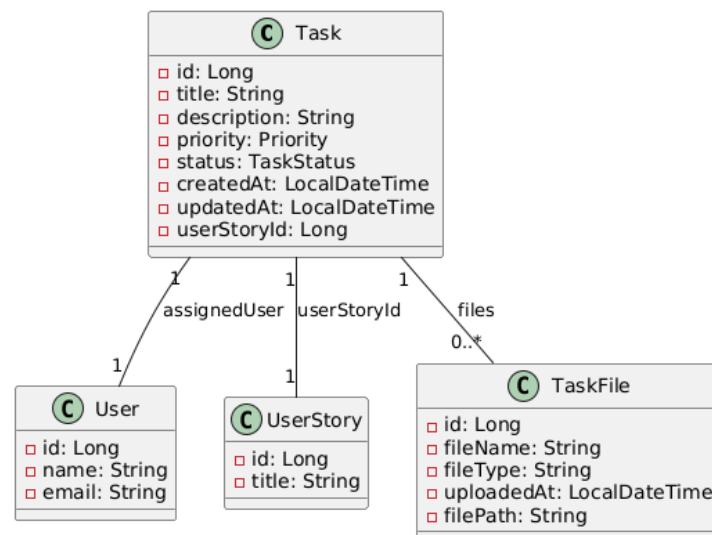


Figure 3: Klassendiagramm der implementierten Entitäten im Sprint 3

Modellierte Klassen und Verantwortlichkeiten

- **Task:** Repräsentiert eine Aufgabe im System. Eine Aufgabe hat Attribute wie Titel, Beschreibung, Priorität, Status sowie Erstellungs- und Aktualisierungszeitpunkte. Zusätzlich besitzt sie zwei zentrale Beziehungen:
 - **assignedUser:** Ein **User**, der für die Aufgabe verantwortlich ist (1:1 Beziehung).
 - **userStoryId:** Eine Referenz auf eine **UserStory**, der die Aufgabe zugeordnet ist.
 - **files:** Eine Liste von **TaskFile**-Objekten, die die zugehörigen Dateien zur Aufgabe darstellen (1:n Beziehung).
- **User:** Stellt einen Benutzer dar, mit Attributen wie ID, Name und E-Mail. Jeder Task wird einem Benutzer zugewiesen.
- **UserStory:** Beschreibt eine User Story, bestehend aus einer ID und einem Titel. Tasks können mit einer User Story verknüpft werden, um die Nachverfolgbarkeit zu verbessern.

- **TaskFile:** Repräsentiert eine Datei, die mit einer Aufgabe verknüpft ist. Enthält Informationen wie Dateiname, Dateityp, Upload-Zeitpunkt und Dateipfad.

Die Beziehungen zwischen den Klassen wurden entsprechend den Anforderungen modelliert:

- Eine Aufgabe (**Task**) ist genau einem Benutzer (**User**) zugewiesen.
- Eine Aufgabe kann mit genau einer User Story (**UserStory**) verlinkt werden.
- Eine Aufgabe kann mehrere Dateien (**TaskFile**) besitzen.

Erweiterungen gegenüber dem ursprünglichen Entwurf

Im Vergleich zum ursprünglichen Klassendiagramm aus den vorherigen Sprints wurden im Sprint 3 folgende Erweiterungen vorgenommen:

- **Hinzufügen der Entität UserStory:** Aufgaben (**Task**) können nun einer User Story zugeordnet werden, um die Nachvollziehbarkeit der Anforderungen zu verbessern.
- **Erweiterung von TaskFile:** Neue Attribute wie `filePath` und `uploadedAt` wurden eingeführt, um vollständige Dateiverwaltungsinformationen zu speichern.
- **Verbesserung der Datei-Upload-Logik:** Anstelle eines einfachen Dateinamens erfolgt jetzt ein echter Datei-Upload mit Pfadspeicherung auf dem Server.
- **Reduktion auf Kerndomänenmodelle:** Service-Schichten wurden aus dem Diagramm entfernt, um eine klarere Darstellung der Entitätsbeziehungen zu gewährleisten.

Modellierung verfeinerter Interfaces und Datenstrukturen

Verfeinerung der Interfaces zwischen den Komponenten

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

User Story – Interface Zuordnung

US-2.1 und US-2.2: Zur Umsetzung der User Stories **US-2.1** (User Stories erstellen) und **US-2.2** (User Stories priorisieren) wurden neue Interfaces und Datenstrukturen entworfen, die sowohl die Wartbarkeit als auch die Erweiterbarkeit des Systems unterstützen. Die nachfolgenden Komponenten stellen zentrale Bausteine des Backlog-Management-Moduls dar.

UserStoryRepository (Implementiert durch: Spring Data JPA)

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

- `findAll()` – Gibt alle gespeicherten User Stories zurück.
- `findById(id)` – Sucht eine User Story anhand ihrer ID.
- `save(story)` – Speichert oder aktualisiert eine User Story.
- `deleteById(id)` – Löscht eine User Story anhand der ID.

BacklogService

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

- `getAllUserStories()` – Gibt eine Liste aller User Stories zurück.
- `createUserStory(...)` – Erstellt eine neue User Story.
- `updateUserStory(...)` – Aktualisiert Beschreibung und Priorität einer User Story.
- `deleteUserStory(...)` – Entfernt eine User Story aus dem Backlog.

