Kurze Dokumentation – Tasktracker

1. Einleitung

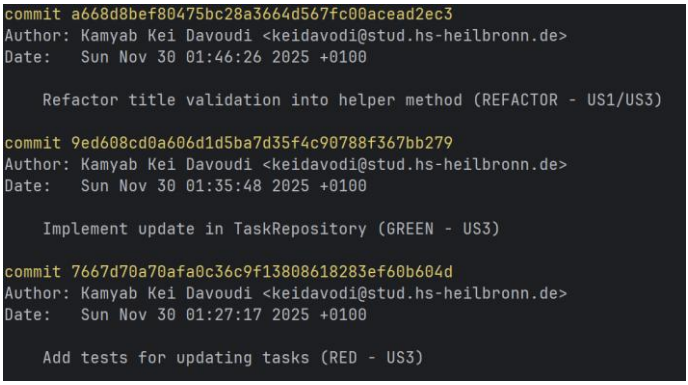
Im Rahmen dieser Übung wurde ein kleiner „Tasktracker“ entwickelt, der das Anlegen, Bearbeiten, Anzeigen und Verwalten von Aufgaben ermöglicht. Die Entwicklung erfolgte bewusst strukturiert und methodisch: Zum einen mit testgetriebener Entwicklung (TDD), zum anderen mit einer klar getrennten Architektur, die unterschiedliche Speicherarten unterstützt. Ziel war eine verständliche, wartbare und erweiterbare Software, die alle Anforderungen der User Stories zuverlässig erfüllt und technisch sauber umgesetzt ist.

2. Anforderungen und User Stories

Die Funktionalität des Systems basiert auf fünf User Stories, die gleichzeitig als Grundlage für die Testfälle dienten.
US1 beschreibt das Erstellen neuer Aufgaben mit Titel, Beschreibung und dem Standardstatus „offen“.
US2 verlangt die Ausgabe aller im System gespeicherten Aufgaben.
Mit US3 müssen bestehende Aufgaben nachträglich aktualisiert werden können, einschließlich Titel, Beschreibung und Status.
US4 definiert das Löschen nicht mehr benötigter Aufgaben.
US5 fordert eine Filterfunktion, die ausschließlich Aufgaben im Status „offen“ zurückgibt.
Diese Stories decken alle zentralen CRUD-Operationen ab und bilden den roten Faden der Implementierung.

3. Vorgehen nach TDD

Die Entwicklung folgte konsequent dem klassischen TDD-Zyklus. Zu jeder User Story wurde zunächst ein fehlschlagender Test geschrieben (RED), anschließend die minimale funktionierende Implementierung ergänzt (GREEN) und schließlich der Code strukturell überarbeitet (REFACTOR). Durch dieses Vorgehen entstand eine klare, kurze und nachvollziehbare Codebasis. Die Tests sorgten dafür, dass Fehler früh sichtbar wurden und die Implementierung jederzeit stabil blieb. Das InMemory-Repository diente dabei als ideales Testobjekt, da es schnell, unabhängig und ohne externe Abhängigkeiten arbeitet.



4. Architektur und Repository-Konzept

Das Projekt basiert auf einer klaren Trennung zwischen Fachlogik und Persistenz. Kern der Lösung ist das TaskRepository, das die verfügbaren Operationen definiert: Aufgaben anlegen, auslesen, aktualisieren, löschen und nach Status filtern. Darauf aufbauend wurden zwei austauschbare Repository-Implementierungen erstellt.

Das **InMemoryTaskRepository** arbeitet rein mit einer Java-Liste und eignet sich hervorragend für Unit-Tests und schnelle Entwicklungszyklen. Es war der zentrale Bestandteil während der TDD-Phase. Das **SqlTaskRepository** greift hingegen über JDBC auf eine echte Microsoft-SQL-Server-Datenbank zu. Beide Klassen implementieren dieselbe Schnittstelle, sodass sie jederzeit austauschbar bleiben. Diese Struktur erleichtert Erweiterungen und entspricht grundlegenden Prinzipien guter Softwarearchitektur (z. B. Trennung von Verantwortlichkeiten und Offen-/Geschlossen-Prinzip).

