
Frontend тестирање со Puppeteer и Jest



Изработила:
Вероника Огњановска 181045

Ментор:
доц. д-р Христина Михајлоска

Август 2021

Содржина

Абстракт	3
Вовед	3
Методологија	4
Puppeteer	4
Инсталирање	5
Работа со Puppeteer	5
Добивање содржина на страницата	6
Најчести методи	8
Jest	9
Инсталирање	9
Работа со Jest	9
Методи	10
Совпаѓачи	12
Mock функции	13
Тестирање на React апликација со Puppeteer и Jest	15
Почетни поставувања	15
Тестирање на Header опции	17
Тестирање на Books опции	19
Тестирање на Categories страна	20
Тестирање на View Book опции	22
Тестирање на Edit Book опции	26
Тестирање на Add/Delete Book опции	29
Извршување на сите тестови	31
Дискусија	33
Користена литература	34

Абстракт

Тестирањето е важен дел при развивањето на апликации и овозможува откривање на грешки во софтверот, притоа давајќи можност истите да бидат поправени пред да влезат во производство. Со frontend тестирање, може однапред да се потврди дека визуелно претставените објекти што се користат функционираат како што е предвидено. Како цел на оваа семинарска работа е најпрво да се разгледаат можностите на Puppeteer и Jest како два JavaScript-рамки што се користат за тестирање, како и да се прикажат дел од истите преку тестирање на React апликација.

Вовед

Со развивање на се поголемиот број софтверски апликации и алатки, нивното одржување станува доста покомплексно. Сето тоа доведува до потреба од тестирање за да се обезбеди и истовремено да се задржи квалитетот на софтверските производи.

Како најважниот дел за корисниците на даден софтвер е соодветниот визуелен приказ на информации. Поради ова, при тестирање на софтверот треба да се провери и како изгледа и функционира визуелниот дел на системот. За да се обезбеди непрекорен графички кориснички интерфејс (GUI), frontend тестирањето е задолжително.

На кратко, frontend тестирањето дава потврда дека она што луѓето го гледаат на страницата и карактеристиките/функционалностите што ги користат на него функционираат како што е предвидено. За да се овозможи полесно автоматизирано frontend тестирање, развиени се голем број рамки (анг."frameworks"), алатки и дополнителни библиотеки. Дел од истите се Jest, Puppeteer, Selenium, Jasmine и останати.

Цел на овој труд е да го продлабочи знаењето на читателот на темата frontend тестирање преку работа со JavaScript-рамките Puppeteer и Jest. Исто така читателот треба да добие основни познавања за функционалностите кои овие рамки/алатки ги нудат на корисникот.

Методологија

Тестирањето ни дава потврда дека сите делови од апликацијата правилно функционираат, како и дека тие делови работат заедно како што се очекува. При тестирање на веб-апликации, важен дел е тестовите да вклучуваат и симулираат сценарија за реални корисници. Постојат голем број на различни начини и рамки кои ни нудат алатки за да се постигне оваа симулација и автоматизација на соодветните тестови^[1]. Меѓу најпопуларните JavaScript рамки за тестирање се Puppeteer и Jest, кои имаат свои специфични функции и се силни алатки сами за себе, но комбинирани заедно, ни даваат моќна алатка за тестирање на модерните веб-апликации.

Во оваа семинарска работа, ќе се разгледаат Puppeteer и Jest во Node JS базиран проект. Соодветно потребни се неколку пакети да се симнат. Најпрво, потребно е да има npm кое претставува помошна алатка за менаџирање со пакети. Може да се симне заедно со NodeJS на следниот линк '<https://www.npmjs.com/get-npm>'. За започнување на NodeJS проект, потребна е командата 'npm init'. Сите останати пакети, ќе бидат дадени понатаму со соодветните команди.

Puppeteer

Puppeteer е JavaScript Node библиотека, развиена од страна на Google, која дава интерфејс (API) за контрола на веб-прелистувачот Chrome, поточно headless Chrome или Chromium. Повеќето рачни операции извршени во прелистувачот Chrome можат да се автоматизираат со помош на Puppeteer^[2], кој притоа го користи протокол Chrome DevTools.

Во основа, Puppeteer е алатка за автоматизација, а не алатка за тестирање. Како таква, има голема популарност за операции како што се web scraping, автоматизација на функции, генерирање PDF датотеки, тестирање на перформанси, итн.

Инсталирање

```
npm install puppeteer
```

За користење на Puppeteer, потребно е да се инсталира преку извршување на горенаведената команда, и притоа се презема и неодамнешна верзија на Chromium^[3].

Работа со Puppeteer

Најпрво е потребно да се направи import на Puppeteer библиотеката во JavaScript датотеката преку require функцијата^{[4][5]}.

```
const puppeteer = require('puppeteer');
```

Преку методот launch(), се креира инстанца од прелистувачот. На методот може да се пренесе дополнителен објект со опции. Една опција е '{ headless:false }', која овозможува прикажување на Chrome инстанцата при извршување на операциите.

