Testing in TypeScript

Unit Testing

Unit Tests helfen, Fehler frühzeitig zu erkennen und sorgen für eine höhere Codequalität.

Test-Patterns und Best Practices

- AAA-Pattern (Arrange, Act, Assert)
- Given-When-Then für bessere Lesbarkeit

Test-Daten generieren (Dummies, Mocks, Stubs, Fakes)

- Dummy: Platzhalter ohne echte Funktionalität
- Stub: Liefert vordefinierte Werte zurück
- Mock: Simuliert Verhalten, prüft Aufrufe
- Fake: Alternative Implementierung, z. B. In-Memory-Datenbank

Unit Testing mit Jest (für Node.js-Projekte)

1. Jest und TypeScript-Support installieren:

```
npm install --save-dev jest ts-jest @types/jest
```

2. Jest konfigurieren:

```
/** @type {import('ts-jest').JestConfigWithTsJest} */
module.exports = {
 testEnvironment: "node",
  transform: {
    "^.+\.tsx?$": ["ts-jest",{}],
  },
  testPathIgnorePatterns: ["<rootDir>/deno.tests"]
};
```

3. Skript in package.json hinzufügen:

```
"scripts": {
  "test": "jest"
}
```

Unit Testing mit Deno Test

Deno hat ein eingebautes Test-Framework, daher brauchen wir keine externe Bibliothek.

Tests ausführen:

```
deno test
```

Test-Setup mit BeforeEach & AfterEach

- `beforeEach`: Initialisierung vor jedem Test.
- `afterEach`: Aufräumen nach jedem Test.

Test Coverage messen

Mit Jest

```
npx jest --coverage
```

Mit Deno

```
deno test --coverage
```