BHT Berliner Hochschule für Technik

Studiere Zukunft

Team 1:

1. Sönmez Süner, 896545
2. Jost Nadwornik, 104710
3. Volodymyr Rusobrov 100222
4. Oksana Orel, 108949
5. Magdalena Kretschmer, 104474

Fitnesstracker – Finale Dokumentation

Projektziel: Transparente Kalorienbilanz für den Nutzer. Umgesetzte Funktionen: Aktivität erfassen, Ernährung erfassen, Grunddaten eingeben, Kalorienauswertung anzeigen

Technologie-Stack: Java mit JavaFx, IntelliJ und Visual Code

Repository: Fitness-Tracker-BHT

Inhalt

Anforderungsanalyse und Konzeption

Requirements Engineering

Stakeholder-Identifikation

Zu den Stakeholdern gehören:
Endnutzer, Produktmanagement /Auftraggeber, Entwicklerteam (Frontend und Backend), Fachanwender (Ernährungsberater & Fitnesscoach), Datenschutzbeauftragte/ Compliance-Stelle, Marketing / Vertrieb, Gesundheitsorganisationen/ Krankenkassen

Funktionale Anforderungen

LF-01: Trainingsdaten erfassen: Nutzer können ihre Schritte, Kalorien Zeit für Training eintragen
LF-02: Nutzerregistrierung und -login: Nutzer können sich mit einem Benutzernamen und Passwort registrieren und einloggen
LF-03: Trainingsdaten in Grafiken darstellen: Die Daten können vom Nutzer als Grafiken aufgerufen werden
LF-04: Nutzerinformationen erfassen: Der Nutzer kann wichtige persönliche Informationen in die App eintragen
LF-05: Berechnung von täglichen Bedarfen: Aus vorhanden Daten wird dem Nutzer die berechneten Kalorien- und Wasserbedarfe angezeigt
LF-06: Mahlzeiten/Kalorien erfassen: Der Nutzer kann eintragen, was er gegessen hat bzw. wie viel Kalorien er zu sich genommen hat
LF-07: Bewegungsdaten erfassen: Der Nutzer kann Bewegungsdaten erfassen manuell oder durch Verbindung mit Schrittzähler oder anderen Aktivitätsmessern

Nicht-funktionale Anforderungen

Usability: Die Bedienung soll intuitiv und barrierearm sei, wichtige Eingaben sollten nicht mehr als 3 Klicks benötigen
Security/Legal: Nutzerdaten müssen DSGVO-konform gespeichert werden
Security: Passwörter müssen verschlüsselt gespeichert werden

Anforderungskatalog

Siehe Anforderungskatalog in unserem Git (Link für die Übersichtlichkeit): Anforderungskatalog

Use-Case-Modellierung

Use-Case-Diagramm

Siehe einzelne Diagramme in Git (Link für die Übersichtlichkeit): Use-Case-Diagramme

Detaillierte Use-Cases

UC-01	Aktivität erfassen	Nutzer erfasst eine Aktivität (z. B. Joggen, Radfahren oder Training) manuell über die Aktivitätseingabemaske.
UC-02	Ernährung erfassen	Nutzer dokumentiert, was er gegessen oder getrunken hat, inklusive geschätzter Kalorien.
UC-03	Grunddaten eingeben	Nutzer gibt persönliche Basisdaten ein, die zur Berechnung des Kalorienverbrauchs dienen.
UC-04	Kalorienauswertung anzeigen	Nutzer sieht eine grafische Auswertung des täglichen Energiehaushalts (Kalorienaufnahme vs. Kalorienverbrauch).

UML-Modellierung und Design

Statische Modellierung

Klassendiagramm

Link für bessere Auflösung: Klassendiagramm

Design-Entscheidungen

Die Klassenstruktur wurde so gewählt, dass jede Entität (User, ActivityRecord, FoodRecord, Report) eine klar abgegrenzte Verantwortung hat. Der User verwaltet persönliche Daten und aggregierte Berechnungen, während ActivityRecord und FoodRecord einzelne Aktivitäten bzw. Mahlzeiten erfassen.
Die Klasse Report dient zur periodischen Auswertung dieser Daten, um Kalorienverbrauch und -aufnahme zu vergleichen.
Alternativ hätte man eine zentralisierte „Tracker"-Klasse erstellen können, die alle Berechnungen übernimmt; jedoch fördert die aktuelle Aufteilung bessere Modularität, Wartbarkeit und spätere Erweiterbarkeit (z. B. durch neue Record-Typen oder Report-Formate).