Исто така се користи операторот await за да чека на 'Promise', и истиот може да се користи само во 'async' блок. Со тоа програмата 'чека' додека ветувањето не даде повратен резултат. Треба да се напомене дека само блокот на функцијата 'async' чека, а не целото извршување на програмата.

```
(async () => {  
  const browser = await puppeteer.launch({  
    headless:false  
  });  
})();
```

Следно, можеме да го користиме методот newPage() на browser објектот на прелистувачот за да добиеме објект од нова страница. И како следен чекор, преку методот goto() над објектот на страницата за да ја вчитаме новата страница зависно URL линкот кој е предаден како параметар.

```
(async () => {  
  const browser = await puppeteer.launch({  
    headless:false  
  });  
  page = await browser.newPage();  
  await page.goto('https://www.finki.ukim.mk/mk');  
})();
```

Можеме да користиме и ветувања, наместо асинхронизирање/чекање, но користењето на 'await' го прави кодот многу почитлив.

```
((() => {
  puppeteer.launch().then(browser => {
    browser.newPage().then(page => {
      page.goto('https://www.finki.ukim.mk/mk')
        .then(() => {
          //...
        })
    })
  })
})())
```

Откако ќе се заврши со сите потребни операции, го повикуваме методот close() на прелистувачот, за затворање на истиот.

```
(async () => {
  const browser = await puppeteer.launch({
    headless:false
  });
  page = await browser.newPage();
  await page.goto('https://www.finki.ukim.mk/mk');
  await browser.close();
})();
```

Добивање содржина на страницата

Објектот 'page' од горенаведениот код, претставува страница со соодветното URL од која можеме да ја добиеме содржината на истата страницата преку повикување на методот evaluate().

```
const result = await page.evaluate(() => {
  //...
})
```

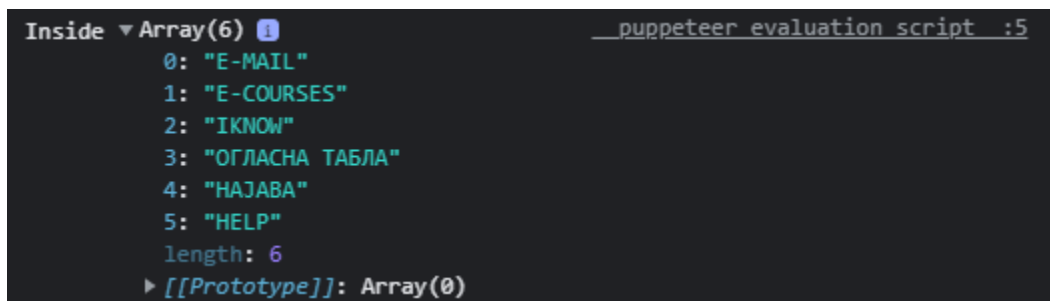
Овој метод, како аргументи, прима callback функција, каде што го додаваме кодот потребен за преземање на потребните елементи од страницата и враќаме соодветен 'result' објект како резултат на повикот evaluate().

```
const result = await page.evaluate(() => {
  const element =[...document.querySelectorAll('.leaf')]
    .map(element => {
      return
    element.querySelector('a').innerText
    });
  console.log('Inside', element);
  return element;
});
console.log(result);
```

Output in IDE console:

```
[ 'E-MAIL', 'E-COURSES', 'IKNOW', 'ОГЛАСНА ТАБЛА', 'НАЈАВА', 'HELP' ]
```

Во кодот од callback функцијата од evaluate методот, се контролира 'page' објектот што ја представува новата страница. Доколку како дел од овој код, сакаме да испечатиме информации во конзола со методот console.log(), соодветните информации ќе бидат испечатени во конзолата од Chrome page која ја контролираме со Puppeteer.



```
Inside ▾ Array(6) ⓘ
  0: "E-MAIL"
  1: "E-COURSES"
  2: "IKNOW"
  3: "ОГЛАСНА ТАБЛА"
  4: "НАЈАВА"
  5: "HELP"
  length: 6
  ▸ [[Prototype]]: Array(0)
__puppeteer_evaluation_script__:5
```

Најчести методи

Puppeteer нуди голем број на методи за добивање на содржината на дадена страница. Еден од нив е и претходно искористениот `evaluate()` метод, кој евалуира функција за страницата во која имаме пристап до DOM објектот. Сличен на `evaluate()` методот е методот `evaluateHandle()`. Како разлика помеѓу овие два методи е резултатот што истите го враќаат. Доколку вратиме DOM објект, со `evaluate()` методот, добиваме празен објект, додека со `evaluateHandle()` методот, соодветно се враќа објектот.

Методот `page.$eval()` кој прима минимум 2 аргументи, селектор за методот `querySelectorAll()` и `callback` функција на која се проследени резултатите од селекторот и дополнителни аргументи, дококу се проследени на `eval` функцијата.

Соодветно, методот `page.$()` овозможува пристап до методот `querySelector()`, и соодветно `page.$$()` за методот `querySelectorAll()`. Над објектот 'page' можат да се повикаат и методите `click()`, `content()`, `emulate()` и `exposeFunction()`.

`Click()` е метод за симулација на кликување на глумче, каде може да се постави лево или десно копче, колку пати е кликнато и време помеѓу кликувања. Преку `content()` методот може да се превземе HTML кодот на страницата, додека со `emulate()` методот, може да се направи симулација на специфичен уред. `ExposeFunction()` е метод кој овозможува да се додаде нова функција во контекстот на веб-прелистувачот, за понатамошно повикување на истата.

Со помош на методите `goBack()` и `goForward()`, може да се навигира низ историјата на прелистувачот, додека со методите `focus()` и `hover()` се поставува фокусот и се симулира глумчето над HTML елементот.

Доста користени методи при работа со Puppeteer се методите за отворање нова страна `goto()`, повторно вчитување `reload()`, креирање на PDF документ од страницата со методот `pdf()`, како и правење на PNG слика од страната со `screenshot()` методот. За методите `pdf()` и `screenshot()` потребно е поставување на `path` вредност кој ја означува патеката каде ќе биде зачуван pdf документот или сликата.

