FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Možnosti testování *React Native* aplikací

KIV/ASWI a KIV/TSP1 – Týmový projekt

Vypracoval: Tomáš Zikmund (*One team to rule them all*)
Datum: 11. 5. 2023

Obsah

Úvod 3
Možnosti testování 3
Manuální testování 3
Testování uživatelského rozhraní (UI) 4
Testování funkcionality aplikace 4
Testování výkonu aplikace 4
Testování přechodů mezi obrazovkami 4
Unit Testing 4
Jest 4
Enzyme 4
React Native Testing Library 4
React Test Renderer 5
UI Testing 5
Detox 5
Appium 5
TestCafe 5
Testování integrace 6
Testování výkonu 6
Základní *cookbook* pro testování 7
Použití frameworku JEST pro testování 8
Zpracovávané zdroje 11

Úvod

Testování aplikací vytvořených v React Native může být provedeno pomocí různých nástrojů a technik. Níže jsou uvedeny některé z nejčastějších metod, které lze použít k účinnému testování aplikací vytvořených v React Native.

1. **Manuální testování** – Manuální testování je tradičním způsobem, jak otestovat aplikace vytvořené v React Native. Tento proces zahrnuje ruční ověření aplikace pomocí různých scénářů, které zahrnují různé funkce a vlastnosti aplikace. Tento přístup je užitečný pro ověření funkcionality a uživatelského rozhraní aplikace.

2. **Testování jednotek (Unit testing)** - Testování jednotek je proces ověřování jednotlivých komponent a modulů aplikace. V React Native může být tento proces proveden pomocí knihoven jako *Jest* nebo *Mocha*. Tento přístup je užitečný pro ověření správné funkce kódu a rychlé nalezení chyb.

3. **Testování uživatelského rozhraní (UI testing)** - Testování uživatelského rozhraní je proces ověřování vizuálního vzhledu a chování aplikace. V React Native může být tento proces proveden pomocí knihoven jako *Detox* nebo *Appium*. Tento přístup je užitečný pro ověření, zda uživatelské rozhraní funguje správně a odpovídá návrhu.

4. **Testování integrace** – Testování integrace je proces ověřování, zda všechny komponenty aplikace spolupracují tak, jak by měly. V React Native může být tento proces proveden pomocí knihoven jako Enzyme nebo React Testing Library. Tento přístup je užitečný pro ověření, zda komponenty aplikace spolupracují správně a data se správně předávají mezi nimi.

5. **Testování výkonu** – Testování výkonu je proces ověřování, jak rychle a efektivně aplikace pracuje při zatížení. V React Native může být tento proces proveden pomocí nástrojů jako jsou například React Native Performance nebo Detox. Tento přístup je užitečný pro ověření, zda aplikace pracuje dostatečně rychle a efektivně při různých zátěžových situacích.

Kombinace různých přístupů k testování aplikací vytvořených v React Native může zvýšit účinnost a úspěšnost testování. Například můžeme použít manuální testování pro ověření uživatelského rozhraní a základní funkčnosti aplikace. Testování jednotek může být použito pro testování samostatných částí kódu a rychlé nalezení chyb. Testování uživatelského rozhraní může být použito pro ověření vizuálního vzhledu a chování aplikace. Testování integrace může být použito pro ověření, zda všechny komponenty aplikace spolupracují správně. Testování výkonu může být použito pro ověření, jak rychle a efektivně aplikace pracuje při zatížení.

Důležité je také implementovat testy již během vývoje aplikace a průběžně je aktualizovat. To umožní rychleji odhalit a opravit chyby a zlepšit kvalitu aplikace.

Kromě výše uvedených přístupů můžeme také využít automatizované testování pomocí různých nástrojů a knihoven, jako jsou například Selenium, Appium nebo Detox. Tyto nástroje umožňují rychle a efektivně ověřit funkčnost aplikace a ušetřit čas a úsilí při testování.

Kombinace různých přístupů a nástrojů k testování aplikací vytvořených v React Native může zlepšit úspěšnost a kvalitu testování a pomoci předejít chybám a problémům s aplikací.

Možnosti testování

Manuální testování

Manuální testování je proces testování aplikace ručně bez použití automatizovaných nástrojů. Pro manuální testování React Native aplikací existuje několik možností:

Testování uživatelského rozhraní (UI)

Testování uživatelského rozhraní se zaměřuje na ověření, zda aplikace vypadá a funguje, jak by měla. Testování uživatelského rozhraní lze provést ručně pomocí emulátorů nebo fyzických zařízení, na kterých bude aplikace spuštěna. Testující mohou provádět různé akce, jako jsou klepnutí na tlačítka, posunutí obrazovky nebo zadání textu a ověřit, zda aplikace reaguje tak, jak by měla.

