Dokumentation Sprint 3

Scrum Board Projekt

Bearbeitet von:
Eren Temizkan
Matrikelnummer: 223201982
Gruppe 3

Universität Rostock
Sommersemester 2025

Inhaltsverzeichnis

1 Einleitung

Im dritten Sprint ging es um einen wichtigen Schritt in der Weiterentwicklung des Scrum Board Projekts. Nachdem im zweiten Sprint bereits zentrale Funktionen wie das Account- und Task-Management umgesetzt worden waren, lag der Schwerpunkt dieses Sprints auf organisatorischen und kollaborativen Erweiterungen. Im Rahmen dessen wurden sämtliche User Stories von US-4.1 bis US-10.1 erfolgreich realisiert.

Trotz reduzierter Teamaktivität im weiteren Projektverlauf konnten alle geplanten Aufgaben rechtzeitig abgeschlossen und die vorgesehenen Funktionalitäten vollständig integriert werden.

Der Fokus dieses Sprints lag vorrangig auf der Back-End-Logik, auch wenn das Front-End im Verlauf ebenfalls auch mehrere neue Features erhalten hat.

2 1.1 Änderungen an den User Stories

Im Laufe des Sprints ergaben sich einige wichtige Änderungen und Konkretisierungen der ursprünglichen User Stories. Diese Änderungen basieren auf technischen Erkenntnissen, Usability-Überlegungen sowie Rückmeldungen aus der Testphase.

- **US-1.3 – Passwort zurücksetzen:** Ursprünglich war nur vorgesehen, dass Nutzer ihr Passwort zurücksetzen können. Im Sprint wurde jedoch festgestellt, dass ein vollständiger Token-basierter Workflow nötig ist, inklusive Formularanzeige, Token-Validierung und Passwortänderung.
  ⇒ Die Story wurde erweitert, um realistische Sicherheitsanforderungen (Token, Ablaufzeit, Verknüpfung mit E-Mail-Adresse) zu berücksichtigen.

- **US-4.4 – Aufgaben verschieben:** Anfänglich war geplant, dass Aufgaben nur zwischen festen Listen (Backlog → Sprint Backlog → Done) verschoben werden. Im Sprint zeigte sich jedoch, dass eine flexible, statusbasierte Verschiebung (z.B. „in Bearbeitung“, „unter Review“) sinnvoller ist.
  ⇒ Die Story wurde erweitert, um generische Statuslisten zu unterstützen.

- **US-5.3 – Rollenverwaltung:** Ursprünglich war nur eine manuelle Rollenzuweisung über die Datenbank angedacht. Während der Entwicklung wurde entschieden, diese Funktion auch über die Oberfläche verfügbar zu machen, z.B. in einem Modal-Fenster.
  ⇒ Die Story wurde um eine Benutzeroberfläche für Rollenzuweisungen ergänzt.

- **US-6.1 – Nutzer-Task-Zuordnung:** Während der Planung war nur eine 1-zu-1-Zuordnung vorgesehen. Im Sprint wurde jedoch erkannt, dass Pair Programming oder Gruppenarbeit eine Mehrfachzuordnung erfordern.
  ⇒ Die Story wurde entsprechend erweitert, um mehreren Nutzern eine Task zuweisen zu können.

- **US-10.1 – Benachrichtigungen:** Die ursprüngliche Idee sah E-Mail-Benachrichtigungen vor. Im Sprint zeigte sich jedoch, dass ein In-App-System mit visuellem Feedback (Badges, Toast-Nachrichten) besser zur Nutzererfahrung passt.
  ⇒ Fokus der Story wurde auf UI-basierte Benachrichtigungen verschoben.

✓

3 Begründung und Sprint-Planung

1+1/2

Im dritten Sprint lag der Fokus darauf, zentrale organisatorische Funktionen in das Scrum Board zu integrieren, damit die Zusammenarbeit im Team strukturiert und effizient unterstützt wird. Dafür wurden besonders solche Features ausgewählt, die direkt zur Aufgabenzuweisung, Statusverfolgung und Kommunikation zwischen Teammitgliedern beitragen.

Konkret ging es um die Umsetzung folgender Funktionen:

- Aufgabenstatus ändern und anzeigen,
- Aufgaben bestimmten Nutzer:innen zuweisen,
- Aufwand für Aufgaben schätzen und dokumentieren,
- Benachrichtigungen bei Änderungen versenden,
- sowie eine Filter- und Suchfunktion für Aufgaben integrieren.

Die Planung erfolgte so, dass zuerst die grundlegenden Strukturen geschaffen wurden – also die neuen Entitäten wie `Estimation`, `Notification` und `TaskAssignment`. Dadurch konnte sichergestellt werden, dass spätere Funktionen wie die Benachrichtigungslogik oder Filteroptionen auf einer stabilen Datenbasis aufbauen.

Die Umsetzung erfolgte bewusst in kleinen, abgeschlossenen Schritten, sodass jede Funktion direkt nach der Implementierung getestet und überprüft werden konnte. Auf diese Weise entstand nach und nach ein stabiles und zuverlässiges System.

Da der Sprint von einer einzelnen Person durchgeführt wurde, war eine klare Priorisierung besonders wichtig. Die Aufgaben wurden daher so geplant, dass sie nacheinander und ohne große Abhängigkeiten umgesetzt werden konnten. Dadurch ließ sich die verfügbare Zeit gut einteilen, und es war möglich, flexibel auf neue Anforderungen oder kleinere Probleme zu reagieren.

Der Ablauf des Sprints gliederte sich in vier Phasen:

1. Modellierung der benötigten Datenklassen und Anpassung des Datenmodells,
2. Entwicklung der zugehörigen Services und Controller,
3. Erweiterung des Frontends mit Thymeleaf, Bootstrap und JavaScript,
4. abschließend Tests zur Sicherstellung der Funktionalität und Qualität.

Durch diese strukturierte Vorgehensweise konnte sichergestellt werden, dass die wichtigsten Funktionen wie geplant umgesetzt und in das bestehende System integriert wurden.

colortbl

4 Aufwandschätzung

Diese folgende Tabelle zeigt eine geschätzte Aufwandseinschätzung (Story Points) für die einzelnen User Stories. Die Farben verdeutlichen den geschätzten Schwierigkeitsgrad:
gering (1–4 SP) mittel (5–7 SP) hoch (8+ SP)

ID	Beschreibung	Story Points
US-4.1	Listen für Tasks und User Stories anlegen	5
US-4.2	Aufgabenstatus definieren	3
US-4.3	Aufgabenstatus ändern können	5
US-4.4	Tasks zwischen Listen verschieben	8
US-4.5	Filter- und Sortierfunktionen für Listen	8
US-5.1	Nutzerprofile einsehen	3
US-5.2	Eigenes Profil bearbeiten	5
US-5.3	Rollen und Rechte ändern (Admin-Funktion)	8
US-6.1	Nutzer einzelnen Tasks zuordnen	5
US-6.2	Übersicht zu Nutzer-Task-Zuweisungen	5
US-7.1	Mehrere Nutzer für eine Task zuweisen	5
US-7.2	Übersicht über Pair-Programming-Tasks	3
US-8.1	Aufwandsschätzung pro Task anlegen	3
US-8.2	Tatsächliche Bearbeitungszeit dokumentieren	5
US-8.3	Abweichungen zwischen Aufwand und Zeit visualisieren	8
US-9.1	Aufgaben nach Priorität filtern	3
US-9.2	Aufgaben nach Status filtern	3
US-9.3	Zuständige Person anzeigen	3
US-10.1	E-Mail/In-App-Benachrichtigungen bei Änderungen	8
	Gesamtsumme	**98 SP**

5 Modellierte Klassen und ihre Verantwortlichkeiten

Comment

Typ: Entität

Verantwortlichkeiten:

- Speicherung von Kommentaren mit Text, Erstellungsdatum und Autor.
- Verknüpfung mit einer spezifischen Task über `task_id`.
- Nutzung durch `CommentController` und `CommentService`.

