Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/05BE36djuza

АУТОМАТСКО ТЕСТИРАЊЕ МИКРО СЕРВИСА АЛАТИМА ЗА ТЕСТИРАЊЕ ПРИХВАТЉИВОСТИ

AUTOMATIC TESTING OF MICRO SERVICES WITH TOOLS FOR ACCEPTANCE TESTING

Никола Ђуза, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *У овом раду је извршена анализа и поређење тренутно најпопуларнијих JavaScript алата за тестирање прихватљивости веб страница. За анализу алата је коришћен пројекат за продају међународног путног осигурања који се аутоматски тестира са свим алатима који су анализирани. Пројекат који се тестира поседује микро сервисну архитектуру, где се сваки сервис налази унутар свог контејнера. Рад улази у детаље писања и покретања аутоматизованих тестова прихватљивости уз алате, како у локалном окружењу, тако и у окружењима за континуалну интеграцију.*

Кључне речи: *Аутоматско тестирање, тестирање прихватљивости, микро сервисна архитектура*

Abstract – *This thesis analyzes and compares most popular JavaScript tools for acceptance testing. A project for selling travel insurance with automated tests has been used to analyze all the tools. Project that's being tested is micro service based and every service is inside its own container. Thesis goes into details on how to write and run tests using these tools, both in local environment and in continuous integration environments.*

Keywords: *Automated testing, acceptance testing, micro service architecture*

1. УВОД

Тестирање је важна активност у развоју софтвера која чини развој софтвера више поузданим. Тестирање добија на значају све више како расте вредност софтвера. Избегавање тестирања софтвера често доводи до грешака у софтверу које могу проузроковати велике новчане штете. Из овог разлога се у последње време не штеди на тестирању и активностима које оно подразумева.

Како комплексност и вредност софтвера расте, тако расту и начини на које се тај софтвер може тестирати. Поред мануелног тестирања, доста је заступљено аутоматско тестирање. Аутоматско тестирање помаже у одржавању поузданости и једноставности програмског кода јер се може покретати приликом сваке измене у изворном коду софтвера у процесу итеративног развоја софтвера.

НАПОМЕНА:
Овај рад је проистекао из мастер рада чији ментор је био др Бранко Милосављевић, ред. проф.

Аутоматски тестови често помажу у проналажењу грешака пре него што се то деси од стране корисника софтвера, и то у раним фазама развоја. Једна врста аутоматских тестова су интеграциони тестови који тестирају софтвер на сличан начин на који корисник треба да користи софтвер. Овај рад ће објаснити које активности и начини се могу користити како би се тестирао сет микро сервиса уз помоћ алата за тестирање прихватљивости. Такође ће фокус бити на аутоматским тестовима, како их писати и покретати, како у локалном, тако и у другим окружењима.

1.1 Шта је тестирање?

Тестирање се дефинише као активност којом се проверава да ли добијени резултат одговара очекиваном резултату и да ли је систем без грешака. Тестирањем се такође врши верификација софтвера и да ли он извршава оно за шта је дефинисан. Битно је напоменути да је тестирањем готово немогуће открити све грешке у систему [1]. Разлог за ово је јер не можемо предвидети све гране извршавања софтвера или постоји пропуст у дизајну софтвера. За разлику од физичких система, већина дефеката у софтверу се дешава због грешке у дизајну, а не у процесу израде.

