Nutrition Tracker

Eine Webanwendung zur Analyse von Aminosäurenprofilen

Projektdokumentation

Verfasser: Hellen Ghirmai

22.03.2025

Inhaltsverzeichnis

1 Einführung

Ziel des Projekts ist die Berechnung des aufgenommenen Aminosäurenprofils auf Basis verzehrter Lebensmittel sowie die Ermittlung individueller Bedarfdeckungen. Die Berechnung des täglichen Bedarfs an Aminosäuren erfolgt ernährungswissenschaftlich auf Grundlage von Referenzwerten pro Kilogramm Körpergewicht. Diese Werte wurden im Rahmen der gemeinsamen Expert Consultation von WHO, FAO und UNU festgelegt. Sie dienen als wissenschaftlich fundierte Referenz für den Aminosäurenbedarf Erwachsener1. Die Aminosäure Glycin ist in dieser WHO-Tabelle nicht enthalten, da sie als nicht-essentiell gilt.

Für diese Arbeit wurde dennoch ein Basiswert für Glycin verwendet, weil Studien zeigen, dass der Körper oft nicht genug davon selbst herstellen kann[2].
Die hinzugefügten Lebensmittel weisen ihre Aminosäurenwerte pro 100 g aus, wobei die Daten aus einer veganen Lebensmitteldatenbank[3] für essentielle Aminosäuren sowie einer separaten Quelle[4] für Glycin stammen.
Die Anwendung erlaubt die Eingabe von Namen, Gewicht und Alter – wobei derzeit das Alter nicht zur Berechnung verwendet wird. Eine Erweiterung der Anwendung für Kinder ist jedoch denkbar, da sich deren Bedarf nach altersabhängigen Körperzusammensetzungsmodellen richtet[5].

---

[1] WHO/FAO/UNU. *Protein and Amino Acid Requirements in Human Nutrition. WHO Technical Report Series Nr. 935.* Genf: World Health Organization; 2007. Tabelle 23, S. 150. Verfügbar unter: https://apps.who.int/iris/handle/10665/43411. [Zugriff am: 20. März 2025].

[2] Meléndez-Hevia, E., de Paz-Lugo, P., Cornish-Bowden, A., & Cárdenas, M. L. (2009).
A weak link in metabolism: the metabolic capacity for glycine biosynthesis does not satisfy the need for collagen synthesis.
Journal of Biosciences, 34(6), 853–872.
DOI: 10.1007/s12038-009-0095-7

[3] Vegane Proteinquellen. *Aminosäuregehalt in pflanzlichen Lebensmitteln.* 2025. Verfügbar unter: https://vegane-proteinquellen.de/aminosaeuregehalt-lebensmittel/. [Zugriff am: 20. März 2025].

[4] vegnt.com: *Top 100 vegane Lebensmittel mit dem höchsten Glycin-Gehalt pro 100 g*, 2025. Online verfügbar unter: https://vegnt.com/de/minerals/food-all/Glycine-desc/ [Zugriff am 20. März 2025].

[5] World Health Organization, Food and Agriculture Organization of the United Nations, and United Nations University. *Protein and Amino Acid Requirements in Human Nutrition.* WHO Technical Report Series No. 935. Geneva: WHO, 2007, S. 168. Verfügbar unter: https://iris.who.int/handle/10665/43411. [Zugriff am: 21. März 2025].

Drei spezifische Decorator-Klassen passen den individuellen Tagesbedarf basierend auf Nutzerparametern dynamisch an:

- Athlet-Decorator: Erhöht den Bedarf an verzweigtkettigen Aminosäuren (BCAAs: Leucin, Isoleucin, Valin), da Studien zeigen, dass sportlich aktive Personen insbesondere höhere Mengen an BCAAs zur Muskelregeneration benötigen[6].
- Veganer-Decorator: Verstärkt den Bedarf an limitierenden Aminosäuren (v. a. Lysin und Methionin), da diese in pflanzlicher Kost häufig unterrepräsentiert sind[7].
- Longevity-Decorator: Reduziert gezielt Methionin und erhöht Glycin, da eine methioninarme Ernährung mit erhöhter Langlebigkeit in Verbindung gebracht wird und Glycin dabei eine ausgleichende Rolle im Stoffwechsel spielt[8].

1.1 Domainmodell

Das Domainmodell verknüpft ein Benutzerprofil (User) mit Ernährungseinträgen (NutritionLog), denen über eine Zwischentabelle (NutritionLogFoodItem) konkrete Lebensmittel (FoodItem) mit aminosäurespezifischen Profilen zugeordnet werden. Die Entität AminoAcidRequirement liefert davon unabhängig Basiswerte für die bedarfsorientierte Berechnung und wird ausschließlich über den Service-Layer eingebunden.
Fachliche Constraints wie ein gültiger Name, ein plausibles Alter sowie ein realistisch begrenztes Körpergewicht werden in der Anwendung auf zwei Ebenen validiert.
Im Frontend erfolgt die Validierung über Vuetify-Formularregeln mit definierten Eingabebeschränkungen.
Im Backend sind die DTOs zusätzlich mit Bean Validation-Annotationen versehen.
Bei Verletzung dieser Regeln wird eine entsprechende Exception ausgelöst und ein strukturierter Fehler zurückgegeben.

---

[6] Shimomura, Y., Yamamoto, Y., Bajotto, G., Sato, J., Murakami, T., Shimomura, N., Kobayashi, H., & Mawatari, K.: Nutraceutical effects of branched-chain amino acids on skeletal muscle. *Journal of Nutrition*, 136(2), 529S–532S, 2006. Online verfügbar unter: https://doi.org/10.1093/jn/136.2.529S [Zugriff am 21. März 2025].

[7] Mariotti, F., & Gardner, C. D. (2019). *Dietary Protein and Amino Acids in Vegetarian Diets—A Review*. Nutrients, 11(11), 2661. Verfügbar unter: https://doi.org/10.3390/nu11112661. [Zugriff am 21. März 2025].