TaskFile

Typ: Entität

Verantwortlichkeiten:

- Repräsentation hochgeladener Dateien mit Dateiname, MIME-Typ und Pfad.
- Verknüpfung mit Aufgaben (`Task`).
- Darstellung und Downloadfunktion im Frontend.

PasswordResetToken

Typ: Entität

Verantwortlichkeiten:

- Speicherung von UUID-Token mit Ablaufdatum.
- Verbindung mit einem Benutzer zur Passwortzurücksetzung.
- Prüfung der Token-Gültigkeit im Passwort-Reset-Workflow.

UserDTO & RegisterRequest

Typ: Data Transfer Objects

Verantwortlichkeiten:

- `UserDTO`: Übertragung von sicheren, gefilterten Benutzerinformationen.
- `RegisterRequest`: Erfasst Registrierungsdaten (Name, E-Mail, Passwort, Rolle).

CommentService

Typ: Service

Verantwortlichkeiten:

- CRUD-Operationen für Kommentare.
- Filterung nach Task-ID, Validierung von Autorisierung.

TaskFileService

Typ: Service

Verantwortlichkeiten:

- Verarbeitung von Datei-Uploads und -Downloads.
- Verknüpfung der Datei mit der jeweiligen Aufgabe.

NotificationService

Typ: Service

Verantwortlichkeiten:

- Versand von E-Mail-Benachrichtigungen via SMTP bei Aufgabenänderungen.
- Unterstützt Ereignisse wie neue Aufgaben, Statusänderung oder Zuweisung.

AccountService

Typ: Service

Verantwortlichkeiten:

- Registrierung, Authentifizierung und Passwortzurücksetzung.
- Generierung und Validierung von Passwort-Reset-Tokens.

UserController

Typ: Controller

Verantwortlichkeiten:

- Verwaltung von Benutzerprofilen und Rollen.
- Aufruf von Services zur Passwortänderung und Profilbearbeitung.

PasswordResetController

Typ: Controller

Verantwortlichkeiten:

- Bereitstellung der API zur Anforderung eines Passwort-Reset-Links.
- Überprüfung der Token-Gültigkeit und Zurücksetzen des Passworts.

---

6 Verfeinerung der Interfaces zwischen den Komponenten

Um die Kommunikation zwischen den Komponenten effizient und skalierbar zu gestalten, wurden im dritten Sprint zusätzliche spezialisierte Interfaces eingeführt und mit Methoden versehen.

User Story – Interface Zuordnung

US-6: Datei-Upload zu Aufgaben Interface: `TaskFileService`

- `saveFile(MultipartFile file, Long taskId)` – Speichert eine Datei auf dem Server und erstellt die Metadaten in der Datenbank.

```
public void saveFile(MultipartFile file, Long taskId) throws IOException
    {
    String fileName = file.getOriginalFilename();
    Path targetLocation = Paths.get(uploadDir).resolve(fileName);
    Files.copy(file.getInputStream(), targetLocation, StandardCopyOption
        .REPLACE_EXISTING);

    TaskFile taskFile = new TaskFile(fileName, file.getContentType(),
        targetLocation.toString(), taskId);
    taskFileRepository.save(taskFile);
}
```

Listing 1: TaskFileService.java

US-7: Kommentare zu Aufgaben Interface: `CommentService`

- `addCommentToTask(Comment comment)` – Speichert einen Kommentar in der Datenbank und verknüpft ihn mit einer Aufgabe.

```
public void addCommentToTask(Comment comment) {
    commentRepository.save(comment);
}
```

Listing 2: CommentService.java

US-8: Projektbezogene Aufgabenverwaltung Interface: `ProjectService`

- `getTasksForProject(Long projectId)` – Gibt alle Aufgaben zurück, die einem bestimmten Projekt zugeordnet sind.

```
public List<Task> getTasksForProject(Long projectId) {
    return taskRepository.findByProjectId(projectId);
}
```

Listing 3: ProjectService.java

US-9: Subtasks zu Aufgaben Interface: `SubtaskService`

- `getSubtasksForTask(Long taskId)` – Gibt alle Subtasks zurück, die einer Hauptaufgabe zugeordnet sind.

```
public List<Subtask> getSubtasksForTask(Long taskId) {
    return subtaskRepository.findByTaskId(taskId);
}
```

Listing 4: SubtaskService.java

Diese Schnittstellenstruktur ermöglicht eine modulare Erweiterung und klare Trennung der Verantwortlichkeiten. Die gewählte Architektur erlaubt dabei eine gezielte Testbarkeit und vereinfachte Wartung jeder Funktionseinheit.

7 User Story – Aufgabenfilterung [US-9.1 – US-9.3]

Die Implementierung der Filter- und Suchfunktion ermöglicht eine prioritäts- und statusbasierte Darstellung sowie gezielte Suche nach zuständigen Nutzern. Dies verbessert die Übersichtlichkeit und Arbeitsverteilung erheblich.

8 Verantwortlichkeiten der Interfaces

- **CommentRepository:**
  Verwaltet Kommentare zu Aufgaben.
  Beispiel: `findByTask(task)` – Kommentare zu einer Aufgabe abrufen.
- **TaskFileRepository:**
  Speichert Datei-Metadaten zu Tasks.
  Beispiel: Standard-CRUD via `JpaRepository`.
- **TaskListRepository:**
  Zugriff auf Backlog- und Sprintlisten.
  Beispiel: `findAll()` – alle Listen abrufen.
- **SubtaskRepository:**
  Verknüpft Subtasks mit Hauptaufgaben.
  Beispiel: `findByTask(task)` – Subtasks einer Aufgabe.
- **PasswordResetTokenRepository:**
  Verwaltung von Reset-Tokens inkl. Ablauf.
  Beispiel: `findByToken(token)` – Token-Validierung.
- **ProjectRepository:**
  Zugriff auf Projektobjekte mit Aufgabenbezug.
  Beispiel: `findById(id)` – Projekt abrufen.
- **TaskRepository:**
  Filterung und Verwaltung von Aufgaben.
  Beispiel: `findByAssignedUserId(id)` – Aufgaben eines Nutzers.
- **UserRepository:**
  Zugriff auf Benutzerdaten.
  Beispiel: `findByEmail(email)` – Nutzer per E-Mail suchen.
- **EstimationRepository:**
  Zugriff auf Schätzungsdaten für Aufgaben.
  Beispiel: `findByTask(Task)` – Schätzung zur gegebenen Aufgabe abrufen.

9 Informationsbedarf der Komponenten

Im dritten Sprint wurden neue Datenstrukturen und Komponenten eingeführt oder signifikant erweitert, um die Anforderungen aus User Stories 4 bis 10 umzusetzen. Der Informationsbedarf dieser Komponenten ergibt sich aus ihrer spezifischen Funktion innerhalb des Systems:

- **Comment:** Speichert nutzerspezifische Kommentare zu Aufgaben. *Benötigte Felder:* `taskId`, `content`, `authorId`, `timestamp`.
  Verwendet durch: `CommentController`, `CommentService`, `CommentRepository`.
  Dient der Anzeige und Bearbeitung von Kommentaren im Detailbereich einer Aufgabe.

- **TaskFile:** Repräsentiert eine Datei inklusive Metadaten und Verknüpfung zur Aufgabe.
  Benötigte Felder: `filename`, `uploadDate`, `path`.
  Verwendet durch: `TaskFileService`, `TaskFileRepository`, `TaskFileController`.
  Ermöglicht Datei-Uploads über das Frontend sowie Anzeige/Download im Aufgaben-Detailfenster.