UserStoryForm

```
public class UserStoryForm {
    String title;
    String beschreibung;
    int priorit t;

    // Getter und Setter
}
```

- Dient als DTO zur Übertragung von Formularwerten im UI.
- Schützt die UserStory-Entität vor direkter Manipulation.

Priorität (Enum)

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

- Definiert Prioritätsstufen für User Stories.
- Ermöglicht einfache Filterung und Sortierung im UI.

Verantwortlichkeiten der Interfaces

Die im Rahmen von US-2.1 und US-2.2 eingesetzten Interfaces und Klassen sind so gestaltet, dass sie eine klare Trennung der Verantwortlichkeiten sicherstellen. Die wichtigsten Aufgabenverteilungen sind wie folgt:

- **UserStoryRepository**
Dient als Schnittstelle zur Datenbank und ermöglicht CRUD-Operationen auf User Stories. Die Implementierung erfolgt automatisch durch Spring Data JPA, wodurch Standardoperationen ohne zusätzlichen Code bereitgestellt werden.
- **BacklogService**
Enthält die Geschäftslogik für das Erstellen, Bearbeiten und Löschen von User Stories. Die Service-Schicht isoliert die Anwendungslogik vom Controller und Repository, was die Testbarkeit und Wiederverwendbarkeit erhöht.
- **UserStoryForm**
Wird als Datenübertragungsobjekt (DTO) zwischen dem Frontend und dem Controller verwendet. Dadurch wird verhindert, dass die Entität `UserStory` direkt vom Benutzer manipuliert wird. Dies verbessert die Sicherheit und Strukturierung der Datenverarbeitung.
- **Priorität (Enum)**
Definiert feste Werte für die Priorität einer User Story. Die Verwendung eines Enums reduziert Fehlerquellen durch Freitexteingaben und ermöglicht eine einheitliche Darstellung sowie Sortierung im UI.

Relevante User Stories:

US-6.1, US-3.1, US-3.2, US-10.1: Im Rahmen der Umsetzung der User Stories US-6.1, US-3.1, US-3.2 und US-10.1 wurden spezifische Interfaces und Datenstrukturen modelliert, um Aufgaben effizient verwalten, Benutzer zuordnen, Dateien hochladen und Aufgaben mit User Stories verknüpfen zu können.

Die folgenden Interfaces und Datenstrukturen wurden angepasst bzw. erstellt:

TaskRepository

```
@Repository
public interface TaskRepository extends JpaRepository<Task, Long> {
    List<Task> findByAssignedUserId(Long userId);
    List<Task> findByStatus(TaskStatus status);
    List<Task> findByPriority(Priority priority);
}
```

Das **TaskRepository** verwaltet Aufgaben im System. Es stellt Methoden bereit, um Aufgaben nach Benutzer, Status oder Priorität effizient abzufragen. Diese Funktionen sind zentral für die Verwaltung und das Filtern von Aufgaben in der Aufgabenübersicht.

UserRepository

```
public interface UserRepository extends JpaRepository<User, Long> {
    User findByEmail(String email);
}
```

Das **UserRepository** ermöglicht den Zugriff auf Benutzerinformationen, insbesondere zur Zuordnung von Aufgaben zu bestimmten Benutzern über deren E-Mail-Adressen.

UserStoryRepository

```
@Repository
public interface UserStoryRepository extends JpaRepository<UserStory, Long> {
    // Standard JpaRepository methods
}
```

Das **UserStoryRepository** dient zur Verwaltung der User Stories, die als Referenz für Aufgaben verwendet werden. Es stellt sicher, dass Aufgaben mit existierenden User Stories verknüpft sind.

User Profile – Interface Zuordnung

Zur Umsetzung der User Stories **US-5.1** (Nutzerprofile anzeigen) und **US-5.2** (Nutzerprofile bearbeiten) wurden spezifische Komponenten implementiert, die sowohl die Trennung zwischen UI, Service und Persistenz als auch die Erweiterbarkeit des Systems unterstützen. Die folgenden Klassen bilden zentrale Bausteine des User-Profile-Moduls.

ProfileController

```
@GetMapping("/profile")
public String showProfile(Model model, HttpSession session) { ... }

@PostMapping("/upload-photo")
public String uploadProfilePicture(@RequestParam("profilePictureFile") MultipartFile
    file, ...) { ... }

@PostMapping("/update-bio")
public String updateBio(@RequestParam("bio") String bio, ...) { ... }

@PostMapping("/change-password")
public String changePassword(...) { ... }

@GetMapping("/view/{userId}")
public String viewOtherProfile(@PathVariable Long userId, ...) { ... }
```

- `showProfile(...)` – Lädt die Profilseite des aktuell eingeloggten Benutzers.
- `uploadProfilePicture(...)` – Speichert das hochgeladene Profilbild im Dateisystem.
- `updateBio(...)` – Aktualisiert die Biografie des Nutzers.
- `changePassword(...)` – Ändert das Passwort nach Validierung.
- `viewOtherProfile(...)` – Zeigt das Profil eines anderen Teammitglieds an.

TeamController

```
@GetMapping("/team")
public String showTeam(Model model, HttpSession session) { ... }
```

- `showTeam(...)` – Zeigt eine Übersicht aller registrierten Nutzer in einer HTML-Ansicht.