Dynamische Modellierung

Die nachfolgenden Diagramme werden zur besseren Darstellung verlinkt.

Sequenzdiagramme

Sequenzdiagramm UC-01

Sequenzdiagramm UC-02

Sequenzdiagramm UC-04

Zustandsdiagramme

Zustandsdiagramm

Interaktionsbeschreibung

Nutzer erfasst eine Aktivität (Typ, Dauer, Intensität, Datum). Das System berechnet den Kalorienverbrauch und speichert den ActivityRecord. [UC-01]

Nutzer fügt einen Ernährungseintrag hinzu (Food, Portion, kcal/Einheit). Das System berechnet die Gesamt-kcal und aktualisiert die Tagesbilanz. [UC-02]

Nutzer ruft die gewünschte grafische Darstellung auf (Verbrauchte <-> Eingenommene Kalorien täglich/monatlich). Das System berechnet die Kalorien nach den Einträgen des Nutzers und stellt sie grafisch dar. [UC-04]

Implementierung und Versionsverwaltung

Repository-Management

Workflow

Wir haben eine main-Branch, weiterhin wird für jedes Feature eine feature-Branch angelegt, da wir jetzt zur Programmierung übergehen haben wir vor eine dev-Branch einzuführen. Alle feature-Branches werden erst in der dev-Branch getestet bevor sie in die main-Branch gemerged werden.

Für unsere Commit-Conventions gilt: Die Commit-Message wird in wenigen englischen Worten und im Präsens formuliert. Sie beschreibt knapp und eindeutig, welche Änderungen vorgenommen wurden.

Die Commit-Convention haben wir erst nach einer Recherche eingefügt, deswegen sind die ersten Commits noch in der Vergangenheitsform formuliert.

Für jede feature-Branch bevor sie sie gemerged wird, wird ein pull-Request erstellt und es wird ein Reviewer zugewiesen. Die Reviewer rotieren durch das gesamte Team.

Dokumentationsstruktur

Die gesamte Struktur ist in unserem GitHub hinterlegt: Struktur des Git

Projektstruktur

Architektur

Wir haben uns für die MVC-Architektur entschieden, weil es trennt, klar zwischen Datenlogik, Benutzeroberfläche und Steuerung. Dadurch ist unser Projekt übersichtlicher und besser wartbar.

Implementierte Komponenten

Implementierte Klassen: UserProfile, BmrCalculator, ActivityTypes, FoodTypes, FoodRecord, ActivityRecord, ActivityCalculator

In Bearbeitung: DailySummary

Code-Qualität

Gewählte Design Pattern: Repository Pattern & Observer Pattern (Genaue Beschreibung unter folgendem Link: Design Patterns)

Wir haben uns beim Programmieren an unser Architektur Modell, unsere Design Patterns und an OOP versucht zu halten. Wir haben geplant beim Refactoring zu prüfen, ob wir uns an die genannten Dinge gehalten haben.

Testing-Strategie

Im Rahmen der Testentwicklung wurden Unit- und Integrationstests für die fachliche Logik der Anwendung implementiert. Ziel der Tests ist es, die korrekte Berechnung von Grundumsatz, Kalorienaufnahme und Kalorienverbrauch sowie die fehlerfreie Verarbeitung der zugehörigen Datenmodelle sicherzustellen.

Getestet werden ausschließlich Klassen der Geschäftslogik (Model- und Serviceklassen). Die grafische Benutzeroberfläche ist nicht Bestandteil der Tests, da sie keine fachliche Logik enthält und automatisierte Tests hier keinen Mehrwert bieten würden.

Testvorgehen

Für bereits implementierte Funktionen wurden Unit-Tests nachträglich erstellt, um die bestehende Logik abzusichern. Für neu entwickelte Features wird das Test-Driven-Development-Verfahren angewendet, bei dem Tests vor der eigentlichen Implementierung definiert werden. Dieses Vorgehen unterstützt eine klare Spezifikation der Anforderungen und erleichtert die Wartbarkeit des Codes.

Testergebnisse

Die implementierten Unit- und Integrationstests wurden erfolgreich ausgeführt. Alle Tests sind ohne Fehler durchgelaufen, wodurch die korrekte Funktionsweise der getesteten Berechnungen und Datenverarbeitungen bestätigt wurde. Zu sehen innerhalb der Workflows in unserem Repository: Testergebnisse über Actions

KI-Werkzeuge im Entwicklungsprozess

Einsatzbereiche

Requirements Engineering

Die Stakeholder-Simulation ist in diesem Dokument: Stakeholer-Simulation

Modellierung

Wir haben eine KI verwendet, um die UML-Diagramme zu validieren, aber keine zur Erstellung.