Testování funkcionality aplikace

Testování funkcionality se zaměřuje na ověření, zda aplikace plní svůj účel a funguje tak, jak by měla. Testování funkcionality aplikace lze provádět ručně pomocí různých scénářů. Například, pokud se jedná o aplikaci pro nákupní seznam, testující mohou zadat seznam položek a ověřit, zda aplikace správně zobrazuje položky a umožňuje je odebrat.

Testování výkonu aplikace

Testování výkonu aplikace se zaměřuje na ověření, jak rychle a efektivně aplikace pracuje při zatížení. Testování výkonu aplikace lze provádět ručně pomocí různých scénářů, jako je například rychlé posouvání obrazovky nebo zadávání velkého množství dat. Testování výkonu lze také provádět s využitím externích nástrojů.

Testování přechodů mezi obrazovkami

Testování přechodů mezi obrazovkami se zaměřuje na ověření, zda přechody mezi různými obrazovkami v aplikaci fungují tak, jak by měly. Testování přechodů mezi obrazovkami lze provádět ručně pomocí různých scénářů, jako je například přechod mezi obrazovkou se seznamem položek a obrazovkou s detaily položek.

Pro manuální testování React Native aplikací je důležité mít k dispozici emulátory a fyzická zařízení, na kterých lze aplikaci testovat, a také seznam testovacích scénářů a požadavků na aplikaci.

Unit Testing

Testování jednotek (Unit testing) je proces testování jednotlivých částí kódu aplikace (tzv. jednotek), který umožňuje identifikovat chyby a problémy v rané fázi vývoje. Pro testování jednotek v React Native aplikacích existuje několik nástrojů a knihoven, které usnadňují proces testování.

Jest

Jest je nejpopulárnější nástroj pro testování React Native aplikací. Jest podporuje testování jednotek (*Unit testing*) i testování chování (*Behavior testing*) pomocí tzv. snapshot testů, které porovnávají výstupy kódu s předem definovanými očekávanými výsledky. Jest umožňuje i asynchronní testování pomocí tzv. *async*/*await* funkcí.

Enzyme

Enzyme je knihovna pro testování uživatelského rozhraní (UI testing) React Native aplikací. Enzyme umožňuje simulovat uživatelské akce, jako například klepnutí na tlačítko nebo zadání textu. Knihovna umožňuje také testovat stavy komponent a ověřovat, zda se komponenty správně aktualizují v závislosti na změně stavů.

React Native Testing Library

React Native Testing Library je další knihovna pro testování uživatelského rozhraní React Native aplikací. Knihovna se zaměřuje na testování chování uživatelského rozhraní a je navržena tak, aby testy co nejvíce simulovaly reálné uživatelské interakce s aplikací. Knihovna také umožňuje testovat přístupnost aplikace pro uživatele se zdravotním postižením.

React Test Renderer

React Test Renderer je nástroj pro testování React Native komponent bez nutnosti spouštět celou aplikaci. Nástroj umožňuje ověřit, jak se komponenty vykreslují a aktualizují v závislosti na změnách stavů.

Pro testování jednotek v React Native aplikacích je důležité mít důkladně napsané testy, které pokrývají co nejvíce funkcionalit aplikace. Testy by měly být spouštěny pravidelně během vývoje aplikace a měly by být součástí procesu integrace a nasazení aplikace.

UI Testing

Testování uživatelského rozhraní (UI testing) React Native aplikace se zaměřuje na ověření, zda uživatelské rozhraní aplikace funguje podle očekávání a zda splňuje požadovaná kritéria kvality a přístupnosti. Pro testování uživatelského rozhraní existuje několik nástrojů a knihoven, které umožňují simulovat uživatelské akce a ověřit správné fungování komponent.

Detox

Detox je nejpopulárnější nástroj pro testování uživatelského rozhraní React Native aplikací. Detox umožňuje simulovat uživatelské akce, jako například klepnutí na tlačítko, zadání textu nebo posouvání obrazovky. Nástroj také umožňuje ověřit správné zobrazení uživatelského rozhraní v závislosti na změně stavů a interakcí uživatele s aplikací.

