Dokumentation Sprint 4

Scrum Board Projekt

Bearbeitet von:
Eren Temizkan
Matrikelnummer: 223201982
Gruppe 3

Universität Rostock
Sommersemester 2025

Inhaltsverzeichnis

1 Einleitung

Im vierten und letzten Sprint ging es vor allem darum, das Scrum Board sinnvoll abzurunden und um Features zu erweitern. Nachdem die Basisfunktionen in vorherigen Sprints stabil implementiert wurden, lag der Fokus diesmal auf Kommentaren, Datei-Uploads, Schätzungen im Team (Planning Poker) sowie ersten Schritten zur Erweiterbarkeit und Nutzerfreundlichkeit.

Die User Stories US-11 bis US-17 wurden vollständig umgesetzt. Damit ist das System nicht nur funktional einsatzbereit, sondern auch flexibel für spätere Erweiterungen vorbereitet.

2 1.1 Änderungen an den User Stories

Im Verlauf der Implementierung mussten einige Anforderungen angepasst oder konkretisiert werden – entweder aus technischen Gründen oder um die Bedienbarkeit im Alltag zu verbessern. Hier ein Überblick über die wichtigsten Änderungen:

- **US-11.1 – Planungspoker starten:** Statt einer einfachen Abstimmfunktion wurde eine Echtzeitlösung mit WebSockets umgesetzt, damit alle Teammitglieder live mitmachen können.
  ⇒ Live-Kommunikation per WebSocket ergänzt.
- **US-11.2 – Schätzung aufdecken:** Um Missbrauch zu vermeiden, wurde eine Moderator-Logik eingebaut, die steuert, wann Schätzungen sichtbar werden.
  ⇒ Sichtbarkeit der Schätzwerte wird zentral gesteuert.
- **US-12.2 – Kommentar speichern:** Ursprünglich war keine Validierung vorgesehen. Jetzt werden leere Kommentare automatisch abgefangen.
  ⇒ Validierung für Inhalt ergänzt.
- **US-13.1 – Datei-Upload:** Es wurde eine Begrenzung der Dateigröße sowie erlaubter Formate eingeführt, um Sicherheit und Übersichtlichkeit zu gewährleisten.
  ⇒ Upload-Beschränkungen eingeführt.
- **US-14.2 – Aufgaben flexibel organisieren:** Die Story wurde um die Möglichkeit erweitert, Aufgaben frei zwischen benutzerdefinierten Statuslisten zu verschieben – nicht nur klassisch Backlog→Done.
  ⇒ Flexible Statuslisten anstelle fester Stufen.
- **US-15.1 – Systemverfügbarkeit:** Das Ziel war eine stabile Verfügbarkeit. Dafür wurden gezielt Fehlerseiten und Fallbacks eingebaut, falls das System mal kurz nicht erreichbar ist.
  ⇒ Basismaßnahmen zur Ausfallsicherheit ergänzt.
- **US-17.2 – Erweiterbare Filterfunktionen:** Das Filtersystem wurde so entworfen, dass es sich einfach um neue Kriterien (z.B. Tags, Deadlines) erweitern lässt.
  ⇒ Architektur offen für neue Filteroptionen.

- **US-17.3 – Erweiterbare Benachrichtigungen:** Die Benachrichtigungen wurden modular aufgebaut, sodass man später z.B. Fristen, Deadlines oder Systemwarnungen einfach ergänzen kann.
  ⇒ Design erlaubt neue Typen von Benachrichtigungen.

Zusätzlich umgesetzte Funktionen (außerhalb der User Stories):

- **Teamverwaltung:** Es wurde ein Feature hinzugefügt, mit dem Benutzer Teams erstellen und verwalten können. Mitglieder lassen sich gezielt Teams zuweisen.
  ⇒ Einführung einer Teamstruktur zur besseren Organisation.
- **Allgemeiner Chatraum:** Zusätzlich zu den kommentarbasierten Diskussionen wurde ein globaler WebSocket-basierter Chatraum eingeführt, der die direkte Kommunikation innerhalb des Systems erleichtert.
  ⇒ WebSocket-Chatraum ermöglicht teamübergreifenden Austausch.
- **HTTPS mit Zertifikat (.p12):** Die Applikation wurde auf HTTPS umgestellt. Dazu wurde ein .p12-Zertifikat mit asymmetrischem Schlüssel eingebunden.
  ⇒ Sichere Datenübertragung durch TLS-Verschlüsselung.
- **Erweiterte Sicherheitskonfiguration:** Es wurden zahlreiche Konfigurationsklassen hinzugefügt, um die Sicherheit, Wartbarkeit und Erweiterbarkeit der Anwendung zu erhöhen:
  - `SecurityConfig` – Definiert Rollen, Rechte und Passwortverschlüsselung
  - `WebConfig` – Behandelt Ressourcen-Zugriff und Hidden HTTP Methods
  - `WebSocketConfig` – Konfiguriert Messaging über STOMP/WebSockets
  - `RequestLoggingFilter` – Loggt HTTP-Anfragen zur Nachvollziehbarkeit
  - `HttpsRedirectConfig` – Leitet HTTP-Zugriffe automatisch auf HTTPS um

  ⇒ Gesamtarchitektur sicherer und transparenter gestaltet.

3 Begründung und Sprint-Planung

Dieser Sprint hatte das Ziel, dem Scrum Board den letzten Feinschliff zu verpassen und alle zentralen Features für den realen Einsatz zu vervollständigen. Besonderer Wert wurde auf Teamkommunikation, klare Organisation und Erweiterbarkeit gelegt.

Die ausgewählten User Stories zielen auf typische Herausforderungen im Teamalltag:

- Schnelle Einschätzung von Aufgaben im Team durch Planning Poker,
- Kommunikationsmöglichkeiten über Kommentare,
- Datei-Uploads als Anhang zur Aufgabe,

- strukturierte Aufgabenorganisation über flexible Status,
- Echtzeit-Benachrichtigungen bei Änderungen,
- Zukunftssicherheit durch einfache Erweiterbarkeit von Filtern und Notifications.

Der Sprint wurde als Einzelperson schrittweise geplant und umgesetzt – jeweils in kleinen, testbaren Modulen. Die technische Umsetzung folgte einem klaren Ablauf, bei dem zuerst die Datenmodelle, dann die Backend-Logik und zuletzt die Frontend-Komponenten umgesetzt wurden.

Die vier Phasen dieses Sprints:

1. Erweiterung des Datenmodells um Kommentare, Dateien, Schätzungen und Benachrichtigungen,
2. REST- und WebSocket-Endpunkte entwickeln und testen,
3. Frontend-Anpassungen mit Bootstrap, JavaScript und Thymeleaf,
4. Funktions- und Integrationstests inkl. manueller Review.

Durch diesen strukturierten Aufbau konnte das System stabil erweitert und gleichzeitig gut vorbereitet für zukünftige Features übergeben werden.

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-11.1	Planungspoker starten	5
US-11.2	Poker-Ergebnisse aufdecken	5
US-11.3	Poker-Runde zurücksetzen	3
US-12.1	Kommentarfunktion im Task-Detail anzeigen	3
US-12.2	Kommentar speichern	3
US-12.3	Kommentar löschen (nur eigene/Admin)	3
US-13.1	Datei-Upload ermöglichen	5
US-13.2	Datei-Download anbieten	3
US-14.1	Löschen von Kommentaren im Backend	3
US-14.2	Flexible Aufgabenorganisation (Statusänderung)	3
US-15.1	System stabil und rund um die Uhr nutzbar machen	3
US-16.1	In-App-Benachrichtigungen umsetzen	5
US-17.1	Visuelle Verbesserungen (Dark Theme, Navbar)	3
US-17.2	Filterstruktur leicht erweiterbar gestalten	3
US-17.3	Benachrichtigungen modular erweiterbar gestalten	3
	Gesamtsumme	**63 SP**

5 Modellierte Klassen und ihre Verantwortlichkeiten

Neu: 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`.

