Berliner Hochschule für Technik

Finale Dokumentation

Team 3

René Pettelkau, Matrikelnummer 112385

Leo Ogger, Matrikelnummer 100706

Marty Jabbusch, Matrikelnummer 104616

Leonel Konla Kamta, Matrikelnummer 112391

Patrick Rahn, Matrikelnummer: 100927

Repository: `https://github.com/leonelkonla/Softwaretechnik-Wise25-26`

Techonologie-Stack	
Programmiersprache	Java
IDE	Intellij
Framework	Maven(mit Jacoco)

Projektziele:

Das Ziel des Projekts ist es, eine ganzheitliche Fitness-App zu entwickeln, in der nicht nur die sportlichen Aktivitäten eingetragen werden, sondern diese auch automatisch erfasst, aufbereitet und analysiert werden können. Die App soll Nutzer dabei unterstützen, gesunde Routinen zu entwickeln. Gleichzeitig soll sie helfen, den Überblick über verschiedene Gesundheitsdaten zu behalten und die eigene Entwicklung aufzuzeichnen.

Inhaltsverzeichnis

1 Anforderungsanalyse und Konzeption

1.1 Requirements Engineering

1.1.1 Stakeholder-Identifikation

- Endnutzer
- Produktmanagement
- Entwickler
- Marketing & Vertrieb
- Datenschutz /Legal
- Gesundheitsexperten
- Investoren/Geschäftsführung

1.1.2 Funktionale Anforderungen

- LF10: Nutzerregistrierung und Anmeldung
- LF20: Datenerfassung
- LF30: Datenspeicherung und Datenschutz
- LF40: Datenvisualisierung
- LF50: Datenexport

1.1.3 Nicht-funktionale Anforderungen

- Benutzerfreundlichkeit
- Sicherheit: Sichere Speicherung der Nutzerdaten (DSGVO-konform)
- Kurze Ladezeiten und stabile Performance

1.1.4 Anforderungskatalog

Anforderungskatalog Link:
`https://github.com/leonelkonla/Softwaretechnik-Wise25-26/blob/main/docs/lastenheft.md`

1.2 Use-Case-Modellierung

1.2.1 Use-Case Diagramm

Das Use-Case Diagramm* ist im Anhang zu finden.

1.2.2 Detaillierte Use-Cases

Use-Case	*Beschreibung*
Nutzer registrieren	Neuer Nutzer möchte sich im Programm registrieren
Schrittzähler	Der Nutzer möchte seine täglichen Schritte protokollieren
Kalorienzähler	Der Nutzer kann seine täglichen Kalorien speichern

2 UML-Modellierung und Design

2.1 Statische Modellierung

2.1.1 Klassendiagramm

Das Klassendiagramm* ist im Anhang zu finden.

2.1.2 Design-Entscheidungen

Ein Nutzer soll die Möglichkeit haben beliebig viele Übungen/Trainings zu erstellen und dies soll auch für die Pläne gelten. In unterschiedlichen Lebenslagen will man seine sportliche Aktivitäten anpassen können. Der Nutzer kann sich beliebig viele Pläne erstellen und experimentieren was am besten passt. Die Statistik analysiert beliebig viele Übungen hinsichtlich ihrer Häufigkeiten und Zeiträume.

Die Übung(Exercise) Klasse ist eine abstrakte Oberklasse und von ihr erben drei Subklassen (WeightExercise, CardioCalisthenicsExercise, CardioRunningExercise). Die Oberklasse und Subklassen führen eine 'ist-ein' Beziehung. Der Vorteil ist, dass man gemeinsame Attribute nur einmal definieren muss und spezifische dann in den jeweiligen Subklassen.

Zwischen Übung und Plan besteht eine Komposition. Ein Plan besteht aus mindestens einer Übung. Die Übungen gehören jedoch immer zu einem Plan und sind damit abhängig vom Plan.

