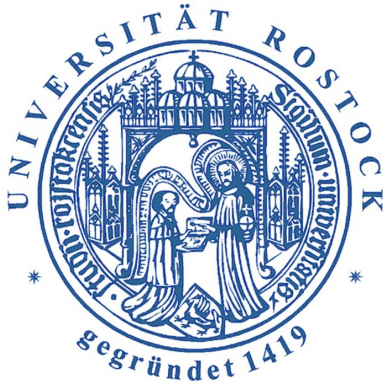
Schön, dass ihr das nochmal mit Latex geschrieben habt!



Softwaretechnik WS 2024-25
Projektabgabedokument Entwurf

Übung-2 Team-1

Dogukan Karakoyun, 223202023
Hüseyin Kayabasi, 223201801
Helin Oguz, 223202103
Eren Temizkan, 223201982
Cagla Yesildogan, 223201881

Inhaltsverzeichnis

1 Initialer Backlog

1.1 User Stories

1. **Account Management**

   - **ID: US-1.1**
     Als neuer Nutzer möchte ich einen Account anlegen können, damit ich die Plattform nutzen und auf personalisierte Inhalte zugreifen kann.
     Traced to: ANF-001
   - **ID: US-1.2**
     Als registrierter Nutzer möchte ich mich mit meinen Zugangsdaten einloggen können, damit ich auf meine Daten und personalisierte Inhalte zugreifen kann.
     Traced to: ANF-001
   - **ID: US-1.3**
     Als registrierter Nutzer möchte ich mein Passwort zurücksetzen können, falls ich es vergessen habe, damit ich wieder Zugriff auf meinen Account erhalte.
     Traced to: ANF-001

2. **Backlog Management**

   - **ID: US-2.1**
     Als Product Owner (PO) möchte ich neue User Stories im Backlog anlegen können, damit ich die Anforderungen des Projekts dokumentieren und verwalten kann.
     Traced to: ANF-002
   - **ID: US-2.2**
     Als Product Owner (PO) möchte ich die Prioritäten der User Stories im Backlog festlegen können, damit das Team an den wichtigsten Aufgaben zuerst arbeitet.
     Traced to: ANF-002

3. **Task Management**

   - **ID: US-3.1**
     Als Teammitglied möchte ich Tasks erstellen können, damit ich die Arbeit in kleinere Einheiten aufteilen kann.
     Traced to: ANF-003
   - **ID: US-3.2**
     Als Teammitglied möchte ich Tasks mit den entsprechenden User Stories verlinken können, damit die Zusammenhänge klar ersichtlich sind.
     Traced to: ANF-003
   - **ID: US-3.3**
     Als Teammitglied möchte ich die Tasks priorisieren können, damit ich sicherstellen kann, dass die wichtigsten Aufgaben zuerst erledigt werden.
     Traced to: ANF-003
   - **ID: US-3.4**
     Als Teammitglied möchte ich Tasks als „complete“ oder „incomplete“ markieren können, damit der Fortschritt der Arbeit klar sichtbar ist.
     Traced to: ANF-003

4. **Task/Story Listen**

   - **ID: US-4.1**
     Als Teammitglied möchte ich mehrere Listen für Tasks und User Stories erstellen können, damit ich zwischen Projekt Backlog und Sprint Backlog klar unterscheiden und diese unabhängig voneinander verwalten kann.
     Traced to: ANF-004
   - **ID: US-4.2**
     Als Teammitglied möchte ich verschiedene Status für Tasks anlegen können (z. B. in Bearbeitung, unter Review, fertiggestellt), damit der Fortschritt besser dokumentiert werden kann.
     Traced to: ANF-004
   - **ID: US-4.3**
     Als Teammitglied möchte ich den Status von Tasks ändern können, damit der Fortschritt der Arbeit aktuell gehalten werden kann.
     Traced to: ANF-004
   - **ID: US-4.4**
     Als Teammitglied möchte ich Tasks zwischen den Listen (z. B. Projekt Backlog, Sprint Backlog oder Status-Listen) hin- und herbewegen können, damit ich Aufgaben flexibel organisieren und an den aktuellen Arbeitsablauf anpassen kann.
     Traced to: ANF-004
   - **ID: US-4.5**
     Als Teammitglied möchte ich eine übersichtliche Darstellung der Listen und Tasks mit Filter- und Sortieroptionen sehen können, damit ich den Fortschritt des Projekts besser nachvollziehen kann.
     Traced to: ANF-004

5. **Nutzer Profile**

   - **ID: US-5.1**
     Als Teammitglied möchte ich Profile anderer Nutzer mit persönlichen Informationen (z. B. Name und Rolle) ansehen können, damit ich die Rollen und Verantwortlichkeiten meines Teams besser verstehe.
     Traced to: ANF-005
   - **ID: US-5.2**
     Als Nutzer möchte ich mein eigenes Profil bearbeiten können, damit ich meine Informationen aktuell und korrekt halten kann.
     Traced to: ANF-005
   - **ID: US-5.3**
     Als Administrator möchte ich die Rollen der Nutzer (z. B. Tester, PO, Entwickler) festlegen und anpassen können, damit die Verantwortlichkeiten innerhalb des Teams klar definiert sind.
     Traced to: ANF-005

6. **Nutzer-Task Zuordnung**

- **ID: US-6.1**
  Als Teammitglied möchte ich Nutzer zu spezifischen Tasks zuordnen können, damit klar ersichtlich ist, wer für welche Aufgabe verantwortlich ist.
  Traced to: ANF-006
- **ID: US-6.2**
  Als Teammitglied möchte ich eine visuelle Übersicht der Nutzer-Task Zuordnungen (z. B. in einer Tabelle oder einem Dashboard) haben, damit ich schnell erkennen kann, wer für welche Aufgabe verantwortlich ist.
  Traced to: ANF-006

7. **Pair Programming**

- **ID: US-7.1**
  Als Teammitglied möchte ich mehreren Nutzern eine einzelne Task zuordnen können, damit Pair-Programming oder Teamarbeit an einer Aufgabe geplant und umgesetzt werden kann.
  Traced to: ANF-007
- **ID: US-7.2**
  Als Teammitglied möchte ich eine Übersicht über alle Tasks haben, bei denen Pair-Programming geplant ist, damit ich sehen kann, welche Aufgaben in Zusammenarbeit erledigt werden.
  Traced to: ANF-007

8. **Schätzungstracking**

- **ID: US-8.1**
  Als Teammitglied möchte ich zu jeder Task eine Aufwandsschätzung hinzufügen können, damit der voraussichtliche Zeitbedarf klar dokumentiert ist.
  Traced to: ANF-008
- **ID: US-8.2**
  Als Teammitglied möchte ich die tatsächliche Bearbeitungszeit einer Task dokumentieren können, damit ich nachvollziehen kann, wie lange die Lösung tatsächlich gedauert hat.
  Traced to: ANF-008
- **ID: US-8.3**
  Als Teammitglied möchte ich eine Visualisierung der Abweichungen zwischen geschätzter und tatsächlicher Bearbeitungszeit sehen, damit ich die Genauigkeit unserer Schätzungen analysieren und verbessern kann.
  Traced to: ANF-008

9. **Suche und Filter**

- **ID: US-9.1**
  Als Nutzer möchte ich Aufgaben nach Prioritäten filtern können, damit ich effektiver arbeiten kann.
  Traced to: ANF-009
- **ID: US-9.2**
  Als Nutzer möchte ich Aufgaben nach ihrem Status filtern können, damit ich

sicherstellen kann, dass die Aufgaben in der richtigen Reihenfolge erledigt werden.
Traced to: ANF-009

- **ID: US-9.3**
  Als Benutzer möchte ich die Person sehen können, die einer Aufgabe zugeordnet ist, damit ich mit der zuständigen Person Kontakt aufnehmen kann.
  Traced to: ANF-009

10. **Benachrichtigungen**

- **ID: US-10.1**
  Als Nutzer möchte ich vom System per E-Mail oder in der App benachrichtigt werden, wenn Aufgaben anstehen, damit ich rechtzeitig eingreifen kann, ohne Fristen zu verpassen.
  Traced to: ANF-010

11. **Agile Practices**

- **ID: US-11.1**
  Als Product Owner möchte ich, dass das System Agile Praktiken wie Planning Poker für die Aufwandsschätzung unterstützt, damit mein Team die Aufgaben priorisieren und effizient planen kann.
  Traced to: ANF-011
- **ID: US-11.2**
  Als Product Owner möchte ich, dass mein Team Retrospektiven und andere Besprechungen über das System durchführen kann, damit Verbesserungsvorschläge dokumentiert und Entscheidungen nachverfolgt werden können.
  Traced to: ANF-011

12. **Synchronisation**

- **ID: US-12.1**
  Als Benutzer möchte ich, dass meine Änderungen an einer Aufgabe für alle Teammitglieder innerhalb von maximal 2 Sekunden synchronisiert und sichtbar sind, damit alle auf dem Laufenden sind und Missverständnisse vermieden werden.
  Traced to: ANF-012

13. **Parallele Bearbeitung**

- **ID: US-13.1**
  Als Benutzer möchte ich, dass mehrere Nutzer gleichzeitig an einer Aufgabe arbeiten können, damit die Arbeit schneller abgeschlossen werden kann.
  Traced to: ANF-013

14. **Multi-Nutzer**

- **ID: US-14.1**
  Als Admin möchte ich, dass mehrere Benutzer gleichzeitig am System arbeiten können, damit parallele Aufgabenbearbeitung ermöglicht wird.
  Traced to: ANF-014

- **ID: US-14.2**
  Als Admin möchte ich, dass die Benutzer die Aufgaben flexibel organisieren können, damit die Arbeit im Team strukturiert und nachvollziehbar bleibt.
  Traced to: ANF-014

15. **Verfügbarkeit**

- **ID: US-15.1**
  Als Nutzer möchte ich zu jeder Tageszeit auf das System zugreifen und meine Aufgaben bearbeiten können, damit ich flexibel arbeiten kann.
  Traced to: ANF-015

16. **Updates**

- **ID: US-16.1**
  Als Benutzer möchte ich, dass Änderungen in den Aufgabenlisten innerhalb von maximal 2 Sekunden synchronisiert und für alle Nutzer sichtbar werden, damit der Projektfortschritt immer aktuell bleibt.
  Traced to: ANF-016

17. **Änderungen**

- **ID: US-17.1**
  Das System soll nach seiner Freigabe leicht erweiterbar sein, sodass neue Listentypen hinzugefügt werden können, um verschiedene Arten von Aufgaben und Prozessen flexibel abzubilden.
  Traced to: ANF-017
- **ID: US-17.2**
  Das System soll nach seiner Freigabe leicht erweiterbar sein, sodass neue Filteroptionen hinzugefügt werden können, damit Nutzer relevante Aufgaben schneller finden und priorisieren können.
  Traced to: ANF-017
- **ID: US-17.3**
  Das System soll nach seiner Freigabe leicht erweiterbar sein, sodass neue Benachrichtigungstypen hinzugefügt werden können, damit das Team besser über wichtige Ereignisse und Fristen informiert wird.
  Traced to: ANF-017

1.2 Anforderungen

1. **Nutzerregistrierung und Authentifikation**

- **ID: ANF-1.1**
  Nutzer können ein Konto erstellen, indem sie persönliche Informationen wie Name und E-Mail-Adresse angeben.
- **ID: ANF-1.2**
  Nutzer können sich mit ihrer E-Mail-Adresse und ihrem Passwort einloggen.
- **ID: ANF-1.3**
  Nutzer können ihr Passwort zurücksetzen, wenn sie es vergessen haben.