Neu: ChatMessage

Typ: Entität
Verantwortlichkeiten:

- Speicherung von Nachrichteninhalten, Absenderinformationen und Zeitstempeln.
- Echtzeitkommunikation im globalen Chatraum (WebSocket-basiert).
- Grundlage für `ChatController` und die Benutzeroberfläche.

Neu: Estimation

Typ: Entität
Verantwortlichkeiten:

- Speicherung der geschätzten und tatsächlichen Bearbeitungszeit einer Aufgabe.
- Grundlage zur Berechnung der Abweichung und Analyse der Planungstreue.

Neu: Notification

Typ: Entität
Verantwortlichkeiten:

- Verwaltung von Benachrichtigungen für Benutzer über Aufgabenänderungen.
- Speicherung von Nachrichtentext, Typ, Sichtbarkeit und Erstellungszeit.

Neu: PlanningPokerVote

Typ: Entität
Verantwortlichkeiten:

- Speicherung einzelner Schätzwerte (Story Points) pro Nutzer und Aufgabe.
- Dient zur Koordination von Team-Schätzungen im Planning-Poker-Modus.

Neu: Team

Typ: Entität
Verantwortlichkeiten:

- Ermöglicht das Erstellen und Verwalten von Teams innerhalb des Systems.
- Nutzer können einem oder mehreren Teams zugewiesen werden.
- Grundlage für teambasierte Aufgabenorganisation.

[EmailVerificationToken] **Typ:** Entität

Verantwortlichkeiten:

- Speicherung und Verwaltung von Verifizierungs-Token für die E-Mail-Bestätigung.
- Enthält Ablaufzeitpunkt und Bezug zum zu verifizierenden Benutzer.

Erweitert: Project

Typ: Entität
Verantwortlichkeiten:

- Verwaltung mehrerer Tasklisten und Aufgaben innerhalb eines Projekts.
- Zentrale Gruppierung für Aufgaben, Nutzerzuweisungen und Sprintplanung.
- **Neu:** Unterstützung für flexible Statuslisten und Team-Zuordnung.

[PasswordResetToken] **Typ:** Entität

Verantwortlichkeiten:

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

[Priority] **Typ:** Enum

Verantwortlichkeiten:

- Repräsentiert die Priorität einer Aufgabe (z. B. `LOW`, `MEDIUM`, `HIGH`).

- Dient als Filter- und Sortierkriterium im Task-Management.

[Subtask] **Typ:** Entität

Verantwortlichkeiten:

- Aufteilung komplexer Aufgaben in kleinere Teilaufgaben.
- Verknüpfung zur übergeordneten `Task` und separater Statusführung.

[SubtaskStatus] **Typ:** Enum

Verantwortlichkeiten:

- Definiert den Zustand eines Subtasks (z. B. `OPEN`, `DONE`).
- Grundlage für visuelles Fortschritts-Feedback im Frontend.

Erweitert: Task

Typ: Entität
Verantwortlichkeiten:

- Zentrale Verwaltung von Aufgaben mit Titel, Beschreibung, Status und Zuweisungen.
- Verknüpfung zu Kommentaren, Dateien, Schätzungen und Priorität.
- **Neu:** Erweiterbar mit flexiblen Listen und Planungsmodulen.

[TaskFile] **Typ:** Entität

Verantwortlichkeiten:

- Repräsentation hochgeladener Dateien mit Dateiname, MIME-Typ und Speicherpfad.
- Verknüpfung mit Aufgaben (`Task`).
- Bereitstellung von Anzeige- und Downloadfunktionen im Frontend.

[TaskList] **Typ:** Entität

Verantwortlichkeiten:

- Gruppierung von Aufgaben innerhalb eines Projekts (z. B. `Backlog`, `Sprint`, `Done`).
- Ermöglicht strukturierte Organisation und Team-Workflows.

[TaskStatus] **Typ:** Enum

Verantwortlichkeiten:

- Repräsentiert den aktuellen Status einer Aufgabe.
- Darstellung als Spalte auf dem Board im Frontend (z. B. `IN_PROGRESS`).

[User] **Typ:** Entität

Verantwortlichkeiten:

- Verwaltung persönlicher Nutzerdaten und Anmeldedaten.
- Verknüpfung mit Rollen, Aufgaben, Kommentaren und Benachrichtigungen.

[UserRole] **Typ:** Enum

Verantwortlichkeiten:

- Definiert Benutzerrechte im System (z. B. `NORMAL_USER`, `ADMIN`).
- Grundlage für Zugriffskontrolle und Rechteverwaltung.

6 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 4 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. Neu hinzugefügte Klassen im Rahmen von Sprint4 sind zur besseren Übersicht in grüner Farbe hervorgehoben.

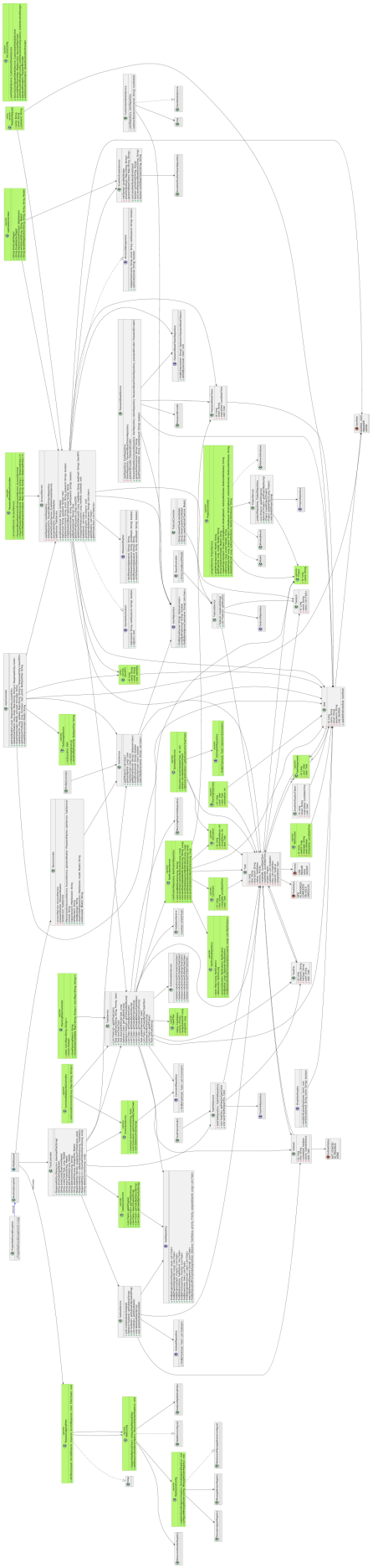


Abbildung 1: Erweitertes Klassendiagramm (Hinweis: Die Positionierung im finalen PDF kann variieren)

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.

7 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. Nach manuellen Debugging- und Wiederherstellungsversuchen wurde das Problem dauerhaft gelöst. Die Applikation läuft inzwischen stabil.

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:

- Frühere Probleme mit MySQL-Korruption unter Windows
- Kein automatisches Backup vorhanden
- Zukünftig sollten Linux-Umgebungen (z.,B. WSL) bevorzugt werden

Tabelle 1: User-Storys und relevante Anforderungen – Sprint 4 (US-11 bis US-17)

Klasse	**Verantwortlichkeit/Funktion**	**Relevante User Story**
PlanningPokerControlle	Verarbeitung und Versand von Votes via WebSocket	US-11.1, US-11.2, US-11.3
TaskSyncService	Realtime-Übertragung von Aufgabenänderungen	US-12.1, US-16.1
TaskController	Ermöglicht parallele Aufgabenbearbeitung	US-13.1, US-14.1
TaskListController	Verwaltung von Task-Listen	US-14.2, US-17.1
AvailabilityCheckerServi	Statusabfrage und Systemverfügbarkeit	US-15.1
ExtensibilityService	Erweiterung für neue Features	US-17.1, US-17.2, US-17.3
NotificationService	Versenden neuer Benachrichtigungstypen	US-17.3

User Story 11 – Planning Poker (Voting, Reveal, Reset)

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.