Како што е претходно наведено, Puppeteer нуди голем број на методи за контролирање на прелистувачот и листа од истите може да видите на линкот '<https://pptr.dev/#?product=Puppeteer&show=api-class-page>'.

Jest

Jest е JavaScript тест-тркач (анг. 'test runner'), односно JavaScript библиотека за креирање, извршување и структурирање тестови^[6].

Како предности што Jest ги нуди се брзина и паралелно извршување на тестови изолирани еден од друг, моќна mockup поддршка, без потребна конфигурација, како и поддршка за snapshot тестирање кое е доста важно од страна на React.

Инсталирање

```
npm install jest
```

За да се користи, потребна е команата од горниот дел за инсталација. За полесно менаџирање со сите Jest тестови, потребно е конфигурирање на 'npm' тест скриптата со доленаведените промени во 'package.json' датотеката.

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

Работа со Jest

Како и останатите алатки за тестирање на софтвер, Jest нуди листа на методи кои можат да се користат при пишување на тестови. Овие методи Jest ги додава во глобалната околина. Со ова, нема потреба од формално да се направи import или require на потребните методи за нивно понатамошно користење, но зависно преференции може да се направи експлицитно импортирање на истите од '@jest/globals'. Повеќе информации за Jest методите може да се најдат на <https://jestjs.io/docs/api>^[7].

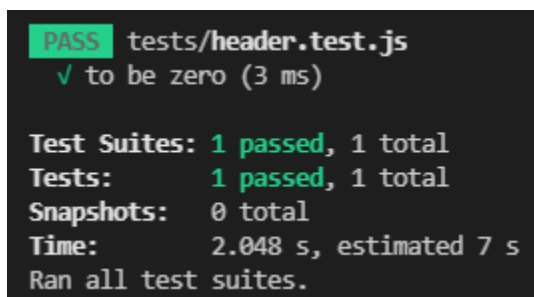
Методи

Во една тест датотека, најважно е да има метод кој ќе го изврши тестот^[8]. Jest го нуди својот test() метод кој како аргументи прима име на тестот, функција која ги содржи очекувањата од тестот и како трет аргумент, опционално прима истекот на времето (во милисекунди) за одредување колку долго да чека пред да прекине со извршување. Test() методот има и свој псевдонимот it().

Во NodeJS, како што беше претходно напоменато, со наредбата 'npm test' може да се извршат сите тестови. Како добра практика е датотеките кои ги содржат тестовите да бидат именувани со '*.test.js'.

Пример за користење на test() методот, е соодветниот код подолу каде што е прикажано очекување на методот zero() да врати 0. При извршување на 'npm test', соодветниот тест ќе биде извршен и резултатот ќе биде прикажан во NodeJS конзолата.

```
const zero = ()=>{
  return 0;
};
test('to be zero', () => {
  expect(zero()).toBe(0);
});
```



```
PASS tests/header.test.js
  ✓ to be zero (3 ms)

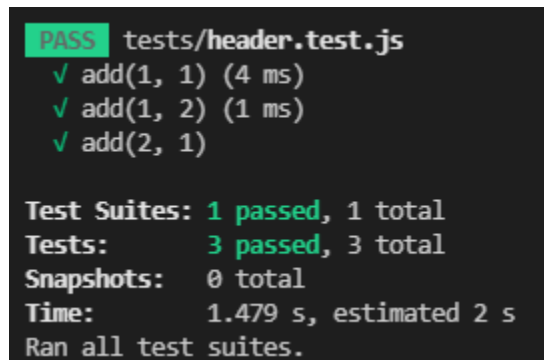
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        2.048 s, estimated 7 s
Ran all test suites.
```

Зависно начинот на извршување на потребниот тест, како и податоците што се потребни, Jest нуди методи како test.concurrent(), test.concurrent.each(table)() и останати други кои нудат можност за истовремено (анг. 'concurrently') извршување на тестовите. Двата методи примаат име на тестот, асинхрона функција и опционален аргумент за истек на времето test.concurrent.each(table)() прима и дополнителен аргумент table кој претставува низа од низи(матрица) со податоци кои се предаваат на тестот, со што истиот тест се извршува повеќе пати, но со различни податоци.

Во соодветниот пример, аргументот 'table' е претставен со низа од низи, каде секоја низа содржи 3 вредности, и тоа a и b како вредности кои понатаму се предаваат на функцијата add(), и третата вредност што ја представува сумата на a и b која се очекува во соодветниот тест.

```
const add = (a, b) => {
  return a + b;
};
```

```
test.concurrent.each([
  [1, 1, 2],
  [1, 2, 3],
  [2, 1, 3],
])('add(%i, %i)', async (a, b, expected) => {
  expect(add(a, b)).toBe(expected);
});
```



За полесна работа со тестовите, 'describe' методот дава прилика за групирање на истите во блокови. Има и повеќе различни имплементации на 'describe' методот, но најосновниот е прикажан подолу. Исто така, со помош на овој метод, може и хиерархиски да се групираат тестовите.

```
describe('my block, () => {
  test('test 1', () => {
    //...
  });
  test('test 2', () => {
    //...
  });
});
```

Како и сите останати рамки за тестирање, Jest има методи кои ни овозможуваат извршување на код пред сите тестови заедно или пред секој тест посебно, како и по завршувањето на тестовите. Ова се постигнува со методите `afterAll(fn, timeout)`, `afterEach(fn, timeout)`, `beforeAll(fn, timeout)` и `beforeEach(fn, timeout)`.