- **PasswordResetToken:** Temporäre Datenstruktur zur sicheren Passwortzurücksetzung über einen Token.
  Benötigte Felder: `token`, `user`, `expiryDate`.
  Verwendet durch: `PasswordResetService`, `PasswordResetController`, `PasswordResetTokenRepository`.
  Wird beim Anfordern und Einlösen von Reset-Links verwendet.

- **Notification:** Interne Nachrichtenstruktur zur Anzeige systembezogener Ereignisse im Frontend, z. B. *„Datei erfolgreich hochgeladen“* oder *„Kommentar hinzugefügt“*.
  Verwendet durch: `NotificationService`.
  Wird dynamisch erzeugt und nicht persistent gespeichert.

- **Project:** Wurde im Sprint 3 funktional erweitert zur Gruppierung von Aufgaben unter gemeinsamen Projekten.
  Benötigte Felder: `name`, `description`, `creatorId`.
  Verwendet durch: `ProjectService`, `ProjectController`, `TaskService`.

- **Frontend-Komponenten:** Neue Funktionen und Darstellungen in `taskdetails.html` und `tasks.html` benötigen synchronisierte Informationen aus den Controller-Schichten.
  Verwendet: `th:each`, `th:if`, REST-Integration mit JavaScript-Fetch-Calls.

Diese neuen Datenstrukturen ergänzen die im vorherigen Sprint etablierten Kernkomponenten. Sie ermöglichen eine feinere Benutzerinteraktion, unterstützen neue Funktionalitäten wie Datei-Upload, Kommentare, Passwortzurücksetzung und Projektstrukturierung.

Erweiterung des Klassendiagramms

Im aktuellen Sprint wurde das Klassendiagramm erweitert. Es umfasst nun zusätzliche Klassen wie `Comment`, `Estimation`, `TaskAssignment`, `Notification`, `TaskList` und `PasswordResetToken`. Zudem wurden alle zugehörigen Services und Repositories modelliert und Abhängigkeiten ergänzt.

Hinweis: Die meisten modellierten Klassen entsprechen in Name und Struktur direkt den implementierten Java-Klassen. In einigen Fällen (z. B. vereinfachte Namen im Klassendiagramm) kann es zu kleinen Abweichungen kommen. Die Zuordnung bleibt jedoch durch Struktur und Funktion eindeutig nachvollziehbar.

Besonders hervorzuheben ist die Einführung der Klasse `TaskAssignment`, die eine **m:n-Beziehung** zwischen `User` und `Task` korrekt abbildet. Alle unnötigen Beschriftungen wie `contains` wurden entfernt, um ein professionelles UML-Diagramm zu gewährleisten.

10 Datenpersistenz und Speicherung

Im dritten Sprint wurde die Applikation weiterhin mit einer lokalen **MySQL**-Datenbank betrieben. Dabei kam es mehrfach zu schwerwiegenden Problemen mit der Datenkonsistenz und Datenbankkorruption, insbesondere nach Systemabstürzen oder unerwarteten Neustarts. Die Erfahrung mit MySQL war daher geprägt von mehreren Debugging- und Wiederherstellungsversuchen, die tieferes technisches Verständnis und manuelle Eingriffe erforderten.

Datenbankprobleme mit MySQL

Die Datenbank wurde lokal unter Windows betrieben. Wiederholt traten folgende Fehlermeldungen auf:

```
ERROR 1017 (HY000): Can't find file: './db/tablename.frm' (errno:
13)
```

Dies war ein Hinweis auf beschädigte `.frm`- oder `.ibd`-Dateien. Die Ursache lag häufig an einem inkonsistenten Zustand im `data/`-Ordner von MySQL.

Manuelle Fehlerbehebung

Um die Datenbank wieder nutzbar zu machen, wurden folgende Debugging-Schritte mehrfach angewendet:

1. Aktivieren eines sicheren Startmodus durch Bearbeiten der Datei `my.ini`:

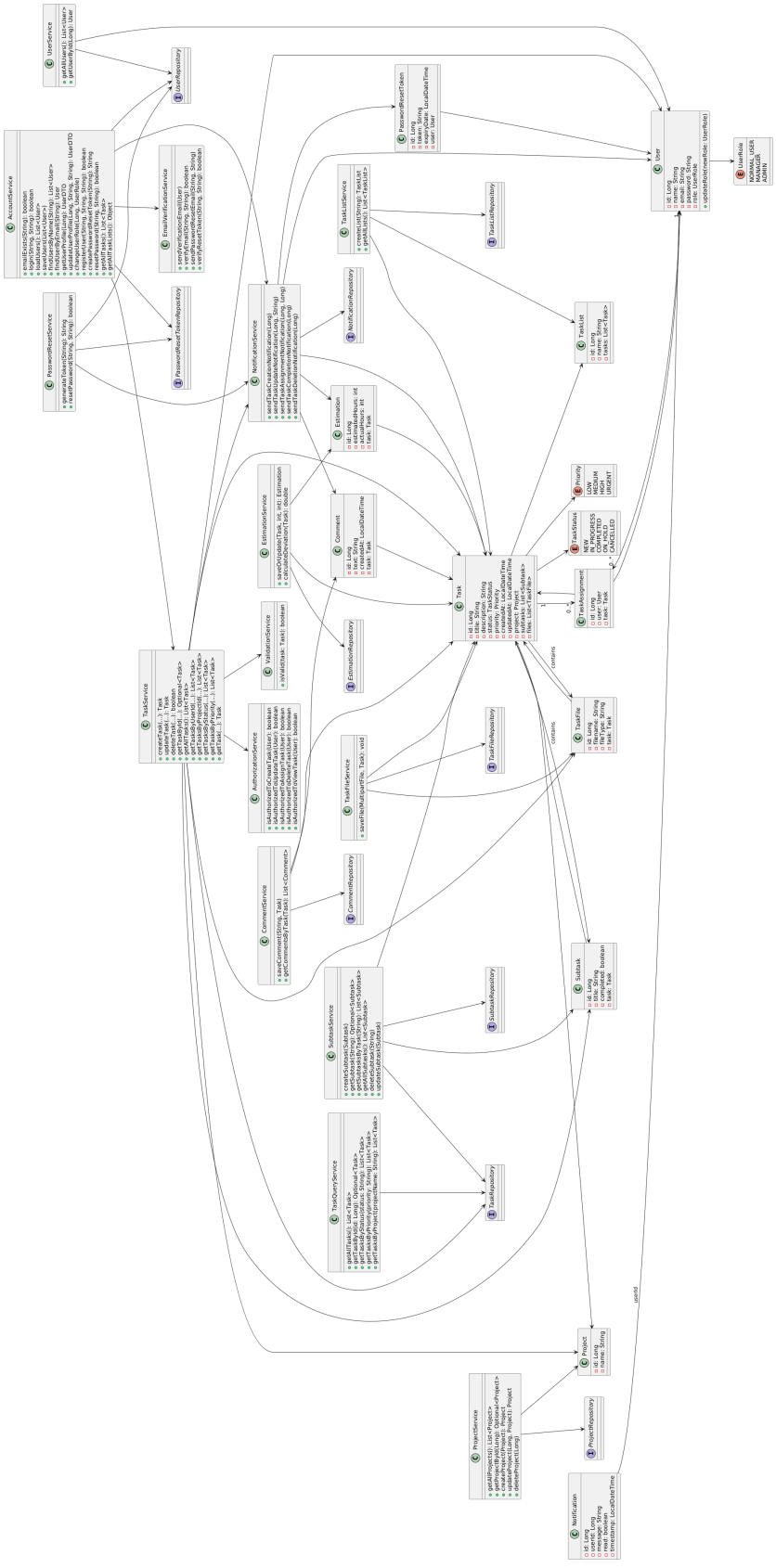


Abbildung 1: Erweitertes Klassendiagramm (Das Diagramm wird trotzt allem irgendwie komischerweise falsch positioniert)

```
[mysqld]
innodb_force_recovery = 1
```