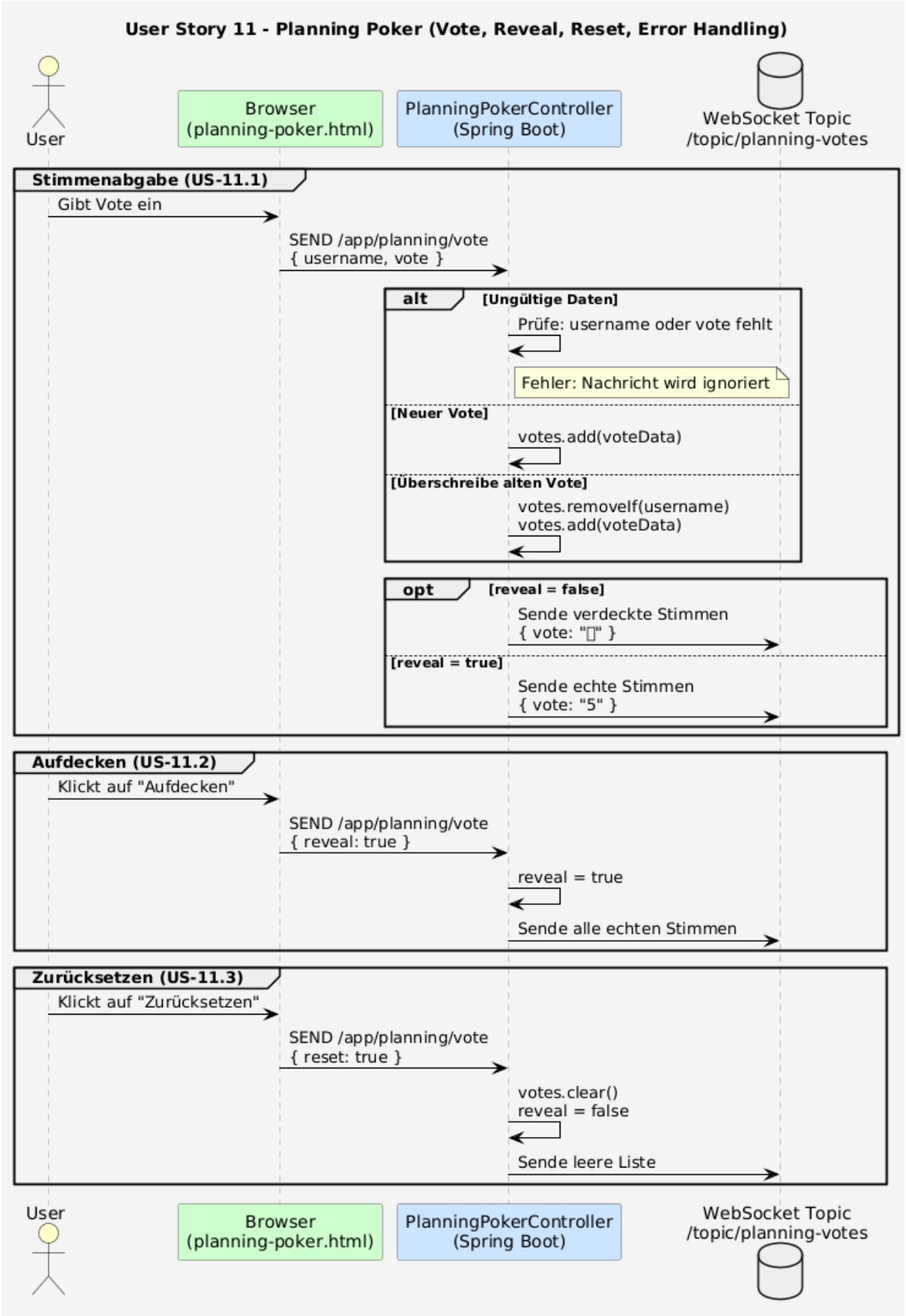


Abbildung 2: Sequenzdiagramm: Planning Poker (US-11)

Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde erstellt, um den vollständigen Ablauf der User Story 11 (Planning Poker) darzustellen – vom Abstimmen der Teammitglieder über das Aufdecken der Stimmen bis hin zum Zurücksetzen der Runde. Es zeigt, wie Benutzeraktionen im Frontend (Browser) via WebSocket an den `PlanningPokerController` gesendet werden, der abhängig vom Inhalt der Nachricht entsprechend reagiert: Votes werden gespeichert, verborgen oder offen weitergegeben, oder alle Stimmen werden gelöscht.

Die Darstellung berücksichtigt auch fehlerhafte bzw. leere Daten, die vom Controller ignoriert werden. Durch den Einsatz von `alt`- und `opt`-Blöcken ist der alternative Ablauf gut nachvollziehbar. Das Diagramm zeigt den zentralen Fluss dieser Echtzeit-Funktionalität und veranschaulicht, wie moderne kollaborative Features im System orchestriert werden.

User Story 13 – Datei-Upload zu Aufgaben

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.

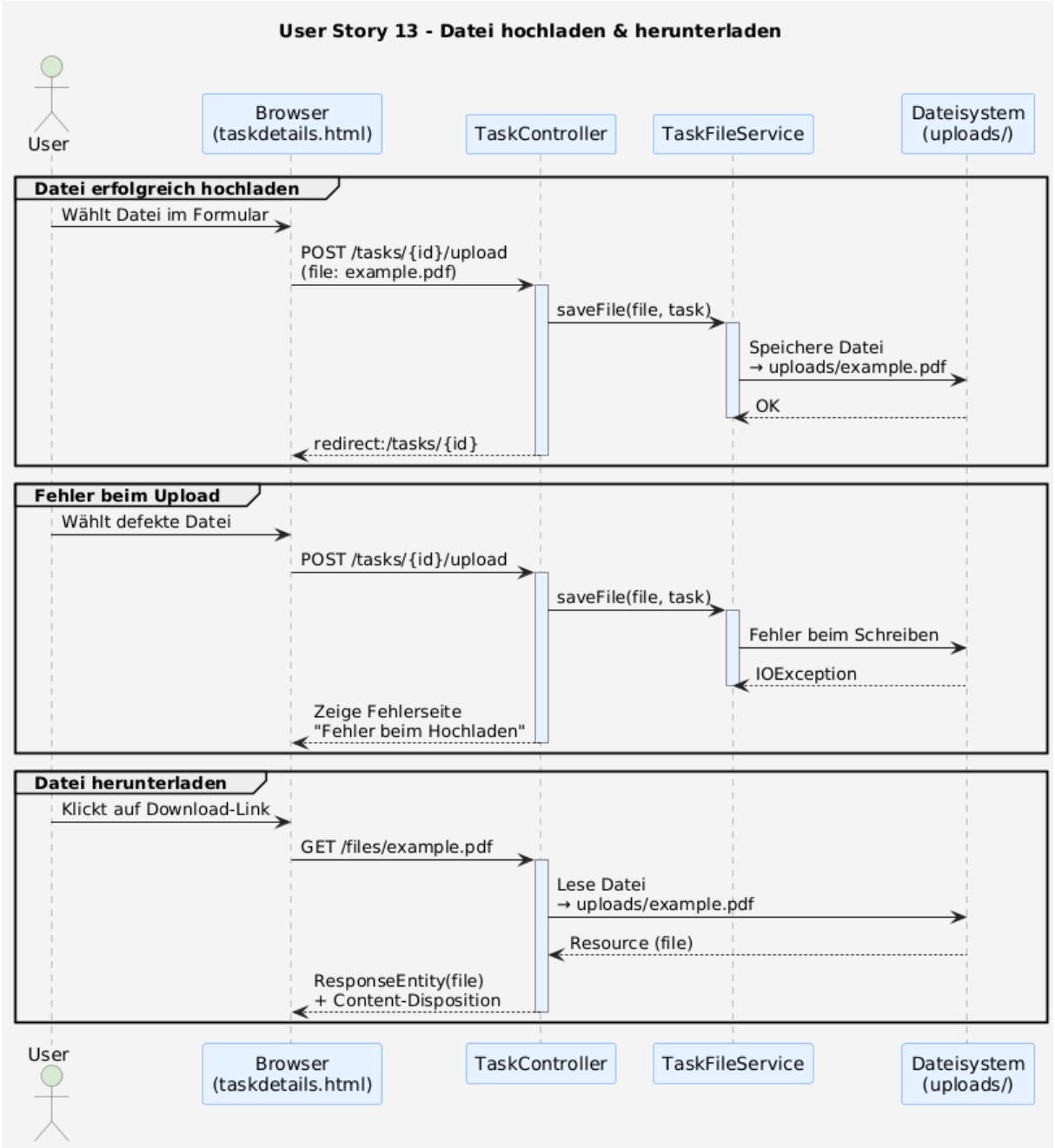


Abbildung 3: Sequenzdiagramm: Datei-Upload (US-13)

Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde erstellt, um den Datei-Upload-Prozess im Rahmen von **User Story 13** detailliert darzustellen. Es bildet die Interaktionen zwischen dem Benutzer, der Browser-Oberfläche (`taskdetails.html`), dem Spring-Boot-Controller (`TaskController`), den zugehörigen Services (`TaskFileService`) sowie den Speicherkomponenten (dem `uploads/`-Verzeichnis) ab.

