Team 3: Teammitglieder

Maria Lorenz - Matrikelnummer: 70494042
Jesse Khala - Matrikelnummer: 20242922
Thomas Brehmer - Matrikelnummer: 104492
Stephan Wolf - Matrikelnummer: 915660

Moderne Softwareentwicklung | BHT MIM 20 W25

FoodRescue - Zwischendokumentation

Inhaltsverzeichnis

Einführung und Kontext **3**
Repository Setup und Grundlagen **3**
CI/(CD-)Implementation **3**
Domain-Modellierung und Event-Struktur **6**
Projekt-Implementierung **10**
LLM-Integration **11**
Ausblick: **11**

Einführung und Kontext

Problemstellung: Die Erstellung einer Plattform zur Vermeidung von Lebensmittelverschwendung.
Konkrete Zielsetzung: Überschüssige Lebensmittel werden angeboten, Nutzer können diese reservieren und abholen.

Technologie-Stack:

Bereich	Technologien
Backend	Java 21, Spring Boot 3.3.4
Frontend	HTML/CSS, JavaScript
Testing	JUnit 5, Mockito, Spring MockMvc
Build	Maven, Spotless (Google Java Format), JaCoCo (Coverage)
CI/CD	GitHub Actions, GitHub Pages

Tab. 1: Technologie-Stack

Repository Setup und Grundlagen

Repository:
https://github.com/futurefounder/moderne-softwareentwicklung-mim-20-w25-team-3-foodrescue

CI/(CD-)Implementation

Branch Protection:

- Restrict deletions auf `main`
- Require pull request bevor gemergt wird

Automatisierte Checks (bei jedem PR):

- Maven Build & Tests (Unit, Controller, Integration)
- Spotless Code-Formatierung (Google Java Format)
- Deployment Check (nur auf `main`)

Herausforderung	Lösung
Kollaborative Fehler	Branch Protection + CI/CD Checks
Dokumentation vergessen	Maven Site automatisch bei Deployment generiert
Lange Build-Zeiten	Maven-Cache in GitHub Actions
Unterschiedliche Formatierung	Spotless erzwingt Google Java Format
Dependencies neu laden	Maven-Cache reduziert Build-Zeit