2. Start des Servers mit reduziertem InnoDB-Recovery-Level:

```
mysqld --console --innodb_force_recovery=1
```

3. Exportieren der noch lesbaren Tabellen (falls möglich) über die Kommandozeile oder MySQL Workbench.

4. Wenn kein Export mehr funktionierte:

   - Stoppen des MySQL-Dienstes.
   - Vollständiges Löschen des Ordners:

     ```
     C:\ProgramData\MySQL\MySQL Server 8.0\data
     ```

   - Danach Neuinitialisierung mit:

     ```
     mysqld --initialize
     ```

   - Und Erstellung eines neuen Users:

     ```
     CREATE USER 'admin'@'localhost' IDENTIFIED BY 'passwort';
     GRANT ALL PRIVILEGES ON *.* TO 'admin'@'localhost';
     ```

Persistente Daten (in MySQL gespeichert)

Die folgende Logik wurde in Datenbanktabellen umgesetzt:

- **User**: Login-Informationen, Rollen, Name, Mail, Passwort-Hash
- **Task**: Titel, Beschreibung, Priorität, Status, Verknüpfung zu User und Projekt
- **Subtask**: Teilaufgaben zu einem Task, inkl. Status
- **TaskFile**: Metadaten zu Dateiuploads, inkl. Speicherpfad
- **Comment**: Kommentare zu Aufgaben, Autor und Inhalt
- **PasswordResetToken**: Token mit Ablaufdatum zur Passwortzurücksetzung
- **Estimation**: Aufwandsschätzungen mit Ist- und Soll-Werten

Vorteile und Herausforderungen

Vorteile:

- Gute Integration mit Spring Boot via JPA/Hibernate
- Vollständige Kontrolle über Tabellen, Daten und Benutzer

Herausforderungen:

- Manuelle Wartung notwendig bei MySQL-Korruption
- Kein automatisches Backup vorhanden
- Zeitintensive Wiederherstellungen nach Problemen

Alternative und Ausblick

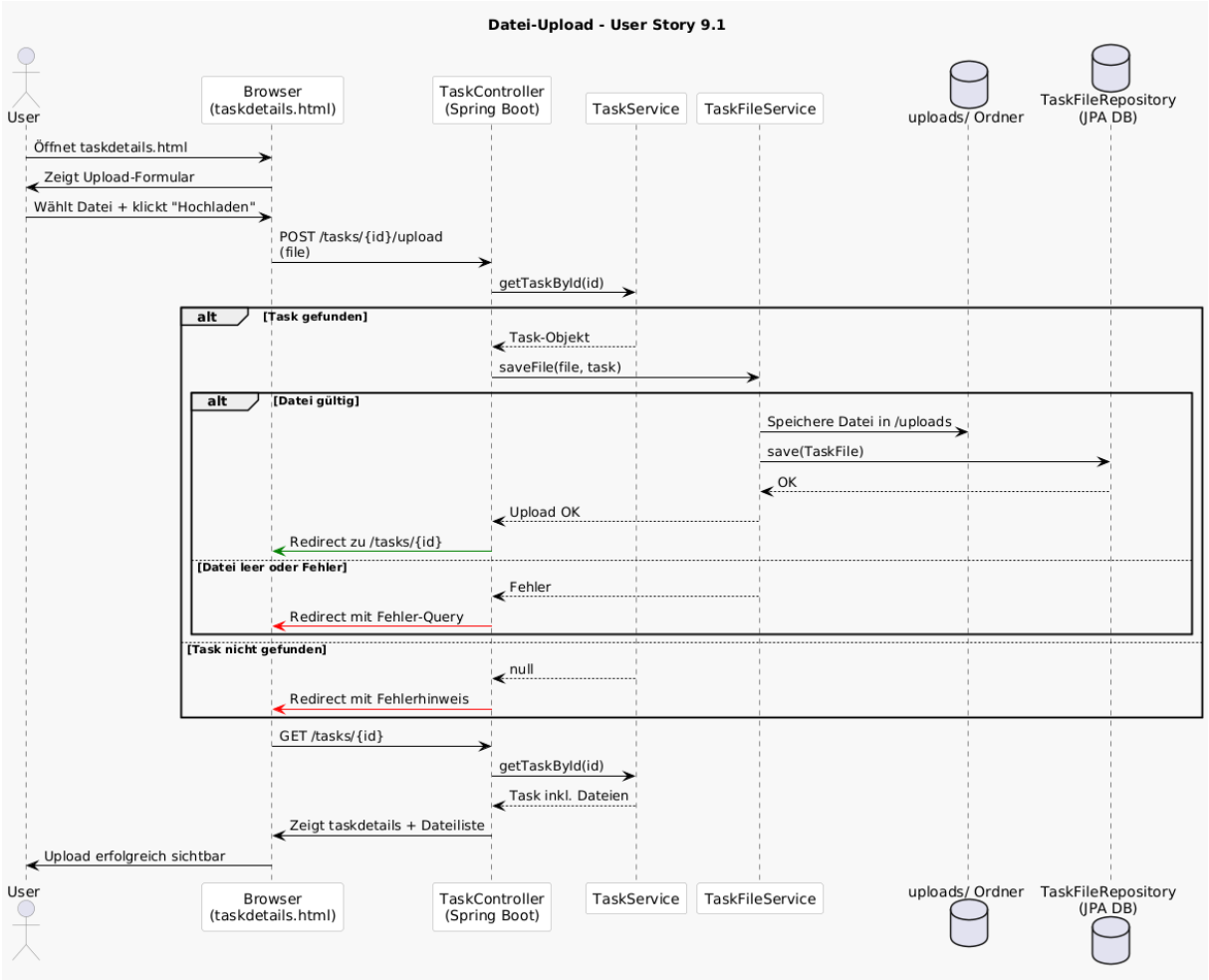
In solchen Fällen hätte sich der Einsatz einer virtualisierten Umgebung oder eines dedizierten Linux-Subsystems (z. B. Debian unter WSL oder VirtualBox) als deutlich robuster erwiesen. Bei einem vollständigen Systemabsturz unter Windows ist es vorgekommen, dass die MySQL-Datenbankstruktur inkonsistent wird oder vollständig zerstört wird — insbesondere wenn Schreibprozesse unterbrochen werden. In einer isolierten Umgebung wie einer virtuellen Maschine mit ext4-Dateisystem wären solche Schäden nicht so kritisch.

User Story 9.4 – Datei-Upload für Aufgaben

Hinweis: Diese User Story 9.4 ist im ursprünglichen Entwurf nicht enthalten, wurde jedoch hier implementiert, da ihre Funktionalität im implementierten System vorhanden ist.