2. **User Story- und Backlog-Management**

   - **ID: ANF-2.1**
     Der Product Owner (PO) kann neue User Stories im Backlog erstellen.
   - **ID: ANF-2.2**
     Der PO kann User Stories im Backlog priorisieren.
   - **ID: ANF-2.3**
     Das Team kann bestehende User Stories im Backlog aktualisieren oder löschen.

3. **Task Management**

   - **ID: ANF-3.1**
     Das System erlaubt es, Tasks als Verfeinerung der User Stories zu erstellen.
   - **ID: ANF-3.2**
     Tasks können mit User Stories verlinkt werden.
   - **ID: ANF-3.3**
     Tasks können priorisiert werden.
   - **ID: ANF-3.4**
     Tasks können als „complete“ oder „incomplete“ markiert werden.

4. **Task/User Story Listen**

   - **ID: ANF-4.1**
     Das System unterstützt mehrere Listen für Tasks und User Stories.
   - **ID: ANF-4.2**
     Tasks können verschiedene Status haben.
   - **ID: ANF-4.3**
     Der Status eines Tasks kann geändert werden.
   - **ID: ANF-4.4**
     Tasks können zwischen verschiedenen Listen verschoben werden.
   - **ID: ANF-4.5**
     Eine Übersicht über Listen und Tasks soll verfügbar sein.

5. **Nutzer Profile**

   - **ID: ANF-5.1**
     Nutzer können Profile mit persönlichen Informationen ansehen.
   - **ID: ANF-5.2**
     Nutzer können ihr eigenes Profil bearbeiten.
   - **ID: ANF-5.3**
     Administratoren können Nutzerrollen definieren und anpassen.

6. **Nutzer-Task Zuordnung**

   - **ID: ANF-6.1**
     Das System erlaubt es, Nutzer zu spezifischen Tasks zuzuordnen.
   - **ID: ANF-6.2**
     Eine visuelle Übersicht der Nutzer-Task Zuordnung soll verfügbar sein.

7. **Pair Programming**
    - **ID: ANF-7.1**
      Mehrere Nutzer können einer einzelnen Task zugeordnet werden.
    - **ID: ANF-7.2**
      Eine Übersicht über alle Pair-Programming Tasks soll verfügbar sein.
8. **Schätzungs-Tracking**
    - **ID: ANF-8.1**
      Tasks können mit Aufwandsschätzungen versehen werden.
    - **ID: ANF-8.2**
      Tatsächliche Bearbeitungszeiten können dokumentiert werden.
    - **ID: ANF-8.3**
      Eine Visualisierung der Schätzungsabweichungen soll verfügbar sein.
9. **Suche und Filter**
    - **ID: ANF-9.1**
      Aufgaben können nach Prioritäten gefiltert werden.
    - **ID: ANF-9.2**
      Aufgaben können nach Status gefiltert werden.
    - **ID: ANF-9.3**
      Nutzer für Aufgaben können angezeigt werden.
10. **Benachrichtigungen**
    - **ID: ANF-10.1**
      Nutzer sollen per E-Mail oder in der App benachrichtigt werden.
11. **Agile Practices**
    - **ID: ANF-11.1**
      Das System soll Agile Praktiken wie Planning Poker unterstützen.
    - **ID: ANF-11.2**
      Retrospektiven und andere Besprechungen sollen dokumentiert werden können.
12. **Synchronisation**
    - **ID: ANF-12.1**
      Änderungen sollen in Echtzeit für alle Nutzer sichtbar sein.
13. **Parallele Bearbeitung**
    - **ID: ANF-13.1**
      Mehrere Nutzer können parallel an Aufgaben arbeiten.
14. **Multi-Nutzer**
    - **ID: ANF-14.1**
      Das System unterstützt mehrere Nutzer gleichzeitig.

- **ID: ANF-14.2**
  Nutzer können Aufgaben flexibel organisieren.

15. **Verfügbarkeit**

    - **ID: ANF-15.1**
      Das System soll jederzeit verfügbar sein.

16. **Updates**

    - **ID: ANF-16.1**
      Änderungen in Aufgabenlisten werden synchronisiert.

17. **Änderungen**

    - **ID: ANF-17.1**
      Neue Listentypen können hinzugefügt werden.
    - **ID: ANF-17.2**
      Neue Filteroptionen können hinzugefügt werden.
    - **ID: ANF-17.3**
      Neue Benachrichtigungstypen können hinzugefügt werden.

Sehr schön überarbeitet!