Appium

Appium je open-source nástroj pro testování mobilních aplikací, který podporuje testování uživatelského rozhraní React Native aplikací. Appium umožňuje simulovat uživatelské akce, jako například klepnutí na tlačítko, zadání textu nebo posouvání obrazovky. Nástroj také umožňuje ověřit správné zobrazení uživatelského rozhraní v závislosti na změně stavů a interakcí uživatele s aplikací.

TestCafe

TestCafe je nástroj pro testování webových a mobilních aplikací, který podporuje testování uživatelského rozhraní React Native aplikací. TestCafe umožňuje simulovat uživatelské akce a ověřit správné zobrazení uživatelského rozhraní v závislosti na změně stavů a interakcí uživatele s aplikací. Nástroj také umožňuje testovat přístupnost aplikace pro uživatele se zdravotním postižením.

Pro účinné testování uživatelského rozhraní je důležité mít vytvořené scénáře, které pokrývají co nejvíce funkcionalit aplikace a simulují různé scénáře použití aplikace. Testy by měly být spouštěny pravidelně během vývoje aplikace a měly by být součástí procesu integrace a nasazování aplikace. To zajišťuje, že chyby v uživatelském rozhraní jsou odhaleny co nejdříve a mohou být opraveny ještě předtím, než jsou vydány do produkce.

Další faktor, který ovlivňuje účinnost testování uživatelského rozhraní, je vytvoření stabilního prostředí pro testování. To znamená, že prostředí pro testování by mělo být co nejvíce podobné prostředí, ve kterém bude aplikace vydána do produkce. K tomu může být použita simulace konkrétních hardwarových a softwarových konfigurací nebo virtualizace operačního systému.

Je také důležité vytvořit dobře organizovanou sadu testů, která pokrývá různé scénáře použití a zajistí, že aplikace splňuje požadovaná kritéria kvality a přístupnosti. Testy by měly být psány s ohledem na opětovné použití a údržbu a měly by být spouštěny automaticky jako součást procesu integrace a nasazování.

Testování uživatelského rozhraní je důležitou součástí procesu vývoje React Native aplikací. Použití nástrojů jako Detox, Appium nebo TestCafe umožňuje efektivní testování uživatelského rozhraní a umožňuje odhalit chyby a nedostatky v aplikaci co nejdříve.

Testování integrace

Testování integrace React Native aplikace je proces, při kterém se testují interakce mezi různými komponentami a moduly aplikace a zajišťuje se, že aplikace pracuje správně jako celek. Tento typ testování se často nazývá také jako End-to-End (E2E) testování, protože se zaměřuje na testování aplikace jako celek, od uživatelského rozhraní až po databázi a serverovou část.

Pro testování integrace se často používají nástroje jako *Cypress* nebo *Selenium*, které umožňují psát testy pomocí jazyků jako JavaScript nebo TypeScript. Tyto nástroje umožňují simulovat různé scénáře použití aplikace a ověřit, zda jsou všechny části aplikace integrovány správně a komunikují mezi sebou bez chyb.

Při testování integrace je důležité mít vytvořené testovací scénáře, které pokrývají co nejvíce funkcionalit aplikace a simulují různé scénáře použití. Tyto scénáře by měly být spouštěny pravidelně během vývoje aplikace a měly by být součástí procesu integrace a nasazování aplikace. To zajišťuje, že chyby v integraci aplikace jsou odhaleny co nejdříve a mohou být opraveny ještě předtím, než jsou vydány do produkce.

Je také důležité vytvořit stabilní prostředí pro testování integrace, které by mělo být co nejvíce podobné prostředí, ve kterém bude aplikace vydána do produkce. K tomu může být použita simulace konkrétních hardwarových a softwarových konfigurací nebo virtualizace operačního systému.

Testování integrace je důležitou součástí procesu vývoje React Native aplikací a umožňuje zajištění kvality a spolehlivosti aplikace. Použití nástrojů jako Cypress nebo Selenium umožňuje efektivní testování integrace a umožňuje odhalit chyby a nedostatky v aplikaci co nejdříve.

Testování výkonu

Testování výkonu React Native aplikace je proces, při kterém se testuje, jak dobře aplikace pracuje při různých zátěžových situacích a zajišťuje se, že aplikace pracuje dostatečně rychle a efektivně, aby uspokojila potřeby uživatelů. Testování výkonu lze rozdělit do několika kategorií, například testování odezvy uživatelského rozhraní, testování odezvy sítě nebo testování využití paměti.