Implementierung

Bei der Implementierung haben wir die KI nur als Unterstützung genutzt. (Zum Beispiel, bei der ordentlichen Implementierung von MVC)

Testing

Wir haben die Testfälle selbst erstellt und anschließend mit der KI geprüft. Wir haben uns auch unterstützen lassen, was die Abdeckung betrifft.

Konkrete Beispiele

Beispiel 1:

Aufgabenstellung Übungsblatt_W2 Aufgabe 1b: Interaktive Vertiefung: Wählen Sie 2 Stakeholder aus und führen Sie ein simuliertes "Interview" mit der KI: Lassen Sie die KI in die Rolle des Stakeholders schlüpfen. Stellen Sie 3-5 konkrete Fragen zu deren Bedürfnissen. Dokumentieren Sie Frage und Antwort. Beispiel-Prompt: "Du bist jetzt ein 45-jähriger Fitness-Enthusiast, der die App nutzen möchte. Ich stelle dir Fragen zu deinen Anforderungen. Antworte aus dieser Perspektive. Frage 1: Was ist dir bei einer Fitness-App am wichtigsten?" c) Reflexion (3-4 Sätze): Wie realistisch waren die KI-generierten Stakeholder-Perspektiven? Was könnten echte Stakeholder-Interviews zusätzlich liefern, was die KI nicht kann?

Verwendete KI: ChatGPT 5.0 (Bezahlversion)

Reflexion: Der 1. Stakeholder ist etwas unrealistisch in unseren Augen. Viel zu kulant. Zeit und Geld wirkte etwas unrealistisch, aus persönlichen Erfahrungen raus. Die 2. Stakeholderin klingt schon deutlich realistischer. Aber auch hier sind die Zahlen vielleicht etwas unrealistisch.

Bei einer KI weiß man nie genau, ob sie die Antworten so formuliert, damit du glücklich und zufrieden mit den Antworten bist. In einem echten Gespräch hast du jemanden vor dir, der entweder etwas zu verlieren hat oder wirklich etwas brauchbares haben möchte.

Beispiel 2:

Aufgabenstellung Übungsblatt_W2 Aufgabe 7: Generative KI im Requirements Engineering (ca. 20 Min.). Jetzt kommt die KI ins Spiel! Nutzen Sie eine Generative KI Ihrer Wahl (ChatGPT, Claude, Gemini, etc.) für diese Aufgabe. a) Wählen Sie Aufgabe 2b (funktionale & nicht-funktionale Anforderungen) oder Aufgabe 3 (User Stories). Formulieren Sie einen präzisen Prompt für die KI, der Ihr Projekt beschreibt, den gewünschten Output spezifiziert und die Rolle der KI definiert (z.B. "Du bist ein Requirements Engineer..."). Beispiel-Prompt: „Du bist ein erfahrener Requirements Engineer. Ich entwickle eine Fitness-Tracking-App, die [Beschreibung]. Erstelle mir 10 User Stories nach dem Format: "Als [Rolle] möchte ich [Funktionalität], damit [Nutzen]." Fokussiere dich auf [spezifischer Bereich].“ Führen Sie den Prompt aus und dokumentieren Sie: Ihren vollständigen Prompt, Das KI-Ergebnis (komplett oder wesentliche Teile) sowie Ihre Bewertung: • Was war gut? Was war unbrauchbar? • Vergleich zu Ihrem manuellen Ergebnis aus Aufgabe 2b oder 3 • Was würden Sie für die weitere Projektarbeit übernehmen? • Was würden Sie auf keinen Fall übernehmen und warum? • Wie würden Sie den Prompt verbessern?