Совпаѓачи

До сега видовме и примери за очекувања или expect методот. Методот expect() се користи секој пат кога сакате да тестирате вредност. Тој дава пристап до множество од 'совпаѓачи' (анг. 'matchers') што ви дозволуваат да потврдите нешто во врска со вредноста. На следниот линк <https://jestjs.io/docs/expect> може да видите листа од 'совпаѓачи'.

Во претходните примери, заедно со expect(), беше искористен toBe(). Користењето на истите, дозволува да се провери дали вредноста проследена како аргумент во методот expect() е еднаква со вредноста проследена во методот toBe().

По слична логика се водат и методите за споредба на броеви toBeGreaterThan(), toBeGreaterThanOrEqual(), toBeLessThan() и toBeLessThanOrEqual(), методите за проверка на true false во бинарен контекст, toBeTruthy() и toBeFalsy(). Како и да биде 'null', 'undefined', 'defined', да не е број или да е инстанца од соодветна класа, со методите toBeNull(), toBeUndefined(), toBeDefined(), toBeNaN(), toBeInstanceOf(Class).

Практичен метод да се знае е методот 'not' кој дозволува да го тестирате спротивното сценарио. Методот 'not' може да се поврзе на сите останати методи на многу едноставен начин, само со додавање пред другиот метод.

```
expect(notZero()).not.toBe(0);
```

Понудени се и методи кој ни даваат потврда за дали дадена функција била повикана, колку пати и со кои аргументи. Соодветно тоа се методите toHaveBeenCalled(), toHaveBeenCalledTimes(number) и toHaveBeenCalledWith(arg1, arg2, ...), или со своите псевдоними toBeCalled(), toBeCalledTimes() и toBeCalledWith() соодветно.

Jest има голем број на „совпаѓачи“, но ни дозволува и креирање на сопствени преку методот expect.extend(). Овој метод, како аргумент прима објект во кој се наведени „совпаѓачите“ во следниот формат:

```
expect.extend({
  yourMatcher(x, y, z) {
    return {
      pass: true,
      message: () => '',
    };
  },
});
```

Како резултат од 'совпаѓачот' е објект со pass вредност коа означува дали има совпаѓање/потврда или не, како и 'message' функција која враќа порака за грешка доколку pass има вредност false.

Jest нуди и погенерални 'совпаѓачи'. Expect.anything() дава потврда за се што не е 'null' или 'undefined', додека expect.any(constructor) за се што е креирано со проследениот конструктор.

```
expect.any(Number)
```

Доколку сакаме на даден објект да провериме таканаречена длабока еднаквост ('deep'), тоа можеме да го постигнеме со toEqual(value). Од друга страна, доколку имаме функција од која очекуваме да фрли исклучок, за соодветно тестирање се користи toThrow(error) или под псевдонимот toThrowError(error?). За тестирање на грешки, мора да го поставите кодот во функција, инаку грешката нема да биде фатена и тестот ќе падне.

```
const throwError = () => {
  throw new Error('error');
};
test('throws an error', () => {
  expect(() => {
    throwError();
  }).toThrow();
});
```

Mock функции

Mock функциите овозможуваат тестирање на врските помеѓу кодот со бришење на вистинската имплементација на функција, зачувување на повици до функцијата (и параметрите пренесени во тие повици) и овозможување конфигурација за време на тестирање на повратните вредности. Mock функции се познати и како 'шпиони', бидејќи тие дозволуваат шпионирање на однесувањето на функцијата што индиректно се повикува со некој друг код, наместо само да го тестирате излезот.

Лесно креирање на mock функција е преку методот jest.fn(). Ако не е дадена имплементација, mock функцијата ќе се врати 'undefined' кога ќе се повика. Имплементација на може да се предаде преку jest.fn(implementation) или jest.fn().mockImplementation(implementation).

```
const mockFn = jest.fn().mockImplementation(  
  scalar => 42 + scalar  
);  
// or: jest.fn(scalar => 42 + scalar);
```

Соодветната имплементација на mock функцијата ќе биде извршена кога mock функцијата се повикува.

```
const a = mockFn(0);  
// a === 42
```

Mock функциите се важен дел при тестирање при тестирање на кодот и соодветните функциите. За повеќе информации за mock функции и останати методи понудени од Jest може да се најдат на следниот линк '<https://jestjs.io/docs/mock-function-api>'.

Тестирање на React апликација со Puppeteer и Jest

Заедно, рамките Puppeteer и Jest^[9], овозможуваат тестирање на било која frontend апликација. За приказ на овие можности, во продолжение следи пример тестирање на frontend функционалностите на една React апликација.

Соодветната апликација е е-библиотека која овозможува функционалности за преглед на листа од книги, преглед на листа од категории на книги, додавање на нова книга, менување, бришење и преглед на детали за соодветна книга, бришење и додавање на примероци од книга.