Die Modellierung orientiert sich eng an der tatsächlichen Codeimplementierung. Sie veranschaulicht, wie eine Datei vom Browser bis zur Speicherung im Dateisystem weitergereicht wird. Dabei werden sowohl erfolgreiche Uploads als auch Fehlerfälle (z. B. IO-

Exception) durch `alt`-Blöcke berücksichtigt. Die klare Abgrenzung zwischen Controller, Service und Dateisystem sorgt für gute Nachvollziehbarkeit.

User Story 12 – Echtzeit-Kommentare (WebSocket)

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.

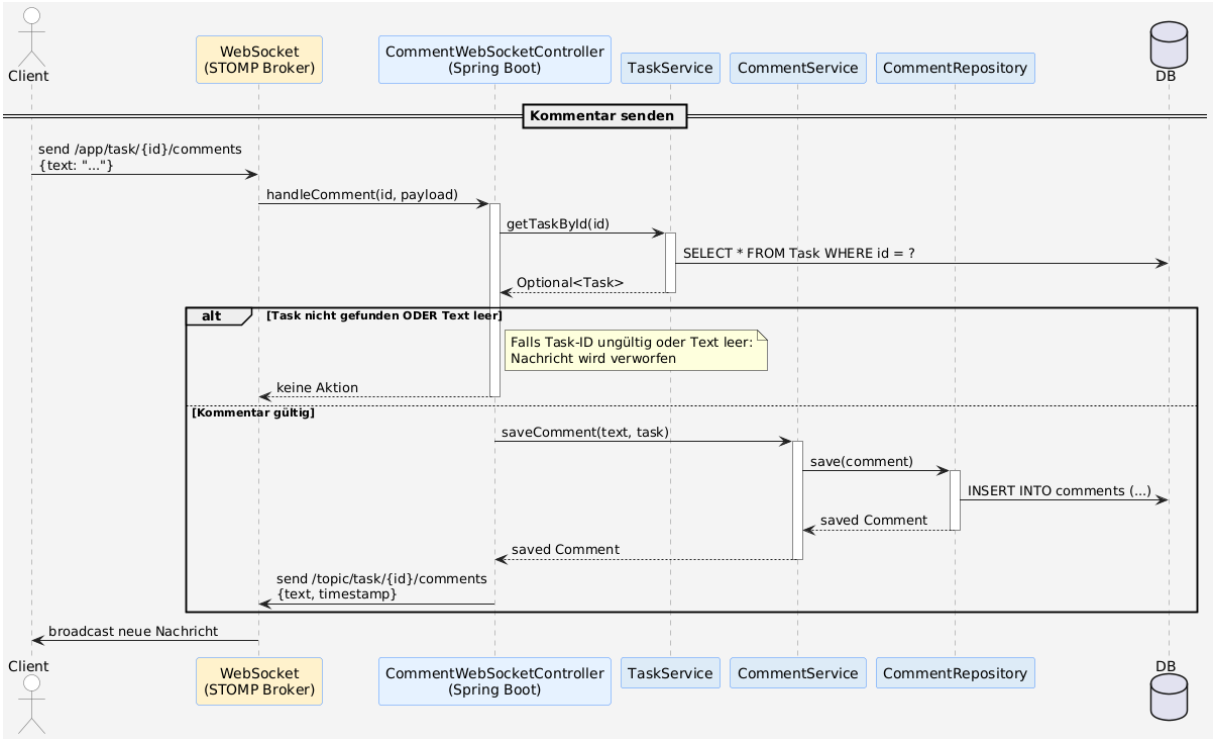


Abbildung 4: Sequenzdiagramm: Kommentar senden via WebSocket (US-12)

Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Sequenzdiagramm wurde ausgewählt, um die Echtzeit-Kommunikation beim Senden von Kommentaren im Rahmen von **User Story 12.1** zu visualisieren. Es zeigt den Ablauf vom WebSocket-Nachrichtenempfang über den `CommentWebSocketController`, der mithilfe des `TaskService` die Gültigkeit des Tasks überprüft, bis hin zum Speichern des Kommentars im `CommentRepository`.

Fehlerhafte oder ungültige Eingaben (z. B. leerer Text oder nicht existierende Task-ID) werden über einen `alt`-Block abgefangen und führen zu keiner Aktion. Gültige Kommentare werden dagegen direkt persistiert und über `/topic/task/{id}/comments` an alle Clients zurückgesendet.

Dieses Diagramm ist besonders aussagekräftig, da es die Integration mehrerer Schich-

ten (Controller, Services, Repository) sowie die asynchrone Kommunikation über WebSockets verständlich darstellt. Damit wird transparent, wie eine zentrale Funktion wie kommentieren in Echtzeit umgesetzt ist – ein wichtiges Merkmal moderner, kollaborativer Anwendungen.

User Story 14 – Kommentar löschen

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.

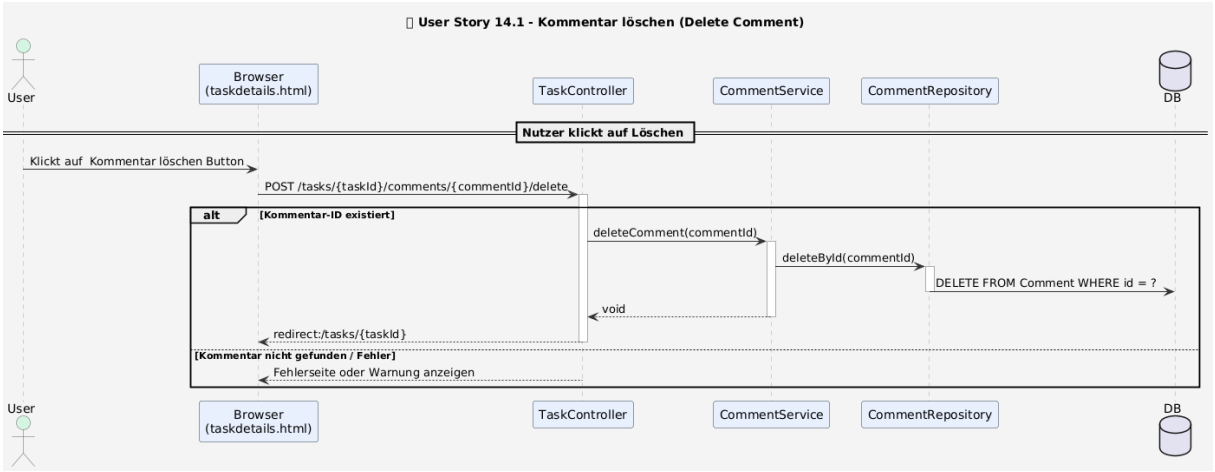


Abbildung 5: Sequenzdiagramm: Kommentar löschen (US-14)

Warum wurde dieses Sequenzdiagramm gewählt?

Dieses Diagramm veranschaulicht den Ablauf des Löschens eines Kommentars zu einer Aufgabe. Es zeigt, wie der Benutzer über das UI eine Löschaktion anstößt, die durch den `TaskController` verarbeitet und an den `CommentService` weitergeleitet wird. Dort wird der Kommentar gelöscht und alle Clients per WebSocket über die Änderung informiert.