1.3 Use Case Diagramm

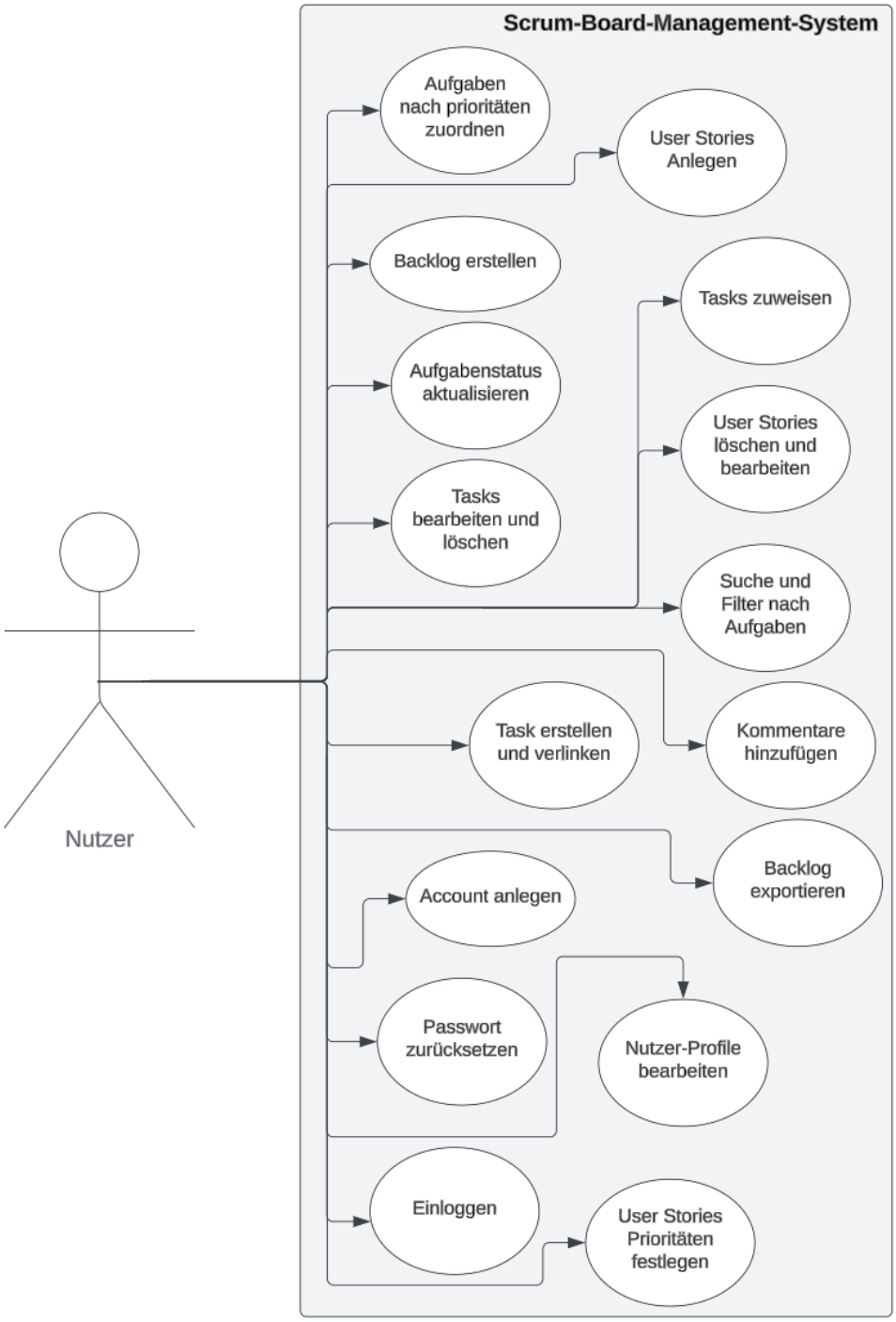


Abbildung 1: Use Case Diagramm

1.4 Klassendiagramm

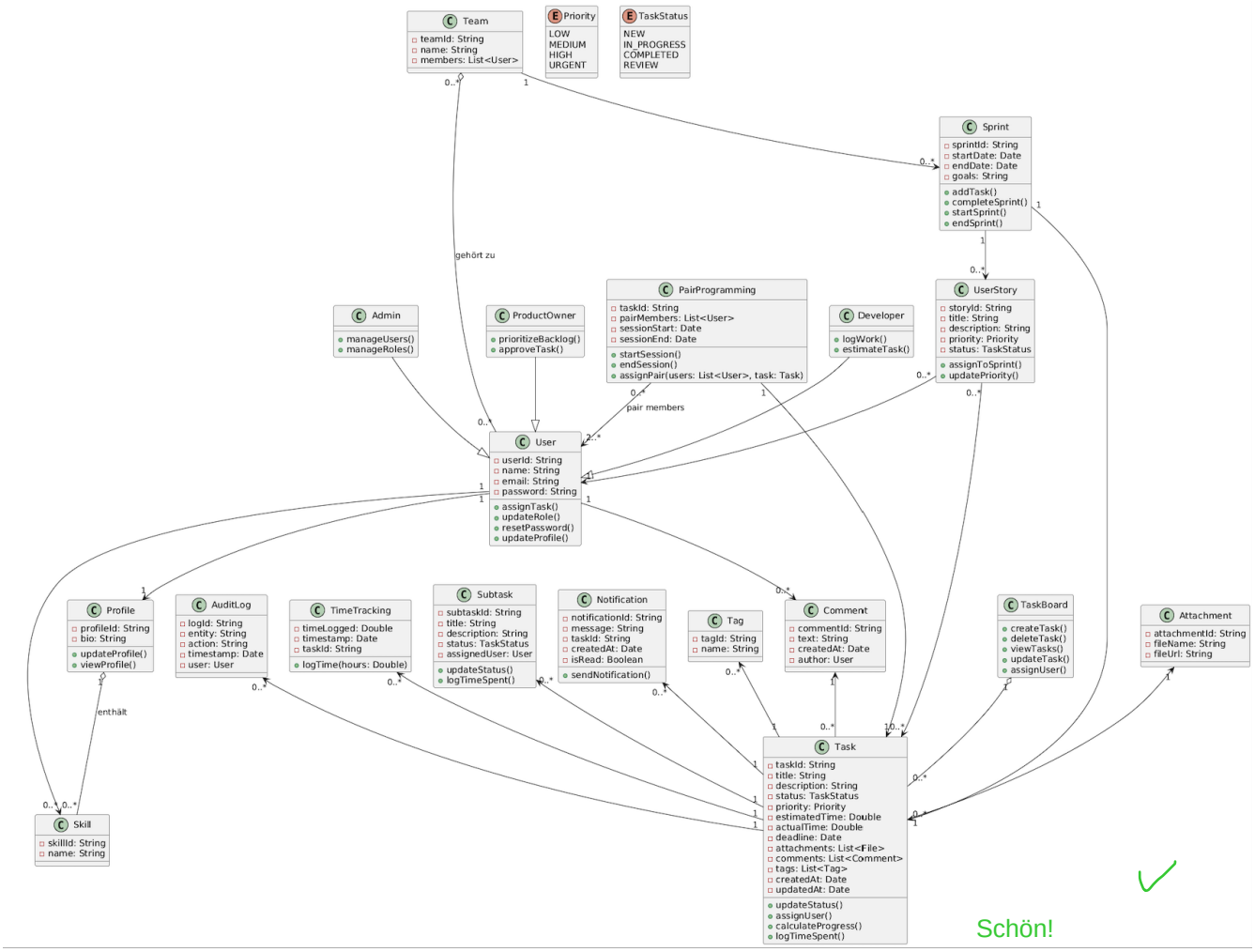


Abbildung 2: Klassendiagramm, das die Domäne des Projektes erfasst

2 Entwurf der Architektur

2.1 Geplantes Architekturmuster und geplante Hauptkomponenten

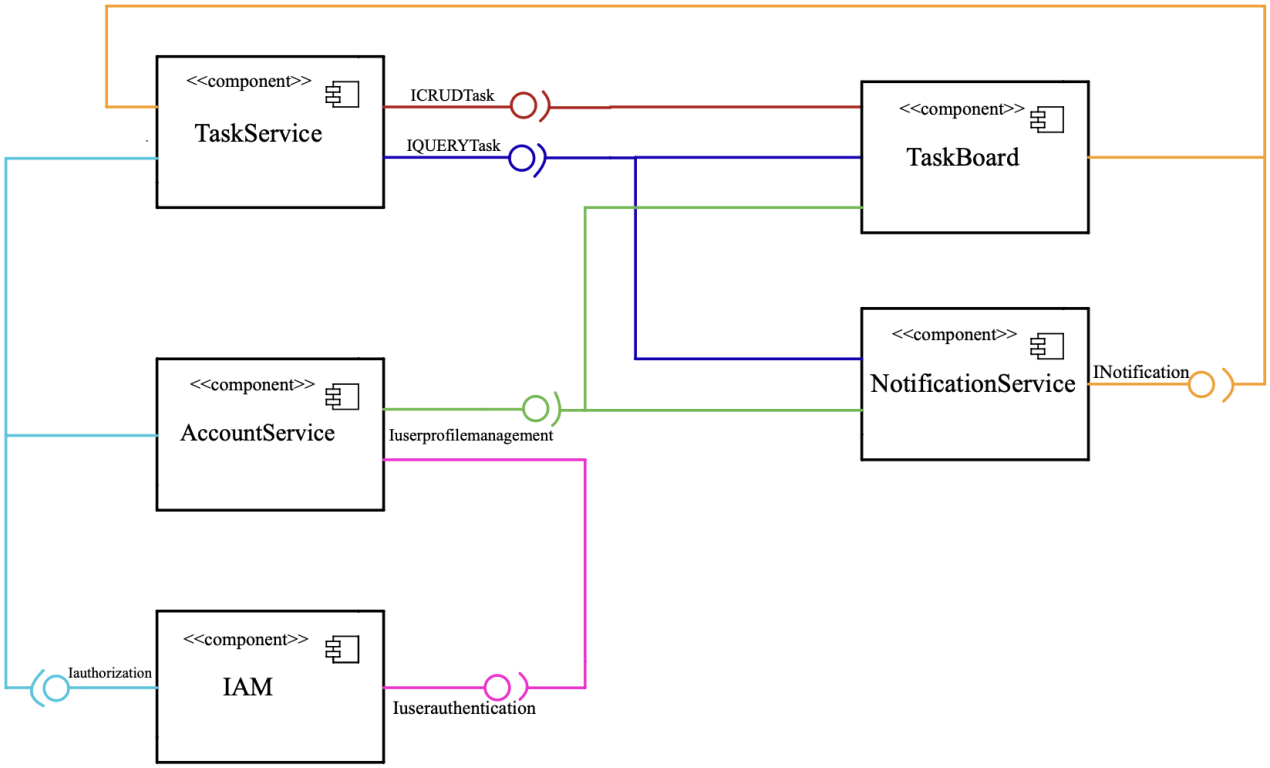
1. **Architekturmuster**

   - *Gewähltes Architekturmuster:*
     Für dieses System wurde das *Layered Architecture*-Muster ausgewählt, da es klare Vorteile in Bezug auf Modularität, Flexibilität und Testbarkeit bietet.
   - **Modularität:**
     - Durch die Trennung in verschiedene Schichten (Presentation, Business Logic und Data Access) können einzelne Teile des Systems unabhängig voneinander entwickelt und gewartet werden.
   - **Flexibilität:**

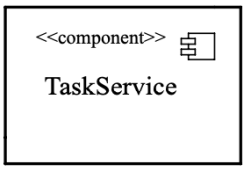
- Änderungen in einer Schicht (z. B. Benutzeroberfläche) wirken sich nicht direkt auf andere Schichten (z. B. Datenzugriff) aus. Dies ermöglicht eine einfache Anpassung an zukünftige Anforderungen.

- **Testbarkeit:**
  - Jede Schicht kann isoliert getestet werden, was die Qualitätssicherung verbessert und Fehler schneller identifizieren lässt.

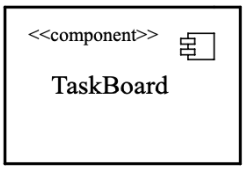
Komponentendiagramm



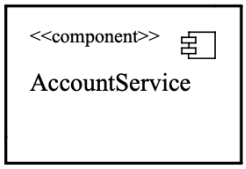
Die Verantwortlichkeiten der Hauptkomponenten



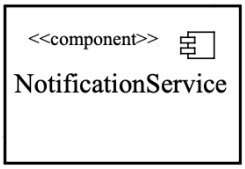
-Verantwortlich für die Task Management
-Suche und Filter



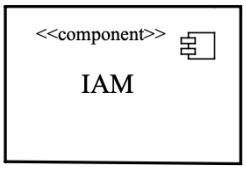
-Visualisierung von Tasks



-Verwaltung von User Account



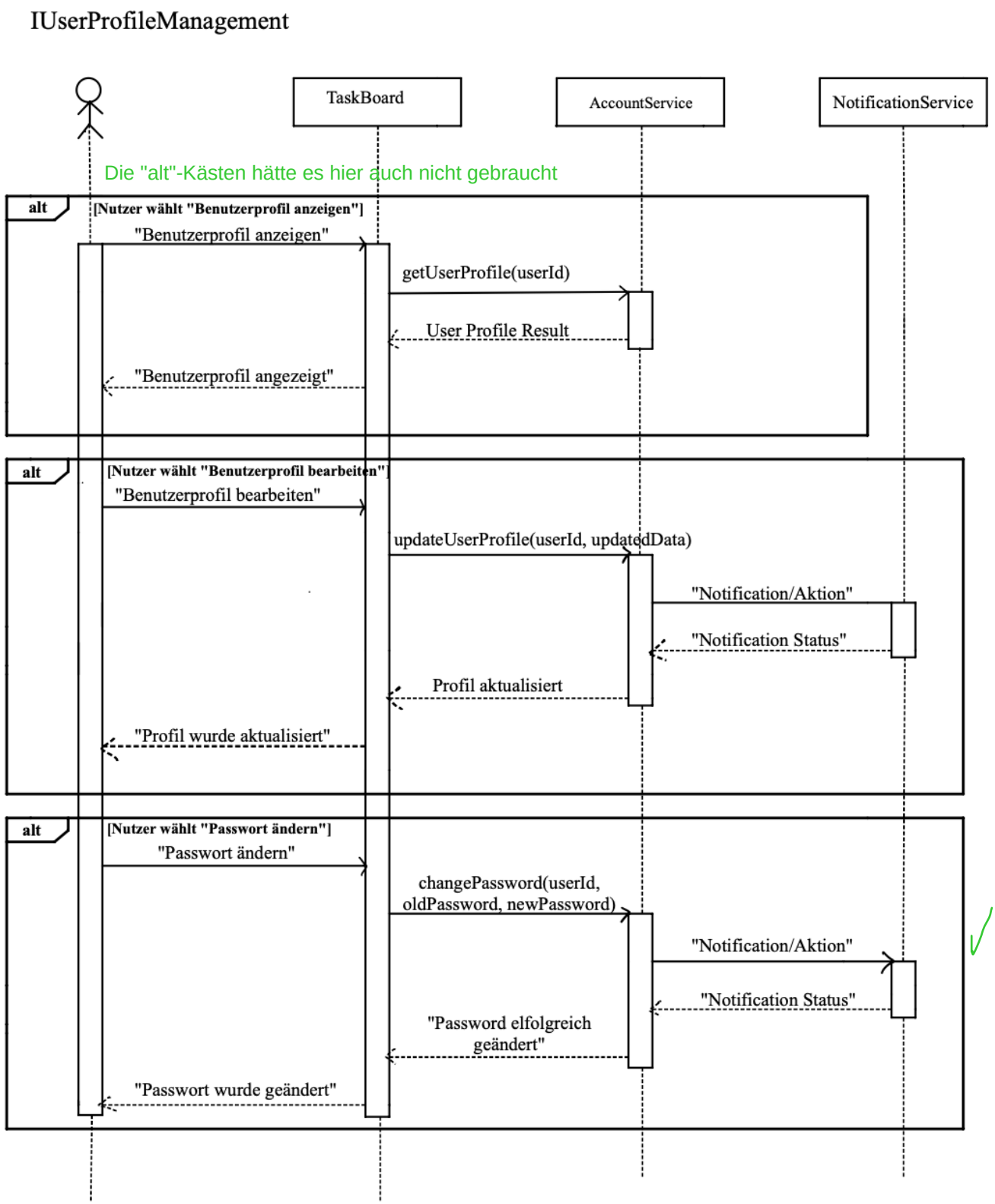
-Zuständig für das Versenden von Benachrichtigungen