Tab. 2: Herausforderung - Lösung

Aktuelle Build-Zeit: ~2:40 Minuten auf `main`

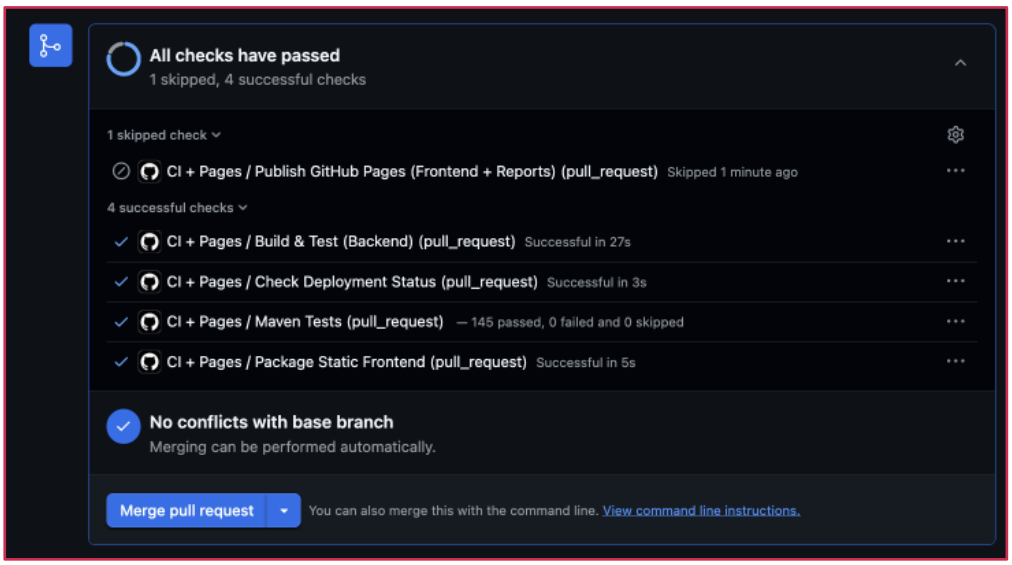

Abb. 1: Beispiel Checks eine Pull Requests

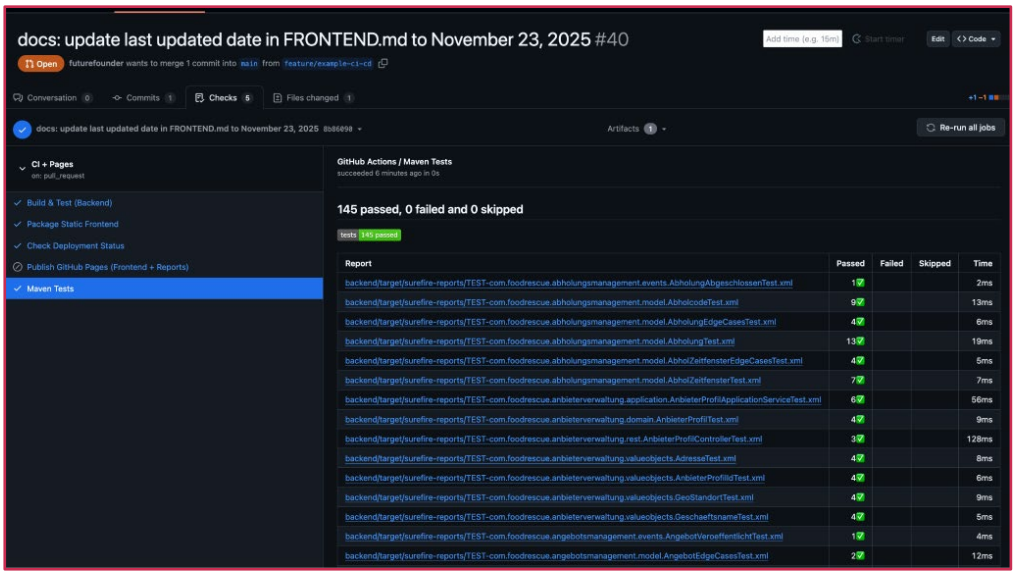

Abb. 2: Beispiel Actions-Log für Maven Tests

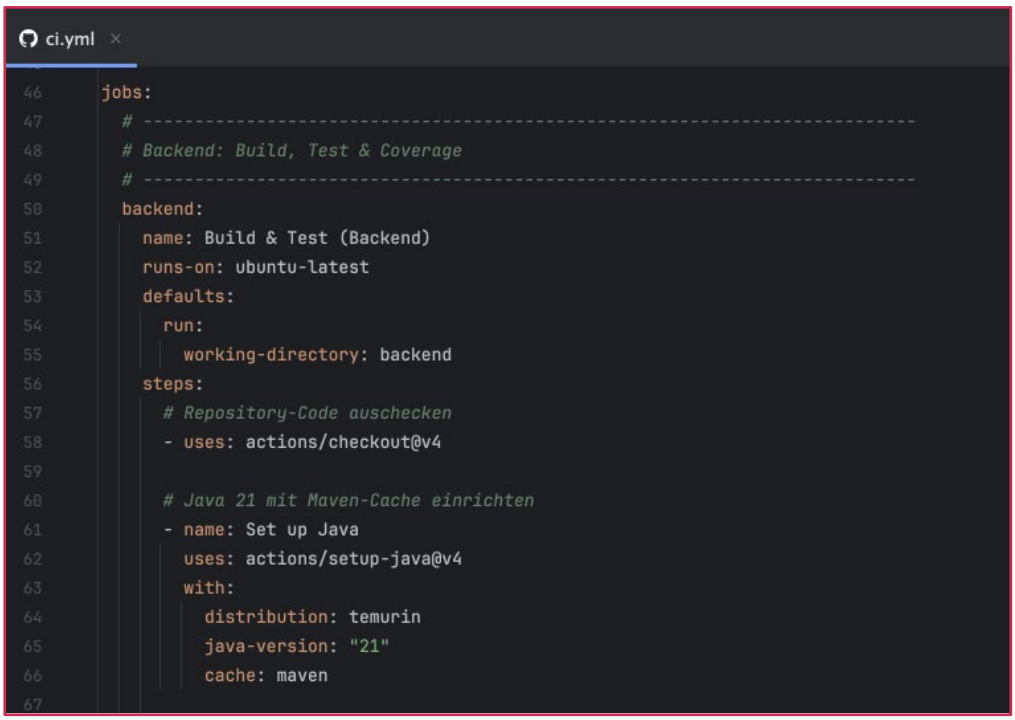

Abb. 3: Beispiel Checks eine Pull Requests

Domain-Modellierung und Event-Struktur

Bounded Context	Zugehöriges Domain Event	Verantwortlichkeit
Angebotsmanagement	Angebot veröffentlicht	Verwaltung von Angeboten
Reservierungsmanagement	Reservierung erstellt	Verwaltung von Reservierungen
Abholungsmanagement	Abholung abgeschlossen	Verwaltung von Abholungen
Userverwaltung	Benutzerkonto angelegt	Verwaltung von Benutzerkonten

Tab. 3: DDD-Struktur: Bounded Contexts & Domain Events

Strukturierung: Package-Struktur spiegelt Bounded Contexts wieder (`src/main/java/com/foodrescue/[context]/`)