AccountService

```
public User getUserByEmail(String email) { ... }
public User getUserById(Long id) { ... }
public void saveUser(User user) { ... }
public boolean changePassword(String email, String oldPassword, String newPassword) {
    ... }
```

- `getUserByEmail(...)` – Findet Nutzer anhand ihrer E-Mail-Adresse.
- `getUserById(...)` – Gibt den Benutzer basierend auf seiner ID zurück.
- `saveUser(...)` – Speichert Profiländerungen wie Biografie oder Profilbildpfad.
- `changePassword(...)` – Führt Passwortprüfung und -änderung durch.

UserRepository (Spring Data JPA)

```
Optional<User> findById(Long id);
User getUserByEmail(String email);
User save(User user);
```

- `findById(...)` – Sucht Nutzer anhand der ID.
- `getUserByEmail(...)` – Holt Nutzer über E-Mail.
- `save(...)` – Persistiert Änderungen im Nutzerobjekt.

User (Entity)

```
private String name;
private String email;
private String bio;
private String profilePictureUrl;
private UserRole role;
```

- Repräsentiert die Benutzerentität mit allen relevanten Profildaten.
- Wird sowohl zur Anzeige als auch zur Bearbeitung im Profil verwendet.

HttpSession (Session Handling)

```
String email = (String) session.getAttribute("loggedInUser");
```

- Ermöglicht die Identifikation des aktuell eingeloggtten Nutzers.
- Wird genutzt, um sicherzustellen, dass nur autorisierte Benutzer ihre eigenen Profile bearbeiten können.

Sequenzdiagramm

US-2.1 und US-2.2 – Backlog Management

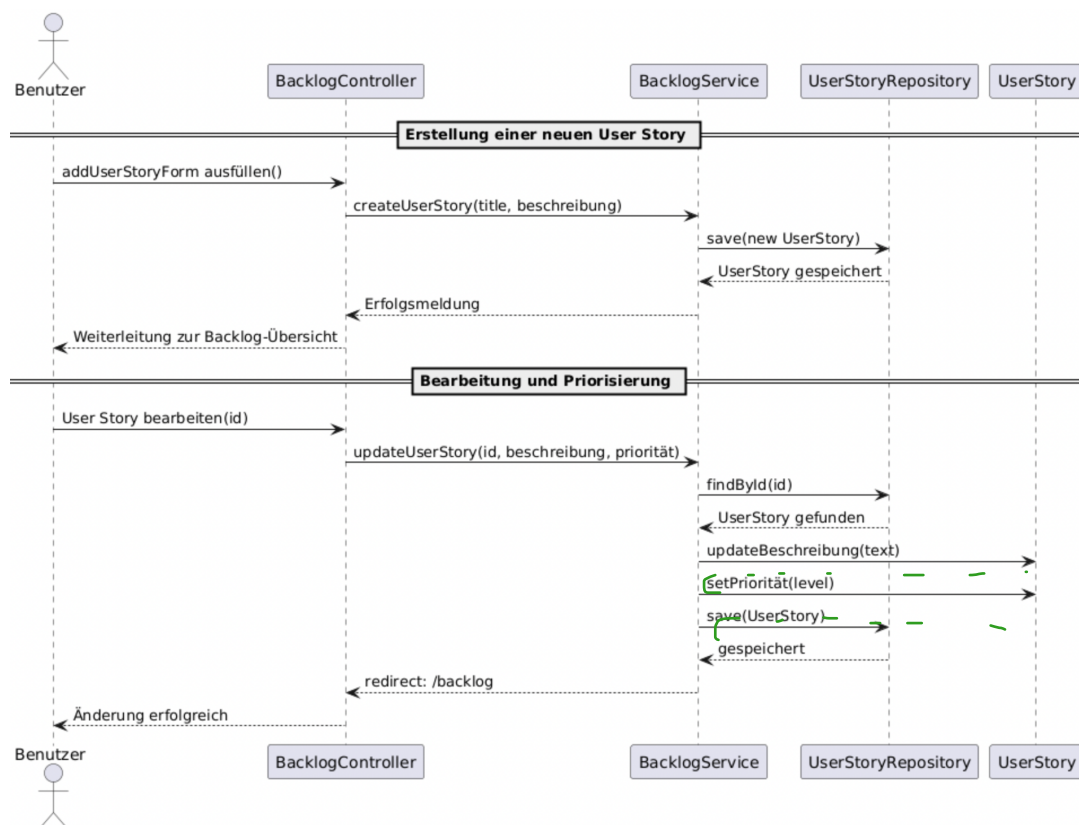


Figure 4: Sequenzdiagramm zur Erstellung und Priorisierung von User Stories

Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde erstellt, um die Benutzerinteraktionen beim Erstellen und anschließenden Bearbeiten einer User Story im Kontext von US-2.1 und US-2.2 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 bearbeitet wird.

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

Das kombinierte Sequenzdiagramm deckt sowohl die Erstellung (US-2.1) als auch die Priorisierung und Bearbeitung (US-2.2) von User Stories im Backlog ab. Es zeigt die Interaktion des Benutzers mit den Anwendungsschichten und demonstriert, wie Datenflüsse zwischen den Schichten strukturiert verlaufen. Durch die Visualisierung wird deutlich, wie neue Anforderungen systematisch integriert und bestehende erweitert werden können.