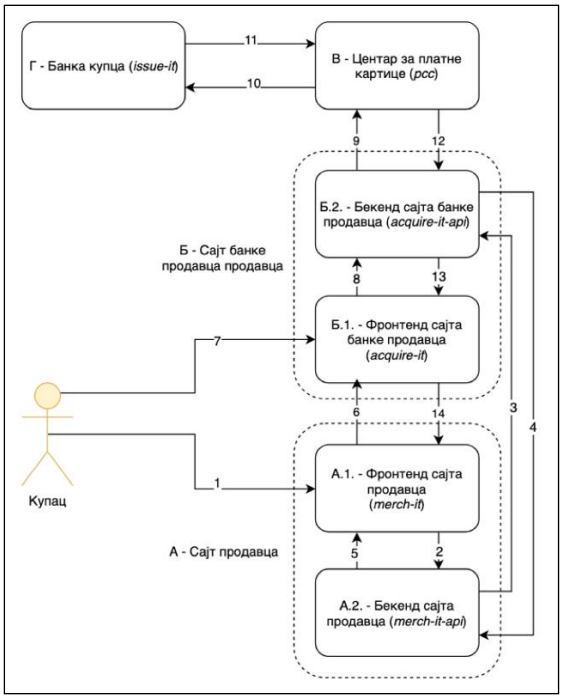
2. СПЕЦИФИКАЦИЈА ПРОЈЕКТА

У овом поглављу ће се описати пројекат над којим су тестирани алати за тестирање прихватљивости. У питању је пројекат са сложеном архитектуром микро сервиса. Пројекат је рађен у *Ruby* и *JavaScript* програмским језицима. Такође је коришћена *Docker* алат за креирање софтверских контејнера. За имплементацију корисничког интерфејса је коришћен *JavaScript* језик и *React* библиотека. За израду бекенд дела је коришћен *Ruby* програмски језик и *Ruby on Rails* фрејмворк. Као база података за чување података је коришћена *PostgreSQL* систем за управљање базама података.

2.1 Архитектура пројекта

Пројекат је сервис за продају путног осигурања корисницима. Сценарио који се разматра у слици 1. је следећи: Купац који има отворени рачун у банци Б (Банка купца) врши неку куповину међународног путног осигурања преко корисничко интерфејса код Продавца и плаћа платном картицом посредством веб апликације Банке продавца за наплату на интернету. Рачун продавца се налази у банци А (Банка продавца).

Комуникација између банке купца (банка Б) и банке продавца (банка А) врши се посредством Центра за платне картице. На слици 1. су приказани сви сервиси који постоје у овом пројекту, као и комуникација описана у претходном примеру куповине међународног осигурања.



Слика 1. *Дијаграм тока плаћања са рачуна из банке купца*

2.2 Контејнеризација уз помоћ *Docker* алата

У традиционалном развоју софтвера, програмски код и софтвер који је развијен у једном окружењу често зна да има доста дефеката кад се покрене на неком другом систему. Инжењери софтвера ово решавају тако што се софтвер покреће у контејнерима у облаку. Контејнери пружају логичко паковање у ком је софтвер на неки начин издвојен од окружења у ком заправо ради. Овакво раздвајање дозвољава апликацијама које су у контејнерима да се лакше лансирају на друго платформе, небитно да ли је у питању приватни дата центар, јавни облак, или чак на рачунару свог колеге. Идеја је да имплементација апликација не зависи од окружења у ком се налази.

Контејнери се често пореде са виртуалним машинама (енгл. *virtual machine – VM*). Виртуалне машине су гостујући оперативни систему као што је *Linux* или *Windows* који је покренут на домаћинском оперативном систему са контролисаним приступом хардверу домаћина. Слично виртуалним машинама, контејнери дозвољавају да се апликације запакују са свим библиотекама и осталим потребним софтвером, пружајући изоловано окружење за покретање саме апликације (сервиса).

3. АНАЛИЗА АЛАТА ЗА ТЕСТИРАЊЕ ПРИХВАТЉИВОСТИ

У овом поглављу ће бити коришћен претходно описан пројекат како би се анализирали најпопуларнији алати (често се називају и фрејмворци и библиотеке) за тестирање прихватљивости унутар *JavaScript* заједнице. Алати који се користе најчешће данас су *Cypress, Puppeteer, Nightwatch.js, TestCafe, CodeceptJS* [2] и њих ћемо анализирати и касније упоредити. Биће искоришћен претходно описани пројекат за продају осигурања како би се написали и покретали тестови прихватљивости тј. *e2e* тестови. Тестови ће бити покретани у локалном окружењу, као и у окружењу на два позната сервиса за континуалну интеграцију – *Semaphore CI* и *Travis CI.*

3.1 *Cypress*

Cypress је *JavaScript* фрејмворк за e2e тестирање. Тренутно је најпопуларнија библиотека за e2e тестирање [3]. Главна разлика у односу на конкуренцију је што не користи *Selenium* за тестирање. Већина библиотека користи *Selenium* и то је узрок њихових мана у већини случајева. Како *Selenium* покреће команде преко мреже и контактира претраживач споља, Cypress се налази унутар претраживача и одатле покреће тестове.