Kernelemente: Aggregate, Entities, Value Objects, Domain Events pro Context

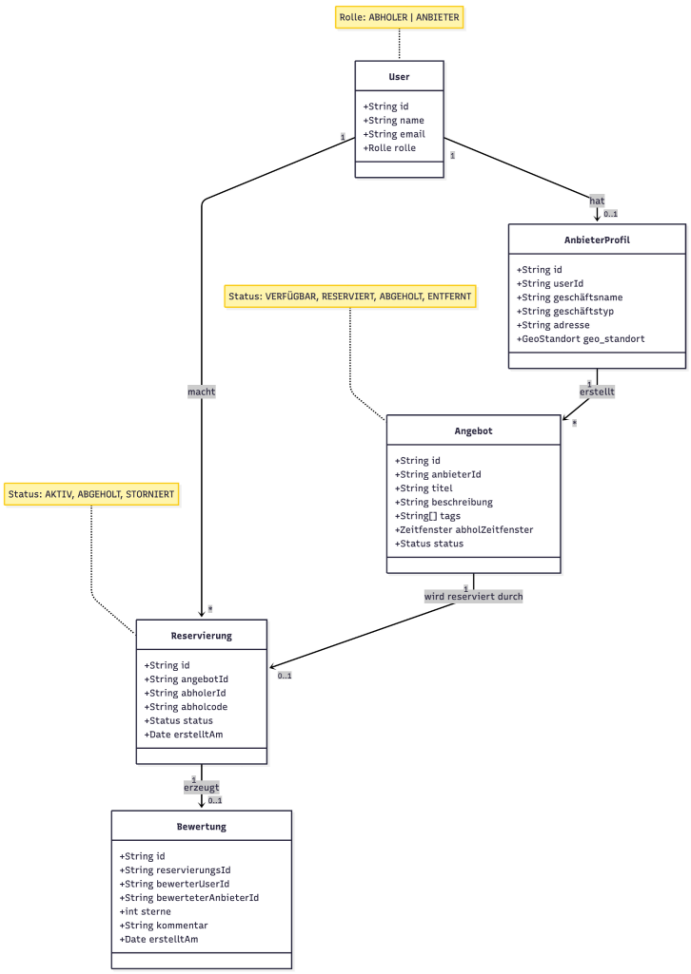

Abb. 4: Klassendiagramm nach DDD via Mermaid Chart

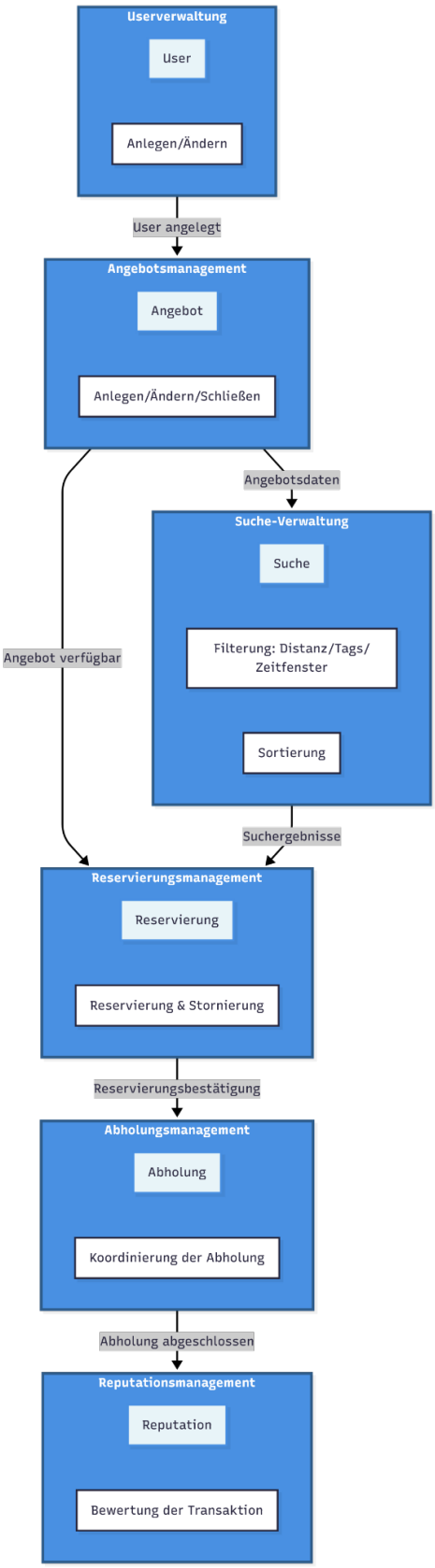

