SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Automatizované testovanie (unit, integračné, End-to-EnD A Výkonnostné testy)

Dokumentácia k semestrálnemu Zadaniu Predmetu Architektúra Softvérových systémov 2022/2023

Obsah

[Zoznam skratiek a značiek 2](#_Toc121876911)

[Úvod 1](#_Toc121876912)

[1 Technická špecifikácia 2](#_Toc121876913)

[1.1 Informácie o kóde softvéru 2](#_Toc121876914)

[1.2 Testovacie nástroje použité v ukážkach 2](#_Toc121876915)

[2 Unit testy 3](#_Toc121876916)

[3 Integračné testy 4](#_Toc121876917)

[4 End-to-end testy 5](#_Toc121876918)

[5 Výkonnostné testy 6](#_Toc121876919)

[Zoznam použitej literatúry 7](#_Toc121876920)

Zoznam skratiek a značiek

**Úvod**

Dokument súži ako návod pre realizáciu praktických ukážok z prezentácie na tému Automatizované testovanie (unit, integračné, end-to-end a výkonnostné testy), ktorá bola odprezentovaná autormi v rámci semestrálneho zadania na predmete Architektúra softvérových systémov.

Kapitola Technická špecifikácia obsahuje informácie o kóde demonštračného softvéru a zoznam použitých testovacích nástrojov. V ďalších častiach je dokumentácia delená do kapitol podľa typu vykonaných testov.

1. Technická špecifikácia
   1. Informácie o kóde softvéru

Použitý programovací jazyk: Java

Použité frameworky:

Kód je v čase publikácie tohto dokumentu dostupný na:

<https://github.com/j-puszter/testovanie>

* 1. Testovacie nástroje použité v ukážkach

Unit testy:

Integračné testy:

End-to-end testy

Výkonnostné testy: Jmeter

* 1. Demo aplikácia

Jednotlivé typy testov sme testovali na demo aplikácii. Fotky zdrojových kódov v tejto dokumentácii sú z tejto aplikácie. Na nasledujúcej fotke je vidieť štruktúra projektu a jeho rozdelenie na zdrojový kód aplikácie a testy.

Obrázok, na ktorom je text

Automaticky generovaný popis

1. Unit testy
   1. Teória

Unit testovanie je typ testovania softvéru, pri ktorom sa testujú jednotlivé jednotky alebo komponenty. Účelom je overiť, či každá jednotka softvérového kódu funguje podľa očakávania. Unit testovanie sa vykonáva počas vývoja (počas fázy kódovania) aplikácie vývojármi. Unit testy izolujú časť kódu a overujú jeho správnosť. Jednotkami softvéru považujeme individuálne funkcie, metódy, procedúry, moduly alebo objekty. Testovanie softvéru začína ešte pred dokončením aplikácie. Týmto spôsobom sú chyby odhalené skôr, ako sa stratia v kódoch.

Unit testovanie je prvou vrstvou celého testovacieho procesu, ktorým musí softvér prejsť pred jeho spustením a vydaním. Toto predbežné testovanie často vykonáva tím vývojárov alebo softvérový inžinier, ktorý napísal kód softvéru. Vyššia úroveň povedomia o zložitosti programu zvyšuje šance na vykonanie dôkladnej práce. Inžinieri zabezpečenia kvality sú tiež vyškolení na vykonávanie unit testov. Tieto testy je možné vykonávať manuálne, avšak zvyčajne sú automatizované. To zaisťuje že časti softvéru spĺňajú očakávania.

Unit testy tvoria základ, na ktorom sú postavené všetky ostatné testy. Ich presnosť a dôkladnosť je významným faktorom softvérového vývoja. Ovplyvňujú, ako dobre sa dajú vykonať ostatné testy a tiež výkon softvéru ako celku.