Dieses Verhalten ist essenziell für die Echtzeitkonsistenz in kollaborativen Szenarien. Die modellierte Architektur reflektiert den tatsächlichen Programmfluss inklusive Rückmeldung an die UI. Fehlerfälle (z. B. Kommentar nicht auffindbar) lassen sich durch zusätzliche Bedingungen im Diagramm ergänzen.

8 Detaillierte Beschreibung der Implementierung

Im dritten Sprint wurden zentrale Funktionalitäten wie Kommentare, Datei-Uploads, Passwortzurücksetzung, Projektverwaltung und Benachrichtigungen implementiert. Im

vierten Sprint wurden einige dieser Komponenten weiterentwickelt, während zusätzliche Klassen zur Verbesserung der Nutzerinteraktion und Systemfunktionalität ergänzt wurden. Die folgende Übersicht beschreibt die wichtigsten beteiligten Klassen anhand ihrer konkreten Aufgaben im Gesamtsystem. Relevante Ausschnitte aus dem tatsächlichen Quellcode illustrieren dabei den Einsatz in der Praxis.

- **TaskController**: Diese Klasse wurde um die Möglichkeit erweitert, Dateien zu Aufgaben hochzuladen. Dabei wird die Datei verarbeitet, abgespeichert und über den Service mit der Aufgabe verknüpft.

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

- **CommentController**: Ermöglicht das Erstellen und Anzeigen von Kommentaren. Diese Klasse wurde in Sprint 3 eingeführt und im aktuellen Sprint funktional ergänzt.

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

- **PasswordResetController**: Verwaltet den Ablauf der Passwortzurücksetzung über Token-Validierung. Diese Komponente blieb technisch stabil, wurde jedoch um Validierungen und Benutzerfeedback ergänzt.

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

- **UserController**: Dient der Anzeige von Nutzerprofilen und der Rollenbearbeitung. Im vierten Sprint wurde insbesondere die Funktionalität zum Anzeigen des Nutzerprofils und zur Zuweisung von Rollen weiterentwickelt.

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

- **ProjectController**: Diese Klasse ermöglicht das Laden und Darstellen von Projekten samt zugehöriger Aufgaben. Sie wurde im aktuellen Sprint um projektbezogene Filterfunktionen ergänzt.

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

- **ViewController**: Stellt zentrale Views wie Login, Registrierung und Reset-Formulare bereit. Diese Klasse ist bewusst einfach gehalten und hat sich im Sprint 4 nicht verändert.

```
@GetMapping("/register")
public String registerPage() {
    return "register";
}
```

- **NotificationService**: Dient dem Versand von E-Mail-Benachrichtigungen. Dieser Service wurde im vierten Sprint um neue Benachrichtigungstypen erweitert, insbesondere im Zusammenhang mit Kommentaren oder Aufgabenveränderungen.

```
public void sendCommentNotification(User user, Task task, String
    commentText) {
    // SMTP-Versandlogik
}
```

- **AccountService**: Beinhaltet Kernlogik für Authentifizierung, Registrierung und Passwortänderung. Die Passwortlogik wurde im vierten Sprint durch eine verbesserte Token-Verwaltung erweitert.

```
public void resetPassword(String token, String newPassword) {
    // Passwort zur cksetzen, Validierung durchf hren
}
```

- **DTOs und Request-Klassen**: Diese Datenübertragungsobjekte wurden sowohl in Sprint 3 als auch 4 genutzt. In Sprint 4 wurde insbesondere die Trennung zwischen interner Entität und externer Repräsentation geschärft.
  - `RegisterRequest.java` – Felder: Name, E-Mail, Passwort
  - `UserDTO.java` – Felder: ID, Name, Rolle

9 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.
- **Gradle:** Als Build- und Abhängigkeitsmanagement-Tool wurde `Gradle` verwendet, insbesondere zur Konfiguration von Projektmodulen, Testausführung und Build-Optimierung über das `Java Plugin`. Die Verwaltung externer Bibliotheken (Spring Boot, Web, Security, usw.) erfolgt ebenfalls darüber.
- **LLM-Unterstützung bei Schlüsselgenerierung:** Während der Konfiguration von HTTPS und der Erstellung des asymmetrischen `.p12`-Schlüsselpaares kamen große Sprachmodelle (LLMs, z. B. ChatGPT) unterstützend zum Einsatz..

  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.

10 3.2 Dokumentation der Codequalität

Abweichungen des Codes zur geplanten Architektur

a) Beschreibung der Abweichung

Im dritten und vierten Sprint wurde die ursprünglich monolithische Controller-Architektur stärker modularisiert. Während zentrale Komponenten wie `UserController`, `TaskController` und `TaskRestController` bestehen blieben, wurden zahlreiche neue Controller-Klassen eingeführt oder ergänzt, um neue Features besser zu kapseln oder technische Herausforderungen zu umgehen.

Zusätzlich entstanden temporäre oder unterstützende Komponenten, z. B. zur Fehlerbehebung, Protokollierung von WebSocket-Aktivitäten oder zur Realisierung von globalen Kommunikationsfunktionen. Folgende zusätzliche Klassen weichen von der ursprünglichen Planung ab:

- `PlanningPokerController`
- `UserApiController`

- `PasswordResetRestController`
- `CommentWebSocketController`
- `WebSocketConfig`
- `ChatMessageController` (für globalen Chatraum)

b) Begründung der Abweichung

Diese Erweiterungen und Anpassungen waren notwendig, um die geplanten Features wie Live-Kommentare, Planning Poker, Passwortzurücksetzung oder globale Benutzerkommunikation technisch korrekt und testbar zu implementieren.

- `PlanningPokerController`: Ursprünglich war geplant, Votes direkt über den bestehenden `TaskController` zu verarbeiten. Aufgrund der starken Trennung von WebSocket-Kommunikation und regulärer Aufgabenlogik wurde jedoch ein eigener Controller implementiert, der ausschließlich für die Verwaltung von Stimmen, Reveal und Reset innerhalb des Planning Poker verantwortlich ist.
- `CommentWebSocketController`: Das Kommentarsystem sollte ursprünglich rein über HTTP-Anfragen funktionieren. Da aber WebSocket-Kommunikation für Live-Updates notwendig wurde, musste eine eigene WebSocket-Empfangskomponente eingeführt werden. Diese war notwendig, da das Kommentieren über WebSocket zunächst nicht korrekt funktionierte (Kommentare erschienen nicht oder doppelt).
- `UserApiController`: Diese REST-basierte Alternative zum klassischen `UserController` wurde während der Testphase eingeführt, um API-Zugriffe besser zu debuggen. Sie erlaubt einfache JSON-GET/PUT-Zugriffe auf Userdaten und erleichtert somit automatisierte Tests und externe API-Nutzung.
- `PasswordResetRestController`: Ursprünglich war der gesamte Reset-Prozess in einer einzigen Controllerklasse geplant. Aufgrund der Trennung von HTML-Views und API-Zugriffen sowie Problemen mit Formular-Validierung im Frontend wurde eine REST-Variante hinzugefügt, die reine JSON-Daten verarbeitet.
- `WebSocketConfig`: Die WebSocket-Konfiguration wurde über die standardmäßige Spring Boot Konfiguration hinaus erweitert. Dies war nötig, um gezielte `STOMP`-Themen für Kommentare, Planning Poker, globale Chats und andere Echtzeit-Features zu konfigurieren. Probleme mit CORS, Sessionbindung und Broadcast-Topics wurden durch diese Klasse gelöst.
- `ChatMessageController`: Zur Implementierung eines allgemeinen globalen Chatraums wurde diese neue Controller-Klasse eingeführt. Sie erlaubt es allen eingeloggten Nutzern, in Echtzeit miteinander zu kommunizieren – unabhängig von Aufgaben oder Projekten. Der Chat basiert vollständig auf WebSocket-Technologie und unterstützt Timestamp-basierte Sortierung der Nachrichten.

c) Weitere technische Aspekte

Ein zusätzlicher Fokus lag auf der korrekten Verarbeitung und Formatierung von Datum und Uhrzeit (z. B. für Kommentare, Aufgaben oder Chatnachrichten). Hierzu wurde sowohl auf `java.time.LocalDateTime` in Kombination mit `@JsonFormat`

geachtet als auch auf die fehlerfreie Anzeige im Frontend durch konfiguriertes Datumsformat in Thymeleaf und JavaScript.

d) Nächste Schritte

Da es sich um den letzten Sprint handelt, wird es streng genommen keine „nächsten Schritte“ im Sinne von weiterer Implementierung geben – es sei denn, jemand möchte sich in seiner Freizeit freiwillig mit Refactoring austoben.