[8] de Cabo, R., & Mattson, M. P. (2019). *Effects of Intermittent Fasting on Health, Aging, and Disease.* New England Journal of Medicine, 381(26), 2541–2551. [Online verfügbar unter: https://doi.org/10.1056/NEJMra1905136 – Zugriff am 21. März 2025]

Die Auswahl von Lebensmitteln ist optional. Fehlende Einträge führen zu einem leeren Ergebnis, stellen jedoch keinen Validierungsfehler dar.
Die Aminosäure-Schlüssel (Map<String, Double> aminoAcidProfile) sind implizit durch die kontrollierte Datenerzeugung (DevDataLoader) auf gültige Werte beschränkt.

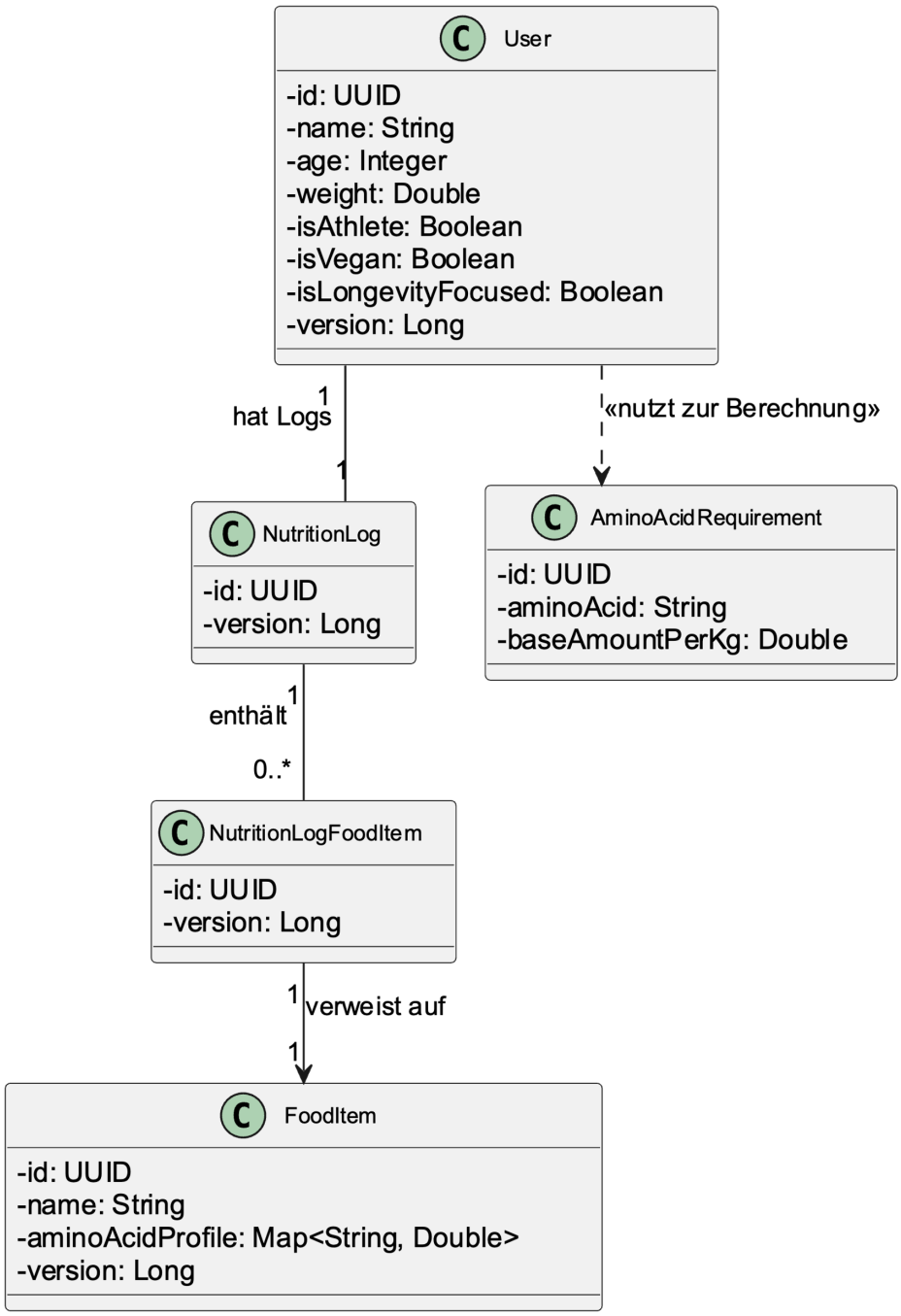


Abbildung1: Domainmodell Nutrition Tracker

1.2 User Stories

1.2.1 User Story 1: Ernährungstracking eines Nutzers

Anforderung:

Als Nutzer möchte ich meine konsumierten Lebensmittel in ein Ernährungstagebuch eintragen, um mein tägliches Aminosäurenprofil analysieren zu können.

Akzeptanzkriterien:

- Der Benutzer sieht eine Liste mit verfügbaren Lebensmitteln und kann daraus eines auswählen.
- Die Lebensmittel werden im NutritionLog gespeichert.
- Aminosäurenprofil der ausgewählten Lebensmittel werden berechnet und angezeigt.

1.2.2 User Story 2: Individuelle Anpassung des Aminosäuren-bedarfs

Anforderung:

Als Benutzer mit spezifischen Ernährungszielen (z. B. Athlet, Veganer) möchte ich meinen Aminosäurenbedarf automatisch anpassen lassen, um eine für mich optimale Versorgung sicherstellen zu können.

Akzeptanzkriterien:

- Ernährungspräferenzen werden berücksichtigt.
- Decorator-Klassen passen die Berechnung an.
- Benutzer erhält eine strukturierte Übersicht mit angepasstem Tagesbedarf.