Kurze Dokumentation – Tasktracker

5. Datenbank und technische Umsetzung

Für die Persistenz wurde mit SQL Server Express eine leichte, lokale Datenbank eingesetzt. Die Datenbank „tasktracker“ enthält eine einzige Tabelle (tasks), die aus einer automatisch generierten ID sowie Titel, Beschreibung und Status besteht. Die Verbindung erfolgt über einen JDBC-Connection-String mit aktivierter Zertifikatvertrauensoption, sodass der lokale Server problemlos erreichbar ist.

Für manuelle Tests wurden einige Datensätze über SQL Server Management Studio eingefügt. Das SqlTaskRepository setzt jede der Repository-Methoden direkt als SQL-Operation um und ist damit ein vollwertiger Persistenzlayer.

```
CREATE TABLE dbo.Tasks (
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Title NVARCHAR(200) NOT NULL,
    Description NVARCHAR(1000) NULL,
    Status NVARCHAR(20) NOT NULL
);
```

6. Build-Prozess, Tests und Coverage

Das gesamte Projekt wird über Gradle verwaltet. JUnit übernimmt die Testausführung, während JaCoCo automatisch die Testabdeckung ermittelt und als HTML-Report bereitstellt. Die Abdeckung ist besonders im InMemory-Repository sehr hoch, da sämtliche Kernfunktionen dort vollständig durch TDD-Tests abgedeckt wurden.

Der Build umfasst das Kompilieren, das Ausführen der Tests und die Erzeugung des Coverage-Berichts. Somit bleibt der Qualitätszustand des Projekts jederzeit nachvollziehbar und reproduzierbar.*Getestet wurde ausschließlich mit dem InMemory-Repository, das wie eine Mock-DB funktioniert und damit die gesamte Logik sehr gut abdeckt. Die SQL-Variante ist für die reale Persistenz gedacht und wird bewusst nicht im Unit-Test-Kontext ausgeführt.*

tasktracker

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
de.tasktracker.repo		30 %		28 %	28	40	93	126	9	17	1	2
de.tasktracker.db		0 %		n/a	2	2	2	2	2	2	1	1
de.tasktracker.model		100 %		n/a	0	9	0	17	0	9	0	1
Total	328 of 508	35 %	33 of 46	28 %	30	51	95	145	11	28	2	4

7. GitHub Actions – Automatisierter Build

Als optionale Erweiterung wurde ein automatischer Build-Prozess mit GitHub Actions eingerichtet. Der Workflow startet bei jedem Push und führt den kompletten Build inklusive Tests aus. Dadurch wird sichergestellt, dass keine fehlerhaften Commits in das Repository gelangen.Er enthält Schritte zum Auschecken des Codes, Einrichten einer Java-Laufzeitumgebung und Ausführen von ./gradlew clean build. Damit entsteht ein transparenter und stabiler Entwicklungsprozess.

8. Fazit

Der entwickelte Tasktracker zeigt, wie sich ein kleines, aber vollständiges Softwaresystem durch TDD, klare Architektur und saubere Persistenzschicht strukturiert aufbauen lässt. Alle User Stories – vom Anlegen über das Aktualisieren bis hin zur Filterung – wurden vollständig umgesetzt. Die Tests stellen sicher, dass das Verhalten jederzeit korrekt bleibt, und die modulare Struktur ermöglicht zukünftige Erweiterungen wie eine GUI oder eine Web-API ohne großen Aufwand.

Der vollständige Quellcode sowie alle Tests und das CI/CD-Setup befinden sich im öffentlichen Repository:
https://github.com/kamyabkd/gse-uebung2-tasktracker