Unit testovanie je súčasťou Test Driven Development (TDD), metodiky, ktorá využíva opakované testovanie na vytváranie kvalitných produktov. Každá jednotka musí byť nezávislá od akéhokoľvek externého faktora alebo kódu, aby testeri mohli jasne interpretovať výsledky.

Vývojári softvéru sa niekedy snažia ušetriť čas pri minimálnom unit testovaní. Toto vedie k opačnému efektu. Nevhodné unit testovanie má za následky vysoké náklady na opravu defektov v neskorších fázach vývoja. Ak sa unit testovanie vykoná na začiatku vývoja, potom to v konečnom dôsledku šetrí čas a peniaze.

* + 1. Ciele unit testovania
* Overenie presnosti častí kódu
* Dosiahnutie samostatných a nezávislých častí kódu
* Identifikovanie chýb na začiatku vývoja softvéru
* Zvýšenie porozumenia kódu programátorom
* Jednoduchšie vykonávanie zmien v kóde
* Opakované používanie kódu
  + 1. Výhody unit testovania
* Vývojári, ktorí chcú zistiť, aké funkcie poskytuje jednotka a ako ju používať, sa môžu pozrieť na unit testy, aby získali jej základné pochopenie
* Unit testy umožňujú programátorovi neskôr zrefaktorovať kód a uistiť sa, že modul stále funguje správne (regresné testovanie). Postup spočíva v napísaní testovacích prípadov pre všetky funkcie a metódy, aby bolo možné kedykoľvek, keď zmena spôsobí poruchu túto poruchu rýchlo identifikovať a opraviť.
* Vzhľadom na modulárny charakter unit testov, môžeme testovať časti projektu bez toho, aby sme čakali na dokončenie ostatných častí.
  + 1. Nevýhody unit testovania
* Nedá sa očakávať, že unit testovanie zachytí každú chybu v programe. Nie je možné vyhodnotiť všetky cesty vykonávania programu ani v tých najtriviálnejších programoch
* Unit testovanie sa zo svojej podstaty zameriava na jednotky kódu. Preto nedokáže zachytiť chyby integrácie alebo rozsiahle systémové chyby.
  + 1. Testom riadený vývoj

Unit testovanie v TDD (test driven development) zahŕňa rozsiahle používanie testovacích frameworkov. Framework pre unit testy sa používa pre vytvorenie automatizovaných unit testov. Tieto frameworky nie sú jedinečné pre TDD, ale sú preň nevyhnutné.

Testom riadený vývoj obnáša:

* Testy sú napísané pred kódom
* Testy sa ťažko spoliehajú na frameworky
* Všetky triedy softvéru sú testované
* Je možná rýchla a jednoduchá integrácia
  + 1. Mockovanie objektov

Unit testovanie sa spolieha na vytváranie mockovaných objektov na testovanie častí kódu, ktoré ešte nie sú súčasťou kompletnej aplikácie. Mockované objekty dopĺňajú chýbajúce časti programu. Môžeme mať napríklad funkciu, ktorá potrebuje premenné alebo objekty, ktoré ešte nie sú vytvorené. Pri testovaní sa budú vytvárať vo forme mockovaných objektov vytvorených výlučne na účely testovania vykonaného v danej časti kódu.

* + 1. Praktiky pri unit testovaní
* Návrh vhodných názvov testov
* Vytvorenie jednoduchých testov
* Vytvorenie deterministických testy
* V teste sa venujeme jedinému prípadu použitia
* Zameranie sa na maximálne pokrytie testov
* Návrh testov aby boli čo najrýchlejšie
* Minimalizácia závislostí testov
* Automatizácia testov
  + 1. Známe nástroje pre unit testovanie
* JUnit – testovací framework pre programovací jazyk Java
* NUnit – testovací framework pre všetky .Net jazyky
* DBUnit – rozširenie JUnit, pre veľke databázov riadené projekty
* HTMLUnit – nástroj na testovanie, používa sa na testovanie web aplikácií
* PHPUnit – testovací framework pre programovací jazyk PHP
* SimpleTest – testovací framework pre programovací jazyk PHP
* Embunit – testovací nástroj pre programovací jazyk C a C++
  1. Príklad unit testovania v Jave

Pre ukážku unit testovania v jave použijem knižnicu JUnit 5. Je to najpoužívanejší java framework pre unit testovanie. Pravidlá spúšťania testov definujeme pomocou anotácií a samotné testy vyhodnocujeme pomocou funkcií z triedy Assertions.