1.2.3 User Story 3: Vergleich von Ernährung und Tagesbedarf

Anforderung:

Als Nutzer möchte ich mein tatsächliches Aminosäurenprofil mit meinem berechneten Tagesbedarf vergleichen, um zu sehen, ob ich Defizite oder Überschüsse habe.

Akzeptanzkriterien:

- Tagesbedarf wird angezeigt.
- Der Benutzer erhält eine prozentuale Übersicht aller Aminosäuren im Vergleich zum Tagesbedarf.

2 Backend

2.1 Backend-Komponenten

Das UML-Komponentenmodell zeigt die modulare Struktur des NutritionTracker-Backends. Es stellt dar, wie die Berechnungskomponenten über das Interface AminoAcidCalculator verbunden sind. Die Lollipop-Notation kennzeichnet dieses Interface und dessen Implementierungen.

Die Klassen DailyAminoAcidCalculator, AthleteAminoAcidDecorator, VeganAminoAcidDecorator und LongevityAminoAcidDecorator implementieren das Interface und sind als separate Komponenten modelliert.

Die Verbindung zum AminoProfileService erfolgt über das Interface, wodurch eine lose Kopplung erreicht wird. Die Darstellung betont die Austauschbarkeit der Implementierungen und die Trennung zwischen Schnittstelle und konkreter Ausführung.

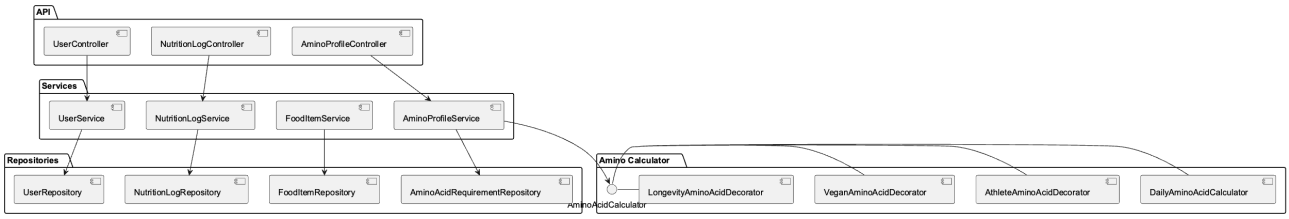


Abbildung 2: UML-Komponentendiagramm des Nutrition Tracker-Backends mit Lollipop-Notation

User, NutritionLog und FoodItem bilden im Modell eine durchgängige Verarbeitungskette. Ein Benutzer erstellt Ernährungseinträge (NutritionLog), die aus verknüpften Lebensmitteln (FoodItem) bestehen.
Die Zwischentabelle NutritionLogFoodItem ermöglicht die flexible Mehrfachzuordnung und dient als zentrale Verbindungseinheit im Modell.

2.2 Backend-Design-Pattern

Der Einsatz des Entwurfsmusters Decorator basiert auf dem Ziel, ein erweiterbares Berechnungsmodell zu schaffen, das unterschiedliche Ernährungstypen und physiologische Anforderungen berücksichtigt, ohne die zugrunde liegende Struktur der Logik zu verändern.

Die Klasse DailyAminoAcidCalculator berechnet den täglichen Bedarf auf Basis des Körpergewichts in Kombination mit einem Referenzwert pro Kilogramm. Sie implementiert das Interface AminoAcidCalculator und dient als Eingabekomponente für die darauf aufbauenden Decorator-Klassen.

Diese erweitern die Grundberechnung zur Laufzeit um zusätzliche Faktoren, z. B. sportliche Aktivität, vegane Ernährung oder Langlebigkeitsstrategien.

Die Auswahl und Kombination der Decorator erfolgt im AminoProfileService anhand der Eigenschaften im UserDTO. Die Dekoration ist rekursiv aufgebaut: Jeder Decorator umschließt die vorangehende Instanz, sodass sich kombinierte Berechnungen für die Dekoratoren abbilden lassen.

Alle Decorator erben von der abstrakten Klasse AminoAcidCalculatorDecorator, die ihrerseits das Interface AminoAcidCalculator implementiert.

Durch diese Struktur wird eine lose Kopplung erreicht und das Prinzip Open/Closed umgesetzt. Im Vergleich zu einer klassischen Verzweigungslogik, wie mit if-/else-Blöcken, verbessert das Pattern die Erweiterbarkeit und Testbarkeit.

2.3 Spezialthema im Backend

Nachfolgend ein kurzer Überblick, wie in diesem Projekt das DTO-Entity-Mapping umgesetzt ist.

2.3.1 Trennung von Entity und DTO

Zur Trennung von Persistenz- und Transportschicht werden im Projekt separate Klassen für die Datenbank-Entities (z. B. User, NutritionLog, FoodItem) und die zugehörigen Data Transfer Objects (DTOs) (UserDTO, NutritionLogDTO, FoodItemDTO etc.) verwendet. Diese Aufteilung dient der Entkopplung interner Datenbankstrukturen von der externen Schnittstellendarstellung und reduziert die Gefahr unbeabsichtigter Datenexponierung.

2.3.2 Abbildung der Felder

Die DTO-Klassen enthalten ausschließlich Felder, die für den jeweiligen Anwendungsfall (z. B. Anzeige, Datenübertragung) erforderlich sind. Interne Beziehungen oder nicht benötigte Attribute der Entity-Klassen werden bewusst ausgelassen.
Beispielsweise enthält UserDTO Felder wie id, name, age und weight, die den gleichnamigen Feldern der Entity User entsprechen.
Das FoodItemDTO wiederum übernimmt lediglich id und name, nicht jedoch komplexe interne Strukturen wie das aminoAcidProfile (Map<String, Double>), da diese Information auf Client-Ebene nicht benötigt wird.