Verwendete KI: CoPilot

Reflexion: Was war gut? Was war unbrauchbar?
Einige KI-Ideen sind logisch und trivial und könnten auch ohne KI-Unterstützung entwickelt werden. Ohne KI würden uns jedoch wichtige Funktionalitäten entgehen.
Wir finden es auch gut, dass Copilot angeboten hat, die Akzeptanzkriterien für die Stories hinzuzufügen. Das könnte auch nützlich sein.
Vergleich zu Ihrem manuellen Ergebnis aus Aufgabe 2b oder 3
Die User-Stories ähneln den Ergebnissen aus Aufgabe 3, einige sind sogar nahezu identisch.
Was würden Sie auf keinen Fall übernehmen und warum?
Uns ist nichts negativ aufgefallen.
Was würden Sie für die weitere Projektarbeit übernehmen?
Wir können die User Stories für „Schlafzeiten Aktivität“ übernehmen.
Wie würden Sie den Prompt verbessern?
Wir würden prompt anfordern, dass die Fitness-Tracker-App neue Ideen für Benutzer enthalten sollte und sich von bereits existierenden Fitness-Tracker-Apps unterscheiden sollte. Vielleicht sollte sie mehr „Gamification“ als andere Apps enthalten.

Kritische Bewertung

Erfolgreich übernommene KI-Vorschläge

Wir haben die KI-Vorschläge zur Verbesserung unseres Klassendiagramms übernommen. Die genauen Vorschläge sind im folgen Dokument zu finden: Klassendiagramm-Review

Verworfene KI-Ergebnisse

Zum Anfang des Projekts haben wir uns von der KI Use-Cases generieren lassen, die wir verworfen haben, da sie zum Beispiel vorgeschlagen hat anstatt eines Ad-basierten Monetarisierungsmodell ein Bezahlmodell zu nutzen, was zu diesem Zeitpunkt noch völlig irrelevant für uns ist.

Lessons Learned

Egal was uns die KI auch ausgibt, es muss immer überprüft und gegengechecked werden.

In erster Linie ist es aber ein super Tool um neue Denkansätze anzuregen und allgemein bei der Arbeit zu unterstützen, solange man die Ergebnisse immer kritisch hinterfragt. Auch ist es sehr hilfreich, wenn man Brainstormen möchte oder „kritisches“ Feedback zu seinen Ergebnissen braucht.

Herausforderungen und Lösungen

Technische Herausforderungen

Umgang mit Git: In unserem Team gab es einige Mitglieder, die zuvor noch nie mit Git gearbeitet haben und wir haben innerhalb des Teams „Mini-Schulungen“ gehabt, die von den Erfahrenen gehalten wurde. Anfänglich mussten wir lernen, wie wir sauber arbeiten (viele haben einfach in main gepushed) mittlerweile arbeiten wir mit unterschiedlichen Branches, Commit-Messages und Reviews.

[Herausforderung 2]: Ausstehend

Team-Organisation

Persönliche Meeting nicht möglich: Wir treffen uns mehrmals wöchentlich (Dienstags und manchmal Samstags/Sonntags) in Discord und besprechen Aufgaben, Lösungen, Herausforderungen und Vorgehensweisen. Die gemeinsame Arbeit auf Git hat in diesem Aspekt auch sehr geholfen. Durch die online Verbindung über Discord, war immer oder sehr zeitnah einer erreichbar, wenn Hilfe benötigt wurde.

Projektergebnis und Reflexion

Erreichte Meilensteine

Meilenstein 1(Zwischenergebnis): PoC des Fitness-Trackers (User Grunddateneingabe und Grundumsatzberechner)

Meilenstein 2 (Ende des Projekts): Aktivitätserfassung, Ernährungserfassung

Nicht erreichte Meilensteine: Auswertung in Form von Diagrammen fehlend, jedoch als Tagesbilanz

Funktionsumfang

Implementierte Features

Grunddaten eingeben und Grundumsatz berechnen. Es können Aktivitäten ausgewählt werden und die Werte werden in der Berechnung des Grundumsatz mit einbezogen, genau wie die Ernährung.

Geplant, aber nicht umgesetzt

Wir werden die Login-Funktion nicht umsetzen, da der Umfang zu groß ist und für unseren PoC auch nicht notwendig ist (Nutzung eines Servers für die Registrierung, Anlegen einer sicheren Datenbank und ähnlich aufwändige Features). Wir haben die grafische Umsetzung in Form von Diagrammen auch aus zeitlichen Gründen nicht umsetzen können, haben jedoch eine Tagesbilanz erstellt.

Lessons Learned

Fachlich: Umsetzung von OOP, Planung mittels UML, Teamorganisation mittels moderner Tools, Kennenlernen verschiedener Software-Architekturen und Design Patterns
Technisch: Umgang mit Git, Umgang mit Jira
Prozessual: Agile Arbeitsweise in Verbindung mit Kanban
KI-Integration: KI ist ein nützliches Tool für Brainstorming neue Ideen sammeln, jedoch unter den kritischen Augen des Nutzers