-Identity and Access Management
-Authorisierung und Ressourcenzugriffskontrolle

2.2 Interfaces zwischen diesen Komponenten

1. IUserProfileManagement



Beschreibung des Sequenzdiagramms für das IUserProfileManagement-Interface

Das von der **AccountService**-Komponente bereitgestellte `IUserProfileManagement`-Interface enthält die folgenden Methoden:

- `getUserProfile(userId)`: Ruft die Profildaten eines bestimmten Nutzers ab.
- `updateUserProfile(userId, updatedData)`: Aktualisiert die Profildaten eines Nutzers.
- `changePassword(userId, oldPassword, newPassword)`: Ändert das Passwort eines Nutzers.

Dieses Interface wird von der **TaskBoard**-Komponente und der **NotificationService**-Komponente verwendet.

Ablauf im Sequenzdiagramm

Wenn ein Nutzer über das **TaskBoard** eine der genannten Operationen durchführen möchte, klickt er auf den spezifischen Button für die gewünschte Aktion. Jede Aktion wird unabhängig voneinander ausgeführt:

1. Benutzerprofil anzeigen (`getUserProfile`)
Ablauf:

1. Der Nutzer klickt auf den Button „Benutzerprofil anzeigen“ im **TaskBoard**.
2. Das **TaskBoard** sendet die Botschaft `getUserProfile(userId)` an die **AccountService**-Komponente.
3. Das **AccountService** verarbeitet die Anfrage und gibt die `User Profile Result` zurück.
4. Das **TaskBoard** zeigt die Profildaten dem Nutzer an.

Warum keine NotificationService-Einbindung?
Diese Methode ruft nur vorhandene Profildaten ab und führt keine Änderungen oder kritischen Operationen aus. Eine Benachrichtigung ist daher nicht notwendig.

2. Benutzerprofil bearbeiten (`updateUserProfile`)
Ablauf:

1. Der Nutzer klickt auf den Button „Benutzerprofil bearbeiten“ im **TaskBoard** und gibt die neuen Profildaten ein.
2. Das **TaskBoard** sendet die Botschaft `updateUserProfile(userId, updatedData)` an die **AccountService**-Komponente.
3. Das **AccountService** führt die Aktualisierung durch und gibt die Rückmeldung `Updated User Profile Result` zurück.

4. Zusätzlich wird eine Benachrichtigung über die **NotificationService**-Komponente ausgelöst, die den Nutzer über die Änderung informiert.

5. Das **TaskBoard** zeigt dem Nutzer die Bestätigung der erfolgreichen Aktualisierung an: „Profil wurde aktualisiert“.

3. Passwort ändern (`changePassword`)
Ablauf:

1. Der Nutzer klickt auf den Button „Passwort ändern“ im **TaskBoard** und gibt das alte sowie das neue Passwort ein.

2. Das **TaskBoard** sendet die Botschaft `changePassword(userId, oldPassword, newPassword)` an die **AccountService**-Komponente.

3. Das **AccountService** führt die Änderung durch und gibt die Rückmeldung `Password Change Status` zurück.

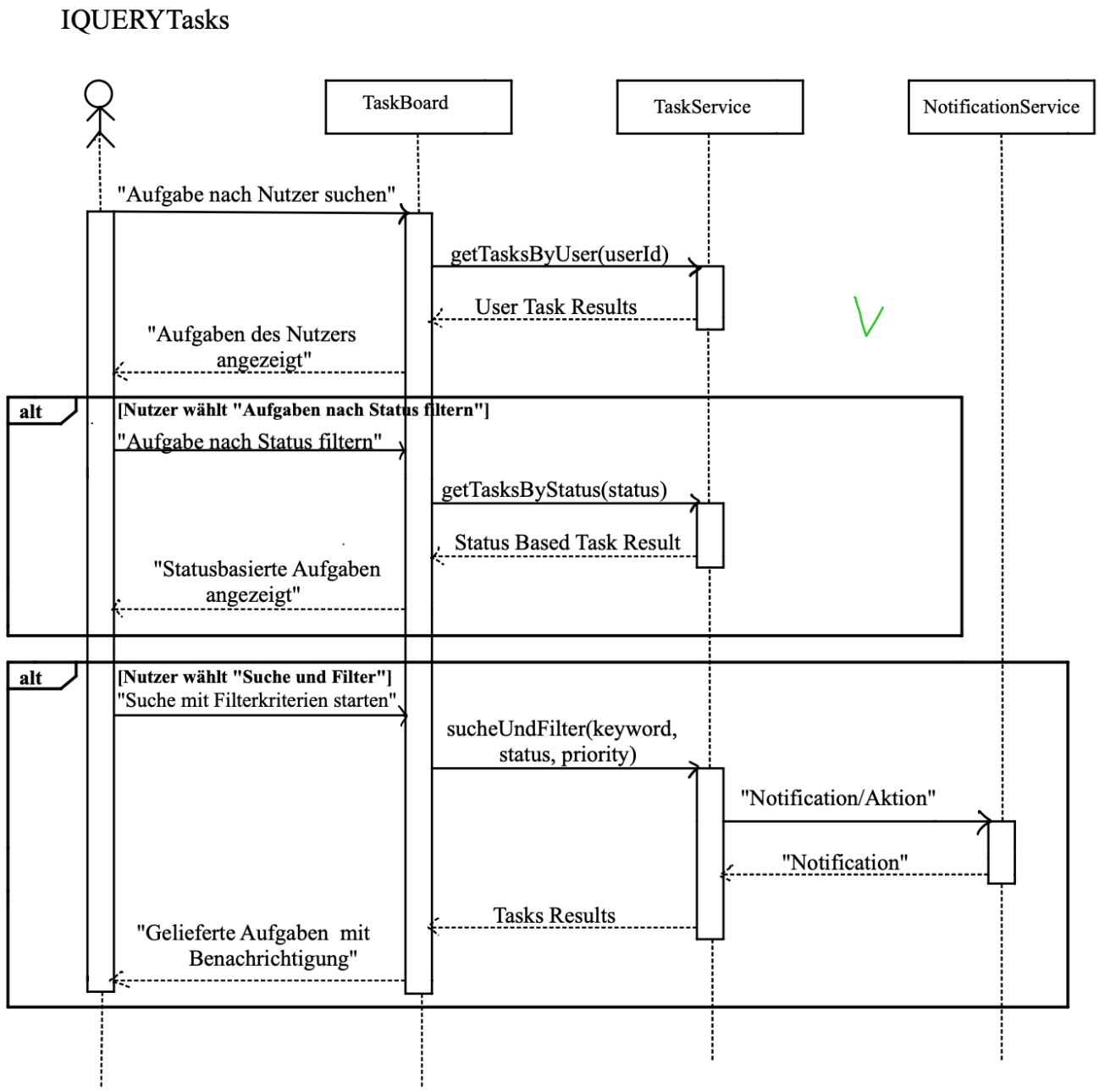
4. Zusätzlich wird eine Benachrichtigung über die **NotificationService**-Komponente ausgelöst, um den Nutzer über die Änderung zu informieren.

5. Das **TaskBoard** zeigt dem Nutzer die Bestätigung der erfolgreichen Passwortänderung an: „Passwort wurde erfolgreich geändert“.

Erklärung der Einbindung der NotificationService-Komponente

Die **NotificationService**-Komponente wird nur in den Methoden `updateUserProfile` und `changePassword` eingebunden, da diese Operationen Änderungen an sensiblen Daten eines Nutzers beinhalten. Diese Änderungen könnten für den Nutzer von hoher Bedeutung sein und erfordern daher eine Benachrichtigung.
Die Methode `getUserProfile` hingegen ruft lediglich vorhandene Daten ab, was keine Benachrichtigung erforderlich macht.

2. IQueryTasks



Beschreibung des Sequenzdiagramms für das IQueryTasks-Interface
Das IQueryTasks-Interface wird von der **TaskService**-Komponente bereitgestellt und enthält die folgenden Methoden:

- `getTasksByUser(userId)`: Gibt alle Aufgaben eines bestimmten Nutzers zurück.
- `getTasksByStatus(status)`: Filtert Aufgaben nach ihrem Status (z. B. „in Bearbeitung“, „abgeschlossen“).
- `sucheUndFilter(keyword, status, priority)`: Führt eine kombinierte Suche basierend auf Schlüsselwörtern, Status und Priorität durch.