2.3.3 Konstruktoren für die Umwandlung

Die Umwandlung von Entities in DTOs erfolgt teilweise über Konstruktoren, die die relevanten Felder extrahieren. So bietet z. B. NutritionLogDTO einen Konstruktor, der ein vollständiges NutritionLog-Objekt entgegennimmt, daraus jedoch nur ID und eine Liste zugehöriger NutritionLogFoodItemDTOs ableitet. Dadurch wird sichergestellt, dass nur kontextrelevante Informationen übergeben werden, ohne eine vollständige Entity-Struktur an den Client zu übertragen. Gleichzeitig bleibt die fachliche Struktur durch klare Zuordnung zwischen DTO und Entity gewahrt.

2.3.4 Reduzierung von Lazy-Loading-Problemen

Im NutritionTracker-Projekt wird bei der Erstellung von DTOs gezielt vermieden, auf nicht initialisierte, lazily geladene Entitäten außerhalb eines aktiven Persistence-Kontextes zuzugreifen. Dies ist relevant, da die Verwendung von FetchType.LAZY in JPA-Relationen zu einer LazyInitializationException führen kann, sobald auf abhängige Objekte zugegriffen wird, ohne dass eine Transaktion aktiv ist. Ein typisches Beispiel betrifft die Klasse NutritionLog, die eine Liste zugehöriger NutritionLogFoodItem-Einträge enthält. Um diese Information im DTO bereitzustellen, ohne das gesamte Objekt zu serialisieren, wird die Umwandlung innerhalb einer Service-Methode mit @Transactional durchgeführt. Dadurch bleibt der JPA-Kontext während des Mappings aktiv und erlaubt den Zugriff auf die lazily geladene Beziehung.

```
@Service
public class NutritionLogService {
    private final NutritionLogRepository nutritionLogRepository;
    public NutritionLogService(NutritionLogRepository nutritionLogRepository) {
        this.nutritionLogRepository = nutritionLogRepository;
    }
    /**
     * Lädt einen NutritionLog und wandelt ihn innerhalb einer Transaktion in ein DTO um,
     * um Lazy-Loading-Probleme bei den zugehörigen FoodItems zu vermeiden.
     */
    @Transactional
    public NutritionLogDTO getLogAsDTO(UUID logId) {
        NutritionLog log = nutritionLogRepository.findById(logId)
                .orElseThrow(() -> new EntityNotFoundException("Log not found"));
        return new NutritionLogDTO(log); // Konstruktor greift auf log.getNutritionLogFoodItems() zu
    }
}
```

```
public class NutritionLogDTO {
    private UUID id;
    private List<NutritionLogFoodItemDTO> foodItems;
/**
     * Konstruktor zum Mapping von NutritionLog zu NutritionLogDTO.
     * Achtung: Der Zugriff auf log.getNutritionLogFoodItems() erfordert einen aktiven
     * JPA-Persistence-Kontext, da die Beziehung standardmäßig mit FetchType.LAZY geladen wird.
     * Die aufrufende Methode sollte daher mit @Transactional versehen sein.
     */
    public NutritionLogDTO(NutritionLog log) {
        this.id = log.getId();
        this.foodItems = log.getNutritionLogFoodItems().stream()
            .map(NutritionLogFoodItemDTO::new)
            .collect(Collectors.toList());
    }
}
```

2.4 Test des Backends

Die folgenden Unit-Tests prüfen zentrale Funktionseinheiten der Anwendung in inhaltlicher Reihenfolge und dokumentieren deren erwartetes Verhalten.

2.4.1 DailyAminoAcidCalculatorTest

Der Test DailyAminoAcidCalculatorTest implementiert die Überprüfung der Basisbedarfsberechnung durch direkte Instanziierung der DailyAminoAcidCalculator-Klasse. Als Eingabe dient ein 70 kg schwerer, nicht spezifizierter Nutzer (UserDTO). Die Methode calculateDailyNeeds() multipliziert das Gewicht mit standardisierten Bedarfssätzen pro Kilogramm Körpergewicht für zehn essentielle Aminosäuren. Die berechneten Ist-Werte werden über assertEquals gegen die aus der Fachliteratur entlehnten Soll-Werte abgeglichen.

2.4.2 AthleteAminoAcidDecoratorTest

Im AthleteAminoAcidDecoratorTest wird die Klasse AthleteAminoAcidDecorator explizit um eine Mock-Instanz des Basiskalkulators gewrappt. Mittels when(...).thenReturn(...) wird ein konstanter Ausgangsbedarf simuliert. Die Methode calculateAminoAcids() erhöht daraufhin BCAAs (Leucin, Isoleucin, Valin) um exakt 30 %, alle übrigen Aminosäuren um 15 %. Dies wird durch eine präzise Map-Vergleichslogik mit Toleranzüberprüfung (0.01) verifiziert.

2.4.3 VeganAminoAcidDecoratorTest

Der Test VeganAminoAcidDecoratorTest folgt derselben Struktur, fokussiert sich jedoch auf gezielte Verstärkung der limitierenden Aminosäuren pflanzlicher Kost. Der Decorator erhöht ausschließlich Lysin und Methionin um 20 % und lässt alle anderen Werte unverändert. Diese Selektivität wird durch den gezielten Vergleich von Einzelschlüsseln der modifizierten Map nachgewiesen, wodurch die korrektive Logik präzise isoliert nachgewiesen wird.

2.4.4 LongevityAminoAcidDecoratorTest

Analog dazu evaluiert LongevityAminoAcidDecoratorTest eine biogerontologische Anpassung: Methionin wird zur potenziellen Lebensverlängerung um 20 % reduziert, Glycin um 25 % erhöht. Auch hier erfolgt die Validierung über manuell konstruierte Mockdaten und

punktuelle Map-Assertions, wodurch die Effektivität der lebensstilbasierten Modulation belegt wird.