Tabelle 1: User-Storys und relevante Anforderungen – Sprint 3

Klasse	**Verantwortlichkeit/Funktion**	**Relevante User Story**
TaskFileController	Verwaltung von Datei-Uploads und Verknüpfung zu Tasks	US-9.1, US-9.2, US-9.3
CommentController	Erstellung und Anzeige von Kommentaren	US-10.1
ProjectController	Erstellung und Verwaltung von Projekten	US-4.1, US-4.2
PasswordResetControlle	Passwort-Reset via Token und E-Mail	US-1.3
AuthController	Authentifizierung (Login, Logout)	US-1.2
ViewController	Darstellung von Formularen (Login, Registrierung, Reset)	US-1.1, US-1.3, US-9.1
NotificationService	Versenden von Benachrichtigungen (Mail)	US-10.1
TaskFileService	Verarbeitung und Speicherung von Datei-Metadaten	US-9.3
AccountService	Benutzer-Registrierung, Passwortverwaltung	US-1.1, US-1.2, US-1.3
TaskFileRepository	Speicherung von Task-bezogenen Dateien	US-9.3
CommentRepository	Speicherung von Kommentaren	US-10.1
PasswordResetToken	Modelliert Token zur Passwortzurücksetzung	US-1.3
RegisterRequest (DTO)	Datenübertragung bei Registrierung	US-1.1
UserDTO	Sichere Datenrückgabe (ohne Passwort)	US-1.2
TaskAssignment	Zuweisung von Nutzern zu Aufgaben (inkl. Pair Programming)	US-6.1, US-7.1
TaskAssignmentService	Validierung und Verwaltung von Zuweisungen	US-6.1, US-7.1, US-7.2
Estimation	Speicherung geschätzter und tatsächlicher Zeiten	US-8.1, US-8.2
EstimationService	Berechnung und Analyse von Aufwandabweichungen	US-8.1, US-8.2, US-8.3
Chart.js (Frontend)	Visualisierung der Abweichungen zwischen Schätzung und Realität	US-8.3



Hinweis: Die meisten modellierten Klassen entsprechen in Name und Struktur direkt den implementierten Java-Klassen. In einigen Fällen (z. B. vereinfachte Namen im Klassendiagramm) kann es zu kleinen Abweichungen kommen. Die Zuordnung bleibt jedoch durch Struktur und Funktion eindeutig nachvollziehbar.

Warum wurde dieses Sequenzdiagramm gewählt?

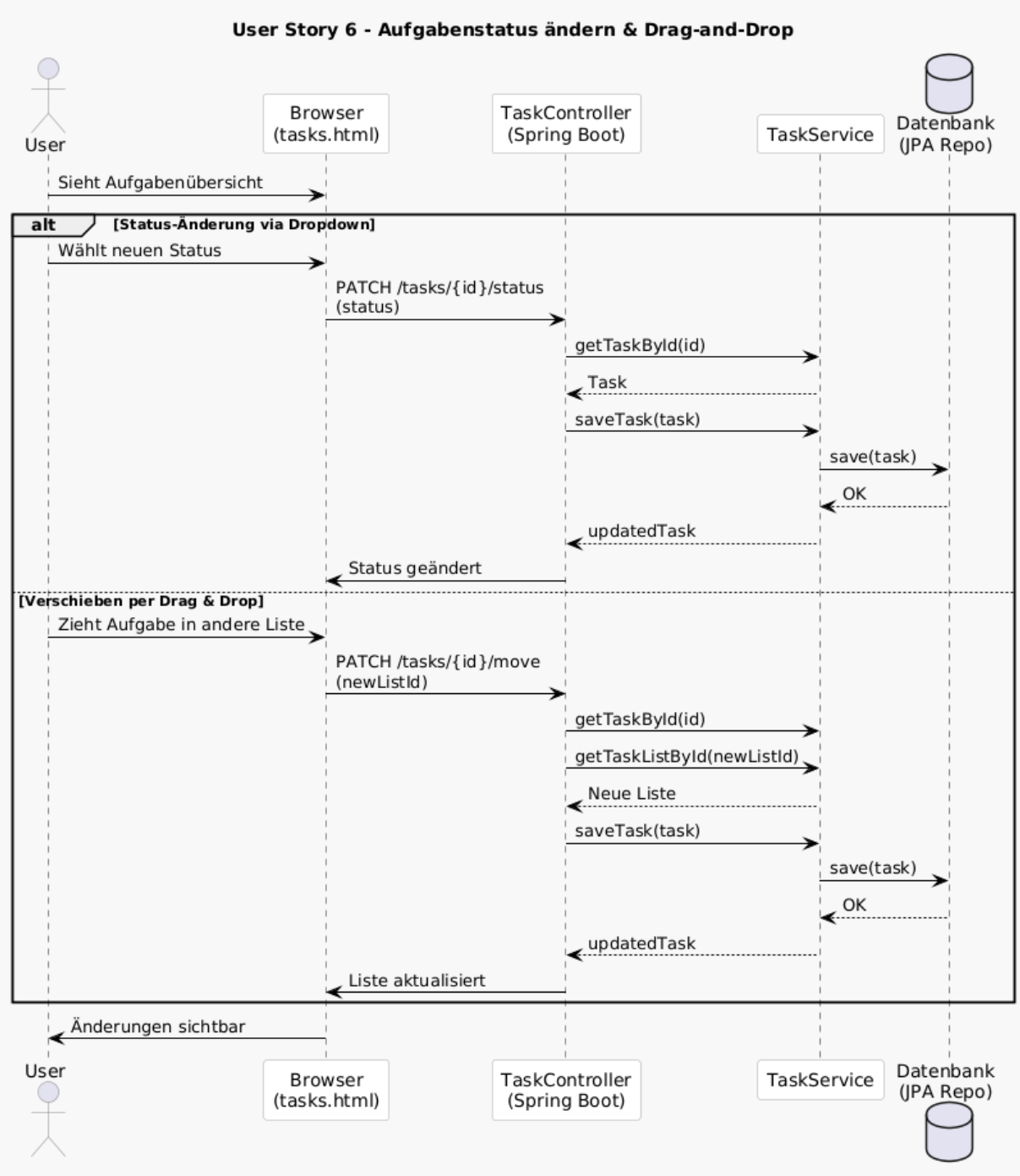
Dieses Sequenzdiagramm wurde erstellt, um den Datei-Upload-Prozess im Rahmen von **User Story 9.1** detailliert darzustellen. Es bildet die Interaktionen zwischen dem Benutzer, der Browser-Oberfläche (`taskdetails.html`), dem Spring-Boot-Controller (`TaskController`), den zugehörigen Services (`TaskService` und `TaskFileService`) sowie den Speicherkomponenten (dem `uploads/`-Verzeichnis und dem `TaskFileRepository`) ab.

Die Modellierung orientiert sich eng an der tatsächlichen Codeimplementierung der Datei-Upload-Funktion. Sie veranschaulicht präzise, wie eine Datei von der Benutzeroberfläche bis zur Speicherung auf der Festplatte und in der Datenbank durchgereicht wird. Dabei werden sowohl erfolgreiche Uploads als auch Fehlerfälle (z. B. leerer Task oder ungültige Datei) durch `alt`-Blöcke berücksichtigt.

Das Diagramm trägt zum besseren Verständnis der Systemarchitektur bei, indem es die Verantwortlichkeiten klar aufteilt: Der Controller orchestriert, die Services führen die Geschäftslogik aus, und Dateien werden getrennt im Dateisystem und in der Datenbank abgelegt. Diese Darstellung unterstützt die Nachvollziehbarkeit, Wartbarkeit und Doku-

mentation der Anwendung.

User Story 6.1 – Aufgabenstatus ändern Drag-and-Drop



Hinweis: Die meisten modellierten Klassen entsprechen in Name und Struktur direkt den implementierten Java-Klassen. In einigen Fällen (z. B. vereinfachte Namen im Klassendiagramm) kann es zu kleinen Abweichungen kommen. Die Zuordnung bleibt jedoch durch Struktur und Funktion eindeutig nachvollziehbar.

Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde erstellt, um die zwei möglichen Methoden zur Änderung des Aufgabenstatus bzw. zur Zuordnung zu einer anderen Taskliste zu modellieren: per Dropdown-Auswahl und per Drag-and-Drop.