Dieses Interface wird von der **TaskBoard**-Komponente verwendet, um die oben genannten Operationen durchzuführen. Ein Nutzer wählt über spezifische Buttons die gewünschte Operation aus:

- Für die Methode `getTasksByUser` klickt der Nutzer auf *„Aufgaben nach Nutzer suchen“*.
- Für die Methode `getTasksByStatus` klickt der Nutzer auf *„Aufgaben nach Status filtern“*.
- Für die Methode `sucheUndFilter` klickt der Nutzer auf *„Suche mit Filterkriterien starten“*.

Das **TaskBoard** leitet diese Anfragen basierend auf der gewählten Aktion an die **TaskService**-Komponente weiter. Je nach Methode gibt die **TaskService**-Komponente folgende Rückmeldungen zurück:

- `UserTaskResults` (für `getTasksByUser`),
- `StatusBasedTaskResult` (für `getTasksByStatus`),
- `Tasks Results` und eine Benachrichtigung über den **NotificationService** (für `sucheUndFilter`).

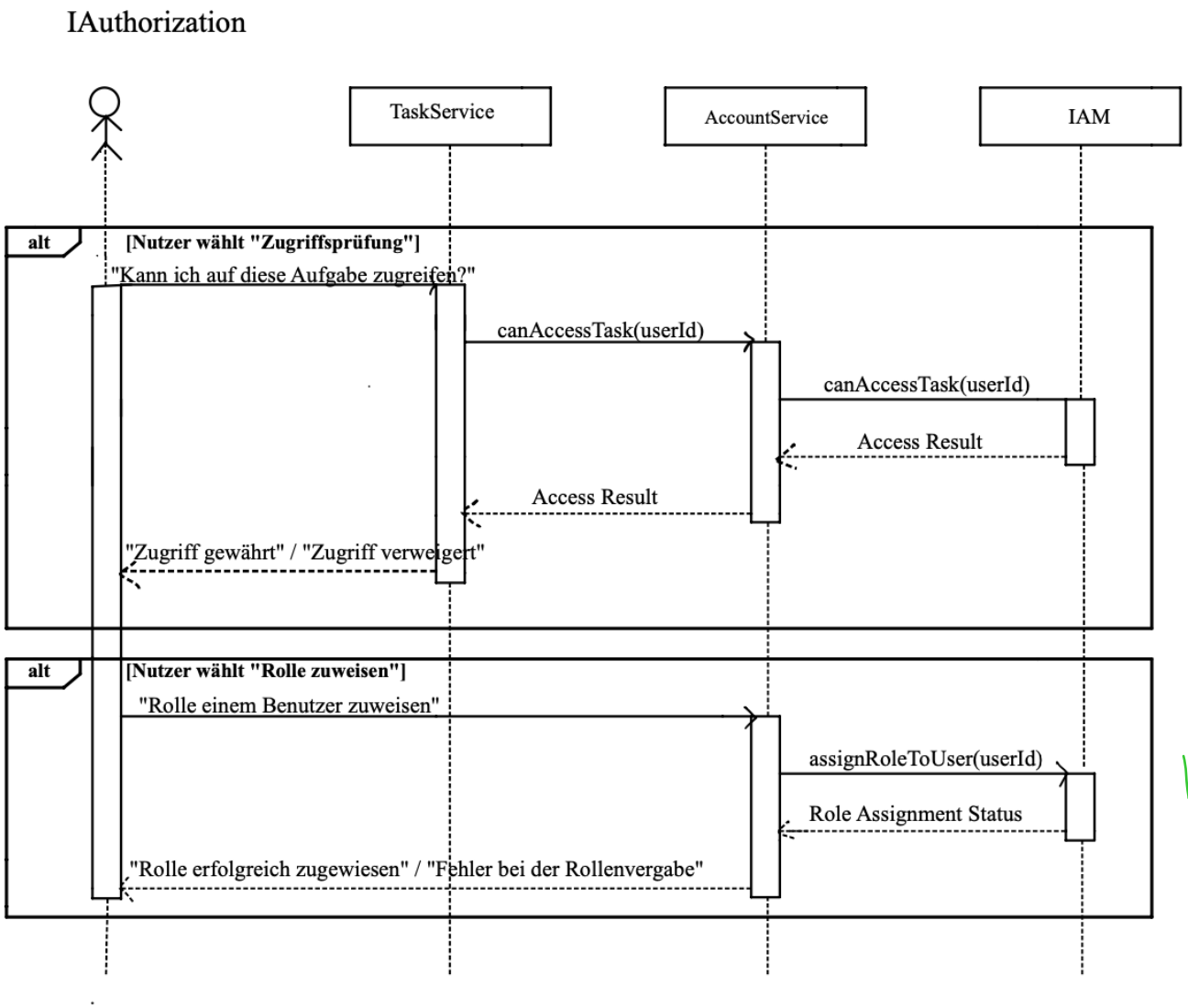
Diese Rückmeldungen ermöglichen es dem **TaskBoard**, dem Nutzer eine abschließende Rückmeldung anzuzeigen.

Erklärung der Einbindung der NotificationService-Komponente
Die Methode `sucheUndFilter` nutzt den **NotificationService**, um den Nutzer während der Suche auf wichtige Ergebnisse (z. B. dringendste Aufgaben) aufmerksam zu machen. Da diese Methode komplexe oder kritische Such- und Filteroperationen beinhaltet, werden Benachrichtigungen generiert.
Andere Methoden wie `getTasksByUser` oder `getTasksByStatus` lösen keine Benachrichtigungen aus, da sie nur einfache Datenabfragen darstellen.
Der **NotificationService** gibt eine Rückmeldung namens `Notification Status` zurück, die dem **TaskBoard** hilft, dem Nutzer wichtige Informationen bereitzustellen.

Ich glaube ich verstehe das jetzt. Geht es dann hier eher um einen parallelen Ablauf von einer Suche,
die möglicherweise länger dauert und gleichzeitigen Benachrichtigungen, die aber unabhängig von der Suche sind?

✓

3. IAuthorization



Beschreibung des Sequenzdiagramms für das IAuthorization-Interface
Das von der **IAM**-Komponente bereitgestellte **IAuthorization**-Interface enthält die folgenden Methoden:

- `canAccessTask(userId)`: Überprüft, ob ein bestimmter Nutzer Zugriff auf eine bestimmte Aufgabe hat.
- `assignRoleToUser(userId)`: Weist einem Nutzer eine spezifische Rolle zu.

Dieses Interface wird von der **TaskService**- und der **AccountService**-Komponente verwendet, um Berechtigungsprüfungen und Rollenzuweisungen durchzuführen.

Ablauf im Sequenzdiagramm
Das Sequenzdiagramm zeigt die Interaktionen zwischen den Komponenten **Nutzer**, **TaskService**, **AccountService** und **IAM**. Jede Aktion wird durch eine direkte Nutzerinteraktion initiiert und unabhängig ausgeführt.

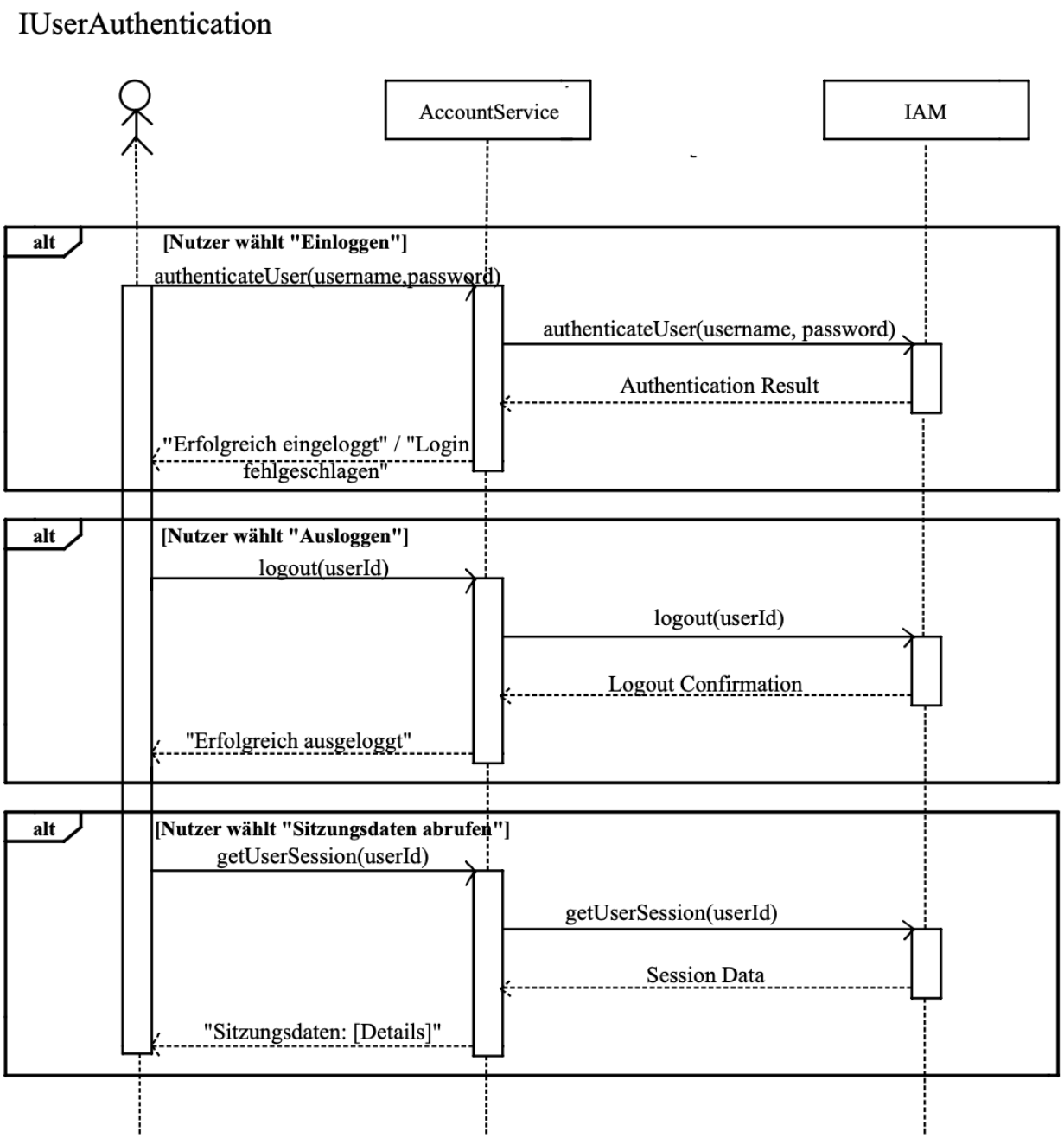
1. Zugriff auf eine Aufgabe prüfen (`canAccessTask`)
Ablauf:

1. Der Nutzer klickt auf den Button *„Zugriff prüfen“* und gibt die entsprechende Aufgabe an.
2. Das **TaskBoard** leitet die Anfrage `canAccessTask(userId)` an die **TaskService**-Komponente weiter.
3. Die **TaskService**-Komponente sendet die Anfrage an die **AccountService**-Komponente.
4. Die **AccountService**-Komponente kommuniziert mit der **IAM**-Komponente, um die Berechtigungen zu überprüfen.
5. Die **IAM**-Komponente gibt das Ergebnis (`AccessResult`) an die **AccountService**-Komponente zurück.
6. Die **AccountService**-Komponente übermittelt das Ergebnis an die **TaskService**-Komponente, die dem Nutzer die Rückmeldung gibt: *„Zugriff gewährt“* oder *„Zugriff verweigert“*.