2.4.5 AminoProfileServiceTest

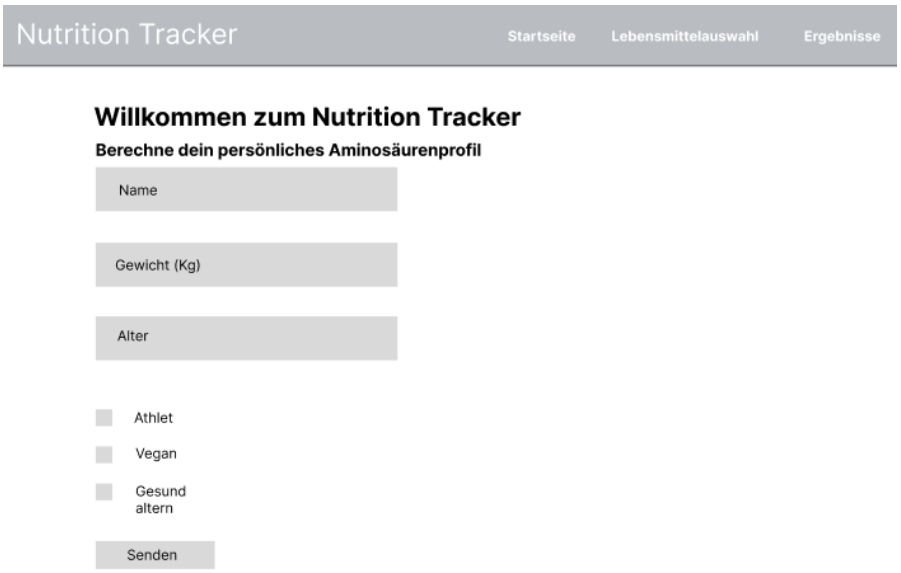
Der abschließende AminoProfileServiceTest bezieht sich auf die Methode calculateAminoAcidSumsForLatestLog() innerhalb des AminoProfileService. Zwei exemplarische FoodItem-Instanzen (z. B. “Tofu”, “Beans”) mit definierten Aminosäureprofilen werden über NutritionLogFoodItem dem Test-Log zugeordnet. Der nutritionLogRepository wird gemockt, um ein Log mit genau diesen Einträgen zurückzugeben. Die Methode summiert die Aminosäuren über alle verknüpften Einträge; die resultierenden Summen werden mit exakt berechneten Sollwerten verglichen (z. B. Leucin = 3.8 g).

3 Frontend

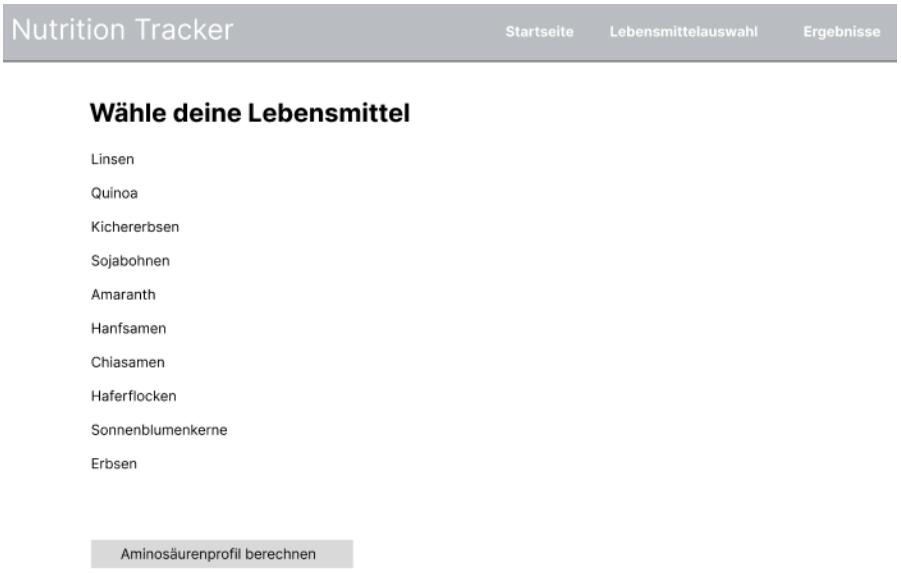
3.1 Frontend: Layout

Im Folgenden werden Mid-Fidelity-Mockups der Anwendung dargestellt.

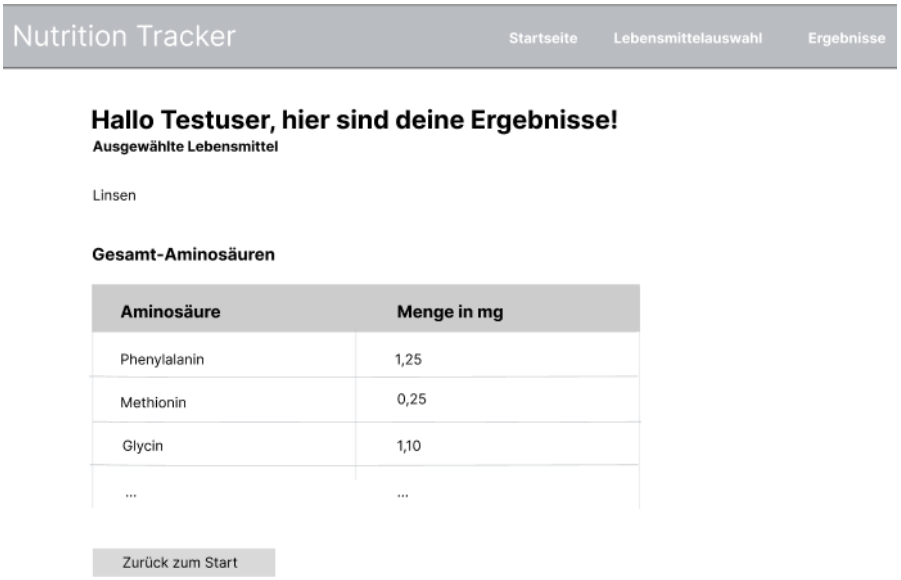
3.1.1 HomeView – Benutzerdaten-Eingabe



3.1.2 FoodSelectionView – Lebensmittelauswahl



3.1.3 ResultView – Auswertung der Aminosäuren



3.2 Frontend: Logik

Das Frontend basiert auf Vue 3, Vuetify und Axios. Die Logik folgt dem Prinzip der Trennung von Darstellung und Funktionalität. API-Aufrufe sind direkt in den Views eingebunden und erfolgen über einen zentralen Axios-Wrapper.