Главни фокус Cypress алата су *e2e* тестови и сав рад је усмерен на побољшавање *e2e* тестова. Да би ово омогућили, Cypress је направљен тако да може да се користи са било којом *JavaScript* библиотеком или фрејмворком за развој графичком интерфејса. Сви тестови који се пишу унутар Cypress фрејмворка морају да се пишу у *JavaScript* програмском језику. Разлог за ово је што се Cypress покреће и ради унутар претраживача, а претраживач најбоље извршава *JavaScript* код.

Велика предност је што кад се Cypress инсталира у оквиру пројекта као библиотека, долази упакован са свих потребним алатима за *e2e* тестирање. Ово драстично смањује време потребно за конфигурацију и инсталацију алата који већ долазе упаковани и спремни за рад од почетка. На слици 2. се налази пример теста написаног у Cypress фрејмворку.

```
describe('Paying for plans', () => {
  it('cheapest one', () => {
    cy.visit('http://localhost:3000')

    cy.contains('Welcome to merch-it')

    cy.get('[data-cy=email]').type('test@example.com')
    cy.get('[data-cy=10000]').click()

    cy.contains('Checkout')
    cy.get('[data-cy=submit]').click()

    cy.get('[data-cy=pan]').type('1')
    cy.get('[data-cy=securityCode]').type('1')
    cy.get('[data-cy=cardHolderName]').type('1')
    cy.get('[data-cy=expirationDate]').type('1')

    cy.get('[data-cy=pay]').click()

    cy.contains("Congrats, you've bought insurance from us!")
  })
})
```

Слика 2. *Пример Cypress тест фајла*

3.2 *Puppeteer*

Puppeteer је *Node.js* библиотека која пружа API високог нивоа за контролисање *Chrome* и *Chromium* преко *Chrome DevTools* протокола. Не плаћа се и *open source* је библиотека доступна на сајту *GitHub.*

Puppeteer се покреће без графичког интерфејса (енгл. *headless*) али се може подесити да ради са графичким интерфејсом у *Chrome* и *Chromium* претраживачима. Он је решење које користи сам *Chrome* претраживач при тестирању њихових функционалности.

Chrome DevTools протокол је сет алата за мерење, инспекцију, проналажење грешака у коду (енгл. *debugging*) и профилисање *Chromium* и *Chrome*. Пример теста написан у *Puppeteer* библитеци је дат на слици 3.

```
test("Balance is too low", async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto("http://localhost:3000");

  await page.type("[data-cy=email]", "test@example.com");
  await page.click("[data-cy='10000']");
  await page.click("[data-cy=submit]");

  await page.waitForSelector("[data-cy=pan]");
  await expect(page).toMatch("PAN");

  await page.type('[data-cy=pan]', '4');
  await page.type('[data-cy=securityCode]', '4');
  await page.type('[data-cy=cardHolderName]', '4');
  await page.type('[data-cy=expirationDate]', '4');

  await page.click('[data-cy=pay]');

  await page.waitForSelector("[data-cy=errorBox]");
  await expect(page)
    .toMatch("Oops! The transaction didn't go through, sorry.");

  await browser.close();
});
```

Слика 3. *Пример Puppeteer тест фајла*

3.3 *Nightwatch.js*

Nightwatch.js је решење за *e2e* тестирање базирано на *Node.js* библиотеци за претраживаче и веб сајтове. Бесплатан је и *open source*. За комуникацију са претраживачем користи моћан *W3C WebDriver API* како би извршавао команде и провере над елементима у *DOM* стаблу веб страница. *W3C WebDriver API* или скраћено *WebDriver* је интерфејс за даљинску контролу претраживача. Он пружа протокол који не зависи од платформе или програмског, већ дозвољава да се даљински диктира понашање веб претраживача [21]. Пример теста написан у *Nightwatch.js* библиотеци је дат на слици 4.

```
module.exports = {
  tags: ['insurance'],

  'Pay for first option' : function (browser) {
    browser
      .url('http://localhost:3000')
      .waitForElementVisible('body', 1000);

    browser.setValue("[data-cy=email]", "test@example.com")
      .click("[data-cy='10000']")
      .click("[data-cy=submit]")
      .waitForElementVisible("[data-cy=pan]")
      .setValue("[data-cy=pan]", '2')
      .setValue("[data-cy=securityCode]", '2')
      .setValue("[data-cy=cardHolderName]", '2')
      .setValue("[data-cy=expirationDate]", '2')
      .click("[data-cy=pay]")
      .assert.containsText("[data-cy=successBox]",
        "Congrats, you've bought insurance from us!")

    browser.end();
  }
};
```