Кодот од тестирањето, како и кодот од React и Spring Boot апликацијата, може соодветно да се најдат на следните линкови:

- <https://github.com/veronikaognjanovska/puppeteer-jest>
- https://github.com/veronikaognjanovska/emit_lab

Почетни поставувања

Тестирањето на оваа React апликација се извршува преку typescript датотека во NodeJS рамката. Како што претходно беше напоменато, потребно е инсталирање на соодветните библиотеки преку следните команди и за полесно менаџирање поставување на следната скрипта^{[10][11][12]}.

```
npm install puppeteer
npm install jest

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

Сите typescript датотеки во кои има најмалку еден тест, се именуваат по конверзијата 'name.test.ts', со кое соодветниот код ќе биде препознат како тест сличај.

Најпрво во соодветната датотека, потребно е да се побара т.е. 'import' на Puppeteer и некои од методите од библиотеката Jest како што се test(), describe(), beforeEach() и afterEach(). Како потребни поставувања за ова тестирање, потребно е да се побара и fs библиотеката за работа со датотечниот систем на вашиот компјутер.

За полесно менаџирање, во истата датотека, глобално се поставени променливите 'browser' и 'page', кои соодветно ќе ги претставуваат објектите од Puppeteer рамката. Променливите 'width' и 'height', како вредности за големината на виртуелната страна од Puppeteer и 'path' променлива која ја чува патеката до која ќе се зачувуваат сликите од екранот.

```
const { beforeEach, afterEach, describe, test } =
  require('@jest/globals');
const puppeteer = require('puppeteer');
const fs = require('fs');

let browser, page;
const width = 1280, height = 800;
const path = 'images/';
```

За потребите на тестирањето, однапред се креираат 5 објекти и тоа bookViewObject, bookEditObject, bookEditObject2, bookAddObject, bookEmptyObject и истите соодветно се претставени во кодот.

Со методите beforeEach() и afterEach(), се поставуваат потребните параметри за почеток на Puppeteer веб-прелистувачот. Поставувањето на 'headless' со вредност 'true', ни означува извршување на тестовите без притоа визуелен приказ на веб-прелистувачот. Со методата setViewport() се поставува висината и ширината на веб-прелистувачот. Останатите методи се соодветно објаснети во делот за Puppeteer.

```
beforeEach(async () => {
  browser = await puppeteer.launch({
    headless: true
  });
  page = await browser.newPage();
  await page.setViewport({
    width: width, height: height
  })
  await page.goto('http://localhost:3000/books');
});

afterEach(async () => {
  await browser.close();
})
```

Тестовите во кодот се поделени во делови зависно делот за кои се наменети. Во соодветни 'describe' делови.

Тестирање на Header опции

За тестирање на 'Header' делот, се искористени 5 тестови.

Првиот тест е именуван 'Left Title', и во истиот, се повикува методата `evaluate()` од 'page' објектот. Во соодветната метода, со помош на `querySelector()` и HTML елементите на страницата, се превзема текстот кој се наоѓа на левата страна на 'header'-от. Со `expect` методата и 'совпаѓачот' `toBe()`, се споредува совпаѓање на текстот со вредноста 'Library'.

Во вториот тест, 'Right Buttons Text', се споредува текстот кои го содржат копчињата од десната страна на header-от со листата ['Books', 'Categories']. За истото, се користи методата `querySelectorAll()`, и соодветно текстот се враќа во листа која понатаму се споредува со помош на `toEqual()`.

```
describe('Header', () => {

  test('Left Title', async () => {
    const result = await page.evaluate(() => {
      return document.querySelector('header span').innerText;
    });
    expect(result).toBe('Library');
  });

  test('Right Buttons Text', async () => {
    const result = await page.evaluate(() => {
      return [...document.querySelectorAll('header a')]
        .map(element => element.innerText);
    });
    expect(result).toEqual(['Books', 'Categories']);
  });

  test('Right Buttons Link', async () => {
    await page.click('header
.btn.btn-outline-light:last-of-type');
    await page.waitForSelector('.categories');
    const url1 = await page.url();
    expect(url1).toBe('http://localhost:3000/categories');
    await page.click('header
.btn.btn-outline-light:first-of-type');
    await page.waitForSelector('.books');
    const url2 = await page.url();
    expect(url2).toBe('http://localhost:3000/books');
  });
});
```