* + 1. Najpoužívanejšie JUnit anotácie
* @Test – Táto anotácia označuje, že metóda je testovacia metóda.
* @ParametrizedTest - Parametrizované testy umožňujú spustiť test viackrát s rôznymi argumentami. Okrem toho je nutné deklarovať aspoň jeden zdroj, ktorý bude poskytovať argumenty pre každé volanie testu a potom tieto argumenty použiť v testovacej metóde.
* @ValueSource – Anotácia špecifikukuje zdroj argumentov pre parametrizované testy.
* @RepeatedTest - Zopakujte test špecifikovaním celkového počtu požadovaných opakovaní.
* @DisplayName - Testovacie triedy a testovacie metódy môžu deklarovať vlastné názvy, ktoré budú zobrazované test runnermi a v test reportoch.
* @BeforeEach - Označuje, že anotovaná metóda by sa mala vykonať pred každou testovacou metódou.
* @AfterEach - Označuje, že anotovaná metóda by sa mala vykonať po každej testovacej metóde.
* @BeforeAll - Táto anotácia označuje metódu, ktorá sa vykoná raz pred všetkými testami
* @AfterAll - Táto anotácia označuje metódu, ktorá sa vykoná raz po všetkých testoch
* @Tag - Túto anotáciu môžeme použiť na deklarovanie tagov pre testy či už na úrovni triedy alebo metódy. Testy potom môžeme pomocou nich filtrovať.
* @Disabled - Anotácia sa používa na zakázanie alebo preskočenie testov na úrovni triedy alebo metódy. Pri deklarácii na úrovni triedy sa preskočia všetky metódy @test. Keď použijeme anotáciu na úrovni metódy, preskočí sa iba anotovaná metóda.
  + 1. Najpoužívanejšie metódy z triedy Assertions
* assertTrue – Overí podmienku či je pravdivá
* assertFalse - Opačná metóda k assertTrue
* assertNull – Overí či hodnota je null
* assertNotNull – Opačná metóda k assertNull
* assertEquals – Overí, či dva parametre sú rovnaké
* assertNotEquals – Opačná metóda k assertEquals
* assertSame – Overí, či hodnoty dvoch parametrov referencujú rovnaký objekt
* assertNotSame – Opačná metóda k assertSame
* assertLinesMatch – Overí, či dva listy stringov sú rovnaké
* assertArrayEquals – Overí, či dve polia sú rovnaké
* assertIterableEquals – Overí, či dva iterable sú rovnaké
* assertThrows – Overí, či funkcia na vstupe vyhodí správnu výnimku
* assertDoesNotThrow – Opačná metóda k assertThrows
* assertTimeout – Overí, či vstupná funkcia skončí pred špecifikovaným timeoutom
* assertAll – Táto funkcia umožňuje vytvorenie skupinových asertions, kde sa spustí každá assertion a ich zlyhania sú hlásené spoločne.
* fail – táto metóda spôsobí zlyhanie testu
  + 1. Testovanie

V triede Calculator sa nachádzajú statické metódy, ktoré počítajú prvočísla. Niektoré metódy majú chybnú implementáciu, pre ukážku funkcionality testov.

Nasledujúce obrázky obsahujú vytvorené ukážkové testy a ich výsledky.

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text, elektronika, snímka obrazovky

Automaticky generovaný popis

1. Integračné testy

Cieľom integračných testov je overenie, či časti softvéru spolu fungujú. Tento typ testovania sa využíva hlavne pri väčších projektoch. Integračnými testami môžeme testovať napríklad endpointy aplikácie a porovnať očakávaný výstup.