US-10.1 - Benachrichtigungssystem

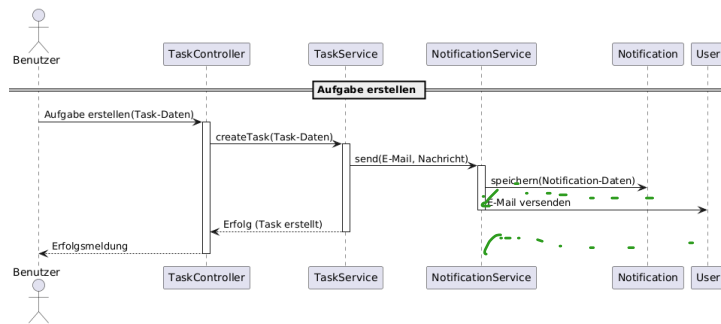


Figure 5: Sequenzdiagramm zu US-10.1 – Benachrichtigungssystem

Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde erstellt, um den Benachrichtigungsprozess des Systems im Kontext von US-10.1 darzustellen. Es zeigt die zentralen Interaktionen beim Erstellen einer Aufgabe und dem automatischen Versand einer Benachrichtigung an den betroffenen Benutzer. Dabei wird deutlich, wie die Verantwortlichkeiten zwischen Controller, Service und Datenhaltung klar getrennt sind und wie das System auf die Benutzeraktion reagiert.

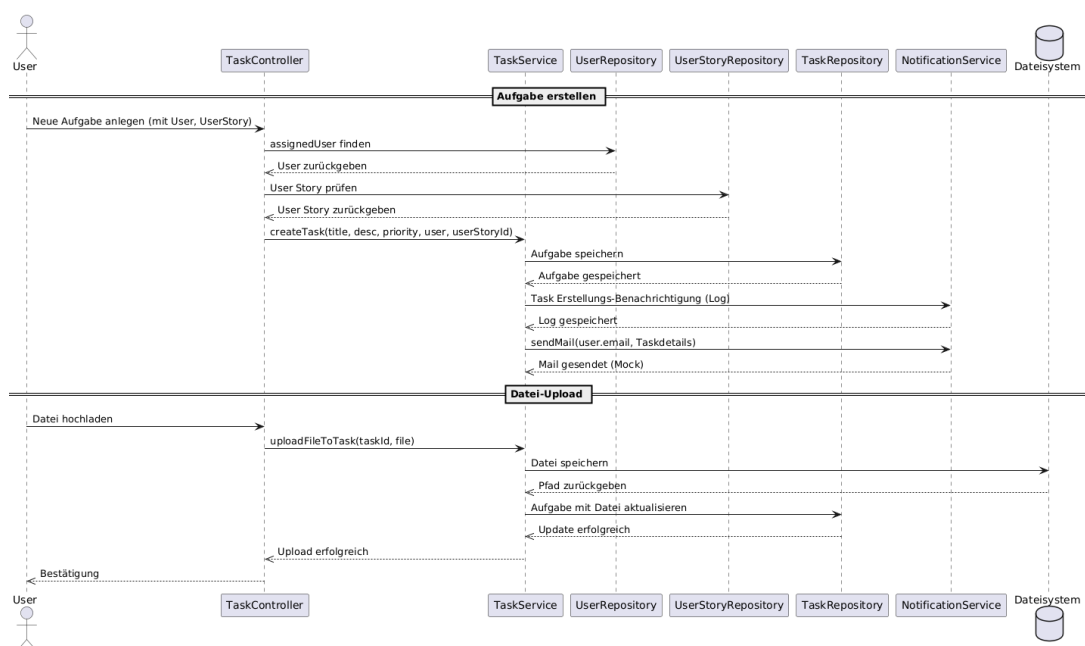
Das Diagramm spiegelt die tatsächliche Implementierung im Code wider und stellt sicher, dass die Modellierung konsistent mit der Systemarchitektur ist.