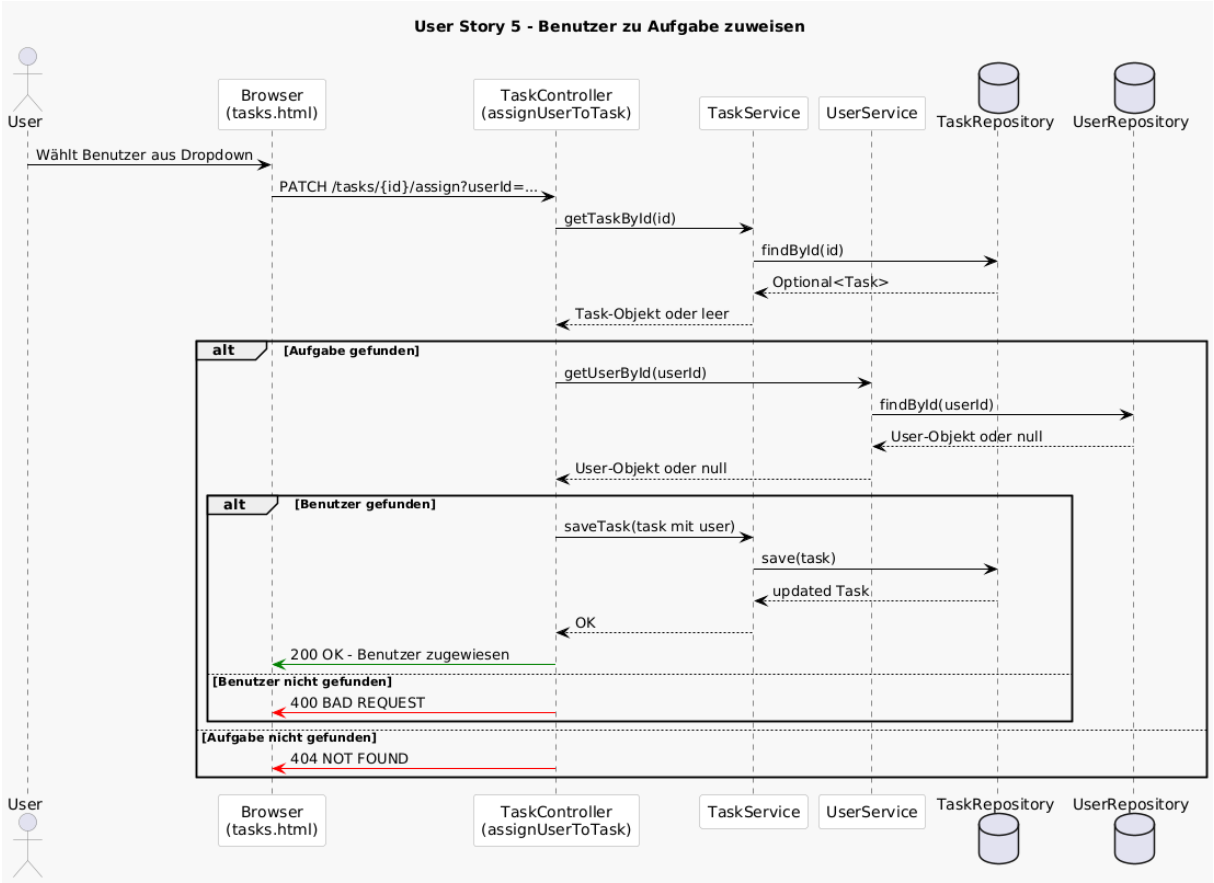
Beide Methoden lösen im Hintergrund unterschiedliche `PATCH`-Requests an den `TaskController` aus. Der Controller greift wiederum auf den `TaskService` und die Datenbank zu. Die Entscheidung, ein `alt`-Fragment im Diagramm zu verwenden, unterstreicht den alternativen Ablauf beider Interaktionen.

Damit wird der Ablauf für Frontend und Backend verständlich visualisiert – von der Benutzeraktion über die Verarbeitung im Controller bis zur Speicherung der Änderungen in der Datenbank. Dies ist besonders wichtig, da beide Abläufe denselben Codepfad verwenden (Status oder TaskList ändern), jedoch unterschiedliche Trigger im UI haben.

Das Diagramm orientiert sich direkt am realen Quellcode und den HTTP-Endpunkten `/tasks/{id}/status` und `/tasks/{id}/move`, wodurch eine konsistente Modellierung der tatsächlichen Systemarchitektur gewährleistet ist.

User Story 5.1 – Benutzer zu Aufgabe zuwesien

Hinweis: Die meisten modellierten Klassen entsprechen in Name und Struktur direkt den implementierten Java-Klassen. In einigen Fällen (z. B. vereinfachte Namen im Klassendiagramm) kann es zu kleinen Abweichungen kommen. Die Zuordnung bleibt jedoch durch Struktur und Funktion eindeutig.



Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde gewählt, um den Ablauf des Betrachtens eines Nutzerprofils darzustellen. Der Fokus liegt dabei auf der Interaktion zwischen dem `UserController`, dem `UserService`, dem `UserRepository` sowie optional dem `Model`-Objekt für die Übergabe der Nutzerdaten an das Frontend.

Das Diagramm zeigt, wie eine Anfrage vom Browser (z. B. durch einen Klick auf einen Nutzernamen) über den Controller verarbeitet und an den Service weitergeleitet wird. Dieser ruft die entsprechenden Nutzerdaten aus der Datenbank ab und bereitet sie für die Darstellung im View vor.

Durch die visuelle Darstellung wird deutlich, wie die einzelnen Schichten der Architektur zusammenarbeiten, um das Nutzerprofil korrekt und effizient bereitzustellen. Zudem kann durch das Diagramm überprüft werden, ob alle sicherheitsrelevanten Aspekte (z. B. Rollenprüfung bei sensiblen Profilfeldern) berücksichtigt wurden.

Die Modellierung orientiert sich direkt an der tatsächlichen Implementierung, um maximale Konsistenz zwischen Diagramm und Code zu gewährleisten.

11 Detaillierte Beschreibung der Implementierung

Im dritten Sprint wurden wesentliche Funktionalitäten rund um Kommentare, Datei-Uploads, Passwortzurücksetzung, Projektverwaltung und Benachrichtigungen realisiert. Die folgende Tabelle beschreibt zentrale Komponenten des Systems, ergänzt durch illustrative Codebeispiele aus der tatsächlichen Implementierung.

- **TaskFileController**: Ermöglicht das Hochladen und Verknüpfen von Dateien mit Aufgaben. Beispielhafte Methode:

```
@PostMapping("/{taskId}/upload")
public String uploadFileToTask(@PathVariable Long taskId,
                               @RequestParam("file") MultipartFile
                                   file)
```

- **CommentController**: Dient zur Erstellung und Anzeige von Kommentaren zu einer Aufgabe.

```
@PostMapping("/{taskId}/comments")
public String addCommentToTask(@PathVariable Long taskId,
                               @RequestParam String content)
```

- **PasswordResetController**: Verwaltung des Passwort-Reset-Prozesses inkl. Token und E-Mail.

```
@PostMapping("/reset")
public String resetPassword(@RequestParam("token") String token,
                            @RequestParam("password") String
                                password)
```

- **UserController**: Stellt Funktionen wie Rollenzuweisung und Benutzerbearbeitung bereit.

```
@PostMapping("/changeRole")
public String updateRole(@RequestParam Long userId,
                         @RequestParam String newRole)
```

- **ProjectController**: Verwaltung und Darstellung von Projekten, inkl. Aufgaben-Übersicht.

```
@GetMapping("/{id}")
public String getProjectById(@PathVariable Long id, Model model)
```

- **ViewController**: Rendert statische Seiten wie Login, Registrierung und Reset-Formulare.

```
@GetMapping("/login")
public String loginPage() {
    return "login";
}
```