Ein Interface ist ein Vertrag. Er besagt, dass der Nutzer sich verpflichtet die Methoden des Exportable Interfaces zu implementieren.

2.2 Dynamische Modellierung

2.2.1 Sequenzdiagramme

Die zwei Sequenzdiagramme* sind im Anhang zu finden.

2.2.2 Interaktionsbeschreibung

1. Nutzer gibt Anmeldedaten in Formular ein und drückt bestätigen.
2. LoginView erzeugt LoginController
3. LoginController ruft UserDatabase auf um Daten zu überprüfen

4. Daten, welche in einer CSV Datei sind, werden überprüft
5. Sind die Daten valide wird changeView Methode aufgerufen und es geht zum Hauptmenü. Sind die Daten jedoch falsch wird eine Exception geworfen und ein Error Label wird sichtbar.

3 Implementierung und Versionsverwaltung

3.1 Repository-Management

3.1.1 Workflow

Alle Beteiligten haben wenig bis gar keine Erfahrung mit diesem Tool. Zielsetzung der Branching-Strategie ist main als Hauptzweig des Projekts(stabile Version) zu nutzen und separate Branches für neue Features anzulegen. Commit Nachrichten sollten so kurz wie möglich aber so ausführlich wie nötig sein.

3.1.2 Dokumentationsstruktur

- README.md: Zweck des Repositories -BHT Projekt, Angabe des Teams, Projektidee, grobe Projektstruktur
- WORKFLOW.md: Rollen im Team, Gemeinsame Verantwortlichkeiten, Tools, Branching Strategie, Arbeitsweise, Reflexion

3.2 Projektstruktur

3.2.1 Architektur

Wir halten uns an die Vorgaben von Maven, da es unser Building Werkzeug ist.
Beispielpfade: src/main/java/com/fitnessapp/model/WeightExercise.java und
src/test/java/com/fitnessapp/model/WeightExerciseTest.jav

Als Softwarearchitektur haben wir uns für MVC (Model View Controller) entschieden.

3.2.2 Implementierte Komponenten

- View: LoginView, MainMenu, ProfileView
- Model: User, Exercise + Subklassen, Plan, Statistic
- Controller: LoginController, MainMenuController, CaloricIntakeController
- Navigation: Navigator
- Tests: UserTest, Exercisesubklassentests, PlanTest, StatisticTest

3.2.3 Code-Qualität

- Java Conventions, Checkstyle Analysis
- JDepend Framework
- Strategy Pattern, Observer Pattern
- SRP Single-Responsibility-Design-Prinzip, DRY Don't Repeat Yourself

Das Ergebnis von JDepend * ist im Anhang zu finden.

3.3 Testing-Strategie

- Unit-Tests: Konstruktoren der Klassen werden getestet und wenn möglich die Methoden.
- Test Coverage: Jacoco Plugin für Maven

Die Ergebnisse der Tests * und Jacoco * ist im Anhang zu finden.

4 KI-Werkzeuge im Entwicklungsprozess

4.1 Einsatzbereiche

- Requirements Engineering: Stakeholder-Simulation, Anforderungen
- Erklärung von Tools und Zusammenhängen z.B Ordnerstruktur für ein Projekt und Details über Maven

4.2 Konkrete Beispiele

- Aufgabenstellung: Stakeholder-Simulation, Tool: ChatGPT, Ergebnis: mehrere plausible Antworten (Endnutzer, Produktmanagement, Entwickler, Marketing etc.), Kritische Reflexion: unvollständige Angaben durch mangelnden Kontext
- Aufgabenstellung: Use-Cases ,Tool: ChatGPT, Ergebnis: Vorschläge für Use-Cases - Systembenachrichtigungen & Erinnerungen senden und Daten exportieren / löschen , Kritische Reflexion: unvollständige Angaben durch mangelnden Kontext

