

FoodRescue - Zwischenpräsentation Handout

Moderne Softwareentwicklung | BHT MIM 20 W25 | Team 3

Team: Maria Lorenz, Stephan Wolf, Thomas Brehmer, Jesse Khala

Problemstellung: Plattform zur Vermeidung von Lebensmittelverschwendung

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

DDD-Struktur: Bounded Contexts & Domain Events

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

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

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

DDD + TDD Kombination

Workflow:

- Fachliche Anforderungen direkt in Tests formulieren (Was soll das Event ausdrücken?)
- Produktiven Code erstellen und anpassen bis Tests "grün" sind
- Refactoring: Code verbessern, Duplizierungen entfernen, Namen optimieren
- 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-Einsatz

Tool	Einsatzbereich	Nutzen
GitHub Copilot	Code-Generierung	Schnellere Entwicklung, Pair-Programming

Tool	Einsatzbereich	Nutzen
SpotBugs	Fehleridentifikation	Nutzen: Nullzeiger, Deadlocks, falsche Library-Nutzung
SonarQube IDE	Echtzeit-Code-Analyse	Probleme beim Schreiben erkennen
ChatGPT	Test-Erstellung, Code-Review	Testfälle, Verbesserungsvorschläge
Claude AI	Diagramm-Erstellung	Klassendiagramme, Bounded Context

Herausforderungen: Qualität abhängig von LLM-Version, Sicherheitsrisiken bei Firmeninterna, Code kann überladen sein → Kontrolle notwendig

CI/CD Pipeline

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`)

Herausforderungen & Lösungen:

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

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

Ausblick

- **CI/CD optimieren:** Build-Zeit reduzieren
- **Backend ausbauen:** Spring Security, Spring Data (aktuell Frontend via `localStorage`)
- **Kern-Features:** Passwort bei Anmeldung, Abholer-Registrierung erweitern
- **Frontend-Framework:** Ggf. Migration zu `React.js/Next.js`

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

Deployment: GitHub Pages <https://futurefounder.github.io/moderne-softwareentwicklung-mim-20-w25-team-3-foodrescue/>