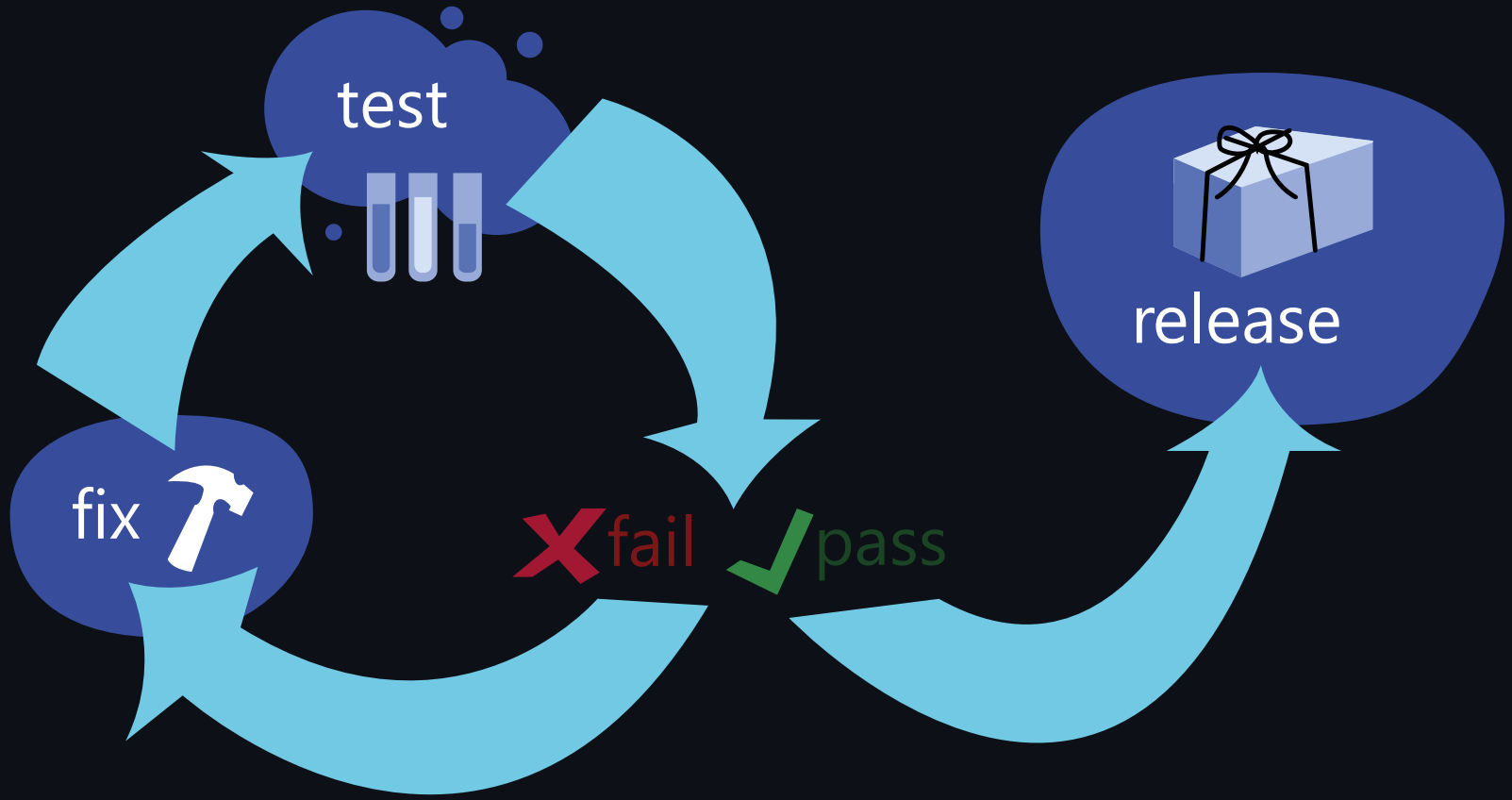
Testing React Native



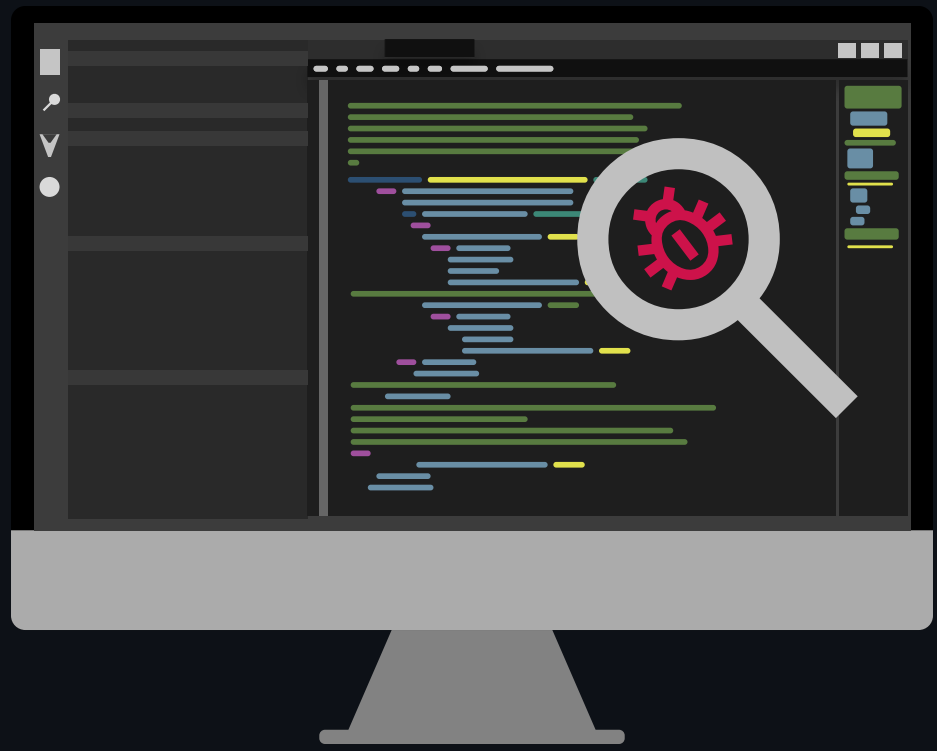
Julian Viechter, Moritz Kummer

Motivation

- Code-Qualität
  - Stellt sicher, dass der Code funktioniert
- Zeitersparnis
  - Probleme können frühzeitig erkannt werden bevor sie zu größeren Problemen werden
- Dokumentation
  - Zeigen wie der Code verwendet werden sollte und welche Ergebnisse zu erwarten sind

Testbarer Code

- Kleine und fokussierte Komponenten / Funktionen
- Pure Functions / Props verwenden
- Trennen von Logic und UI