4.3 Kritische Bewertung

Die Grenzen liegen in mangelnden Angaben zum Kontext und dem daraus entstehenden Risiko, dass KI unvollständige Anforderungen erzeugt. Zudem besteht das Risiko, vertrauliche Projektdaten an Dritte weiterzugeben, wenn gültige Datenschutzgesetze durch die KI verletzt werden. Im wöchentlichen Projektalltag können wir KI nutzen, um Dokumente vorzubereiten, Meetingnotizen zusammenzufassen oder uns Aspekte von Tools und Konzepten erläutern lassen, um ein besseres Verständnis zu schaffen. In Zukunft werden wir mehr KI Modelle nutzen und die unterschiedlichen Ergebnisse untersuchen, um uns ein besseres Bild von der Qualität des generierten Inhalts zu verschaffen.

5 Herausforderungen und Lösungen

5.1 Technische Herausforderungen

- Versionsverwaltung Git: allgemeine Nutzung, Commits, Branching Strategie
- Umlet zur Gestaltung der UML Diagramme
- Maven - Building Tool, Dependency Manager, Project Manager: Installation, Nutzung, Plugins, POM Datei

Lösungsansätze

Wir mussten viele Sachen von Grund auf lernen und versuchen in der kurzen Zeit auch anwenden zu können. Es ist ein schrittweiser Prozess der bis heute anhält. Der erste Schritt war die Dokumentation der Werkzeuge zu lesen, um im nächsten Schritt über systematisches Ausprobieren die Funktionen zu verstehen und dann anzuwenden.

5.2 Team-Organisation

- Extrem heterogenes Team in Bezug auf Alter, Familienstand, Arbeitsverhältnis
- Zeit war ein sehr limitierender Faktor aufgrund der Zusammenstellung des Teams
- Kein einziges Teammitglied hat nennenswerte Programmiererfahrung. Nicht alle Mitglieder haben Programmierung 2 absolviert (gilt nicht als Voraussetzung für Softwaretechnik)

Lösungsansätze

Wir haben versucht wenn es ging die Aufgaben nach Stärken und Interessen aufzuteilen. Dies war jedoch oft die Ausnahme, da wir alle keine Erfahrung in diesem Gebiet haben. Wir hatten ein wöchentliches Meeting am Dienstag um 18:15 direkt nach der Vorlesung und gemeinsame Reviews am Ende der Woche. Aufgrund der Aufstellung des Teams war es jedoch unmöglich mit allen 5 Mitgliedern regelmäßige Meetings zu haben, was den Zeit- und Arbeitsaufwand enorm erhöht hat.

5.3 Projektergebnis und Reflexion

5.4 Erreichte Meilensteine

- Implementierung bestimmter Klassen in Code
- Testing: Unit-Tests, Assertions, Code Coverage mit Jacoco
- Continuous Integration: GitHub Actions
- Verbesserung der Code Qualität über Checkstyle und JDepend
- zwei Use Cases implementiert
- Nicht erreicht: weitere Use-cases

5.5 Funktionsumfang

5.5.1 Implementierte Features

- Das Einloggen eines Nutzers
- Kalorienzähler

5.5.2 Geplant, aber nicht umgesetzt

- Feature: Registrierung eines Nutzers, Begründung: Das Konzept für Sicherheit und Datenbank sind noch nicht ausgereift.
- Feature: Anlegen von Übungen, Begründung: Die genauen Übungen müssen noch festgelegt werden.

5.6 Lessons Learned

Die Softwaretechnik dient der Herstellung von stabiler, sicherer und zuverlässiger Software also ausführbaren Code und umfasst alle Werkzeuge, Prinzipien und Methoden, um dieses Ziel zu erreichen. Weiterhin gehören neben dem ausführbaren Programm auch Aspekte wie Dokumentation, Quellcode und Demo-Programme dazu. Im Allgemeinen unterscheidet man zwischen aktiven Daten wie dem Programm und passiven Daten, die für den Betrieb des Programms benötigt werden.