Слика 4. *Пример Nightwatch.js тест фајла*

3.4 *TestCafe*

TestCafe је бесплатан и *open source Node.js* алат који служи за аутоматизовање *e2e* веб тестова. Тестови се могу писати у *JavaScript* или *TypeScript* програмским језицима. *TestCafe* ради на свим популарним оперативним системима као што су *Windows*, *Mac OS*, и *Linux*. Такође постоји подршка за тестирање мобилних, десктоп, удаљених (енгл. *remote*) и претраживача у облаку (енгл. *cloud*).

За покретање тестова није потребан ни *WebDriver* ни *Selenium*. *TestCafe* користи други приступ при контролисању претраживача, а то је *URL*-rewriting proxy механизам. Користећи овај механизам, акције корисника у тесту су емулиране кроз *DOM API* који је доступан преко драјвер скрипте која је додата путем *proxy*-ја на страницу. На овај начин, подешавања окружења за покретање *TestCafe* тестова је прилично једноставно. Све што је потребно је да се инсталира *TestCafe* пакет са npm регистра *JavaScript* пакета у пројекат. На слици 5 је приказан пример теста написан у *TestCafe* алату.

```
import { Selector } from 'testcafe';

fixture `Insurance`
  .page `http://localhost:3000`;

test('Balance is too low', async t => {
  await t
    .typeText('[data-cy=email]', 'test@example.com')
    .click("[data-cy='10000']")
    .click('[data-cy=submit]')

    .typeText('[data-cy=pan]', '4')
    .typeText('[data-cy=securityCode]', '4')
    .typeText('[data-cy=cardHolderName]', '4')
    .typeText('[data-cy=expirationDate]', '4')

    .click('[data-cy=pay]')
    .expect(Selector('[data-cy=errorBox]').innerText).contains(
      "Oops! The transaction didn't go through, sorry."
    );
});
```

Слика 5. *Пример TestCafe тест фајла*

3.5 *CodeceptJS*

CodeceptJS је фрејмворк за тестирање који користи друге популарне библиотеке за тестирање и додаје *API* високог нивоа који се користи за писање тестова и који повећава читљивост *e2e* тестова. Библиотеке које су укључене су:

- *Protractor* – користи се ова библиотека путем *WebDriver* протокола
- *Puppeteer* – користи се брзи *headless* режим са *Google Chrome* претраживачом за покретање тестова
- *Appium* – библиотека која се користи за тестирање мобилних уређаја
- *Nightmare* – користи *Electron* и *NightmareJS* за покретање тестова

CodeceptJS такође нуди опцију покретања тестова користећи *WebDriver* протокол кроз *webdriverio* библиотеку. Ово је фрејмворк који изолује и прави апстракт од набројаних библиотека, дозвољавајући кориснику да пише тестове у разумљивом језику. Ове библиотеке се зову помагачи (енгл. *helpers*). На слици 6 је приказан пример теста у *CodeceptJS* фрејмворку.

```
Feature('Pay for insurance');

Scenario('Pay for first option', (I) => {
  I.amOnPage('http://localhost:3000')
  I.fillField('[data-cy=email]', 'test@example.com')
  I.click("[data-cy='10000']")
  I.click('[data-cy=submit]')

  I.waitForText('PAN');

  I.fillField('[data-cy=pan]', '2')
  I.fillField('[data-cy=securityCode]', '2')
  I.fillField('[data-cy=cardHolderName]', '2')
  I.fillField('[data-cy=expirationDate]', '2')
  I.click('[data-cy=pay]')

  I.waitForText("Congrats, you've bought insurance from us!");
});
```

Слика 6. *Пример CodeceptJS тест фајла*

4. РЕЗУЛТАТИ АНАЛИЗЕ

На слици 7. су упоређени алати и четири категорије које их карактеришу. Прва категорија је лакоћа подешавања где је узето у обзир колико је лако инсталирати алат у локалном окружењу, али и на сервисима за континуалну интеграцију. Друга категорија је лакоћа писања тестова где се посматра колико је лако савладати синтаксу која долази уз алат, колико је она документована и општи утисак о писању тестова. Ове две категорије носи максимално 5 поена. У трећој категорији се гледа да ли алат подражава тестирање на више претраживача и носи 1 поен, а четврта категорија прати да ли се тестови могу паралелно покретати уз алат и такође носе 1 поен.