2. Rolle zuweisen (`assignRoleToUser`)
Ablauf:

1. Der Nutzer klickt auf den Button *„Rolle zuweisen“* und gibt die erforderlichen Details (z. B. Benutzer-ID und Rolle) ein.
2. Das **TaskBoard** leitet die Anfrage `assignRoleToUser(userId)` an die **AccountService**-Komponente weiter.
3. Die **AccountService**-Komponente sendet die Anfrage an die **IAM**-Komponente, um die Rollenzuweisung durchzuführen.
4. Die **IAM**-Komponente verarbeitet die Anfrage und liefert den `Role Assignment Status` (z. B. erfolgreich oder fehlgeschlagen) an die **AccountService**-Komponente zurück.
5. Die **AccountService**-Komponente gibt das Ergebnis an den Nutzer zurück: *„Rolle erfolgreich zugewiesen“* oder *„Fehler bei der Rollenzuweisung“*.

4. IUserAuthentication



Beschreibung des Sequenzdiagramms für das IUserAuthentication-Interface
Das von der **IAM**-Komponente bereitgestellte **IUserAuthentication**-Interface enthält die folgenden Methoden:

- `authenticateUser(username, password)`: Führt die Authentifizierung eines Nutzers durch.
- `logout(userId)`: Meldet einen Nutzer aus dem System ab.

- `getUserSession(userId)`: Ruft die Sitzungsinformationen eines Nutzers ab.

Dieses Interface wird von der **AccountService**-Komponente verwendet, um die Authentifizierungs- und Sitzungsoperationen durchzuführen.

Ablauf im Sequenzdiagramm
Das Sequenzdiagramm zeigt die Interaktionen zwischen den Komponenten **Nutzer**, **AccountService** und **IAM**. Jede Aktion wird durch einen separaten Schritt ausgelöst, abhängig davon, welche Funktion der Nutzer ausführt.

1. Nutzer-Authentifizierung (`authenticateUser`)
Ablauf:

1. Der Nutzer klickt auf den Button *„Einloggen“* und gibt seinen Benutzernamen sowie das Passwort ein.
2. Das **TaskBoard** leitet die Anfrage `authenticateUser(username, password)` an die **AccountService**-Komponente weiter.
3. Das **AccountService** sendet die Anfrage an die **IAM**-Komponente, um die Authentifizierung durchzuführen.
4. Die **IAM**-Komponente überprüft die Anmeldedaten und gibt das `AuthenticationResult` an das **AccountService** zurück.
5. Das **AccountService** liefert die Rückmeldung an den Nutzer: *„Erfolgreich eingeloggt“* oder *„Login fehlgeschlagen“*.

2. Nutzer-Abmeldung (`logout`)
Ablauf:

1. Der Nutzer klickt auf den Button *„Abmelden“* im **TaskBoard**.
2. Das **TaskBoard** leitet die Anfrage `logout(userId)` an das **AccountService** weiter.
3. Das **AccountService** sendet die Abmeldeanfrage an die **IAM**-Komponente.
4. Die **IAM**-Komponente beendet die Sitzung des Nutzers und gibt die `Logout Confirmation` an das **AccountService** zurück.
5. Das **AccountService** bestätigt dem Nutzer, dass die Abmeldung erfolgreich war: *„Erfolgreich abgemeldet“*.

3. Sitzungsinformationen abrufen (`getUserSession`)
Ablauf:

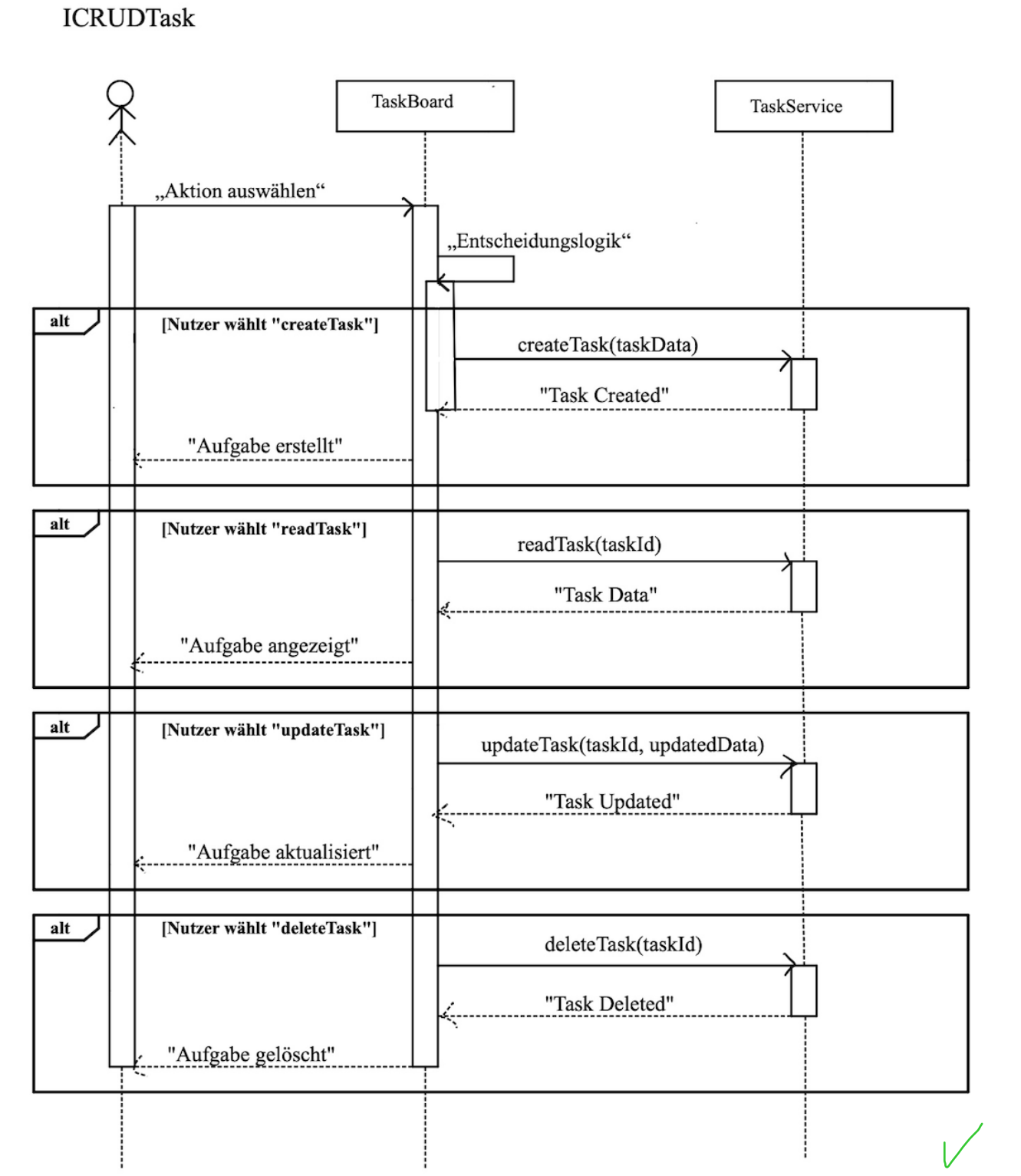
1. Der Nutzer klickt auf den Button *„Sitzungsinformationen anzeigen“*.
2. Das **TaskBoard** leitet die Anfrage `getUserSession(userId)` an das **AccountService** weiter.

3. Das **AccountService** sendet die Anfrage an die **IAM**-Komponente, um die Sitzungsdaten abzurufen.

4. Die **IAM**-Komponente gibt die `Session Data` an das **AccountService** zurück.

5. Das **AccountService** zeigt die Sitzungsdetails dem Nutzer an, z. B.: *„Sitzungs-ID: XYZ, Status: Aktiv“*.

5. ICRUDTask



Beschreibung des Sequenzdiagramms für das ICRUDTask-Interface
Das **ICRUDTask**-Interface wird von der **TaskService**-Komponente bereitgestellt und enthält die folgenden Methoden:

- `createTask(taskData)`: Zum Erstellen einer neuen Aufgabe.

- `readTask(taskId)`: Zum Anzeigen der Details einer bestehenden Aufgabe.
- `updateTask(taskId, updatedData)`: Zum Aktualisieren einer bestehenden Aufgabe.
- `deleteTask(taskId)`: Zum Löschen einer bestehenden Aufgabe.

Dieses Interface wird von der **TaskBoard**-Komponente verwendet, um die verschiedenen CRUD-Operationen durchzuführen.

Ablauf im Sequenzdiagramm
Ein Nutzer initiiert eine CRUD-Operation über das **TaskBoard**, indem er auf den entsprechenden Button klickt:

- Für das Erstellen einer Aufgabe klickt der Nutzer auf *„Neue Aufgabe erstellen“*.
- Zum Anzeigen einer Aufgabe klickt der Nutzer auf *„Aufgabe anzeigen“*.
- Zum Bearbeiten einer Aufgabe klickt der Nutzer auf *„Aufgabe bearbeiten“*.
- Zum Löschen einer Aufgabe klickt der Nutzer auf *„Aufgabe löschen“*.