Es muss viel geplant und erörtert werden, um einen klaren Plan zu haben wie man das Softwareprodukt aufbauen will und dann auch umsetzen kann. Requirements-Engineering und die Analyse mit Use-Cases schafft einen guten Überlick an dem man sich orientieren kann. Um in einem Team effektiv zu arbeiten sind eine Versionsverwaltung wie Git und eine Plattform wie Windows Teams unerlässlich. Das Zeitmanagement ist eine sehr schwierige Aufgabe, die immer wieder Probleme verursacht. Generative KI kann sehr hilfreich sein, um sich ein Bild von einem Thema zu schaffen, erste Denkanstöße zu vermitteln und komplexe Aspekte der Softwaretechnik zu erläutern. Jedoch muss man immer aufpassen und die Ergebnise der KI kritisch bewerten.

Wir haben verschiedene Aspekte von Softwarearchitektur beleuchtet und uns konkrete Prinzipien (DRY, SOLID) und Patterns (Strategy, Observer) angeeignet, um die Qualität zu gewährleisten. Darüber hinaus haben wir uns damit beschäftigt, wie man Code testet und dann über Code Coverage analysiert, wie viel Code bei den Tests durchlaufen wird. Die Automatisierung von Testing und Building ist Teil von Continuous Integration und wurde über GitHub Actions umgesetzt.