Pro testování výkonu se obvykle používají specializované nástroje jako například *Apache JMeter* nebo *LoadRunner*, které umožňují simulovat různé scénáře zátěže a sledovat, jak se aplikace chová při různých zátěžových situacích. Tyto nástroje umožňují měřit rychlost odezvy aplikace, počet transakcí za sekundu, využití paměti, využití sítě a další klíčové metriky výkonu.

Při testování výkonu je důležité mít k dispozici reálná data a scénáře použití, aby se co nejvíce přiblížilo skutečným podmínkám, ve kterých bude aplikace používána. Testy by měly být spouštěny na různých zařízeních a platformách, aby bylo zajištěno, že aplikace pracuje správně v různých prostředích.

Další důležitou součástí testování výkonu je sledování výsledků testů a identifikace klíčových oblastí, které mohou být výkonovými hledisky problematické. To umožňuje vývojářům včas zareagovat na případné nedostatky a optimalizovat aplikaci pro lepší výkon.

Proces testování výkonu je důležitou součástí procesu vývoje React Native aplikací a umožňuje zajištění, že aplikace pracuje dostatečně rychle a efektivně, aby uspokojila potřeby uživatelů. Použití specializovaných nástrojů jako Apache JMeter nebo LoadRunner umožňuje efektivní testování výkonu a umožňuje identifikovat klíčové oblasti, které je třeba optimalizovat pro lepší výkon.

Základní *cookbook* pro testování

Testování mobilních aplikací v React Native může být velmi rozmanité a závisí na konkrétních požadavcích a specifikách dané aplikace. Nicméně, následující "kuchařka" poskytuje základní přehled kroků pro manuální testování mobilních aplikací v React Native:

1. Definujte testovací scénáře: Vytvořte seznam klíčových funkcí, které chcete otestovat. Zamyslete se nad funkcemi, které jsou pro uživatele nejdůležitější. Pro každou funkci vytvořte testovací scénář, který přesně popisuje, jak tuto funkci otestovat.

2. Připravte testovací zařízení: Připravte si testovací zařízení, na kterém budete aplikaci testovat. Měli byste testovat na více zařízeních a různých operačních systémech, aby se vám podařilo odhalit všechny chyby a zda aplikace funguje správně na všech typech zařízení. Měli byste testovat jak na zařízeních s nižším výkonem, tak i na těch výkonnějších.

3. Ověřte základní funkčnost: Ověřte, zda aplikace operuje bez chyb a spouští se správně. Ověřte, zda jsou všechny základní funkce dostupné a fungují správně (testování navigace v aplikaci, ověření funkčnosti tlačítek, vyplnění formulářů a odeslání dat). Vyzkoušejte také aplikaci při různých podmínkách (např. při špatném připojení k internetu).

4. Testujte všechny součásti aplikace, včetně modulů pro přihlášení, správu profilu a nastavení, zobrazování dat a také integraci s externími API.

5. Testujte jednotlivé funkce: Postupně procházejte jednotlivé testovací scénáře a ověřujte správnou funkčnost každého kroku. Zaznamenávejte všechny nalezené chyby a zaznamenejte si je do bug trackingového systému.

6. Testujte s různými daty: Ověřte, jak aplikace funguje s různými typy dat (např. různé velikosti obrázků, různé množství dat v seznamu atd.).

7. Testujte chování aplikace při různých a neočekávaných situacích: Vyzkoušejte, jak aplikace reaguje na neočekávané situace, jako jsou chyby v síti, špatné zadání uživatele nebo neočekávaný vstup dat.

8. Testujte uživatelské rozhraní: Ověřte, zda uživatelské rozhraní aplikace je intuitivní a snadno použitelné. Ověřte také, zda jsou všechny prvky uživatelského rozhraní funkční a reagují na uživatelské vstupy.

9. Otestujte kompatibilitu aplikace na různých zařízeních a operačních systémech. Můžete použít emulátory nebo reálná zařízení. Je důležité zkontrolovat, zda aplikace správně funguje na různých velikostech obrazovky, v různých orientacích a na různých verzích operačních systémů.

10. Pokud aplikace využívá hardwarové senzory (například GPS), otestujte jejich správnou funkčnost a přesnost.

11. Testování výkonu: Prověřte výkon aplikace, zda zvládá správně reagovat na uživatelské interakce a provádět požadované úkoly rychle a efektivně. Můžete použít nástroje pro měření rychlosti a výkonu aplikace, jako jsou například *Profiler* a *Systrace*. Dále pak *React Native Performance Monitor* nebo *Expo DevTools*. Tyto nástroje umožňují sledovat využití procesoru a paměti.