3.1 Konfigurácia

@SpringBootTest – potrebný pre spustenie celej aplikácie, vytvorí ApplicationContext potrebný pre načítanie tried pomocou dependancy injection

@AutoConfigureMockMvc – Automatické nakonfigurovanie MockMvc



3.2 MockMvc

Hlavný vstupný bod pre komunikáciu s aplikáciou. Vďaka MockMvc môžeme volať requesty bez toho aby sa musel spúšťať server.



* 1. Testovanie

Endpointy voláme pomocou metódy perform v ktorej môžeme volať rôzne metódy a následne môžeme zadať čo očakávame, aký status by mal endpoint vrátiť, aké hlavičky by mal vrátiť, aké dáta má vrátiť a ďalšie parametre.

* + 1. Metódy

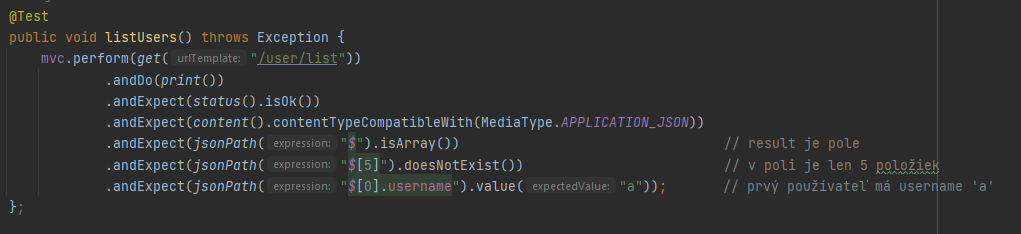
Metódy voláme pomocou triedy MockMvcRequestBuilder ktorý obsahuje statické metódy ako get a post na ktoré je možné následne zadať aký typ dát posielame a vložiť dané dáta.

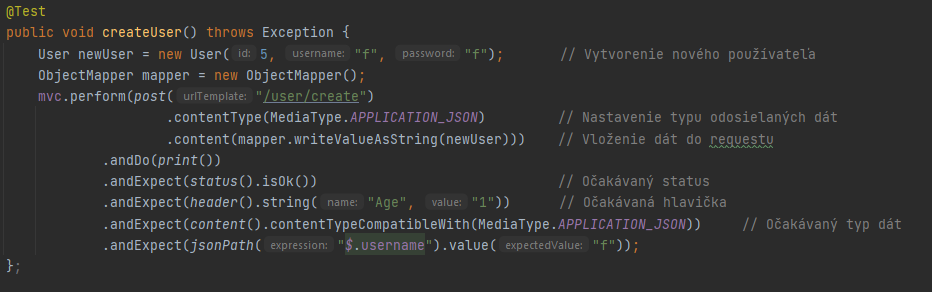
|  |  |
| --- | --- |
| GET |  |
| POST |  |

* + 1. Výstup

Pomocou triedy MockMvcResultHandler a jej statickej metódy print() test zobrazí všetky podrobnosti o requeste. Zároveň je možné porovnávať výsledky s očakávaným výsledkom pomocou triedy MockMvcResultMatchers a jej metód:

* status() – status odpovede
* content()
  + formu odpovede (typ dát, kódovanie...)
  + samotnú odpoveď - porovnanie základného stringu (string())
* header() – hlavičku odpovede
* jsonPath() – porovnanie konkrétnej hodnoty v json-e
  1. Testy





1. End-to-end testy
2. Výkonnostné testy

**Zoznam použitej literatúry**

<https://www.guru99.com/unit-testing-guide.html>

<https://www.spiceworks.com/tech/devops/articles/what-is-unit-testing/>

<https://devqa.io/junit-5-annotations/>

<https://www.appsdeveloperblog.com/an-overview-of-junit5-assertions-with-examples/>