```

test('Screenshot', async () => {
  screenshot = path + 'header.png';
  await page.screenshot({
    path: screenshot,
    'clip': { 'x': 0, 'y': 0, 'width': width, 'height': 55 }
  });
  new Promise((resolve) => {
    fs.access(screenshot, fs.constants.F_OK, (err) => {
      err ? resolve(false) : resolve(true);
    });
  }).then(screenshotMade => {
    expect(screenshotMade).toBeTruthy();
  });
});
});

```

Следниот тест именуван како 'Right Buttons Link', го тестира функционирањето на двете копчиња. Со предавање на класен селектор на методата click() над 'page' објектот, се кликува над избраното копче и се чека да се изврши соодветната акцијата. Во овој случај, се редиректира корисникот на нова страна, најпрво со URL 'http://localhost:3000/categories', додека при клик на другото копче, на страната URL 'http://localhost:3000/books'. Со методите waitForSelector() и url() од објектот 'page', се чека додека не се пронајде соодветно пратениот селектор на HTML страницата, и се превзема URL на соодветната страна која е отворена.

Со последниот тест, се прикажува функционалноста која Puppeteer ја нуди за креирање на слика од моментално отворената страна на веб-прелистувачот. За истата цел, потребна е патека до папката во која ќе се зачува сликата т.е. 'screenshot' променливата ја има таа вредност. На методата screenshot() потребно е да се предаде објект со вредност за 'path' и објект како вредност за 'clip', кој ги претставува почетите координати и големината на сликата т.е. отсечокот кој се зачувува со помош на 'fs' библиотеката за работа со датотечниот систем на вашиот компјутер. За да се зачува сликата и по нејзиното зачувување да се провери дали е успешно, се користи 'Promise' т.е. ветување бидејќи пристапот до сликата преку методата access работи со ветувања при случај на грешка. Во блокот 'then' соодветно се поставува Jest изразот за проверка во комбинација со toBeTruthy() 'совпаѓачот'.

Следната сликата е резултат на овој тест. На истата може да се забележи текстот Library од левата страна и двете копчиња од десната страна.



Тестирање на Books опции

На почетната страна од апликацијата, има листа од сите книги и со пагинација се поставени по две книги во листа. За тестирање на овој дел, има повеќе тестови кои можете да ги видите во кодот, додека тука ќе бидат разгледани само дел од нив. Тестовите вклучуваат тестирање на текстот на насловите на страната и/или насловите на колоните на табелата и текстовите во копчињата.

Еден од тестовите е дали е направена слика од екранот и соодветно следната слика го претставува екранот.

List of Books

Add New Book

ID	Book Name	Book Category	Author Name	Available Copies	
9	Harry Potter and the Philosopher's Stone	FANTASY	J. K. Rowling	2	<div>ViewEditDelete</div>
10	Harry Potter and the Chamber of Secrets	FANTASY	J. K. Rowling	3	<div>ViewEditDelete</div>

back 1 2 next

Тестот 'Table Body First 2 Elements', со помош на Puppeteer ги зима редиците во табелата која ја представува листата од книги, поради пагинацијата тоа се две книги. Соодветното мапирање на вредностите може да го видите во кодот. Како резултат се очекува да бидат вратени листа од два елемента, и елементите се исто така листи со по шест елемента во нив. Тестирањето се извршува со 'совпаѓачот' `toHaveLength()` и `toEqual()` со кој се споредува листата.

```
test('Table Body First 2 Elements', async () => {
  const result = await page.evaluate(() => {
    return [...document.querySelectorAll('.books table tbody
tr')]
      .map(element => {
        return [...element.querySelectorAll('td')]
          .map(td => td.innerText)
      });
  });
  expect(result).toHaveLength(2);
  expect(result[0]).toHaveLength(6);
});
```

```

    expect(result).toEqual([
      ['9', 'Harry Potter and the Philosopher\'s Stone',
        'FANTASY', 'J. K. Rowling', '2', 'ViewEditDelete'],
      ['10', 'Harry Potter and the Chamber of Secrets',
        'FANTASY', 'J. K. Rowling', '3', 'ViewEditDelete']
    ]);
  });
});

```

Следниот тест 'Pagination' како што гласи е за пагинација каде се тестира дали визуелно се претставени елементите кои се потребни за да се прегледаат следните книги од листата. Потребно е да има два или повеќе елементи, од кои првиот гласи 'back', додека последниот гласи 'next'. Бројот на елементи се тестира со `toBeGreaterThanOrEqual()`.

Во понатамошните тестови, се тестира и функционалноста на копчињата, на истиот начин како и копчињата од 'Header' делот.

```

test('Pagination', async () => {
  const result = await page.evaluate(() => {
    return [...document.querySelectorAll('.pagination li')]
      .map(element =>
        element.querySelector('a').innerText);
  });
  expect(result.length).toBeGreaterThanOrEqual(2);
  expect(result[0]).toEqual('back');
  expect(result[result.length - 1]).toEqual('next');
});

```

Тестирање на Categories страна

Од почетните поставувања, Puppeteer не води најпрво кон почетната страна со листата со книги. За тестирање на друга почетна страна, може да се постави кодот во друга датотека, или да се постави дополнителен `beforeEach` блок во 'describe' делот за категории. Овој код ќе биде извршен само пред тестовите од соодветниот 'describe' дел. На следната слика е прикажан екранот од веб-прелистувачот за оваа страна.

Од страната на која се прикажани книгите, преку кликување на 'Categories' копчето од 'Header' делот, веб-прелистувачот не редиректира кон страна на која е прикажана листа од сите категории. Преку тестот 'Elements', се тестира листата на категории.

```

describe('Categories', () => {
  beforeEach(async () => {
    // clicks the first button, first book
    await page.click('.btn.btn-outline-light:last-of-type');
    await page.waitForSelector('.categories');
  });
  test('Elements', async () => {
    const result = await page.evaluate(() => {
      return [...document.querySelectorAll('.categories table
tbody tr td')].map(element => element.innerText);
    });
    expect(result).toHaveLength(7);
    expect(result).toEqual(['NOVEL', 'THRILER', 'HISTORY',
      'FANTASY', 'BIOGRAPHY', 'CLASSICS', 'DRAMA']);
  });
});

```

На следната слика е прикажан екранот од веб-прелистувачот за страната со категории.

List of Categories
Categories
NOVEL
THRILER
HISTORY
FANTASY
BIOGRAPHY
CLASSICS
DRAMA

Тестирање на View Book опции

Преку листата на книги, имаме опции за преглед на детали за дадена книга. За директен пренос на соодветната страна, се користи 'describe' дел со beforeEach блок, како што беше напоменато и во делот за 'Header'. Тестирањето ќе се одвива на втората книга од листата.

На следните три слики е прикажана страната со детали за втората книга.

[Back](#)

Book Info

[Add New Book Print](#)

Book Info

Book name

Harry Potter and the Chamber of Secrets

Category

FANTASY

Author

J. K. Rowling

Author from

Great Britain - Europe

Number Available Copies

3

ID	Bookprint status		
34	AVAILABLE	<button>Mark As Taken</button>	<button>Delete</button>
35	AVAILABLE	<button>Mark As Taken</button>	<button>Delete</button>
38	AVAILABLE	<button>Mark As Taken</button>	<button>Delete</button>

За да се тестира дали точните детали се прикажани во полињата се користи тестот 'View Inputs'. Во овој тест, вредностите од input полињата се превземаат во објект и се споредуваат со претходно споменатите вредности, bookViewObject е соодветниот објект креиран за овој тест. За тестирање искористен е 'совпаѓачот' toMatchObject().