12. Testování bezpečnosti: Zkontrolujte, zda aplikace správně pracuje s uživatelskými daty a chrání je před neoprávněným přístupem. Měli byste také testovat aplikaci na přítomnost možných zranitelností a bezpečnostních chyb.

13. Testování použitelnosti: Otestujte, zda je aplikace intuitivní a snadno ovladatelná pro uživatele. Můžete použít uživatelské testování, dotazníky nebo jiné metody, abyste získali zpětnou vazbu od uživatelů a zlepšili uživatelskou zkušenost.

14. Testování aktualizací: Otestujte, zda aktualizace aplikace správně fungují a nezpůsobují žádné problémy. Měli byste také zkontrolovat, zda aktualizace neovlivní uživatelská data a nastavení.

15. Testování konektivity: Otestujte, jak aplikace pracuje v různých typech sítí, jako jsou například Wi-Fi, mobilní data a bez připojení k internetu. Měli byste také zkontrolovat, jak aplikace reaguje na přerušení a obnovení připojení k síti.

16. Testování různých scénářů použití (*use-cases*): Otestujte různé scénáře použití aplikace, aby bylo zajištěno, že aplikace funguje správně v různých situacích (například při změně jazyka nebo při změně nastavení). Můžete použít příklady scénářů, které jste vytvořili při tvorbě use-case scénářů.

17. Po dokončení manuálního testování můžete použít také automatizované testování pomocí nástrojů jako jsou Jest nebo Appium. Tyto nástroje umožňují automatizovat testování různých scénářů a zrychlit tak proces testování.

Při testování mobilní aplikace v React Native je důležité disponovat různými zařízení a operační systémy pro ověření kompatibility. Také bychom měli mít k dispozici nástroje pro měření výkonu a využití paměti a procesoru aplikace.

Měli bychom také mít na paměti, že testování mobilní aplikace v React Native je proces, který by měl být opakován pravidelně a v průběhu celého vývojového cyklu. To vám umožní odhalit a opravit chyby co nejdříve a zaručit kvalitu aplikace pro uživatele. Kromě manuálního testování bychom také měli zvažovat automatizované testování pomocí nástrojů jako Jest, Detox nebo Appium. Tyto nástroje vám umožní automatizovat testování a snížit tak čas a náklady na testování aplikace.

Testování mobilních aplikací v React Native není omezeno na pouhé manuální testování, ale může být také automatizováno pomocí různých nástrojů, jako jsou Jest, Appium nebo React Native Testing Library. Tyto nástroje umožňují snadné a efektivní testování různých scénářů a usnadňují proces testování.

Použití frameworku JEST pro testování

Pro testování se nejvíce na základě předcházející analýzy hodí testovací framework **Jest**. Jest se často používá pro testování React Native aplikací, včetně aplikací postavených na Native Base a spouštěných v Dev Expo. Jest podporuje testování React Native komponent a funkcionality, jako jsou například ***Redux*** akce a ***reducery***, ***AsyncStorage*** a další. Jest umožňuje psát testy pro React Native aplikace pomocí stejných metod a API jako pro standardní webové aplikace napsané v Reactu, takže jeho použití pro testování Native Base aplikací je velmi přirozené. Pro použití Jest pro psaní testů je nejprve třeba nainstalovat Jest a jeho závislosti v projektu pomocí package manageru jako je například ***npm***. Poté lze vytvořit soubor s testy, které budou testovat jednotlivé části kódu aplikace.

Jest má mnoho funkcí a metod, které lze použít pro psaní testů, včetně asercí pro porovnávání výstupů funkcí, mockování objektů a funkcí a řízení toku testování pomocí tzv. **describe** a **beforeEach** bloků.

Pro zjednodušení tvorby testů může být užitečné použít IDE s podporou pro Jest, jako například *Visual Studio Code*, ***WebStorm*** nebo *Atom* s rozšířením Jest. Tyto editory nabízejí různé funkce jako například možnost spouštění testů přímo v editoru, zvýrazňování syntaxe a automatické doplňování kódu.

Níže je uveden návod obecnějšího rázu, jak lze pro testování React Native aplikací vytvořit automatizované testy. Testovací postup je pouze obecný a bude se lišit nepatrně lišit v závislosti na konkrétní aplikaci.