6 Anhang

Use-Case Diagramm

Abbildung 1: Use-Case Diagramm

Klassendiagramm

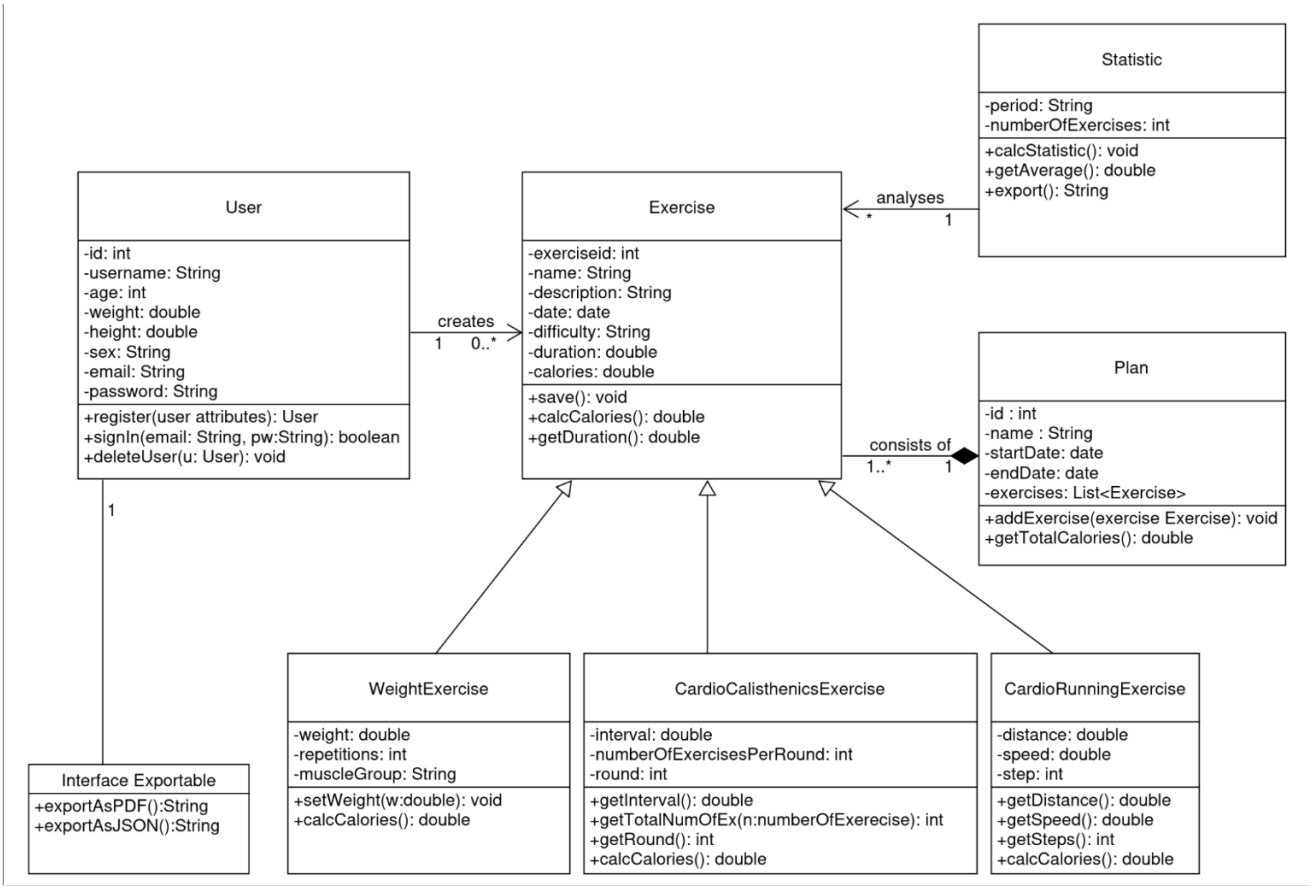

Abbildung 2: Klassen Diagramm

Sequenzdiagramme

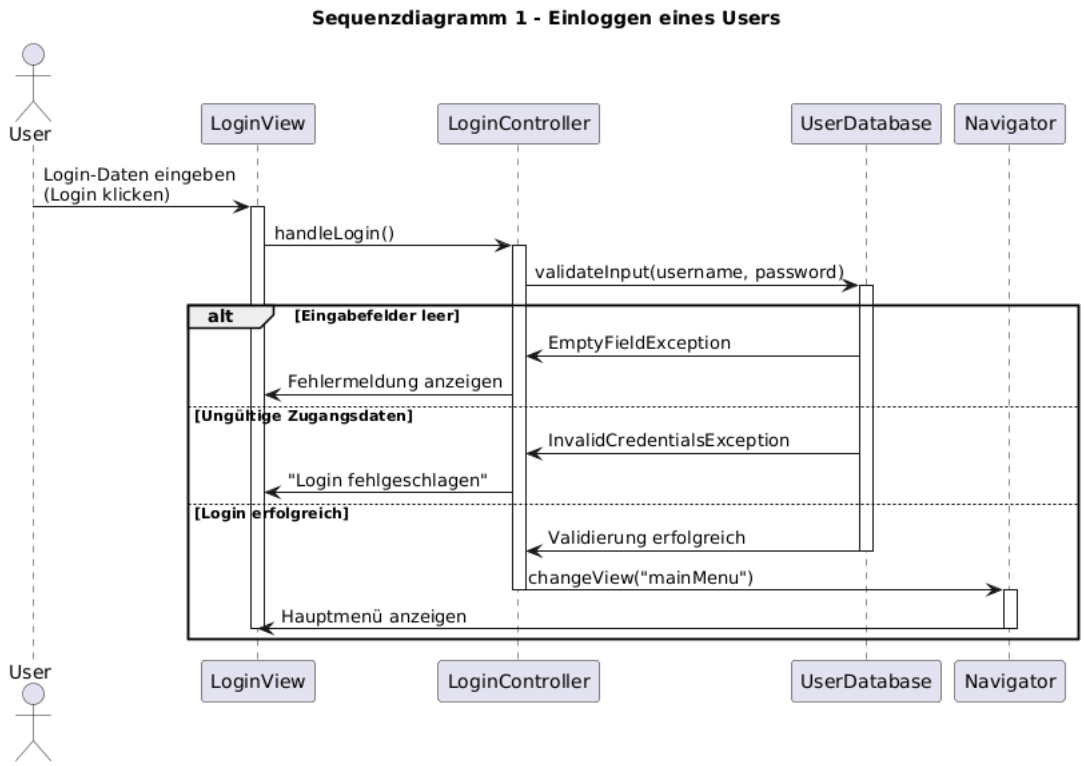

Abbildung 3: Sequenzdiagramm 1

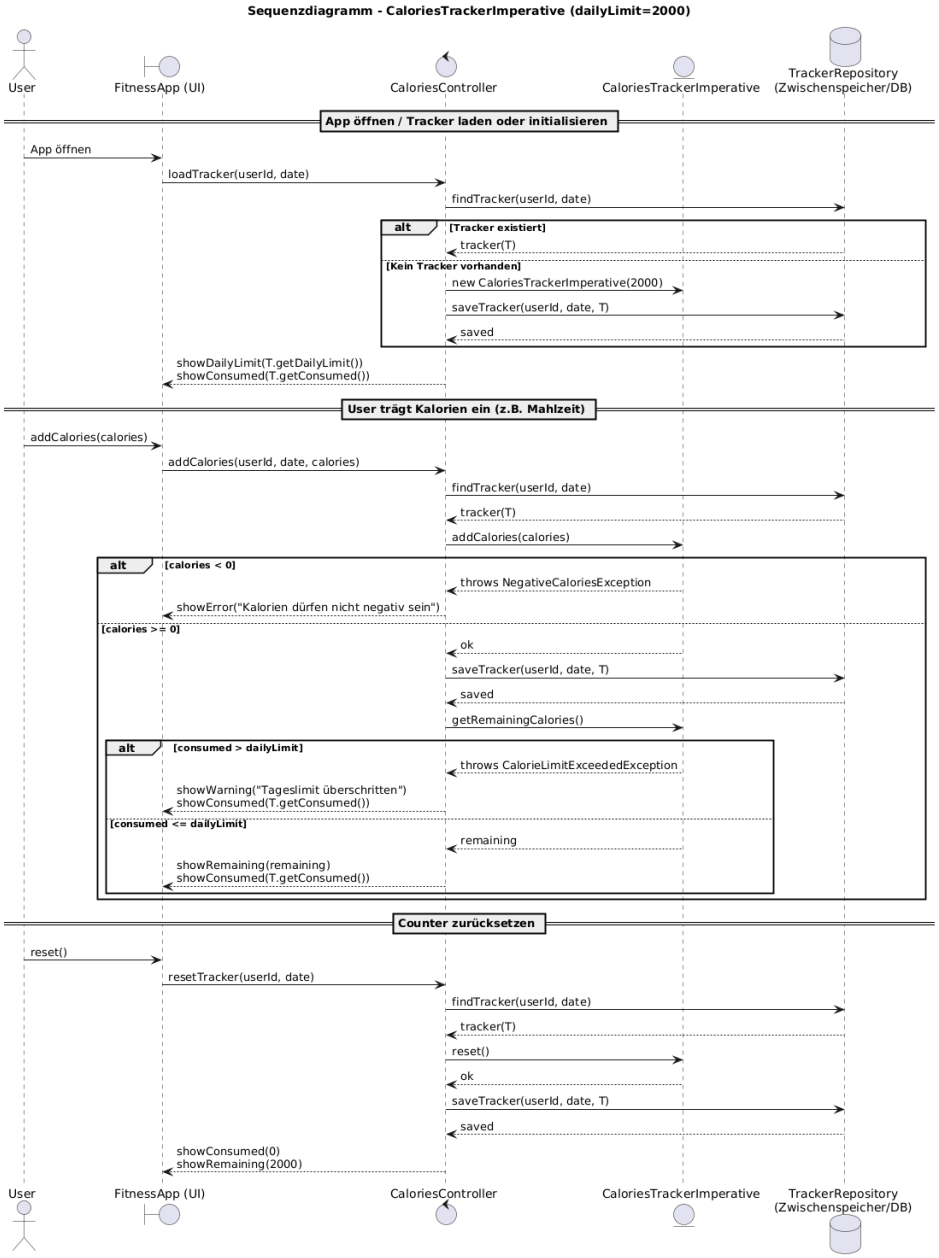

Abbildung 4: Sequenzdiagramm 2

```
---- JDepend Report ----
Package: com.fitapp, Classes: 1, Ca: 0, Ce: 3, I: 1.00, A: 0.00, D: 0.00
Package: com.fitapp.controller, Classes: 4, Ca: 1, Ce: 7, I: 0.88, A: 0.25, D: 0.13