```
const bookViewObject = {
  'Book name': 'Harry Potter and the Chamber of Secrets',
  Category: 'FANTASY',
  Author: 'J. K. Rowling',
  'Author from': 'Great Britain - Europe',
  'Number Available Copies': '3'
};

test('View Inputs', async () => {
  const result = await page.evaluate(() => {
    let object = {};
    [...document.querySelectorAll('.book-view .form-group')]
      .forEach(element => {
        let label =
          element.querySelector('label').innerText;
        let input = element.querySelector('input').value;
        object[label] = input;
      });
    return object;
  });
  expect(result).toMatchObject(bookViewObject);
});
```

На страната може да се забележи дека има две копчиња 'Back' и 'Add New Book Print'. Тестирањето на копчето 'Back' се извршува со следење на URL на страната која е отворена, додека тестирањето на 'Add New Book Print' се следи со преглед на известувањата кои се прикажуваат и додавањето на нов 'Book Print' во листата. Со тестот именуван 'Back Button' се тестира функционалноста на 'Back' копчето.

```
test('Back Button', async () => {
  await page.click('.btn.btn-outline-primary');
  await page.waitForSelector('.books');
  const url = await page.url();
  expect(url).toBe('http://localhost:3000/books');
});
```

За подобра структура на тестовите, тестовите поврзани со 'Book Print' се поставени во хиерархиски во 'View Book' describe делот.

```
describe('View Book', () => {
  ...
  describe('Book Print', () => {
    ...
  })
});
```

Како дел од тестирањето на копчиња, ќе биде прикажан тестот за функционалностите за додавање и бришење на еден 'Book Print'. Во овој дел, за успешноста на извршената акција има два покажувачи. Едниот е додадениот т.е. избришаниот 'Book Print' од листата, додека вториот е известување, визуелно во горниот десен агол. Во следниот тест, најпрво се додава нов примерок со кликување на соодветното копче, и со методата `waitForSelector()` се чека да се покаже известувањето. За тестирање на типот на известувањето, се споредува насловот и пораката на истото со известувањата 'Success!' и 'Book Print added successfully!'. Исто така, за успешноста се проверува и листата на примероци по што се очекува да биде за еден повеќе.

Методата `waitForTimeout()` ни дава прилика да се запре извршувањето на тестот за 6000 милисекунди т.е. 6 секунди. Ова време е потребно за известувањето ќе е претходно добиено да се тргне од екранот.

Во вториот дел од овој тест, имаме бришење на последниот примерок во листата т.е. примерокот кој е претходно додаден со што податоците се враќаат во почетната состојба. Проверувањето на оваа функционалност соодветствува на проверката при додавање на примерок.

```
test('Add/Delete Book Print', async () => {
  await page.click('.btn.btn-outline-success');
  await page.waitForSelector('.notification');
  const result1 = await page.evaluate(() => {
    let notification = document.querySelector('.notification');
    let title = notification.querySelector('.notification__title')
      .innerText;
    let message = notification
      .querySelector('.notification__message').innerText;
    return [title, message];
  });
```



```

});
expected1 = ['Success!', 'Book Print added successfully!'];
expect(result1).toEqual(expect.arrayContaining(expected1));
const result2 = await page.evaluate(() => {
    return document.querySelectorAll('.book-view table tbody
        tr').length;
});
expect(result2).toBe(4);
await page.waitForTimeout(6000);
await page.click('table tbody tr:last-of-type
    .btn-outline-danger');
await page.waitForSelector('.notification');
const result3 = await page.evaluate(() => {
    let notification = document.querySelector('.notification');
    let title = notification.querySelector('.notification__title')
        .innerText;
    let
message=notification.querySelector('.notification__message')
        .innerText;
    let length = document.querySelectorAll('.book-view table tbody
        tr').length;
    return [title, message, length];
});
expected3 = ['Warning!', 'Book Print deleted successfully!'];
expect(result3).toEqual(expect.arrayContaining(expected3));
expect(result3[2]).toEqual(3);
}, 15000);

```

На следните слики, може да се видат известувањата.



Во делот за Jest, беа спомнати повеќе параметри за дадената метода `test()`. Еден од нив е опционалниот параметар за истекот на времето (во милисекунди) за одредување колку долго да чека пред да прекине со извршување на тестот. На горенаведениот тест, овој параметар е предаден со вредност од 15 секунди. Стандардниот период е 5 секунди, но во овој тест имаме исчекување од 6 секунди, со што стандардниот период се надминува и за негово извршување потребно е да се додаде соодветен временски интервал.

Тестирање на Edit Book опции

За секоја книга од листата, поставена е и функционалност податоците за истата да се менуваат.

[Back](#)

Edit Book

Edit Book

Book name

Harry Potter and the Philosopher's Stone

Category

FANTASY

Author

J. K. Rowling

Number Available Copies

2

Submit

За овој дел поставен е describe блокот 'Edit Book'. Како карактеристика за тестирање што не беше прикажана до сега е менувањето на податоци на 'input' елементи. Овие промени можат лесно да се направат во методата `evaluate()` од 'page' објектот.

Во првиот дел од овој тест 'Edit Inputs', се тестираат постоечките податоци од елементите, додека во вториот дел 'Submit Edit Data', се менуваат податоците во елементите. Соодветно, формата се праќа со кликување на копчето 'Submit', и апликацијата успешно не пренесува на страната на која се прикажани книгите. Со ова ние може соодветно да провериме дали книгата што е сменета, е представена во листата со новите податоци.

Во третиот дел 'return data values back', се прави повторно менување на податоците, но овој пат назад во претходните за да и следното извршување на истиот тестот е успешно.