1. Nainstalujte si Expo CLI a vytvořte nový projekt v Dev Expo.
2. Nainstalujte si testingový framework, jako je například Jest, pomocí příkazu npm install --save-dev jest.
3. Vytvořte soubor s testy v adresáři projektu. Například můžete vytvořit soubor s názvem App.test.js, kde App je název testované aplikace.
4. Napište testovací kód pro funkce aplikace. Pro aplikaci napsanou v Native Base můžete testovat například UI prvky, jako jsou tlačítka, textová pole nebo obrázky.
5. Nastavte konfiguraci Jestu v souboru package.json. Nastavte testovací soubor a definujte potřebné moduly a rozšíření.

Testovací JSON pak může vypadat následujícím způsobem:

```
{
  "scripts": {
    "test": "jest"
  },
  "jest": {
    "preset": "react-native",
    "transformIgnorePatterns": [
      "node_modules/(?!(react-native|native-base-shoutem-theme|native-base|react-
navigation|expo|@expo|@expo/vector-icons|react-native-easy-grid|@unimodules))",
    ],
    "globals": {
      "__DEV__": true
    }
  }
}
```

Pro spuštění testu se provede následující posloupnost kroků:

6. Spusťte testy příkazem *npm test.*
7. Pro zobrazení výsledků testů můžete použít některý z testingových reportů, jako je například **Jest HTML Reporter**.
8. Opakujte předcházející kroky 4 až 7 pro všechny části aplikace, které chcete testovat.

Testy v rámci Jest testovacího frameworku se píší v jazyce JavaScript, konkrétně v syntaxi *ES6* nebo novější. Tento framework je napsán v JavaScriptu a je určen pro testování aplikací napsaných v jazyce JavaScript, takže je přirozené, že testy se píší v tomto jazyce. V rámci testů můžete využívat standardní syntaxi jazyka JavaScript, jako jsou například funkce, proměnné, podmínky, cykly, operátory atd. Jest také poskytuje mnoho vestavěných funkcí a metod pro psaní testů, které usnadňují práci s testy a zvyšují jejich efektivitu.

Uvedeme si ještě jednu ukázku zdrojového souboru s testy pro Jest, který by bylo možné umístit do adresáře projektu:

```
// importy závislostí a modulů, které budeme potřebovat
import { render, fireEvent } from '@testing-library/react-native';
import React from 'react';

// importování komponenty, kterou budeme testovat
import MyComponent from '../src/components/MyComponent';

// popis testovací sady
```

```
describe('MyComponent', () => {
  // popis jednotlivých testů
  test('renders correctly', () => {
    const { getByTestId } = render(<MyComponent />);
    const component = getByTestId('my-component');
    expect(component).toBeDefined();
  });

  test('button click updates state', () => {
    const { getByTestId } = render(<MyComponent />);
    const button = getByTestId('my-component-button');
    fireEvent.press(button);
    const message = getByTestId('my-component-message');
    expect(message.props.children).toEqual('Button was pressed');
  });
});
```

V této ukázce jsou dva testy pro komponentu **MyComponent**. První test ověřuje, že komponenta se zobrazí správně. Druhý test simuluje kliknutí na tlačítko v komponentě a ověřuje, že se stav komponenty změní podle očekávání. Samozřejmě se soubory s testy mohou v závislosti na potřebách projektu lišit.

Byly použity informace z následujícího zdroje, dedikovaného testování v Dev Expo za použití Jest: Testing with Jest. Který pojednává o tom, jak nastavit a nakonfigurovat balíček ***jest-expo*** pro psaní unit testů a snapshot testů projektu. Bodová anotace článku vypadá následovně:

1. Jest je populární JavaScriptový testovací framework, který se používá pro testování React Native aplikací, včetně aplikací postavených na Native Base a spouštěných na Dev Expo.
2. Pro spuštění testů v Dev Expo se používá nástroj Expo CLI, který poskytuje sadu příkazů pro spouštění, testování a vývoj aplikací.
3. Pro psaní testů v Jest se používají funkce jako describe() a it() pro popis testovacích případů a očekávaných výsledků.
4. Jest podporuje různé typy testů, včetně testů jednotkových funkcí, testů snapshotů a testů integračních scénářů.
5. Jest lze snadno integrovat s dalšími nástroji, jako je například Enzyme pro testování React komponent, nebo React Native Testing Library pro testování uživatelského rozhraní.
6. Pro psaní testů v Jest je možné použít různá IDE, včetně Visual Studio Code, Atom, nebo WebStorm, která poskytují funkce jako syntax *highlighting*, *autocompletion* a *debugging*.
7. Expo CLI poskytuje několik příkazů pro spouštění testů v Jest, včetně expo test, expo test:android, a expo test:ios, které umožňují spouštět testy na emulátorech nebo fyzických zařízeních.
8. Expo CLI také umožňuje konfigurovat Jest pomocí souboru jest.config.js, kde lze nastavit různá nastavení, včetně nastavení transformace kódu pomocí *babelu*, nastavení modulu pro testování, a nastavení reporteru pro výstup testů.