US-6.1, US-3.1, US-3.2, US-10.1

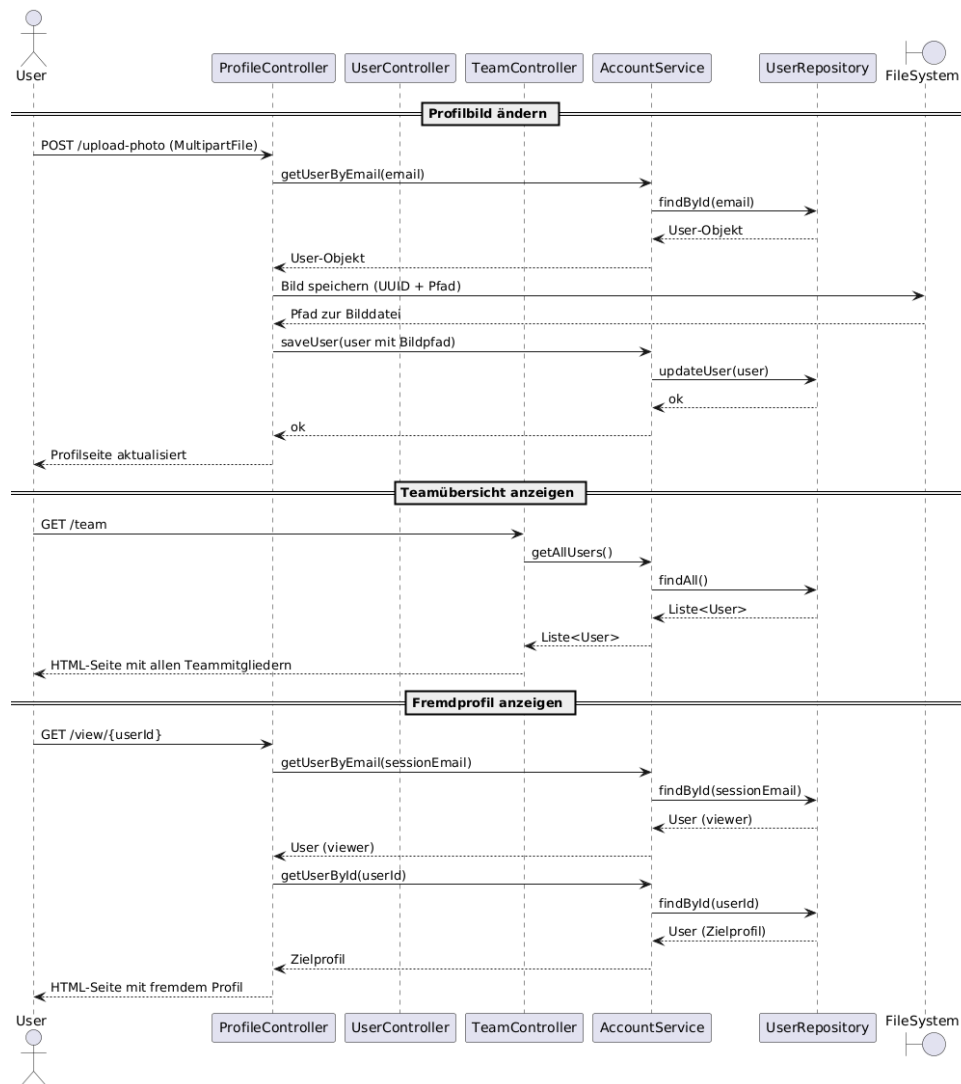
Das folgende Sequenzdiagramm beschreibt die zentralen Abläufe, die im Rahmen von Sprint 3 implementiert wurden. Es zeigt die Erstellung einer neuen Aufgabe durch einen Benutzer, einschließlich der Zuweisung eines Verantwortlichen (US-6.1) und der Verlinkung mit einer User Story (US-3.2).

Nach dem Speichern der Aufgabe wird eine Systembenachrichtigung geloggt und eine simulierte E-Mail an den zugewiesenen Benutzer gesendet (US-10.1). Darüber hinaus wird der Prozess des Datei-Uploads zu einer Aufgabe dargestellt (US-3.1).

Das Modell abstrahiert vom Implementierungsdetail und fokussiert sich auf den wesentlichen Informationsfluss zwischen Benutzer, Controllern, Services und Repositories.



US-5.1, US-5.2 - User Profile



Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde erstellt, um die Umsetzung der User Stories US-5.1 und US-5.2 im Bereich der Nutzerprofile darzustellen. Es zeigt die wichtigsten Interaktionen beim Anzeigen von Teammitgliedern, dem Öffnen fremder Profile sowie dem Hochladen eines neuen Profilbildes durch den Nutzer selbst. Ziel des Diagramms ist es, die Trennung der Verantwortlichkeiten zwischen den Komponenten ProfileController, TeamController, AccountService, UserRepository und dem FileSystem aufzuzeigen.

Besonders deutlich wird durch das Diagramm, wie Benutzeraktionen über definierte HTTP-Endpunkte verarbeitet und anschließend systematisch über die Service- und Persistenzschicht bearbeitet werden. Zusätzlich wird sichtbar, wie externe Ressourcen (z.B. das Dateisystem beim Profilbild-Upload) integriert sind. Die Modellierung spiegelt die tatsächliche Implementierung im Code wider und ist konsistent mit der geplanten Architektur des Systems.

2.2 Tracing der User Storys und relevante Anforderungen

Table 2: Tracing der User-Storys und relevante Anforderungen

Klasse	Verantwortlichkeit/Funktion	Relevante User Story
Task	Speicherung der Aufgabendetails inkl. User-Zuweisung	US-6.1, US-3.1, US-3.2
User	Repräsentation von Nutzern, die Aufgaben zugeordnet werden	US-6.1, US-5.1, US-5.2
UserStory	Verknüpfung von Aufgaben mit User Stories	US-3.2
TaskFile	Verwaltung und Verknüpfung von Dateiuploads mit Aufgaben	US-3.1
NotificationService	(Mock) Senden von Benachrichtigungen bei Aufgabenerstellung	US-10.1
TaskController	Steuerung der User-Interaktionen mit Aufgaben	US-6.1, US-3.1, US-3.2, US-10.1
UserStory	Erstellung und Priorisierung von User Stories im Backlog	US-2.1, US-2.2
TeamController	Anzeige aller Teammitglieder auf einer Teamseite	US-5.1, US-5.2
AccountService	Verwaltung der kompletten Nutzerlogik: Rollenwechsel, Profilspeicherung und Passwortänderung usw	US-5.1, US-5.2
UserController	Steuerung der Interaktion zur Anzeige/Bearbeitung von Nutzerprofilen, inklusive Rollenänderung	US-5.1, US-5.2
ProfileController	Verarbeitung von Profil-spezifischen Aktionen: Anzeige des eigenen Profils, Bild-Upload, Bio- und Passwortänderung	US-5.1, US-5.2
UserRepository	Zugriffsschicht auf die gespeicherten User-Daten, z.B. Benutzer suchen, speichern, aktualisieren	US-5.1, US-5.2



Detaillierte Beschreibung der Implementierung

Im Rahmen der Sprint 3 wurden die folgenden Klassen gezielt modelliert und erweitert:

- **Task:** Die Klasse **Task** wurde um die Attribute **assignedUser** und **userStoryId** erweitert, um eine Aufgaben-Zuweisung zu Nutzern und User Stories zu ermöglichen (US-6.1, US-3.2).
- **User:** Die Klasse **User** repräsentiert die Nutzer, denen Aufgaben zugeordnet werden können (US-6.1).
- **UserStory:** Durch die Einführung von **userStoryId** in der **Task**-Klasse wurde die Verbindung zu User Stories hergestellt (US-3.2).
- **TaskFile:** Für das Dateiupload-Feature wurde die Klasse **TaskFile** modelliert, um Dateien Aufgaben zuzuordnen (US-3.1).
- **NotificationService:** Ein einfacher (Mock-)Service zur Simulation von Benachrichtigungen bei der Erstellung neuer Aufgaben wurde implementiert (US-10.1).
- **TaskController:** Diese Controller-Klasse wurde erweitert, um die neuen Features der Aufgabenverwaltung (User-Zuweisung, User Story-Verlinkung, Dateiupload, Benachrichtigungen) über HTTP-Endpunkte bereitzustellen (alle User Stories).
- **UserStory:** Durch die Einführung von **userStoryForm** und den **BacklogController** wurde die Erstellung, Bearbeitung und Priorisierung von User Stories realisiert (US-2.1, US-2.2). Die Klasse

`UserStory` wurde um Attribute wie `beschreibung`, `priorität`, `createdAt` und `updatedAt` erweitert. Zusätzlich wurde ein Enum `Priorität` eingeführt, um konsistente Prioritätsstufen zu ermöglichen. Über den `BacklogService` wird die Geschäftslogik gekapselt und das `UserStoryRepository` ermöglicht den Zugriff auf persistierte Daten.

- **AccountService:** Die Klasse `AccountService` kapselt die gesamte Geschäftslogik zur Verwaltung von Benutzerkonten. Dazu gehören die Registrierung neuer Nutzer, die Authentifizierung, die Änderung von Passwörtern, die Rollenverwaltung sowie die Aktualisierung von Profilinformationen wie Name, E-Mail, Biografie und Profilbild. Sie übernimmt die Kommunikation mit der Persistenzschicht und stellt sicher, dass alle Änderungen validiert und korrekt gespeichert werden.
- **ProfileController:** Die Klasse `ProfileController` verarbeitet alle HTTP-Anfragen, die das eigene Benutzerprofil betreffen. Dazu gehören das Anzeigen des Profils, das Hochladen eines neuen Profilbildes, das Ändern der Biografie sowie das sichere Ändern des Passworts. Die Geschäftslogik wird an den `AccountService` delegiert.
- **UserController:** Die Klasse `UserController` ist für die Anzeige und Bearbeitung von Benutzerprofilen zuständig. Neben der Anzeige des eigenen Profils ermöglicht sie auch die Bearbeitung von Namen und E-Mail-Adressen sowie – durch Administratoren – die Änderung der Benutzerrollen. Sie greift dabei auf den `AccountService` zurück.
- **TeamController:** Die Klasse `TeamController` ist zuständig für die Anzeige aller Teammitglieder in einer übersichtlichen Darstellung. Über den Team-Endpoint kann der Nutzer alle registrierten Benutzer samt Rolle einsehen. Die benötigten Daten werden über den `AccountService` bereitgestellt.
- **UserRepository:** Die Klasse `UserRepository` stellt eine Abstraktionsschicht für den Datenzugriff dar. Sie ermöglicht das Finden, Speichern und Aktualisieren von Benutzerdaten basierend auf E-Mail oder ID. Sie wird direkt vom `AccountService` verwendet und trennt die Geschäftslogik klar von der Datenhaltung.

3.1 Updates zu genutzten Technologien

Im Rahmen des dritten Sprints wurden keine neuen Technologien eingeführt. Bestehende Technologien wie Spring Boot, Thymeleaf und JPA wurden weiterverwendet und gezielt erweitert, um die neuen Anforderungen effizient umzusetzen.



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: Wir wollten von Anfang an sicherstellen, dass unsere Kernfunktionen sowohl auf Service-Ebene als auch auf API-Ebene korrekt funktionieren. Daher haben wir sowohl Unit-Tests als auch REST-Tests geschrieben:

- **White-box-Tests:** direkt im Service (`TaskService`, `AccountService`), um die Logik zu prüfen
- **Black-box-Tests:** via `MockMvc` für die REST-Controller (`TaskRestController`, `AccountRestController`), um die API wie ein Nutzer zu testen

Testübersicht:

Test-ID	Zeitpunkt	Herkunft / Beschreibung	Ergebnis
TC6	10.06.2025	Unit-Test: <code>AccountService.registerUser()</code> mit gültigen Daten	Pass
TC7	10.06.2025	Unit-Test: <code>AccountService.registerUser()</code> mit existierender E-Mail	Pass
TC8	10.06.2025	Unit-Test: <code>AccountService.login()</code> mit gültigen Anmeldedaten	Pass
TC9	10.06.2025	Unit-Test: <code>AccountService.login()</code> mit ungültigen Anmeldedaten	Pass
TC10	10.06.2025	Unit-Test: <code>AccountService.changePassword()</code> mit gültigem alten Passwort	Pass
TC11	10.06.2025	Unit-Test: <code>AccountService.changePassword()</code> mit falschem alten Passwort	Pass
TC12	10.06.2025	REST-Test: <code>POST /api/account/register</code> mit gültigen Parametern	Pass
TC13	10.06.2025	REST-Test: <code>POST /api/account/register</code> mit existierender E-Mail	Pass
TC14	10.06.2025	REST-Test: <code>POST /api/account/login</code> mit gültigen Anmeldedaten	Pass
TC15	10.06.2025	REST-Test: <code>POST /api/account/login</code> mit ungültigen Anmeldedaten	Pass
TC16	10.06.2025	REST-Test: <code>POST /api/account/changePassword</code> mit gültigen Daten	Pass
TC17	10.06.2025	REST-Test: <code>POST /api/account/changePassword</code> mit falschem alten Passwort	Pass

Table 3: Testübersicht: Gesamtübersicht Task- und Account-Tests

Ablageort der Tests:

- `src/test/java/com/example/demo/service/TaskServiceTest.java`
- `src/test/java/com/example/demo/controller/TaskRestControllerTest.java`
- `src/test/java/com/example/demo/service/AccountServiceTest.java`
- `src/test/java/com/example/demo/controller/AccountRestControllerTest.java`

Reflexion zur Teststrategie:

Automatisierte Tests geben uns Sicherheit bei Änderungen am Code. Durch die Kombination aus Unit-Tests für die Service-Ebene (`TaskService`, `AccountService`) und REST-Tests für die API-Ebene (`TaskRestController`, `AccountRestController`) 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 händisch getestet, ob man Benutzer korrekt registrieren, anmelden und Aufgaben verwalten kann. Dabei wurden sowohl Fehlerfälle als auch normale Abläufe geprüft.

Testergebnisse:

Rolle	Ziel des Tests	Version	Ergebnis
Teammitglied	Test der Validierung (leeres Feld bei Task-Eingabe)	UI vom 01.05.2025	Pass
Teammitglied	Anzeige der Taskliste mit leerer DB	UI vom 01.05.2025	Pass
Teammitglied	Nach Registrierung wird eine Bestätigungs-Mail versendet	UI vom 10.06.2025	Pass
Teammitglied	Passwort vergessen“ löst eine Mail mit Link zum Zurücksetzen aus	UI vom 10.06.2025	Pass
Teammitglied	Passwort lässt sich erfolgreich über den Link in der Mail ändern	UI vom 10.06.2025	Pass

Table 4: Testergebnisse: Manuelle Tests

3.3 Tracing

In unserem Projekt wurde auch in Sprint 3 ein systematisches Tracing zwischen den UML-Diagrammen und der Code-Implementierung durchgeführt. Dafür wurde eine ausführliche Tabelle erstellt, die die Zuordnung zwischen UML-Komponenten und den entsprechenden Code-Klassen sowie -Interfaces dokumentiert.

Diese Tabelle gewährleistet eine präzise Nachvollziehbarkeit zwischen der Modellierung und der tatsächlichen Implementierung – zum Beispiel für Klassen wie `TaskService`, `NotificationService` oder `TaskController`, die in diesem Sprint entwickelt oder erweitert wurden.

GitLab-Link zur Tracing-Dokumentation: [Link](#)

- **Abgedeckte UML-Diagramme:**

- Klassendiagramm (US-6.1, US-3.1, US-3.2, US-10.1, US-2.1, US-2.2, US-5.1, US-5.2)
- Sequenzdiagramm (US-6.1, US-3.1, US-3.2, US-10.1, US-2.1, US-2.2, US-5.1, US-5.2)

3.4 Laufender Prototyp

GitLab-Link zum Prototyp: [Link](#)

Folgende neue Funktionalitäten wurden in Sprint 3 implementiert:

Benutzerperspektive

Beim Erstellen einer neuen Aufgabe kann ein zuständiger Nutzer über ein Dropdown-Menü ausgewählt werden (US-6.1). Nach der Erstellung besteht die Möglichkeit, Dateien hochzuladen (US-3.1) und die Aufgabe mit einer User Story zu verlinken (US-3.2). Zudem wird beim Erstellen automatisch eine Notification (als Log-Eintrag) für den zuständigen Nutzer generiert (US-10.1). Beim Erstellen einer neuen *User Story* (US-2.1) kann der Benutzer Titel und Beschreibung eingeben. Bestehende Stories können anschließend bearbeitet und priorisiert werden (US-2.2).

Beispielhafte Screenshots

Screenshots zur Veranschaulichung der implementierten Funktionalitäten befinden sich im Anhang:

- Aufgabenübersicht mit zugewiesenem Nutzer und Datei-Upload
- Datei-Upload-Formular
- Log-Ausgabe der Benachrichtigung

Installation und Kompilation

Die Installations- und Kompilationsanleitung ist unverändert und weiterhin gültig. Eine detaillierte Schritt-für-Schritt-Anleitung zur Installation und Ausführung befindet sich in der README-Datei: [README.md](#).



