12 Unit Tests und End-to-End-Tests

Thema

Testarten, Mocking, Datenbanktests und Tests über die echte HTTP-Schnittstelle.

Auftrag

Erstelle für einen Service mindestens drei Unit Tests. Isoliere das Repository mit einem Mock und prüfe dabei einen Erfolg, einen fachlichen Fehler und einen Fehler des Datenzugriffs.

Erstelle zusätzlich eine End-to-End-Teststrecke für die API. Starte die Anwendung und Datenbank in einer Testumgebung und führe die Requests zuerst mit `curl`, danach als Postman-Collection mit Post-response-Tests aus.

Mocking und Datenbanken

Begründe für jeden Test, ob du:

- ein Repository oder eine externe Abhängigkeit mockst;
- eine In-Memory-Datenbank verwendest;
- eine echte isolierte Testdatenbank verwendest.

Ein Mock ist sinnvoll, wenn ein Unit Test eine Abhängigkeit isolieren, schnell ausführen oder Fehler gezielt simulieren soll. Eine In-Memory-Datenbank kann für einfache Integrationsfälle hilfreich sein, bildet aber nicht immer das Verhalten der produktiven Datenbank ab. Für ORM-, SQL-, Migrations- und Constraint-Tests ist eine echte isolierte Testdatenbank zuverlässiger.

Abgabe

- Unit-Testdateien mit mindestens drei Fällen;
- Erklärung der verwendeten Mocks;
- Entscheidung und Begründung für Mock, In-Memory-Datenbank oder Testdatenbank;
- `curl`-Teststrecke;
- Postman-Collection mit mindestens fünf End-to-End-Tests;
- kurze Testdokumentation mit bekannten Grenzen.

Lernzielkontrolle

1. Was unterscheidet einen Unit Test von einem End-to-End-Test?
2. Warum werden Abhängigkeiten in Unit Tests gemockt?
3. Was ist der Unterschied zwischen einem Mock, einer In-Memory-Datenbank und einer echten Testdatenbank?
4. Warum kann eine In-Memory-Datenbank einen Test mit PostgreSQL nicht vollständig ersetzen?
5. Was prüft ein End-to-End-Test zusätzlich zu einer einzelnen Funktion?
6. Wie können Postman-Tests Statuscodes und Response-Daten prüfen?

Zeitbudget

Zeit: ca. 6 Stunden