Das **TaskBoard** empfängt die Eingabe des Nutzers und leitet die entsprechende Anfrage an die **TaskService**-Komponente weiter. Je nach gewählter Methode sendet die **TaskService**-Komponente folgende Rückmeldungen zurück:

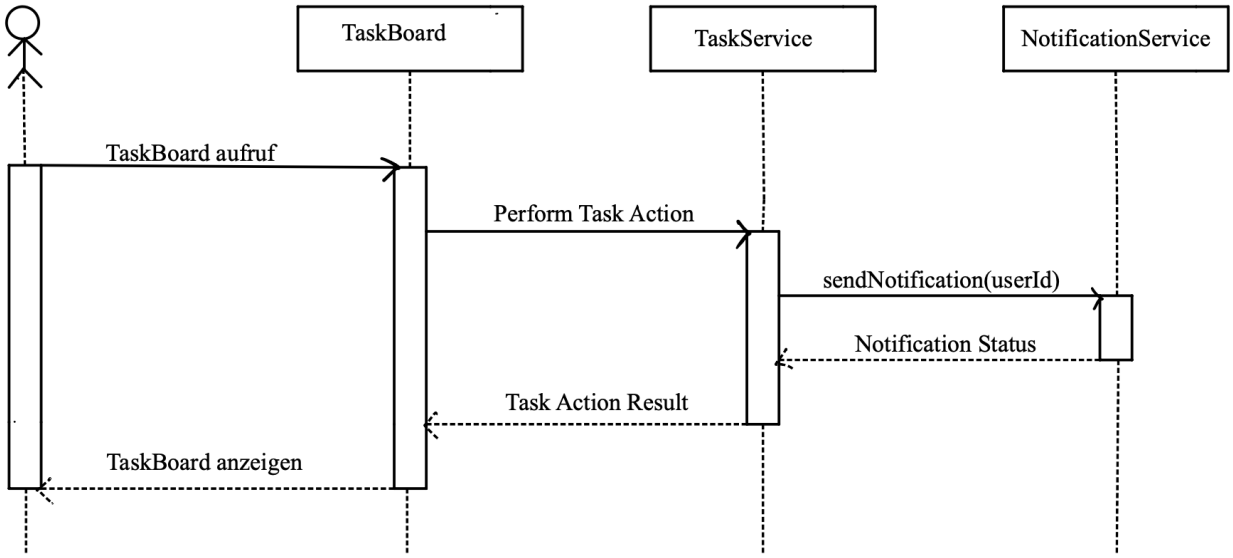
- *„Task Created“* (für die Methode `createTask`),
- *„Task Data“* (für die Methode `readTask`),
- *„Task Updated“* (für die Methode `updateTask`),
- *„Task Deleted“* (für die Methode `deleteTask`).

Details der Rückmeldungen
Diese Rückmeldungen ermöglichen es dem **TaskBoard**, dem Nutzer entweder eine abschließende Bestätigung oder die angeforderten Informationen anzuzeigen:

- Nach der Erstellung einer neuen Aufgabe zeigt das **TaskBoard** die Nachricht *„Aufgabe erfolgreich erstellt“* an.
- Beim Anzeigen der Details einer Aufgabe liefert das **TaskBoard** die spezifischen `Task Data`.
- Nach der Bearbeitung einer Aufgabe wird die Nachricht *„Aufgabe erfolgreich aktualisiert“* angezeigt.
- Beim Löschen einer Aufgabe informiert das **TaskBoard** den Nutzer mit *„Aufgabe erfolgreich gelöscht“*.

6. INotification

INotification



Beschreibung des Sequenzdiagramms für das `INotification`-Interface

Das von der **NotificationService**-Komponente bereitgestellte `INotification`-Interface enthält die Methode:

- `sendNotification(userId, message)`: Diese Methode dient dazu, eine Benachrichtigung mit einer bestimmten Nachricht an den definierten Nutzer zu senden.

Dieses Interface wird sowohl von der **TaskBoard**-Komponente als auch von der **TaskService**-Komponente verwendet.

Ablauf im Sequenzdiagramm

Wenn ein Nutzer über das **TaskBoard** eine Operation durchführen möchte, die eine Benachrichtigung erfordert, wird der folgende Ablauf dargestellt:

1. Der Nutzer sendet eine Botschaft namens `Taskboardaufruf` an das **TaskBoard**.
    - Diese Botschaft signalisiert, dass eine Aktion auf der Ebene des **TaskBoard** durchgeführt werden soll.
2. Das **TaskBoard** leitet die Botschaft in Form von `perform Task Action` an die **TaskService**-Komponente weiter.
    - Die **TaskService**-Komponente ist für die Verarbeitung der spezifischen Aufgabenlogik verantwortlich.

3. Innerhalb der **TaskService**-Komponente wird die Methode `sendNotification(userId, message)` des `INotification`-Interfaces aufgerufen.
    - Diese Botschaft wird an die **NotificationService**-Komponente gesendet, um eine Benachrichtigung zu generieren.
    - **sendNotification(userId, message)** : Diese Methode dient dazu, eine Benachrichtigung mit einer bestimmten Nachricht an den definierten Nutzer zu senden.
4. Die **NotificationService**-Komponente verarbeitet die Anfrage und gibt eine Rückmeldung namens `Notification Status` an die **TaskService**-Komponente zurück.
    - Diese Rückmeldung bestätigt den Status der Benachrichtigung (z. B. erfolgreich gesendet, Fehler, etc.).
5. Basierend auf der Rückmeldung von der **NotificationService**-Komponente gibt die **TaskService**-Komponente eine weitere Rückmeldung namens `Task Action Result` an das **TaskBoard** zurück.
6. Das **TaskBoard** verwendet diese Rückmeldung, um dem Nutzer eine finale Rückmeldung in Form von „Taskboard anzeigen“ zu geben.

Erklärung der Einbindung der NotificationService-Komponente

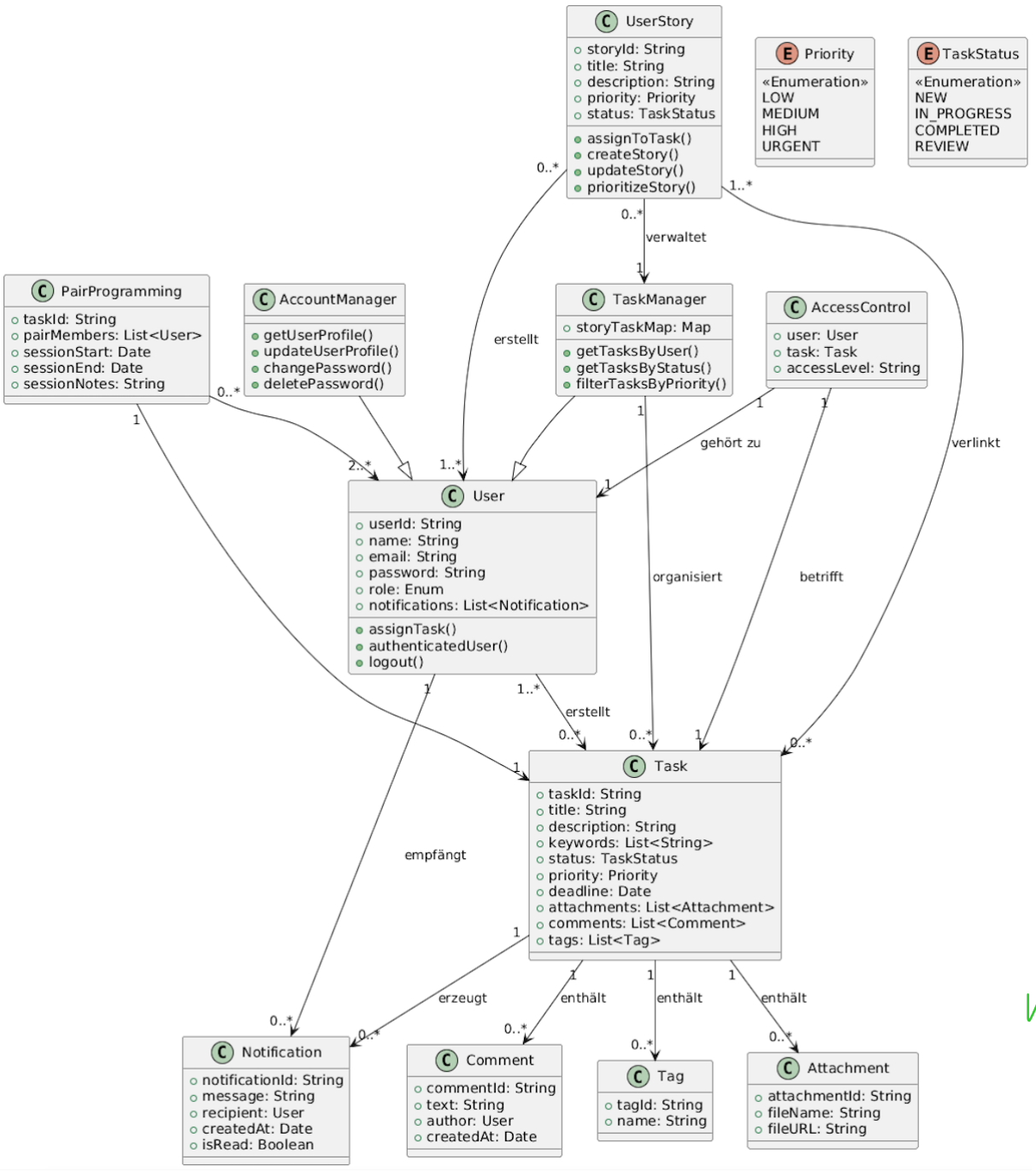
Die **NotificationService**-Komponente wird eingebunden, wenn eine Benachrichtigung erforderlich ist. Dies ist besonders bei Operationen relevant, die die Nutzer direkt betreffen (z. B. Statusänderungen, neue Aufgaben oder wichtige Benachrichtigungen). Sie wird verwendet, um sicherzustellen, dass relevante Nachrichten die Nutzer rechtzeitig erreichen. Das **TaskBoard** empfängt alle Rückmeldungen (z. B. `Notification Status`, `Task Action Result`) von den zuständigen Komponenten. Auf Basis dieser Rückmeldungen ist das **TaskBoard** in der Lage, dem Nutzer eine abschließende Rückmeldung in Form von „Taskboard anzeigen“ zu geben.