- Test Driven Development



Jest

- Eine Test-Laufzeitumgebung von Facebook
- Unterstützt Snapshot-Tests, Unit-Tests, Integrationstests und End-to-End-Tests
- Code Coverage
- Mocking
- Einfach zu konfigurieren und auszuführen

Setup

Testframework Jest installieren

- jest
- @types/jest
- jest-expo

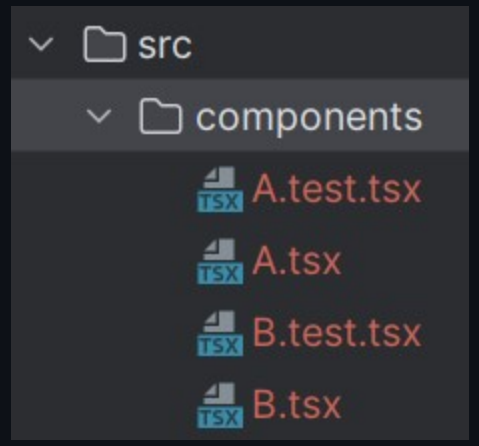
```
npm install jest @types/jest jest-expo --save-dev
```

jest.config.json

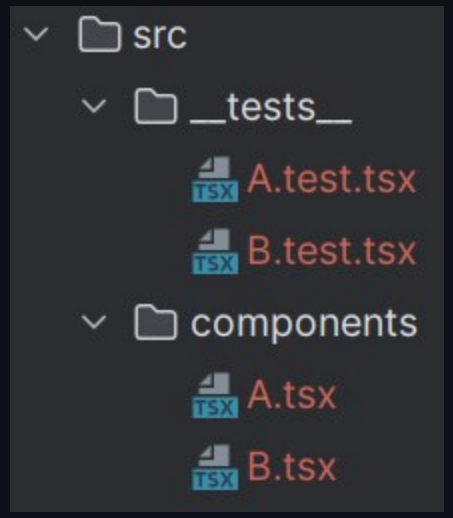
```
{
    "preset": "jest-expo", // "react-native" ohne expo
}
```