Package: com.fitapp.view, Classes: 1, Ca: 1, Ce: 7, I: 0.88, A: 0.00, D: 0.13
Package: com.fitapp.navigation, Classes: 0, Ca: 2, Ce: 0, I: 0.00, A: 0.00, D: 1.00
Package: com.fitapp.model, Classes: 15, Ca: 1, Ce: 6, I: 0.86, A: 0.20, D: 0.06
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  13.995 s
[INFO] Finished at: 2025-12-29T15:36:44+01:00
[INFO] ------------------------------------------------------------------------
patrick@patrick-ubuntu-vm:~/IdeaProjects/softwaretechnik-projekt$
```

Abbildung 5: *JDepend Report

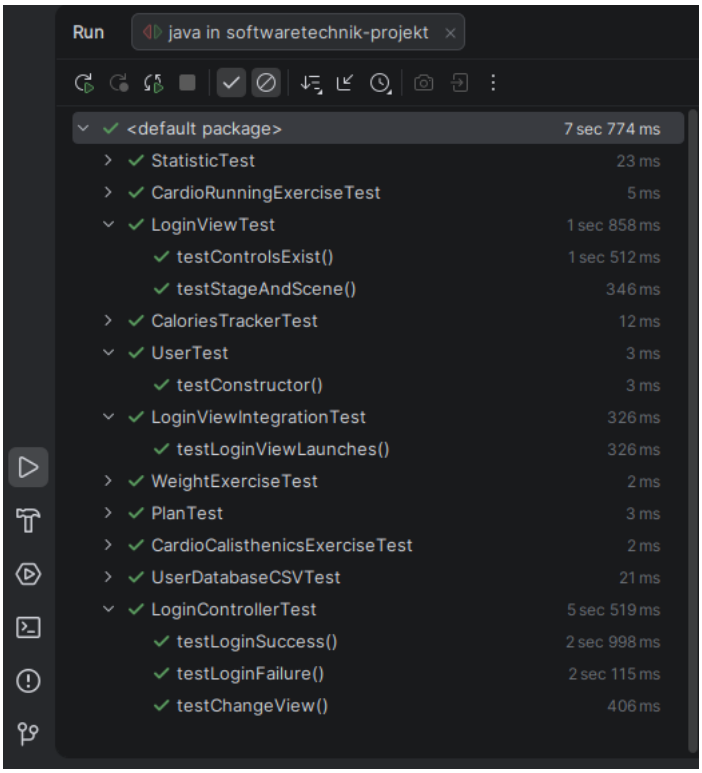

Abbildung 6: *Test Ergebnisse

softwaretechnik-projekt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.fitapp.model		74%		69%	34	87	42	151	28	74	1	14
com.fitapp.controller		27%		n/a	8	14	39	57	8	14	1	3
com.fitapp.navigation		76%		33%	4	6	5	19	0	2	0	1
com.fitapp		0%		n/a	2	2	3	3	2	2	1	1
com.fitapp.view		100%		n/a	0	3	0	13	0	3	0	1
Total	271 of 780	65%	12 of 32	62%	48	112	89	243	38	95	3	20

Abbildung 7: *Jacoco Code Coverage