Verwendete Endpunkte:

- PUT /user-data/update, GET /user-data (Benutzerdaten)
- GET /food-items/all (Lebensmittel)
- POST /nutrition-logs/create, POST /nutrition-logs/{logId}/food-items/{foodItemId} (Lebensmittel hinzufügen in das Ernährungstagebuch)
- POST /amino-profile/sum, GET /amino-profile/daily-needs, POST /amino-profile/coverage (Aminosäurenanalyse)

Modelldaten im Frontend:

- UserModel: Name, Alter, Gewicht, Flags (Athlet, Vegan, Longevity)
- FoodItemModel: ID, Name

- AminoProfileModel: Aminosäuren-Summen, Tagesbedarf, Deckung in %

Logik (Handler) pro View:

- HomeView: toggleDecorator, goToSelection
- FoodSelectionView: toggleFoodSelection, processSelections
- ResultView: closeApp

Utils:

- localStorage
- Axios-Wrapper

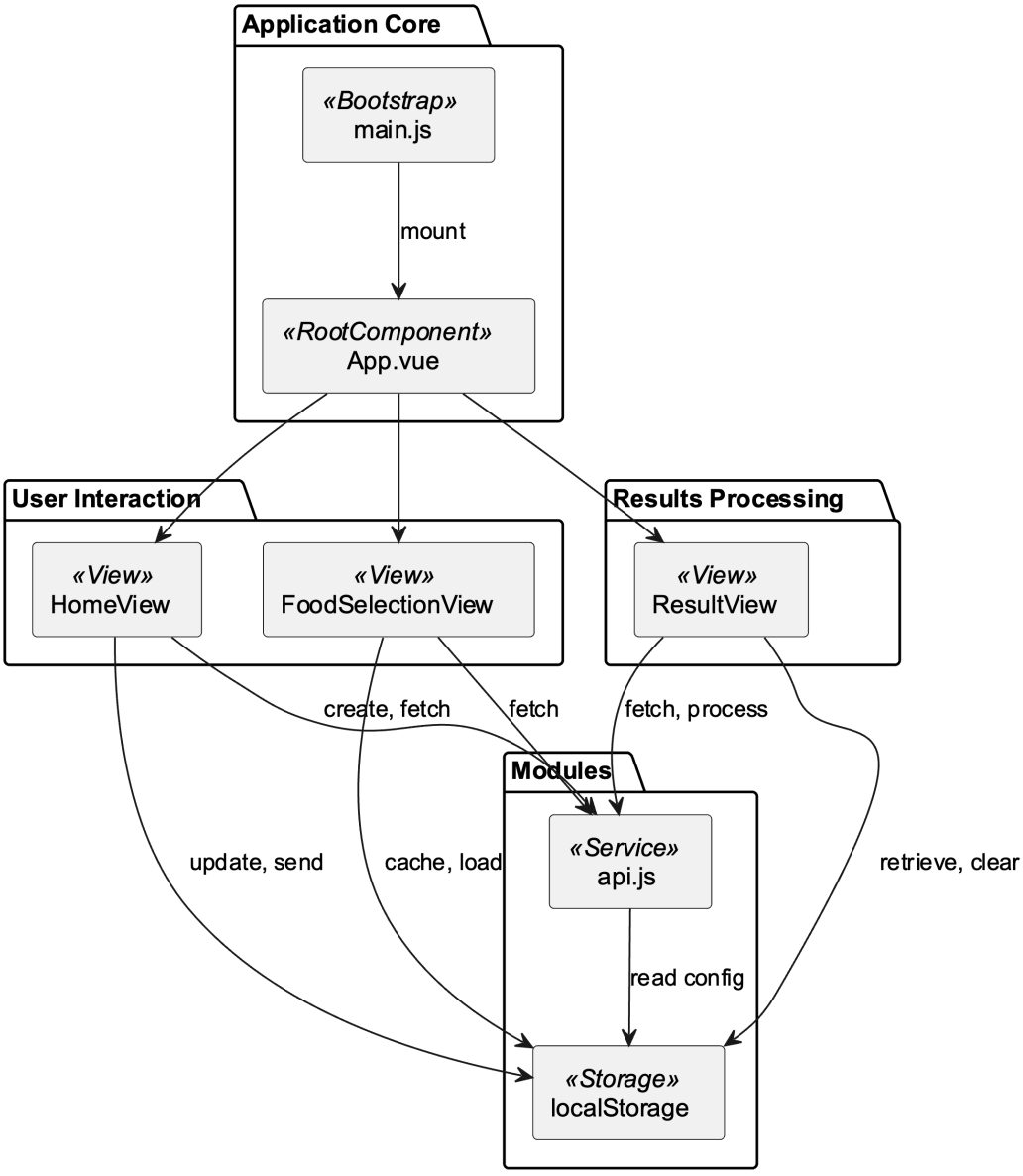


Abbildung 3: UML-Komponentenmodell des Nutrition Tracker-Frontend

3.3 Spezialthema im Frontend

Im Rahmen der Frontend-Implementierung wurde das Vuetify-Framework als zentrales UI-Toolkit verwendet. Vuetify basiert auf dem Material Design-Konzept und bietet eine Vielzahl an vorgefertigten, responsiven und barrierefreien Komponenten, die eine konsistente Benutzeroberfläche ermöglichen.

