# Testkonzept für das Recipe-Planner-Projekt

## Zusammenfassung (Introduction)

Dieses Testkonzept beschreibt die Vorgehensweise zur Qualitätssicherung des Recipe-Planner-Projekts. Ziel ist es, sicherzustellen, dass die Applikation den funktionalen und nicht-funktionalen Anforderungen entspricht. Es wird nach dem Test-Driven-Development-Ansatz (TDD) entwickelt, wobei Unit-, Integration- und End-to-End-Tests zum Einsatz kommen.

## Big Picture - Systemarchitektur mit den Test Items

Das System besteht aus einem React-Frontend und einem Java-Spring-Boot-Backend.

Frontend:

- React-Komponenten, die in `src/components` implementiert sind.  
- Routing- und UI-Elemente basieren auf React und Drittanbieterbibliotheken.

Backend:

- Spring-Boot-Applikation mit Controller-, Service-, Repository- und Domain-Schichten.  
- Kommunikation erfolgt über REST-APIs.

Test Items:

- Frontend-Komponenten (UI-Rendering, Zustand und Verhalten).  
- REST-API-Endpunkte (CRUD-Operationen, Validierung, Fehlerbehandlung).  
- Backend-Logik (Services und Datenbankinteraktion).

## Test Features - Welche Elemente werden getestet

Frontend:  
- Rendering und Funktionalität der Hauptkomponenten (z. B. AddRecipe, Browse, Planer).  
- Navigation zwischen Seiten und Zustandshandling.  
- Validierung von Benutzereingaben.  
  
Backend:  
- REST-Endpunkte auf korrekte Datenverarbeitung und Validierung.  
- Service-Methoden auf korrekte Logik.  
- Datenbankoperationen über Repositorys.

## Features not to be tested - Welche Aspekte nicht getestet werden

- Nicht-kritische Styling-Aspekte im Frontend (z. B. CSS-Details).  
- Drittanbieter-Bibliotheken (nur grundlegende Funktionalität wird überprüft).  
- Deployment-spezifische Konfigurationen.

## Testvorgehen - Nach dem TDD-Ansatz

1. Schreiben von Tests vor der Implementierung:  
 - Jede neue Funktionalität beginnt mit einem fehlgeschlagenen Test.  
2. Implementierung:  
 - Implementierung der Funktionalität bis die Tests bestehen.  
3. Refactoring:  
 - Optimierung des Codes ohne Veränderung der Funktionalität.

Testarten:

- Unit-Tests: Für einzelne Komponenten und Methoden.  
- Integrationstests: Für Schnittstellen zwischen Modulen (z. B. Frontend-API-Aufrufe).  
- End-to-End-Tests: Simulation von Benutzerinteraktionen.

## Kriterien für erfolgreiche / nicht-erfolgreiche Tests

- Erfolgreich: Alle Unit-, Integrations- und End-to-End-Tests bestehen ohne Fehler.  
- Nicht-erfolgreich:  
 - Fehler in kritischen Funktionalitäten.  
 - Nicht-erfüllte Anforderungen aus den User Stories.

## Testumgebung - Tools

- Frontend: Jest und React Testing Library.  
- Backend: JUnit für Unit- und Integrationstests.  
- End-to-End-Tests: Cypress.  
- Datenbank: H2 für In-Memory-Tests.

## Kurze Planung

1. Woche 1: Einrichten der Testumgebung und Schreiben von Unit-Tests für vorhandene Module.  
2. Woche 2: Integrationstests für die REST-APIs.  
3. Woche 3: End-to-End-Tests für den Benutzerfluss.  
4. Woche 4: Fehlerbehebung und Abdeckung verbleibender Szenarien.