Oficiální tutoriál pro testování React aplikací

Následuje stručný popis obsahu webové stránky "***Testing React Apps***" dostupné na adrese: https://jestjs.io/docs/tutorial-react#setup.

Tato webová stránka poskytuje podrobný návod a tutoriál pro používání Jest testovacího frameworku při testování React aplikací. Zahrnuje následující klíčové informace: Setup (konfigurace a nastavení JEST frameworku), testování snapshotů, mockování modulů, DOM testování, Custom transformers apod.

Stránka obsahuje také odkazy na další zdroje a dokumentaci Jestu, které mohou být užitečné pro další prozkoumání a hlubší porozumění Jest testování v kontextu React aplikací.

Další zpracovávané zdroje

- Broker, D. (2022, July 25). *5 Popular Automation Tools Used to Test React Native Apps*. Www.headspin.io. https://www.headspin.io/blog/5-popular-test-automation-tools-for-react-native-apps

Ve článku je prezentováno pět nejpopulárnějších nástrojů pro automatizované testování React Native aplikací. Autor článku popisuje výhody a nevýhody každého z nástrojů a zároveň uvádí i požadavky a další relevantní informace.

Prvním nástrojem, který je popsán, je *Appium*. Tento nástroj podporuje automatizované testování pro mnoho platforem, včetně React Native. K jeho výhodám se řadí podpora mnoha jazyků a možnost použití pro testování v reálném prostředí.

Dalším nástrojem je *Detox*, který byl speciálně navržen pro testování React Native aplikací. Výhodami jsou rychlost a stabilita, díky kterým je vhodný pro testování velkých aplikací.

Třetím nástrojem je *Selenium*. Tento nástroj je velmi rozšířený a podporuje mnoho jazyků a platforem. Jeho výhodou je snadná použitelnost, ale na druhé straně se může stát, že nemusí být optimální pro testování React Native aplikací.

Čtvrtým nástrojem je *Jest*, který byl vytvořen společností Facebook pro testování aplikací napsaných v React Native. Jedná se o jednoduchý a snadno použitelný nástroj, který poskytuje dobrou výchozí konfiguraci pro testování.

Posledním nástrojem, který je v článku popsán, je *Testim.io*. Tento nástroj je vhodný pro testování mobilních aplikací, včetně React Native aplikací. K jeho výhodám se řadí možnost použití na cloudu a podpora pro všechny hlavní mobilní platformy.

Autor článku v závěru shrnuje výhody a nevýhody jednotlivých nástrojů a doporučuje, aby si každý vývojář vybral ten, který nejlépe vyhovuje jeho potřebám a požadavkům.

- *Testing · React Native*. (2023, January 12). Reactnative.dev. https://reactnative.dev/docs/testing-overview

Článek na téma testování v React Native popisuje, jak lze testovat aplikace vytvořené pomocí React Native frameworku. Testování je důležitou součástí vývojového procesu, protože umožňuje zkontrolovat, zda aplikace funguje správně a splňuje očekávání uživatelů.

V článku se popisují různé druhy testování, včetně unit testů, integračních testů a end-to-end testů. Unit testy se zaměřují na testování jednotlivých funkcí, zatímco integrační testy testují interakce mezi různými částmi aplikace. End-to-end testy simuluje uživatelské chování a testují celou aplikaci jako celek.

Další část článku popisuje nástroje a frameworky pro testování v React Native, jako jsou Jest a Detox. Jest je framework pro psaní unit testů, zatímco Detox je framework pro psaní end-to-end testů.

Nakonec článek popisuje některé nejlepší praktiky pro testování v React Native, včetně psaní testů při vývoji aplikace, používání falešných dat pro testování a automatizace testování pomocí nástrojů jako je CI/CD.