1

Hier habt ihr euch sehr viel Arbeit gemacht, was ich leider trotzdem nur mit den vorgesehenen Punkten würdigen kann. Ihr hättet hier nicht unbedingt alle Interaktionen zeigen müssen, z.B. ist das Erstellen, Löschen, Ändern und Ansehen von Tasks sehr ähnlich, sodass dort auch eins gereicht hätte. Es ist trotzdem sehr schön geworden und ich hoffe, dass das Modellieren euch dann wenigstens etwas auf die Klausur vorbereitet hat.

2/2

2.3 Datenstrukturen



Beschreibung der gespeicherten Datenstrukturen

- **User**:
  - **Gespeicherte Attribute:** `userId`, `name`, `email`, `password`, `role`, `notifications`.
  - **Zweck:** Speichert Benutzerinformationen und zugehörige Benachrichtigungen.
- **Task**:

  - **Gespeicherte Attribute:** `taskId`, `title`, `description`, `keywords`, `status`, `priority`, `deadline`, `attachments`, `comments`, `tags`.
  - **Zweck:** Repräsentiert Aufgaben mit Priorität, Status und zusätzlichen Metadaten.

- **Notification**:
  - **Gespeicherte Attribute:** `notificationId`, `message`, `recipient`, `createdAt`, `isRead`.
  - **Zweck:** Speichert Benachrichtigungen, die an Benutzer gesendet werden.

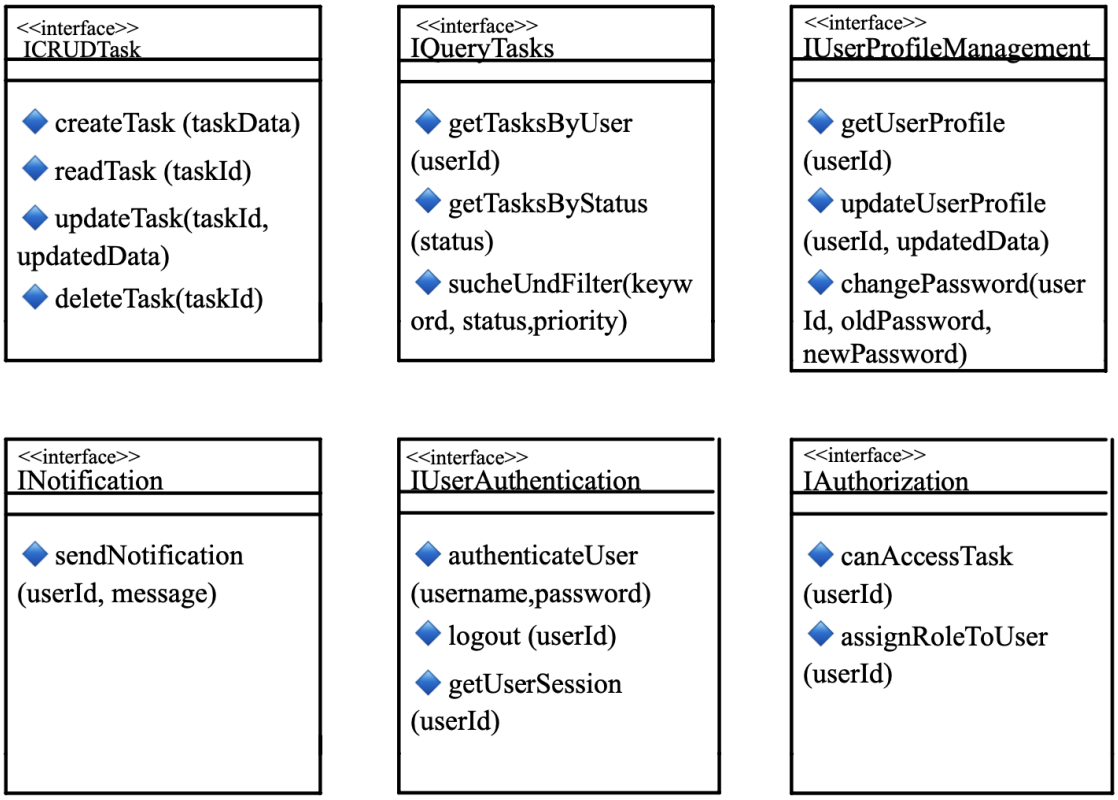
- **PairProgramming**:
  - **Gespeicherte Attribute:** `taskId`, `pairMembers`, `sessionStart`, `sessionEnd`, `sessionNotes`.
  - **Zweck:** Speichert Informationen über Pair-Programming-Sitzungen.

- **AccessControl**:
  - **Gespeicherte Attribute:** `user`, `task`, `accessLevel`.
  - **Zweck:** Speichert Zugriffsrechte eines Benutzers auf bestimmte Aufgaben.

- **Weitere Klassen:**
  - **Comment:** `commentId`, `text`, `author`, `createdAt`.
  - **Tag:** `tagId`, `name`.
  - **Attachment:** `attachmentId`, `fileName`, `fileURL`.

- **Enumerationen:**
  - **Priority:** `LOW`, `MEDIUM`, `HIGH`, `URGENT`.
  - **TaskStatus:** `NEW`, `IN_PROGRESS`, `COMPLETED`, `REVIEW`.

Zuordnung von Datenstrukturen zu den Interfaces



- **IUserProfileManagement:**
  - **Genutzte Datenstrukturen:**
    - **User:** Verwendet für Benutzerprofile (`userId`, `name`, `email`, `password`, `role`).
  - **Begründung:** Dieses Interface verwaltet Benutzerprofile, und alle relevanten Benutzerattribute sind in der `User`-Datenstruktur enthalten.
- **IQueryTasks:**
  - **Genutzte Datenstrukturen:**
    - **Task:** Genutzte Attribute: `taskId`, `status`, `priority`, `keywords`.
    - **User:** Genutzte Attribute: `userId` (für die Zuordnung von Aufgaben zu Benutzern).
  - **Begründung:** Dieses Interface ermöglicht das Abfragen und Filtern von Aufgaben anhand von Benutzerinformationen und Metadaten.
- **IAuthorization:**
  - **Genutzte Datenstrukturen:**
    - **AccessControl:** Genutzte Attribute: `user`, `task`, `accessLevel`.

    - **User:** Genutzte Attribute: `userId`, `role`.
    - **Task:** Genutzte Attribute: `taskId`.
  - **Begründung:** Dieses Interface verwaltet Benutzerzugriffsrechte. Die `AccessControl`-Datenstruktur verbindet Benutzer, Aufgaben und Berechtigungen.

- **IUserAuthentication:**
  - **Genutzte Datenstrukturen:**
    - **User:** Genutzte Attribute: `userId`, `username`, `password`.
    - **PairProgramming:** Genutzte Attribute: `sessionStart`, `sessionEnd` (optional für Sitzungsinformationen).
  - **Begründung:** Dieses Interface authentifiziert Benutzer und kann zusätzlich Sitzungsinformationen verwalten.

- **ICRUDTask:**
  - **Genutzte Datenstrukturen:**
    - **Task:** Genutzte Attribute: `taskId`, `title`, `description`, `priority`, `status`.
  - **Begründung:** Dieses Interface verwaltet Aufgaben. Alle relevanten Attribute sind in der `Task`-Datenstruktur enthalten.

- **INotification:**
  - **Genutzte Datenstrukturen:**
    - **Notification:** Genutzte Attribute: `notificationId`, `message`, `recipient`, `createdAt`, `isRead`.
    - **User:** Genutzte Attribute: `userId` (als Empfänger der Benachrichtigung).
    - **Task (optional):** Genutzte Attribute: `taskId` (falls Benachrichtigungen mit Aufgaben verknüpft sind).
  - **Begründung:** Dieses Interface verwaltet Benachrichtigungen und deren Zuordnung zu Benutzern und Aufgaben.

Sehr gut!

3 Technologien

Soweit absehbar, sind dies die Technologien, die im Laufe der Projektentwicklung implementiert oder ausprobiert werden sollen:

Programmiersprache: Java mit Spring Boot

Java wurde als Programmiersprache gewählt, da die objektorientierte Struktur die Softwareentwicklung erleichtert und die Sprache den Teammitgliedern bekannt ist. Um den Projektstart und das Management zu optimieren, wird das Open-Source-Framework **Spring Boot** eingesetzt, das eine Vielzahl an Bibliotheken mitbringt und damit die Entwicklung deutlich vereinfacht.

Datenbank: MySQL

Für die Datenbank wurde **MySQL** ausgewählt. MySQL ist ein weit verbreitetes und zuverlässiges relationales Datenbankmanagementsystem, das durch seine hohe Performance und Skalierbarkeit überzeugt. Es ermöglicht eine einfache Integration mit Java-Anwendungen und unterstützt die effiziente Speicherung und Abfrage von Daten.

Agile Methode: Scrum

Als agile Methode wurde **Scrum** gewählt, da es durch seine flexible und iterative Struktur teamorientiertes Arbeiten mit regelmäßigen Entwicklungszyklen ermöglicht, sodass wir den Entwicklungsprozess als Team problemlos einsehen und bearbeiten können.

Kommunikationstool: Discord

Die Kommunikation zwischen den Entwicklern erfolgt über **Discord**, da es eine unkomplizierte App ist, mit der alle Teammitglieder bereits vertraut sind. Discord bietet nützliche Funktionen wie Bildschirmfreigabe und den relativ einfachen Austausch von Dateien.

Versionskontrolle: Git mit GitLab

Git zusammen mit **GitLab** wird als Versionskontrolle für eine effektive Verwaltung von Daten und Code verwendet. Der Grund dafür ist, dass Teammitglieder sich gut mit dem Paket auskennen und es bereits für verschiedene Aufgaben genutzt haben.

4 Dokumentation individuelle Beiträge

Prozessschritt	**Verantwortliche Mitglieder**	**Teilnehmende im Meeting**	**Beitragende Inhalte**
Teamkoordination und Planung	Dogukan, Helin, Hüseyin, Eren, Cagla	Dogukan, Helin, Hüseyin, Eren, Cagla	Dogukan, Helin, Hüseyin, Eren, Cagla
Finale Überprüfung	Dogukan, Helin, Hüseyin, Eren, Cagla		

Sehr gute Überarbeitung insgesamt!

10,5/10