Trotzdem ließen sich aus technischer Sicht einige klare Optimierungspunkte für eine potenzielle Weiterentwicklung ableiten:

- Einige spezialisierte Controller wie der `CommentWebSocketController` oder `UserApiController` könnten zukünftig in bestehende Hauptcontroller überführt werden, sobald Stabilität, Testabdeckung und UI-Integration ausreichend abgesichert sind.
- Die `WebSocketConfig` könnte so erweitert werden, dass zusätzliche Sicherheitsfeatures (z. B. Authentifizierungsprüfungen auf Topic-Ebene) realisiert werden.
- Auch die Service-Schicht könnte granularer modularisiert werden, insbesondere bei überladenen Klassen wie `AccountService` oder `NotificationService`.

Für das aktuelle Projektziel wurde jedoch bewusst auf übermäßige Fragmentierung verzichtet, um Klarheit und Implementierbarkeit innerhalb des Sprintzeitraums zu gewährleisten.

Clean Code-Prinzipien

- **Kurze Methodenlängen:** Die meisten Methoden bleiben deutlich unter 30 Zeilen, wodurch Lesbarkeit und Testbarkeit erhöht werden. *Beispiel:* `resetPassword()` im `PasswordResetController` umfasst nur ca. 20 Zeilen und verarbeitet sowohl Token-Validierung als auch Passwortaktualisierung effizient.
- **Aussagekräftige Methodennamen:** Methoden wie `uploadFileToTask()`, `addComment()`, oder `getUserProfile()` sind klar benannt und lassen ihre Funktion sofort erkennen.
- **Klar getrennte Verantwortlichkeiten:** Die neue Controller-Architektur des vierten Sprints spiegelt eine klare Trennung der Module wider:
  - `CommentController` – Kommentare via HTML verwalten
  - `TaskFileController` – Datei-Upload und -Download
  - `PasswordResetController` – Token-basierter Passwort-Reset
  - `UserApiController` – REST-Schnittstellen für Benutzerverwaltung
  - `ViewController` – Routing statischer Seiten
  - `PlanningPokerController` – WebSocket-gestützte Abstimmungen

- **Verwendung von DTOs:** Data Transfer Objects wie `RegisterRequest`, `UserDTO`, `TaskFileResponse` oder `EstimationRequest` helfen, die Trennung zwischen Datenmodell und API-Schnittstelle konsequent durchzuhalten.
- **Ausgelagerte Logik:** Validierungs- und Hilfsfunktionen wurden ausgelagert, etwa:
  - `ValidationService` – Pflichtfeldprüfungen
  - `MailService` – SMTP-Mailversand
  - `TaskFileService` – Dateioperationen
  - `EstimationService` – Aufwandsschätzungen auswerten

Die umgesetzten Prinzipien gewährleisten auch im vierten Sprint eine klare, wartbare und modular erweiterbare Codebasis, selbst bei komplexeren Anforderungen wie WebSocket-Integration oder Datei-Upload.

- **White-box-Tests:** Direkt auf Service-Ebene (z. B. `TaskServiceTest`), `um interne Logik zu testen.`
- `Black-box-Tests: Über MockMvc für REST-Controller, um das Verhalten aus Sicht eines API-Nutzers zu testen:`
  `TaskRestControllerTest, SubtaskControllerTest, AuthControllerTest, UserControllerTest`
- `Konfigurations-Tests: Für Sicherheits-, Web- und WebSocket-Konfigurationen:`
  `SecurityConfigTest, WebConfigTest, WebSocketConfigTest`
- `Controller-Tests: Tests für View- und REST-Controller:`
  `AuthViewControllerTest, BoardControllerTest, CommentWebSocketControllerTest, MainControllerTest, NotificationControllerTest, PlanningPokerControllerTest, ProjectControllerTest, SubtaskControllerTest, TaskListControllerTest, TaskRestControllerTest, UserApiControllerTest, UserControllerTest, ViewControllerTest`
- `Service-Tests: Umfassende Tests auf Service-Ebene:`
  `AccountServiceTest, AuthorizationServiceTest, CommentServiceTest, CustomUserDetailsServiceTest, EmailVerificationServiceTest, FileStorageServiceTest, NotificationServiceTest, PasswordResetServiceTest, SubtaskServiceTest, TaskFileServiceTest, TaskListServiceTest, TaskQueryServiceTest, TaskServiceTest, UserServiceTest, ValidationServiceTest`
- `Integrationstests:`
  `PasswordResetIntegrationTest, DemoApplicationTests`

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.

Tabelle 2: Testübersicht – Ausgewählte Testfälle im 4. Sprint

Test-ID	Datum	Beschreibung	Ergebnis
TC1	20.06.2025	Unit-Test: `TaskService.getTask()` mit gültiger ID	Pass
TC2	20.06.2025	Unit-Test: `TaskService.getTask()` mit ungültiger ID (String)	Pass
TC3	20.06.2025	Unit-Test: `TaskService.getTask()` mit nicht vorhandener ID	Pass
TC4	20.06.2025	REST-Test: POST `/api/tasks` mit gültigem JSON (Task erstellen)	Pass
TC5	20.06.2025	REST-Test: POST `/api/tasks` mit leerem JSON (Validierung greift)	Pass
TC6	20.06.2025	Controller-Test: Registrierung mit gültigen Daten	Pass
TC7	20.06.2025	Controller-Test: Registrierung mit fehlenden Feldern	Pass
TC8	20.06.2025	Controller-Test: Registrierung mit bereits verwendeter E-Mail	Pass
TC9	20.06.2025	SubtaskController: GET `/subtasks/{id}` mit gültiger ID	Pass
TC10	20.06.2025	SubtaskController: GET `/subtasks/{id}` mit ungültiger ID	Pass
TC11	20.06.2025	SubtaskController: POST `/subtasks/create` mit Testdaten	Pass
TC12	20.06.2025	Integrationstest: PasswordReset – Token generieren	Pass
TC13	20.06.2025	Integrationstest: PasswordReset – Passwort zurücksetzen	Pass
TC14	20.06.2025	Controller-Test: GET `/api/users/{id}` gibt korrekte User-Daten zurück	Pass
TC15	20.06.2025	SecurityConfigTest prüft Sicherheitskonfiguration	Pass
TC16	20.06.2025	WebConfigTest prüft Web MVC Konfiguration	Pass
TC17	20.06.2025	WebSocketConfigTest prüft WebSocket Setup	Pass
TC18	20.06.2025	AuthViewControllerTest testet Auth-View-Endpunkte	Pass
TC19	20.06.2025	BoardControllerTest testet Board-spezifische Endpunkte	Pass
TC20	20.06.2025	CommentWebSocketControllerTest testet Kommentar-WebSocket Funktionalität	Pass