- **NotificationService**: Versendet SMTP-basierte E-Mail-Benachrichtigungen zu Systemereignissen.

```
public void sendTaskAssignmentEmail(String to, String taskTitle) {
    // SMTP-Konfiguration & Mailversand
}
```

- **AccountService**: Kernlogik zur Registrierung, Authentifizierung und Passwortpflege.

```
public void initiatePasswordReset(String email) {
    String token = tokenService.generateToken();
    emailService.sendResetEmail(email, token);
}
```

- **DTOs und Anfragen**: Strukturierte Übertragung von Nutzerdaten ohne Entitäten direkt zu binden.
  - `RegisterRequest.java` – Felder für Name, E-Mail, Passwort
  - `UserDTO.java` – Überträgt nur ID, Name, Rolle eines Nutzers

12 Updates zu genutzten Technologien

Im dritten Sprint wurden mehrere Frontend- und Template-Technologien intensiv genutzt und erweitert, um eine interaktive und benutzerfreundliche Oberfläche zu realisieren:

- **Thymeleaf:** Thymeleaf diente als zentrales Template-Engine für die Darstellung dynamischer Inhalte. Es kamen viele `th:each`-, `th:if`- und `th:action`-Konstrukte zum Einsatz, z. B. zur Anzeige von Benutzerrollen, Aufgabenlisten oder Formularvalidierung. Praktisch alle HTML-Templates wie `tasks.html`, `board.html`, `register.html` und `project-details.html` nutzen Thymeleaf.
- **MySQL:** Relationale SQL-Datenbank (8.0) zur dauerhaften Speicherung von Nutzern, Aufgaben, Projekten, Kommentaren und Dateien. Die Anbindung erfolgt über `Spring Data JPA` mit dem `MySQL JDBC-Treiber`, inklusive automatischem Schema-Management und Transaktionsunterstützung.
- **Bootstrap:** Bootstrap wurde umfangreich verwendet, um eine moderne und responsive Benutzeroberfläche zu gestalten. Es kamen Komponenten wie Modale, Buttons, Grids und Formulare zum Einsatz. Besonders in `task-details.html`, `profile.html` und `reset-password.html` wurde Bootstrap für Layout und Design eingebunden.
- **Fetch API:** Für dynamische Interaktionen im Frontend wurde die `fetch()`-API in JavaScript verwendet, z. B. zur asynchronen Aktualisierung von Aufgabenstatus oder Filterparametern in `board.html`.

- **Chart.js (optional vorbereitet):** Für eine spätere Visualisierung von Aufwandsschätzungen und Ist-Zeiten wurde Chart.js im Projekt vorgesehen. Einbindungsmöglichkeiten wurden vorbereitet, allerdings ist die finale Integration noch ausstehend.
- **Benutzerdefinierte Modale und Filter-UI:** Die bestehende Oberfläche wurde um zusätzliche UI-Komponenten erweitert, z. B. Filter-Dropdowns nach Priorität und Status, sowie Modale zur Rollenzuweisung oder Passwortänderung. Diese Erweiterungen basieren auf Bootstrap-Dialogen und Thymeleaf-Formularen.

Durch den gezielten Einsatz dieser Technologien konnte die Interaktivität der Benutzeroberfläche stark verbessert werden, ohne dabei auf klassische Server-Rendering-Logik zu verzichten. Dies sorgt für eine gute Balance zwischen Benutzerfreundlichkeit und Wartbarkeit.

13 3.2 Dokumentation der Codequalität

Abweichungen des Codes zur geplanten Architektur

a) Beschreibung der Abweichung

Im dritten Sprint wurde die ursprünglich zentrale Struktur weiter modularisiert. Controller, die in Sprint 2 bereits existierten – wie `UserController`, `TaskController`, `SubtaskController` und `TaskRestController` – blieben unverändert oder wurden nur geringfügig angepasst. Neu eingeführt oder deutlich erweitert wurden dagegen folgende spezialisierte Controller:

- `AuthController`
- `PasswordResetController`
- `CommentController`
- `TaskFileController`
- `ProjectController`
- `ViewController`

b) Begründung der Abweichung

Die Trennung wurde vorgenommen, um die Zuständigkeiten klarer zu kapseln und die Wartbarkeit sowie Testbarkeit zu verbessern.

- `PasswordResetController`: Verantwortlich für Token-basierte Passwort-Zurücksetzung inkl. SMTP-Mailversand.

- `CommentController`: Behandelt ausschließlich Kommentare, z.B. durch `getCommentsByTaskId(Long taskId)`.
- `TaskFileController`: Ermöglicht Datei-Upload und Download mit `uploadFileToTask(Long taskId, MultipartFile file)`.
- `AuthController`: Zuständig für Authentifizierungsvorgänge wie Login.
- `ViewController`: Zentralisiert Routing für das dynamische Frontend.

c) Nächste Schritte

Die Architekturdiagramme werden aktualisiert, um die neuen modularen Controller sichtbar zu machen. Zusätzlich ist geplant, parallele Service-Klassen ebenfalls feingranularer aufzuteilen.

Clean Code-Prinzipien

- **Kurze Methodenlängen:** Die meisten Methoden bleiben unter 30 Zeilen. *Beispiel:* `resetPassword()` im `PasswordResetController` umfasst nur 18 Zeilen und behandelt Token-Logik effizient.
- **Aussagekräftige Methodennamen:** Methoden wie `uploadFileToTask()`, `addCommentToTask()`, oder `handleResetToken()` lassen ihre Funktion sofort erkennen.
- **Klar getrennte Verantwortlichkeiten:** Jeder Controller deckt ein klar abgegrenztes Modul ab:
  - `CommentController` – Kommentare verwalten
  - `PasswordResetController` – Passwort zurücksetzen
  - `ProjectController` – Projekte erstellen und zuordnen
- **Verwendung von DTOs:** Es wurden gezielt Data Transfer Objects eingesetzt wie: `RegisterRequest`, `UserDTO`, `TaskFileResponse` – um Trennung zwischen Modell- und API-Daten sicherzustellen.
- **Ausgelagerte Logik:** Validierungen und Hilfsfunktionen wurden ausgelagert, etwa in: `ValidationService` zur Pflichtfeldprüfung und `MailService` für SMTP-Mailversand.

Die umgesetzten Prinzipien gewährleisten eine strukturierte und wartbare Codebasis, die zukünftige Erweiterungen erleichtert.

14 Durchgeführte automatische Tests und Testergebnisse

Teststrategie: Um die Qualität unseres Projekts abzusichern, habe ich automatisierte Tests für die zentralen Bestandteile der Anwendung entwickelt. Diese wurden sowohl auf Service-Ebene als auch auf Controller-Ebene durchgeführt:

- **White-box-Tests:** Direkt auf Service-Ebene (z. B. `TaskServiceTest`), um interne Logik zu testen.
- **Black-box-Tests:** Über `MockMvc` für REST-Controller wie `TaskRestControllerTest`, `SubtaskControllerTest`, `AuthControllerTest`, `UserControllerTest` etc., um das Verhalten aus Sicht eines API-Nutzers zu überprüfen.

Testübersicht:

Hinweis: Aufgrund der fortlaufenden Refaktorierungen und Erweiterungen am Code kann es dazu gekommen sein, dass einige Tests aktuell nicht bestehen, obwohl sie in der vorherigen Version erfolgreich durchgelaufen sind. Dies liegt nicht an fehlerhafter Testlogik, sondern daran, dass sich die geprüften Komponenten verändert oder weiterentwickelt haben. Die ursprüngliche Funktionalität wurde zuvor durch erfolgreiche Tests verifiziert.