3.5 Abweichung Sprintplanung

Während des dritten Sprints wurden zusätzliche Aufgaben durchgeführt, die ursprünglich nicht in der Sprint-Planung enthalten waren:

- **Backend-Validierung der User Story:** Überprüfung, ob die ausgewählte User Story tatsächlich existiert, zusätzlich zum Frontend-Dropdown.
- **Implementierung eines echten Datei-Uploads:** Speicherung hochgeladener Dateien über multipartFile statt nur Eingabe von Dateiname und Typ.
- **Download-Funktion für hochgeladene Dateien:** Ermöglicht das Herunterladen der Dateien mit ihrem ursprünglichen Dateinamen.
- **Fehlerbehandlung beim Datei-Upload:** Behandlung von fehlerhaften Uploads und nicht existierenden Verzeichnissen.
- **Erweiterung der Aufgabenübersicht-UI:** Darstellung der hochgeladenen Dateien und der Download-Links in der Übersicht.
- **Log-Benachrichtigung für Task-Zuweisung:** Einführung von Logging bei der Zuweisung von Aufgaben zur besseren Nachvollziehbarkeit.
- **Manuelle Session-Validierung in Profilooperationen:** Zur Sicherstellung, dass Benutzer nur auf eigene Profilfunktionen zugreifen können, wurde eine zusätzliche Session-Validierung in mehreren Controllern implementiert.
- **UUID-basierter Dateiname beim Bild-Upload:** Zur Vermeidung von Dateikonflikten wurde beim Profilbild-Upload die automatische Erzeugung eines eindeutigen Dateinamens (UUID) nachträglich ergänzt.
- **Sichtschutz beim Aufruf fremder Profile:** Eine Kontrollstruktur wurde ergänzt, damit ein Nutzer sein eigenes Profil nicht über den Fremdprofil-Endpunkt anzeigen kann (Vermeidung von Dopplung und Verwirrung).
- **Test-Endpunkt zur Prüfung von Dateiooperationen:** Ein zusätzlicher Endpunkt wurde eingeführt, um Upload-/Schreibrechte im Projektverzeichnis zu prüfen. Dies diente vor allem der Fehlerdiagnose bei Problemen mit Dateispeicherung auf dem Server.

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 3	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-6.1-T1	Erweiterung der Task-Klasse um assignedUserId (User-Zuweisung)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-6.1-T2	Dropdown-Auswahl für Nutzer im Aufgaben-Formular	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-6.1-T3	Validierung der Nutzer-Zuweisung (Backend)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-3.1-T3	Datei-Upload-Feature: Hochladen und Verknüpfen von Dateien (TaskFile)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-3.1-T4	Datei-Download-Feature: Hochgeladene Dateien herunterladen	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-3.2-T1	Erweiterung der Task-Klasse um userId (Verlinkung mit User Story)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-3.2-T2	Dropdown-Auswahl für User Story im Aufgaben-Formular	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-10.1-T1	Implementierung eines Mock-Notification-Services (Log-Ausgabe)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-2.1-T1	Erstellung von User Stories im Backlog (Formular, Speicherung, Übersicht)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-2.2-T1	Bearbeitung und Priorisierung von User Stories (Update, Validierung)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-5.1-T1	Implementierung der Teamseite (TeamController, Übersicht aller Nutzer)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-5.1-T2	Implementierung der Profilansicht für fremde Nutzer (/view/userId)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-5.1-T3	HTML-Oberfläche zur Anzeige von Rollen, Namen und Bio anderer Teammitglieder	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-5.2-T1	Erstellung der Profilseite zur Anzeige eigener Nutzerdaten	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-5.2-T2	Implementierung des Uploads für Profilbilder (inkl. Pfadspeicherung)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-5.2-T3	Backend-Logik und Formular zur Aktualisierung der Biografie	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
US-5.2-T4	Änderung des Profilbildes durch Datei-Upload und Aktualisierung im Benutzerprofil	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
Bonus-T1	Fehlerbehandlung beim Datei-Upload (IOException Handling)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils
Bonus-T2	UI-Erweiterung der Aufgabenübersicht (Dateiliste und Download-Links)	Dogukan, Helin, Hüseyin, Cagla	25% jeweils

Table 6: Beitrag der Teammitglieder zu den im Sprint 3 erledigten Tasks

✓

Super!

20+2/20

Anhang – Screenshots

Aufgabenübersicht										
← Zurück zur Startseite		+ Neue Aufgabe erstellen								
Titel	Beschreibung	Priorität	Status	Zugewiesen an	Erstellt am	User Story	Dateien	Aktion	Verantwortlichen ändern	Datei hinzufügen
Testaufgabe	Dies ist eine Beispielaufgabe.	MEDIUM	COMPLETED	Dogukan	30.04.2025 17:48	—	<ul style="list-style-type: none"> demo.pdf (kein Pfad) demo.pdf 	↗ Wieder öffnen	Dogukan ▼	Datei auswählen K...t Datei hochladen

Figure 6: Aufgabenübersicht mit Datei-Upload und User-Zuweisung

Datei hinzufügen




Figure 7: Datei-Upload-Formular

```

2025-06-06T17:25:32.673+02:00 INFO 19056 --- [nio-8080-exec-3] c.e.demo.service.NotificationService : Task created: [43] Log-Benachrichtigung
2025-06-06T17:25:32.673+02:00 INFO 19056 --- [nio-8080-exec-3] c.e.demo.service.NotificationService : New notification to 60_schlapphut_chrom@icloud.com: You were assigned to task: Log-Benachrichtigung
  
```

Figure 8: Logausgabe der Benachrichtigung beim Erstellen einer Aufgabe

User Stories

ID	Titel	Status	Aktion
Noch keine User Stories vorhanden.			

Figure 9: Leere Backlog-Übersicht mit Eingabefeld zur Erstellung einer neuen User Story (US-2.1)