Tests anlegen: Ordner Struktur

Daneben



Daneben in einem Ordner



In eigenem Ordner

Tests schreiben: Unit Tests

- Testen einzelner Komponenten / Funktionen
- Unabhängig voneinander
- `it` / `test` (synonym)

```
it('should add thousands separators to integers', () => {
    const result = addThousandsSeparators('1234567890');
    const expected = '1,234,567,890';

    expect(result).toBe(expected);
});
```

Ausführen

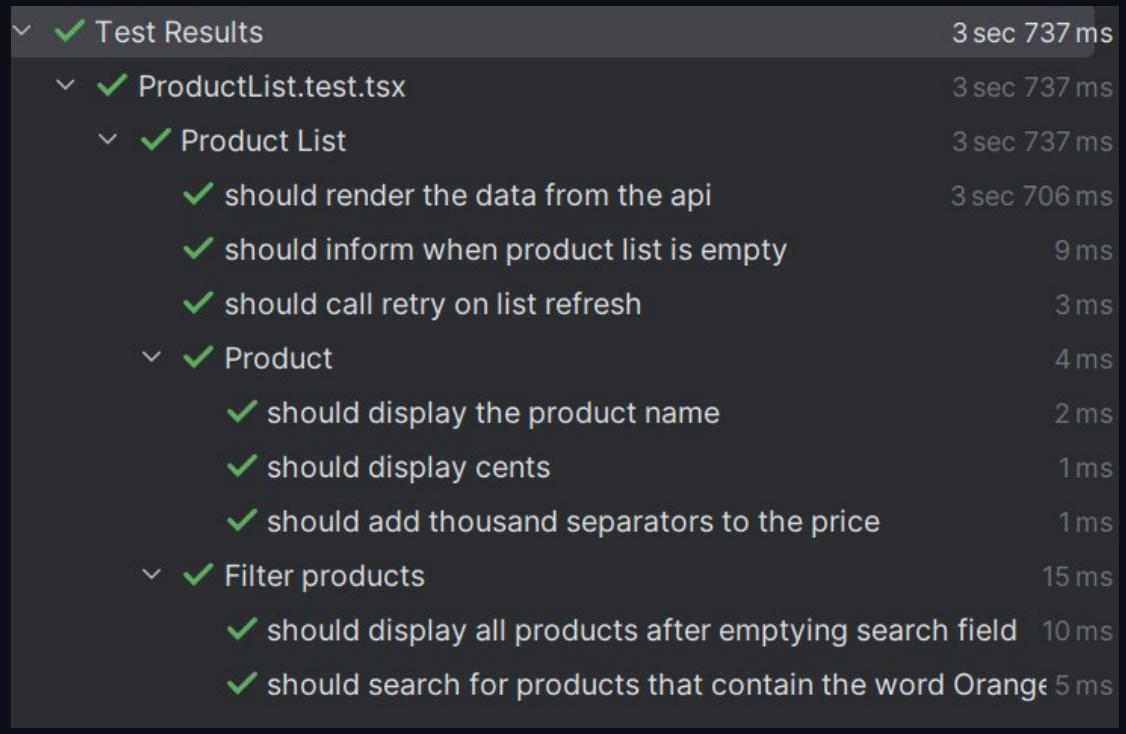
```
npm test
yarn test
```

Matchers

```
expect(x).toBe(y); // Exakte Gleichheit
expect(x).toEqual(y); // Inhaltliche Gleichheit

expect(x).toBeUndefined();
expect(x).not.toBeUndefined(); // negieren mit not

expect(x).toBeTruthy();
expect(x).toBeGreatherThan(y);

expect(x).toContain(y);
expect(x).toThrow(myError);
```

Gruppieren von Tests mit `Describe`

```
describe('Product List', () => {
    it('should render the data from the api');
    it('should inform when product list is empty');
    ...

    describe('Product', () => {
        it('should display the product name');
        ...
    });

    ...
});
```