Die Anwendung nutzt spezialisierte Vuetify-Komponenten gezielt zur Umsetzung der zentralen Use Cases:

- In der HomeView.vue werden Benutzerdaten über ein strukturiertes v-form mit v-text-field-Eingabefeldern aufgenommen. Zusätzlich kommen benutzerdefinierte Checkboxes auf Vuetify-Basis zum Einsatz, um optionale Einstellungen wie “Athlet”, “Vegan” oder “Gesund altern” intuitiv abzubilden.
- Die FoodSelectionView.vue nutzt die Vuetify-Komponenten v-list und v-list-item, um eine klar strukturierte und interaktive Lebensmittelauswahl zu realisieren. Über die dynamische visuelle Hervorhebung ausgewählter Einträge wird die Benutzerführung verbessert.
- In der ResultView.vue werden die berechneten Ergebnisse mithilfe mehrerer v-table-Komponenten dargestellt. Diese Tabellen sind übersichtlich gestaltet und ermöglichen eine differenzierte Darstellung der aufgenommenen Aminosäuren, des Tagesbedarfs sowie der prozentualen Deckung. Zusätzlich wird ein v-alert zur Fehlermeldung und ein v-btn zur Navigation eingesetzt.

Durch den gezielten Einsatz dieser Vuetify-Komponenten konnte eine konsistente, benutzerfreundliche und klar strukturierte Benutzeroberfläche geschaffen werden, ohne eigene Low-Level-UI-Komponenten manuell entwickeln zu müssen. Die Verwendung eines etablierten UI-Frameworks steigert zudem die Wartbarkeit und Erweiterbarkeit der Anwendung.

3.4 Frontend: Test

Der folgende End-to-End-Test wurde mit dem Testing-Framework Cypress im Rahmen der Projektstruktur unter frontend/cypress/e2e implementiert.

Im beforeEach-Block wird ein definierter Testnutzer über einen PUT-Request an das Backend persistiert, um reproduzierbare Ausgangsbedingungen sicherzustellen. Anschließend wird die Benutzerinteraktion in drei Teilschritten simuliert.

Im ersten Schritt erfolgt die Navigation zur Seite /selection, gefolgt von der Validierung, dass mindestens drei auswählbare Lebensmittel im DOM der gerenderten Anwendung vorhanden sind. Dabei wird geprüft, ob die entsprechenden HTML-Elemente nach dem Rendern im Browser verfügbar und visuell zugänglich sind.

Im zweiten Schritt werden gezielt drei Lebensmittel ausgewählt, woraufhin die Berechnung des Aminosäureprofils ausgelöst wird. Die zugehörigen API-Endpunkte (/amino-profile/sum, /amino-profile/daily-needs, /amino-profile/coverage) werden mittels cy.intercept() überwacht. Die erfolgreiche Verarbeitung wird durch die Validierung von HTTP-Statuscodes (200 OK) abgesichert. Ergänzend werden Timeout-Parameter sowie Sichtbarkeitsprüfungen (.should("be.visible")) eingesetzt, um potenzielle Latenz oder Verzögerungen bei der UI-Darstellung abzufangen.

Im dritten Abschnitt wird die Ergebnisseite überprüft. Es erfolgt eine strukturierte Validierung der Begrüßung des Nutzers, der Anzeige der gewählten Lebensmittel sowie der korrekten Darstellung der berechneten Tabelleninhalte (Aminosäuren, Tagesbedarf, prozentuale Deckung). Abschließend wird die Rücknavigation zur Startseite simuliert und ebenfalls vlidiert.