Celkově je testování důležitou součástí vývojového procesu v React Native a tento článek poskytuje užitečné informace a nástroje pro testování aplikací v tomto frameworku.

- Bekkhus, S. (2023, March 6). *Testing React Native Apps · Jest*. Jestjs.io. https://jestjs.io/docs/tutorial-react-native

Jest je nástroj na testování kódu v React Native aplikacích. Používá se k psaní testů pro React komponenty, Redux a další aspekty React Native aplikací. Jeho hlavními výhodami jsou snadná instalace a konfigurace, rychlost a vynikající dokumentace.

Jest podporuje testování pomocí různých typů testů, jako jsou unit testy, snapshot testy a end-to-end (E2E) testy. Unit testy ověřují, zda jednotlivé komponenty fungují správně. Snapshot testy kontrolují, zda se zobrazující se komponenty nezměnily. E2E testy pak zajišťují, že aplikace funguje jako celek.

Jest také obsahuje mnoho užitečných funkcí, jako jsou automatické zjišťování změn, mockování a asynchronní testování. Všechny tyto funkce pomáhají ušetřit čas a snižují nároky na testování aplikací.

Jest se stal populárním nástrojem pro testování React Native aplikací kvůli své jednoduchosti a spolehlivosti. Díky svému rozsáhlému ekosystému a aktivnímu komunitnímu vývoji je Jest vhodným nástrojem pro většinu projektů React Native.

Jest lze použít k testování kódu napsaného v React Native pomocí knihovny react-test-renderer. Pomocí tohoto nástroje můžete testovat chování vaší aplikace a ověřit, zda komponenty fungují správně.

V článku se také dozvíte, jak nastavit Jest pro testování React Native aplikací a jak psát testy pro různé typy komponent, včetně stavových a bez-stavových komponent.

V závěru článku se autor zamýšlí nad tím, jaké jsou výhody použití Jest pro testování React Native aplikací, včetně rychlosti, snadnosti použití a široké podpory komunity.

- Helppi, V.-V. (2016, January 29). *Testing React Native Apps on Android and iOS*. SmartBear.com. https://smartbear.com/blog/testing-react-native-apps-on-android-and-ios/

Článek na SmartBear blogu se zaměřuje na testování React Native aplikací na Androidu a iOS. Článek začíná tím, že vysvětluje, co je React Native a proč by se měla tato technologie používat pro vývoj mobilních aplikací. Poté popisuje základy testování React Native aplikací, včetně testování uživatelského rozhraní a testování logiky aplikace.

V další části článku jsou podrobně popsány nástroje pro testování React Native aplikací, včetně Jest, Appium a Detox. Článek porovnává tyto nástroje podle různých kritérií, jako je jednoduchost použití, rychlost testování a podpora funkcí jako testování gest a průchodů.

Další část článku se zaměřuje na praktické tipy pro testování React Native aplikací, včetně toho, jak vytvářet testovací skripty a jak se vypořádat s problémy jako je asynchronní kód a změny v uživatelském rozhraní.

Nakonec článek ukazuje, jak lze testování React Native aplikací integrovat do kontinuální integrace a jakým způsobem lze používat cloudová řešení pro testování aplikací na více zařízeních.

Celkově jde o užitečný a informativní článek pro vývojáře mobilních aplikací, kteří chtějí testovat své React Native aplikace.

- Joshi, M. (2023, February 3). *End-to-End React Native Testing*. BrowserStack. https://www.browserstack.com/guide/end-to-end-testing-of-react-native-apps

Tento článek pojednává o tom, jak provádět end-to-end (E2E) testování React Native aplikací. E2E testování je způsob, jakým se ověřuje, že aplikace funguje správně v reálném světě, když interaguje s dalšími systémy a službami.

Článek shrnuje několik důležitých kroků pro provádění E2E testování React Native aplikací, jako je vytvoření testovacích scénářů, integrace s nástroji pro automatizaci testů, vytvoření a udržování testovacích prostředí a monitorování výkonu aplikace.

Dále se článek zaměřuje na nástroje, které jsou k dispozici pro E2E testování React Native aplikací, jako je například Jest, Detox nebo Appium. Každý z těchto nástrojů má své výhody a nevýhody, a článek se také zabývá tím, jaký nástroj je nejvhodnější pro konkrétní aplikaci.

Nakonec článek poskytuje tipy pro úspěšné provádění E2E testování React Native aplikací, jako je například použití správných nástrojů a technologií, správné nastavení testovacích prostředí a důkladné monitorování výkonu aplikace.