Component Tests Setup

- @testing-library/react-native
- @testing-library/jest-native (optional für zusätzliche Matcher)
- react-test-renderer
- @types/react-test-renderer

```
npm install --save-dev @testing-library/react-native @testing-library/jest-native react-test-renderer @types/react-test-renderer
```

Wenn @testing-library/jest-native genutzt wird:

jest.config.json

```
{
    "setupFilesAfterEnv": ["@testing-library/jest-native/extend-expect"]
}
```

Component Tests

- Interaktion
  - Testen wie der User mit der Komponente interagiert
  - Was ändert sich durch eine Interaktion (z.B. ein Button wurde gedrückt)
  - Nicht den State change an sich testen sondern was wird dem User präsentiert
- Rendering
  - Aussehen und positionierung von Elementen

Einfacher Render Test

```
const Product: React.FC<ProductProps> = ({ item }) => (
    <View style={styles.product}>
        <Text style={styles.productName}>{item.name}</Text>
        <Text style={styles.productPrice}>
            {addThousandsSeparators((item.price / 100).toString())}€
        </Text>
    </View>
);
```

```
it('should add thousand separators to the price', () => {
    render(<Product item={{ id: '1', name: 'Foo', price: 234599 }} />);

    expect(screen.getByText('2,345.99€')).toBeOnTheScreen();
});
```

Queries

Tests sollten so ähnlich wie möglich dazu sein, wie Benutzer mit dem Code / Komponente / Screen Interagieren.

```
screen.getByText(...) // Empfohlene Methode
screen.getByDisplayValue(...) // TextInput
screen.getByPlaceholderText(...) // TextInput
...

screen.getByRole(...) // Accessibility Role für z.B.Screen Reader

<ActivityIndicator testID={'unique-id'}/>
screen.getByTestId('unique-id') // Geht immer aber nicht empfohlen
```

Komplexer Render Test

Snapshot Testing

- Konvertiert den gerenderten Output zu JSON
- Gespeichert in `__snapshots__/<File>.test.tsx.snap`
- JSON string wird in VCS mit aufgenommen und beim Testen verglichen
- Gut zum absichern gegen unerwartete Änderungen

```
it('renders correctly', () => {
    const tree = renderer
        .create(<Product item={{ id: '1', name: 'Foo', price: 234599 }} />)
        .toJSON();

    expect(tree).toMatchSnapshot();
});
```

Interaktion

Vordefinierte Events

```
const button = screen.getByText('Press me');
fireEvent.press(button);

fireEvent.changeText(input, 'new Text');

fireEvent.scroll(scrollView, {
  nativeEvent: {
    contentOffset: {
      y: 200,
    },
  },
});
```

Beliebige Events

```
fireEvent(button, 'pressIn');
fireEvent(button, 'longPress');
```

Mocking und Interaktion

Erstellung eines Objektes, das den Funktionsumfang von realen Objekten nachahmt.

```
const {data: products, status, retry} = useFetch<ProductI[]>('https://api.example.com/products');
```

```
import * as useFetch from './useFetch';
it('should be able to retry a failed api call', () => {
    const retry = jest.fn(); // mocked function

    // mocked return value of useFetch
    jest.spyOn(useFetch, 'default').mockReturnValueOnce({
        status: useFetch.FetchStatus.Failed,
        retry,
    });

    render(<ProductList />);

    const retryButton = screen.getByText('Retry');
    expect(retryButton).toBeOnTheScreen();
    fireEvent.press(retryButton);
    expect(retry).toBeCalledTimes(1);
});
```

Zusammenfassung

Tests schreiben

- Dateiname: `myComponent.test.ts`
- `test` / `it`
- Gruppieren: `describe`
- `expect().to...`

Render Tests

- `render(<Component/>)`
- Snapshot für komplexen Output

Mocking

Nachahmung des Funktionsumfangs realer Objekte: `jest.fn()`, `jest.spyOn()`

Interaktion

- Button press: `fireEvent.press`
- TextInput: `fireEvent.changeText`
- Scroll: `fireEvent.scroll`

Quellen

Webseiten:

Jest: jestjs.io
Jest Native: github.com/testing-library/jest-native
React Native Testing Library: callstack.github.io/react-native-testing-library/
React Native: reactnative.dev/docs/testing-overview