4 Projektdokumentation und Rückblick

4.1 Projektmanagement

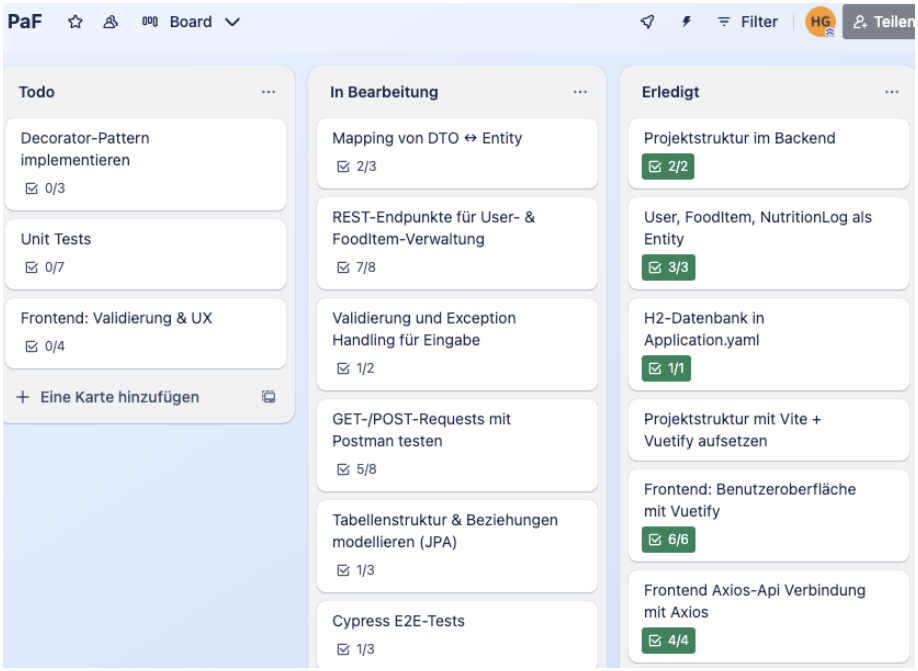


Abbildung 4: Dieses Board zeigt den Stand nach Umsetzung der API-Endpunkte

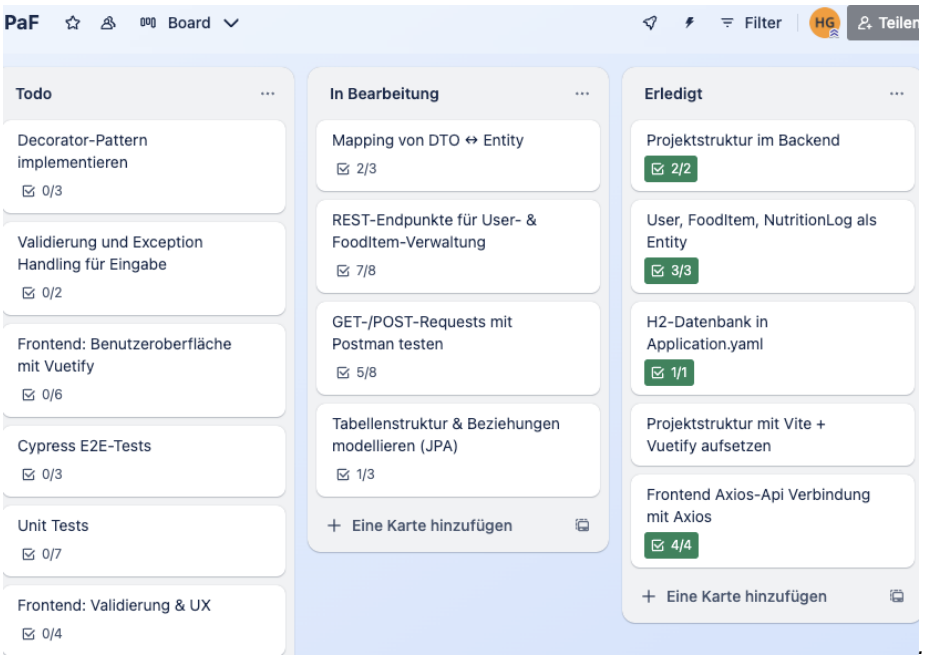


Abbildung 5: Dieses Board zeigt den Projektstand während der End-to-End-Testphase

4.2 Versionierung

Die Commits sind auf zwei nacheinander genutzte Repositories verteilt und dokumentieren den fortlaufenden Entwicklungsprozess.

```
~/Projekte/IntelliJ/NutritionTracker git:[main]
git log --oneline
38fef8c (HEAD -> main, origin/main) Added Vue.js frontend to NutritionTracker
7485f2e Added comments and improved formatting
c4b8884 build decorators in getuser
f258622 functional Version without database reset
eb2ae66 amino acid calculation and API endpoints
dcfa5ee Integrated validation in UserDTO and added GlobalException for user-friendly validation outputs
b47c61e Fixed API endpoints and implemented UserData to allow the frontend to pass user data to the backend
3546931 DTO
ee595a0 feat(tests): Add E2E-like test for DevDataLoader user and amino acid calculations
467329b Fix: Adjusted logging dependencies in pom.xml
069b639 Add functionality to link FoodItems with NutritionLog.
3b42319 Adjusted pom.xml: Made changes
44b504c 🔧 Updated profiles 'dev' & 'test': Refactored application-test setup & fixed user-saving logic
d514382 Add integration testing for UserRepository to validate database interactions
50cf937 Backend improvements: Refactored controllers, updated service logic, revised repository methods, and improved test coverage
07eeeb8 Integrated Angular frontend with Material UI, adjusted code, and verified PostgreSQL connection
d4bceea Implemented Angular in the Frontend directory
14fa159  Add NutritionLogServiceTest with CRUD and calculation tests
530723c Initialize H2 database, update application.yml, and set up initial test data
ebd1c6d Initialize H2 database, update application.yml, and set up initial test data
108a694 Refactored: Simplified logic and corrected Constructor Injection
e8b44e2 feat(logging): Added logging capabilities to StartupRunner for better system monitoring and debugging
a908238 Implemented Decorator Pattern for amino acid calculation
376119c Implemented Lombok configuration and resolved constructor issues: - Added and configured @NoArgsConstructor in affected entities - Ensured Lombok
  plugin is enabled and annotation processing is active - Updated Maven dependencies to address warnings and vulnerabilities
57d6462 Add Builder Pattern Test and Update Maven Config
fb13d27 Implement Builder Pattern for NutritionLog to streamline object creation
f093bbd Add API, Repository, and Service layers for NutritionTracker
```

Abbildung 6: Commit-Historie des ursprünglichen Repositories – frühe Backend-Implementierung mit Strukturaufbau, Datenmodellierung und Einführung der Berechnungslogik

```
~/Projekte/IntelliJ/NutritionTrackerApp git:[main]
git log --oneline
8d81356 (HEAD -> main, origin/main) Test for AminoAcid sum
1e98314 Updated E2E-Test
f553403 Updated validation in HomeView
8e82890 Add new README and LICENSE files
c55bfeb Modified package declaration
af00d60 Unit Tests
16db4ef Correct method invocation
397116a Full end-to-end test with adjustments in the views
48d96e3 Added Cypress tests and updated OpenAPI configuration
a66b9d9 Added OpenAPI YAML for API documentation and updated pom.xml configuration
33595a7 Added frontend and backend
```

Abbildung 7: Commit-Historie des Hauptprojekts – strukturierte Weiterentwicklung mit Fokus auf API-Stabilisierung, Validierungslogik, Decorator-Pattern und End-to-End-Test

4.3 Retrospektive

Im Projektverlauf wurde eine Vielzahl an funktionalen Anforderungen erfolgreich umgesetzt. Besonders hervorzuheben sind die Schichtentrennung im Backend, die strukturierte API-Entwicklung mit DTOs und Repositories sowie die stabile Integration eines Vuetify-Frontends über Axios. Die schrittweise Einführung des Decorator-Patterns ermöglichte eine flexible Berechnungslogik für unterschiedliche Nutzerprofile.

Zu Beginn gestaltete sich die Arbeit mit einem initialen Repository unübersichtlich, weshalb die Entwicklung in ein zweites, strukturierter aufgebautes Repository überführt wurde.

Herausforderungen traten u. a. bei der Validierung und der Behandlung von Lazy-Loading im Zusammenhang mit JPA auf. Diese wurden durch gezielte Backend-Anpassungen, DTO-Nutzung und @Transactional-Annotationen gelöst. Während der End-to-End-Testphase zeigte sich, dass die Rückmeldelogik und API-Kommunikation weiter verbessert werden mussten, was zu mehreren iterativen Anpassungen führte.