Abb. 5: Diagramm zum Bounded Context via Mermaid Chart

Anmeldemaske

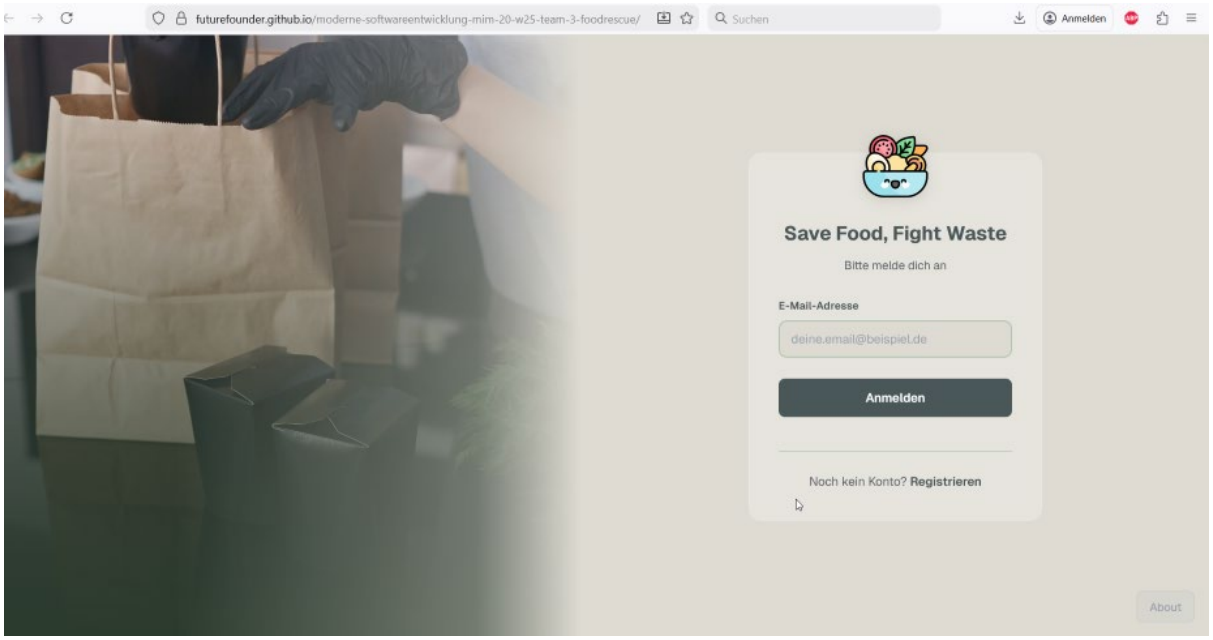

Abb. 6: Anmeldemaske

Registrierungsmaske

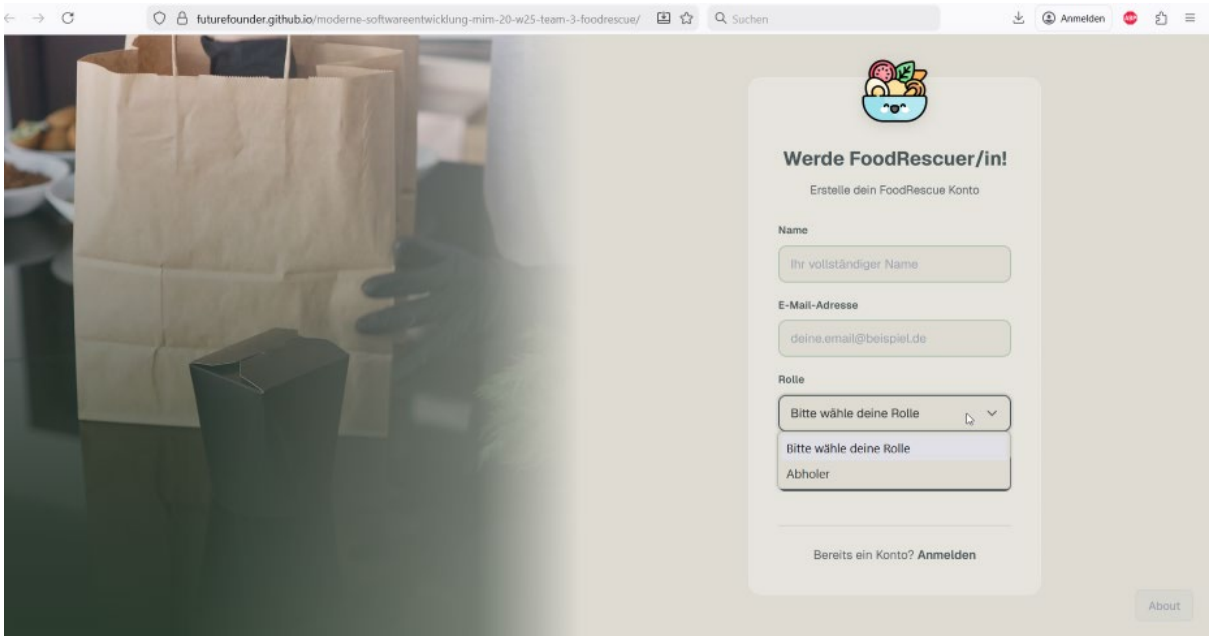

Abb. 7: Registrierungsmaske

Projekt-Implementierung

Projektstruktur:

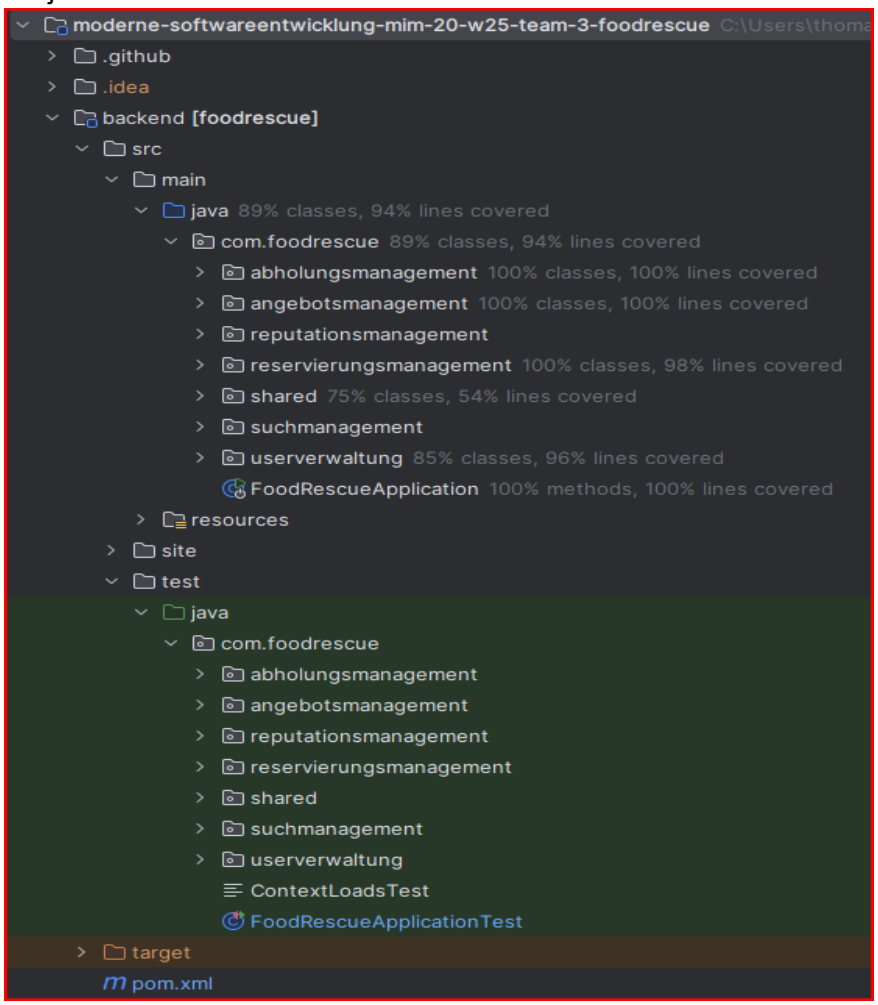

Abb. 8: IDE-Screenshot mit Projektstruktur

Teststrategie:

Workflow:

1. Fachliche Anforderungen direkt in Tests formulieren (Was soll das Event ausdrücken?)
2. Produktiven Code erstellen und anpassen bis Tests "grün" sind
3. Refactoring: Code verbessern, Duplizierungen entfernen, Namen optimieren
4. Edge Cases testen, um Grenzfälle abzudecken

Test-Kategorien:

- Unit-Tests: Business-Logik (z.B. `RescueServiceTest`)

- Controller-Tests: REST-API mit MockMvc (z.B. `HealthControllerTest`)
- Integrationstests: Spring-Kontext-Validierung (z.B. `ContextLoadsTest`)

LLM-Integration

Tool	Einsatzbereich	Nutzen
GitHub Copilot	Code-Generierung	Schnellere Entwicklung, Pair-Programming
SpotBugs	Fehleridentifikation	Schnellere Entwicklung, Pair-Programming
SonarQube IDE	Echtzeit-Code-Analyse	Probleme beim Schreiben erkennen
ChatGPT	Test-Erstellung, Code-Review	Testfälle, Verbesserungsvorschläge
Claude AI	Diagramm-Erstellung	Klassendiagramme, Bounded Context

Tab. 4: LLM im Einsatz

Herausforderungen: Die Qualität ist abhängig von der LLM-Version, die verwendet wird. Sicherheitsrisiken können bei der Weitergabe von Firmeninterna entstehen. Der erstellte Code kann überladen sein, daher ist eine Kontrolle notwendig.

Ausblick:

Woran wir noch arbeiten wollen

- CI/CD Pipeline optimieren (z.B. ‘main’ build liegt aktuell bei ca. 2:40 Minuten)
- Ausbau Backend: Spring Security, Spring Data (aktuell Testing im Frontend via localstorage)
- Kern-Features ausbauen: Passwort bei Anmeldung, Abholer-Registrierung
- Ggf. Frontend-Ausbau via Framework (React.js/Next.js)