Fortsetzung auf nächster Seite

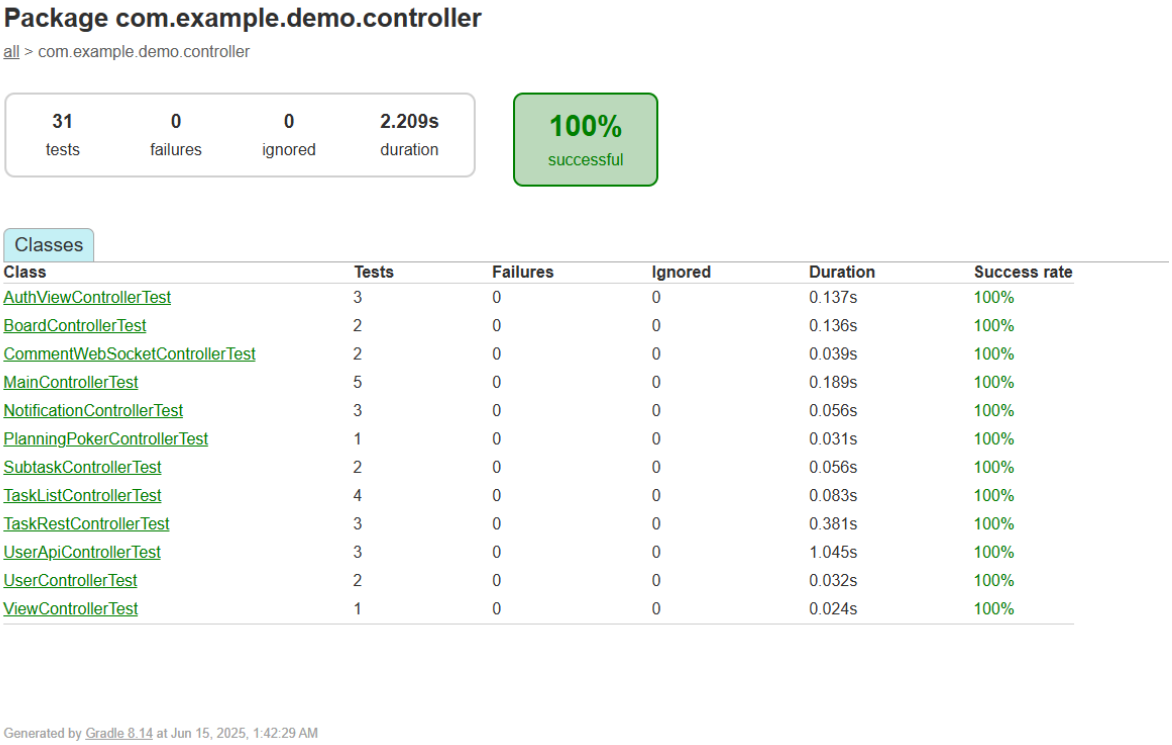
Fortsetzung von vorheriger Seite

Test-ID	Datum	Beschreibung	Ergebnis
TC21	20.06.2025	MainControllerTest testet Hauptseiten-Controller	Pass
TC22	20.06.2025	NotificationControllerTest prüft Notification API	Pass
TC23	20.06.2025	PlanningPokerControllerTest testet Planning Poker Funktionen	Pass
TC24	20.06.2025	ProjectControllerTest testet Projekt-Endpoints	Pass
TC25	20.06.2025	SubtaskControllerTest prüft Subtask API	Pass
TC26	20.06.2025	TaskListControllerTest prüft TaskList API	Pass
TC27	20.06.2025	TaskRestControllerTest prüft Task REST API	Pass
TC28	20.06.2025	UserApiControllerTest prüft User API	Pass
TC29	20.06.2025	UserControllerTest prüft User-Management API	Pass
TC30	20.06.2025	ViewControllerTest prüft UI-spezifische Controller	Pass
TC31	20.06.2025	AccountServiceTest testet Account-Service-Logik	Pass
TC32	20.06.2025	AuthorizationServiceTest testet Autorisierungslogik	Pass
TC33	20.06.2025	CommentServiceTest testet Kommentar-Logik	Pass
TC34	20.06.2025	CustomUserDetailsServiceTest testet CustomUserDetails-Service	Pass
TC35	20.06.2025	EmailVerificationServiceTest testet Email-Verifikation	Pass
TC36	20.06.2025	FileStorageServiceTest testet Datei-Upload/-Speicherung	Pass
TC37	20.06.2025	NotificationServiceTest testet Notification-Logik	Pass
TC38	20.06.2025	PasswordResetServiceTest testet Passwort-Reset-Logik	Pass
TC39	20.06.2025	SubtaskServiceTest testet Subtask-Logik	Pass
TC40	20.06.2025	TaskFileServiceTest testet Task-File-Handling	Pass
TC41	20.06.2025	TaskListServiceTest testet TaskList-Logik	Pass
TC42	20.06.2025	TaskQueryServiceTest testet Task-Abfragen	Pass
TC43	20.06.2025	TaskServiceTest testet Task-Logik	Pass
TC44	20.06.2025	UserServiceTest testet User-spezifische Logik	Pass
TC45	20.06.2025	ValidationServiceTest testet Validierungsregeln	Pass
TC46	20.06.2025	DemoApplicationTests Integrationstest des Gesamtsystems	Pass

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Test-ID	Datum	Beschreibung	Ergebnis
TC47	20.06.2025	PasswordResetIntegrationTest Integrationstest Passwort-Reset Prozess	Pass



Package com.example.demo.controller

all > com.example.demo.controller

31 tests	0 failures	0 ignored	2.209s duration	100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
AuthViewControllerTest	3	0	0	0.137s	100%
BoardControllerTest	2	0	0	0.136s	100%
CommentWebSocketControllerTest	2	0	0	0.039s	100%
MainControllerTest	5	0	0	0.189s	100%
NotificationControllerTest	3	0	0	0.056s	100%
PlanningPokerControllerTest	1	0	0	0.031s	100%
SubtaskControllerTest	2	0	0	0.056s	100%
TaskListControllerTest	4	0	0	0.083s	100%
TaskRestControllerTest	3	0	0	0.381s	100%
UserApiControllerTest	3	0	0	1.045s	100%
UserControllerTest	2	0	0	0.032s	100%
ViewControllerTest	1	0	0	0.024s	100%

Generated by Gradle 8.14 at Jun 15, 2025, 1:42:29 AM

Abbildung 6: Ausgeführte Tests im Browser über `index.html`

Ablageort der Tests:

- `src/test/java/com/example/demo/config/SecurityConfigTest.java`
- `src/test/java/com/example/demo/config/WebConfigTest.java`
- `src/test/java/com/example/demo/config/WebSocketConfigTest.java`
- `src/test/java/com/example/demo/controller/AuthViewControllerTest.java`
- `src/test/java/com/example/demo/controller/BoardControllerTest.java`
- `src/test/java/com/example/demo/controller/CommentWebSocketControllerTest.java`
- `src/test/java/com/example/demo/controller/MainControllerTest.java`
- `src/test/java/com/example/demo/controller/NotificationControllerTest.java`
- `src/test/java/com/example/demo/controller/PlanningPokerControllerTest.java`
- `src/test/java/com/example/demo/controller/ProjectControllerTest.java`
- `src/test/java/com/example/demo/controller/SubtaskControllerTest.java`
- `src/test/java/com/example/demo/controller/TaskListControllerTest.java`
- `src/test/java/com/example/demo/controller/TaskRestControllerTest.java`

- `src/test/java/com/example/demo/controller/UserApiControllerTest.java`
- `src/test/java/com/example/demo/controller/UserControllerTest.java`
- `src/test/java/com/example/demo/controller/ViewControllerTest.java`
- `src/test/java/com/example/demo/service/AccountServiceTest.java`
- `src/test/java/com/example/demo/service/AuthorizationServiceTest.java`
- `src/test/java/com/example/demo/service/CommentServiceTest.java`
- `src/test/java/com/example/demo/service/CustomUserDetailsServiceTest.java`
- `src/test/java/com/example/demo/service/EmailVerificationServiceTest.java`
- `src/test/java/com/example/demo/service/FileStorageServiceTest.java`
- `src/test/java/com/example/demo/service/NotificationServiceTest.java`
- `src/test/java/com/example/demo/service/PasswordResetServiceTest.java`
- `src/test/java/com/example/demo/service/SubtaskServiceTest.java`
- `src/test/java/com/example/demo/service/TaskFileServiceTest.java`
- `src/test/java/com/example/demo/service/TaskListServiceTest.java`
- `src/test/java/com/example/demo/service/TaskQueryServiceTest.java`
- `src/test/java/com/example/demo/service/TaskServiceTest.java`
- `src/test/java/com/example/demo/service/UserServiceTest.java`
- `src/test/java/com/example/demo/service/ValidationServiceTest.java`
- `src/test/java/com/example/demo/DemoApplicationTests.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.