	Лакоћа подешавања	Лакоћа писања тестова	Подршка за више претраживача	Паралелно покретање тестова
Cypress	5/5	5/5	✗	✓
Puppeteer	4/5	3/5	✗	✗
Nightwatch.js	4/5	3/5	✓	✓
TestCafe	5/5	4/5	✓	✓
CodeceptJS	3/5	5/5	✓	✓

Слика 7. *Табела поређења анализираних алата за тестирање прихватљивости*

На основу табеле, може се закључити да на основу оцењених категорија *TestCafe* и *Cypress* односе победу што се тиче најбољих алата за *e2e* тестирање и тестирање прихватљивости. У даљем тексту ће бити објашњен резултат поређења сваког од алата у табели.

Cypress има одличне оцене за подешавање и писање тестова, али нажалост не подржава тестирање са више претраживача (*cross-browser testing*). Ова функционалност је најављена да ће бити имплементирана у будућности. Одлично се показао приликом подешавања на сервисима за континуалну интеграцију. Алат који олакшава *TDD* приступ развоју софтвера јер аутоматски покреће тестове при променама у тестовима пројекта. Одличан за развој иначе јер се покреће у претраживачу и може бити покренут док се развија функционалност и одмах се проверити да ли софтвер ради или не.

Puppeteer на овој табели има најлошију позицију, али то не значи да не постоји случај у ком би се овај алат могао користити. *Puppeteer* специјализује у покретању тестова у *Google Chrome* претраживачу, и у пројекту где је тај претраживач једини захтев, заједно са брзином покретања тестова, *Puppeteer* би могао бити прави избор. Такође његова могућност да се конфигурише на различите начина и слобода која је тиме дата могу се показати као добар избор за одређене тимове који већ имају подешен систем на одређен начин.

Nightwatch.js је добар избор ако је потребно тестирати на више претраживача користећи *WebDriver* протокол јер се *Selenium* не препоручује у тренутној верзији алата. У овој табели, на основу резултата поређења, *Nightwatch.js* заузима претпоследње место.

TestCafe се показао као најкомплетније решење које тренутно постоји у свету *JavaScript e2e* тестирања јер је лак за подешавање и писање тестова, подржава тестирање са више претраживача, и све то може да се извршава у паралели. Одличан избор ако је у пројекту потребно да се тестира софтвер на више претраживача јер његов директни ривал – *Cypress* не подржава ту функционалност.

CodeceptJS је одличан алат који пружа читљиву синтаксу за писање тестова. Подржава више претраживача али је тежи да се подеси у односу на остале алате. Такође постоје нерешене разлике у командама које корисник може да зада у тестовима у зависности од претраживача за који се определи. Због овога *CodeceptJS* заузима друго место на основу резултат поређења у табели. Одличан је избор ако је тимовима потребно да дефинишу читљиве тестове за остатак тима, али из разлога што користи друге алате као помагаче, није успео да се више пласира.

5. LITERATURA

[1] J. Pan, Software Testing (coursework), Carnegie Mellon University, 1999.

[2] V. Zaidman, "An Overview of JavaScript Testing in 2019," Welldone Software, 18 February 2019. https://medium.com/welldone-software/an-overview-of-javascript-testing-in-2019-264e19514d0a

[3] „The State of JavaScript 2018: Testing – Other Libraries," https://2018.stateofjs.com/testing/other-libraries/

Кратка биографија:

Никола Ђуза рођен је у Новом Саду 1993. године. Мастер студије на Факултету техничких наука из области Електротехнике и рачунарства – примењене рачунарске науке уписао је 2016. године након завршених основних академских студија из исте области. контакт: nikoladjuza@uns.ac.rs