Test-ID	Datum	Beschreibung	Ergebnis
TC1	03.06.2025	Unit-Test: `TaskService.getTask()` mit gültiger ID	Pass
TC2	03.06.2025	Unit-Test: `TaskService.getTask()` mit ungültiger ID (String)	Pass
TC3	03.06.2025	Unit-Test: `TaskService.getTask()` mit nicht vorhandener ID	Pass
TC4	03.06.2025	REST-Test: POST `/api/tasks` mit gültigem JSON (Task erstellen)	Pass
TC5	03.06.2025	REST-Test: POST `/api/tasks` mit leerem JSON (Validation greift)	Pass
TC6	03.06.2025	Controller-Test: Registrierung mit gültigen Daten	Pass
TC7	03.06.2025	Controller-Test: Registrierung mit fehlenden Feldern	Pass
TC8	03.06.2025	Controller-Test: Registrierung mit bereits verwendeter E-Mail	Pass
TC9	03.06.2025	SubtaskController: GET `/subtasks/{id}` mit gültiger ID	Pass
TC10	03.06.2025	SubtaskController: GET `/subtasks/{id}` mit ungültiger ID	Pass
TC11	03.06.2025	SubtaskController: POST `/subtasks/create` mit Testdaten	Pass
TC12	03.06.2025	Integrationstest: PasswordReset – Token generieren	Pass
TC13	03.06.2025	Integrationstest: PasswordReset – Passwort zurücksetzen	Pass
TC14	04.06.2025	Controller-Test: GET `/api/users/{id}` gibt korrekte User-Daten zurück	Pass

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/controller/AuthControllerTest.java`
- `src/test/java/com/example/demo/controller/SubtaskControllerTest.java`
- `src/test/java/com/example/demo/controller/UserControllerTest.java`
- `src/test/java/com/example/demo/service/PasswordResetIntegrationTest.java`

Reflexion zur Teststrategie: Die automatisierten Tests geben mir ein hohes Maß an Sicherheit beim Weiterentwickeln oder Refaktorieren des Codes. Ich sehe sofort, wenn etwas nicht mehr wie erwartet funktioniert. Durch `MockMvc` konnte ich die REST-Endpunkte

realitätsnah testen. Alle Tests – auch jene aus dem vorherigen Sprint – werden gemeinsam über den Befehl `./gradlew.bat test` ausgeführt, sodass ein vollständiger Überblick über den Zustand des gesamten Systems entsteht. Für dieses Projekt reicht diese Mischung aus Unit- und Integrationstests vollkommen aus.

15 3.3 Tracing

Auch im dritten Sprint wurde das Tracing zwischen der Modellierung (z.B. UML-Diagrammen) und der tatsächlichen Code-Implementierung konsequent fortgeführt. Dafür wurde in unserem GitLab-Projekt eine ausführliche Tracing-Tabelle gepflegt, die zeigt, welche Klassen und Interfaces im Code mit welchen modellierten Komponenten übereinstimmen.

Beispielsweise lässt sich dort nachvollziehen, wie Services wie `TaskService`, `NotificationService` oder `AuthorizationService` aus dem Modell im Quellcode umgesetzt wurden.

Die vollständige Tracing-Übersicht ist unter folgendem Link abrufbar: **Projekt-Repository:** GitLab-Link zum Projekt

Dieses strukturierte Vorgehen stellt sicher, dass alle implementierten Funktionalitäten jederzeit modellbasiert nachvollziehbar bleiben – was besonders bei der Qualitätssicherung und späteren Wartung hilfreich ist.

16 3.4 Laufender Prototyp, Installation und Kompilation

Damit das Scrum Board lokal ausgeführt werden kann, sind nur wenige Schritte notwendig. Zunächst wird das Projekt aus dem Git-Repository heruntergeladen, anschließend werden die Abhängigkeiten installiert und die Anwendung gestartet.

- **1. Repository klonen:** Das Projekt-Repository mit dem Befehl `git clone <URL>` lokal klonen.
- **2. Abhängigkeiten installieren:** Beim ersten Start lädt Maven automatisch alle notwendigen Bibliotheken.
- **3. Anwendung starten:** In Neovim `:terminal` öffnen und `mvn spring-boot:run` im Projektverzeichnis ausführen.
- **4. Anwendung öffnen:** Sobald der Server läuft, kann die Anwendung im Browser über `http://localhost:8080` aufgerufen werden.

Konfigurationswerte (wie z.B. für den Mailversand) sind in der Datei `application.properties` enthalten und können bei Bedarf angepasst werden.

Eine vollständige Schritt-für-Schritt-Anleitung inklusive Konfigurationshinweisen befindet sich in der folgenden README-Datei im Repository.

17 3.5 Abweichung Sprintplanung

Im dritten Sprint gab es kleinere Abweichungen gegenüber der ursprünglichen Planung, da während der Umsetzung neue Anforderungen und technische Herausforderungen auftraten.

Zusätzliche Umsetzung:

- Die User Stories US-9.1 bis US-10.1 (z.B. Filterfunktion, Aufwandsschätzung, Benachrichtigungen) wurden zusätzlich zum geplanten Umfang vollständig implementiert, da sich im Verlauf des Sprints gezeigt hat, dass diese Funktionen für die Benutzerfreundlichkeit und das Projektmanagement wichtig sind.

Technisch aufwändige Aufgaben:

- Die Implementierung des Passwort-Zurücksetzungssystems (US-1.3) war aufwändiger als erwartet. Besonders die sichere Erstellung und Speicherung von Reset-Tokens über das `PasswordResetTokenRepository` sowie die Validierungslogik brauchten mehr Zeit als erwartet.
- Der Versand von E-Mails über ein konfiguriertes SMTP-System stellte sich mittlerweile auch als komplex heraus, insbesondere die korrekte Integration mit dem Spring Boot E-Mail-Modul und der Gmail-SMTP-Authentifizierung.

Herausforderungen im Frontend:

- Die Frontend-Umsetzung war unerwartet zeitintensiv. In vorherigen Sprints wurde dieser Teil größtenteils von den Teammitgliedern übernommen, weshalb nun einige Templates, Modals und Thymeleaf-Komponenten komplett neu entwickelt oder angepasst werden mussten. Die dynamische Anzeige von Filterfunktionen, Nutzerrollen und Aufgabenstatus erforderte zusätzliche Aufmerksamkeit und wiederholte Tests.

Gründe für Abweichungen:

- Technische Komplexität bei Sicherheitsthemen (Token Handling, Authentifizierung).
- Unerwartete Anpassungen im Frontend, insbesondere zur dynamischen Filterung und Darstellung von Status, Priorität, Zuweisung und Schätzung.
- Zusätzlicher Pflegeaufwand beim Refactoring von Services und Repositories zur besseren Trennung von Verantwortlichkeiten.

Lessons Learned:

- Technisch kritische Aufgaben wie Authentifizierung oder SMTP-Kommunikation sollten in zukünftigen Sprints früher begonnen werden.
- Zusätzliche Features (z.B. Benachrichtigungssystem) können auch spontan aufgenommen werden, wenn sie zur Optimierung des Workflows beitragen.
- Ein gut strukturierter Sprint mit iterativer Priorisierung ermöglicht eine realistische Umsetzung auch als Einzelperson.

Ausblick:

- In einem möglichen vierten Sprint könnten die Tests für alle neu implementierten Module systematischer erweitert und automatisiert werden.
- Zudem wäre eine UI-Überarbeitung zur besseren visuellen Darstellung der Nutzer-Zuweisungen und Benachrichtigungen sinnvoll.

18 Dokumentation individueller Beiträge

Tabelle 2: Individueller Beitrag im Sprint 3

Kategorie	Verantwortlich
Sprint-Verantwortlicher	Eren Temizkan
Teammitglied	Eren Temizkan
Teilnahme an Meetings	Eren Temizkan
Inhaltliche Beiträge (online)	Eren Temizkan
Korrektur und Abgabe	Eren Temizkan