11 3.3 Tracing

Auch im vierten Sprint wurde das Tracing zwischen modellierten Anforderungen (z.B. UML-Diagrammen, Use Cases, Sequenzdiagrammen) und der tatsächlichen Code-Implementierung konsistent weitergeführt. Unser GitLab-Projekt enthält eine fortlaufend gepflegte Tracing-Tabelle, die dokumentiert, wie modellierte Komponenten und Prozesse im Code realisiert wurden.

Besonders im Fokus standen in Sprint 4 neue Funktionalitäten wie Datei-Uploads (US-13), Kommentarfunktionen mit WebSocket-Synchronisation (US-14) und die Umsetzung von Planning Poker (US-11). Diese Funktionen wurden modellseitig durch neue Sequenzdiagramme ergänzt und im Code durch spezialisierte Controller und Services abgebildet:

- `TaskFileController`, `TaskFileService` – Umsetzung von Datei-Uploads gemäß US-13
- `CommentController`, `CommentWebSocketController` – synchrone Kommentare via WebSocket (US-14)
- `PlanningPokerController`, `WebSocketConfig` – Durchführung von Planning Poker Sessions (US-11)
- `PasswordResetRestController` – REST-basierte Variante zur Passwortzurücksetzung
- `UserApiController` – Debugging-Schnittstelle zur Userverwaltung (entstanden durch fehlgeschlagene Tests)

Durch diese Erweiterungen wurden modellierte Systemfunktionen nicht nur erweitert, sondern auch technisch präzisiert. Das Tracing bleibt dadurch ein zentrales Bindeglied zwischen Spezifikation und Implementierung.

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.

12 3.4 Laufender Prototyp, Installation und Kompilation

Der aktuelle Prototyp lässt sich lokal mit wenigen Befehlen starten – ganz ohne zusätzliche Tools. Das Projekt basiert auf `Spring Boot` und nutzt den integrierten `Gradle Wrapper`, was eine einfache Ausführung ohne externes Gradle-Setup erlaubt.

Schritte zur lokalen Ausführung

- **1. Repository klonen:**
  `git clone https://git.informatik.uni-rostock.de/kk2316/projectuebung2_eren.git`
- **2. Projektverzeichnis öffnen:**
  `cd projectuebung2_eren/Team1_demo_project`
- **3. Anwendung starten:**
  Terminal öffnen (Linux, macOS, Windows oder in `Neovim` mit `:terminal`):
  - `./gradlew bootRun` (Linux/macOS)
  - `.\gradlew.bat bootRun` (Windows)
- **4. Anwendung im Browser öffnen:**
  Nach erfolgreichem Start unter entweder: `http://localhost:8080` oder `http://localhost:8443`

Konfiguration und Anpassung

Alle zentralen Einstellungen (Ports, Mailserver, Tokens etc.) befinden sich in: `src/main/resources/application.properties`

Beispielhafte Konfigurationseinträge:

- Mail-Versand (SMTP)
- Server-Port und Session-Timeouts
- Datenbankverbindung (via Spring JPA)

HTTPS-Unterstützung

HTTPS ist optional aktivierbar über einen eingebetteten `.p12`-Keystore mit asymmetrischer Verschlüsselung. Konfiguration über:

```
server.port=8443
server.ssl.enabled=true
server.ssl.key-store=classpath:keystore.p12
server.ssl.key-store-password=yourpass
server.ssl.key-store-type=PKCS12
server.ssl.key-alias=tomcat
```

Listing 1: HTTPS-Konfiguration

Weitere Informationen

Eine vollständige Schritt-für-Schritt-Anleitung befindet sich im Repository unter: `README.md`

13 3.5 Abweichung Sprintplanung

Im vierten Sprint kam es erneut zu kleineren Abweichungen gegenüber der ursprünglichen Planung, da sich während der Umsetzung sowohl technische Herausforderungen als auch neue Optimierungsmöglichkeiten ergaben.

Zusätzliche Umsetzung:

- Die User Stories US-11 (Planning Poker), US-13 (Datei-Upload), US-14 (Kommentar löschen inkl. WebSocket) wurden vollständig umgesetzt – zusätzlich zu bereits geplanten Wartungs- und Refactoring-Aufgaben.
- Darüber hinaus wurden mehrere Features implementiert, die ursprünglich nicht als User Story spezifiziert waren, aber im Verlauf des Sprints einen klaren funktionalen Mehrwert boten:
  - **Globaler Chatraum:** Eine WebSocket-basierte Echtzeitkommunikation, bei der sich alle eingeloggten Nutzer systemweit Nachrichten senden können.

    - **Team-Funktionalität:** Aufgaben, Projekte und Nutzer können nun Teams zugewiesen werden – inkl. Datenmodell und Teamverwaltung.
    - **Datum/Zeit-Handling:** Sämtliche Entitäten (z. B. Kommentare, Chatnachrichten, Benachrichtigungen) wurden mit standardisierten Zeitstempeln (`LocalDateTime`) versehen, um eine konsistente Sortierung und Anzeige zu ermöglichen.
- Zusätzlich wurde mit dem `UserApiController` ein REST-basierter Debug-Endpunkt eingeführt, der ursprünglich nicht vorgesehen war, sich aber als äußerst hilfreich für Tests und Fehleranalyse erwies.

Technisch aufwändige Aufgaben:

- Die WebSocket-basierte Kommentar-Synchronisierung war deutlich aufwändiger als geplant. Die Initialkonfiguration von `WebSocketConfig` sowie die Integration des `CommentWebSocketController` erforderten mehrere Anläufe und Debuggingphasen.
- Der `PasswordResetRestController` musste zusätzlich implementiert werden, da die klassische HTML-basierte Variante nicht alle Anwendungsfälle korrekt abdeckte.
- Die Realisierung des globalen Chatraums erforderte eine stabile STOMP- und Topic-Verwaltung auf Backend-Seite sowie UI-seitige Unterstützung für Live-Updates mit `SockJS`.

Frontend-Herausforderungen:

- Die Integration neuer UI-Komponenten (Modals für Datei-Upload, Planning Poker Voting, Kommentarverwaltung, Teamerstellung) erforderte viele Iterationen im Zusammenspiel mit Thymeleaf, Bootstrap und Controller-Logik.
- Die visuelle Konsistenz mit dunklem UI-Theme, responsivem Design und funktionalem Layout beanspruchte einen unerwartet großen Teil der Zeit.

Gründe für Abweichungen:

- Technische Komplexität bei der WebSocket-Kommunikation und Echtzeit-Aktualisierung im Frontend.
- Notwendigkeit zur Erstellung zusätzlicher Controller (`UserApiController`, `PasswordResetRestController`, `TaskSocketController`) zur besseren Separation und Testbarkeit.
- Feature-Wünsche aus der Entwicklung (z. B. Chat, Teamstruktur, Timestamps) ergaben sich organisch und machten konzeptionell Sinn – auch ohne formale User Story.
- Designtechnische Detailarbeit für eine saubere, intuitive und moderne Nutzeroberfläche.

Lessons Learned:

- Arbeiten im Alleingang hat sich im vierten Sprint als überraschend angenehm, effizient und stressfrei herausgestellt. Ohne Abstimmungsrunden oder Merge-Konflikte konnte ich mich ganz auf Codequalität, Architektur und UI-Details konzentrieren – und das hat richtig Spaß gemacht.

- Eine gute Vorbereitung auf technische Spezialfälle (wie WebSocket-Verbindungen, Zeitformatierung) spart am Ende viel Zeit.
- Guter Code ist nur die halbe Miete – genauso wichtig ist es, dass das System auch angenehm zu bedienen und visuell nachvollziehbar ist.

Ausblick:

- Auch wenn dieser Sprint der letzte im Rahmen des Projekts ist, vielleicht kommt doch noch irgendwelche Kleinigkeiten. Die Architektur wäre jedenfalls bereit dafür.
- Besonders die Tests für neue Komponenten wie Planning Poker, Chat oder Dateihandling könnten noch robuster und umfangreicher ausgebaut werden.

14 Dokumentation individueller Beiträge

Tabelle 3: 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