```
describe('Edit Book', () => {
  beforeEach(async () => {
    // clicks the first button, first book
    await page.click('.btn.edit-btn');
    await page.waitForSelector('.book-edit');
  });

  test('Submit Edit Data', async () => {

    // Edit Inputs

    const result1 = await page.evaluate(() => {
      let object = {};
      [...document.querySelectorAll('.book-edit .form-group')]
        .forEach(element => {
          let label = element.querySelector('label')
            .innerText;
          let input = element.querySelector('input');
          if (label === 'Book name') {
            input = input.placeholder;
          } else if (label === 'Number Available Copies') {
            input = input.value;
          } else {
            input = element.querySelector('select
              option:checked').innerText;
          }
          object[label] = input;
        });
      return object;
    });
    expect(result1).toMatchObject(bookEditObject);
  });
});
```

```

// Submit Edit Data

await page.waitForTimeout(5000);
await page.focus('#name');
page.keyboard.type(bookEditObject2['Book name']);
await page.waitForTimeout(1000);
await page.select('#category', bookEditObject2['Category']);
await page.waitForTimeout(1000);
await page.select('#author', '5');
await page.waitForTimeout(1000);
await page.click('button[type="submit"]');
const url = await page.url();
expect(url).toBe('http://localhost:3000/books');
await page.waitForTimeout(5000);
const result2 = await page.evaluate(() => {
  return [...document.querySelectorAll('.books table tbody
    tr')]
    .map(element => {
      return [...element.querySelectorAll('td')]
        .map(td => td.innerText)
    });
});
await page.waitForTimeout(5000);
expect(result2).toEqual([[ '9', 'Sense and Sensibility',
'CLASSICS', 'Jane Austen', '2', 'ViewEditDelete'], [ '10', 'Harry
Potter and the Chamber of Secrets', 'FANTASY', 'J. K. Rowling', '3',
'ViewEditDelete']]);

// return data values back

await page.waitForTimeout(5000);
await page.click('.btn.edit-btn');
await page.waitForSelector('.book-edit');
await page.focus('#name');
page.keyboard.type(bookEditObject['Book name']);
await page.select('#category', bookEditObject['Category']);
await page.select('#author', '4');
await page.waitForTimeout(1000);
await page.click('button[type="submit"]');

}, 30000);
});

```

Тестирање на Add/Delete Book опции

На почетната страна, за менаџирање со книги, има копче за додавање на нова книга, како и соодветно копче за бришење на дадена книга. Во кодот може да се прегледаат повеќе детали за тестовите за овој дел. Истите се слични со тестот за менување на податоци на дадена книга и тестот за бришење на примерок од книга.

List of Books

Add New Book

ID	Book Name	Book Category	Author Name	Available Copies	
9	Harry Potter and the Philosopher's Stone	FANTASY	J. K. Rowling	2	<div>ViewEditDelete</div>
10	Harry Potter and the Chamber of Secrets	FANTASY	J. K. Rowling	3	<div>ViewEditDelete</div>

back 1 2 next

Add Book

Book name

Enter book name

Category

Choose here

Author

Choose here

Available Copies

Enter number available copies

Submit

Во тестот 'Submit and Delete New Data', како примери за користење на Jest изрази, може да се издвои `stringMatching()`. Со помош на овој метод може да се спореди даден текст со соодветен 'REGEX' израз. Во следниот код тој е представен на местото на уникатниот идентификатор на книгата, кој не може да се претпостави за време на пишување на тестовите.

```
expected1 = [['11', 'A Long Petal of the Sea', 'NOVEL', 'Isabel Allende', '2', 'ViewEditDelete'],  
  [expect.stringMatching(/\d+/), 'Pride and Prejudice', 'CLASSICS', 'Jane Austen', '2',  
  'ViewEditDelete']];  
expect(result1).toEqual(expected1);
```

Последен Jest метод кој ќе биде споменат е `arrayContaining()`. Со овој израз, се очекува низата која се тестира да ги има истите елементи или повеќе од соодветно дадените.

```
expected4 = ['Warning!', 'Book deleted successfully!'];  
expect(result4).toEqual(expect.arrayContaining(expected4));
```

Извршување на сите тестови

Со наредбата `'npm test'` се извршуваат сите тестови во соодветно именуваните датотеки.

Како резултат од извршувањето на тестовите се добива одговор во терминалот на IDE, и истиот е претставен на следната слика.

```

> npm test

> puppeteer-jest@1.0.0 test C:\Users\veron\Documents\GitHub\puppeteer-jest
> jest

PASS tests/book.test.js (101.891 s)
  Header
    ✓ Left Title (1698 ms)
    ✓ Right Buttons Text (1356 ms)
    ✓ Right Buttons Link (1538 ms)
    ✓ Screenshot (1542 ms)
  Books
    ✓ Heading List of Books (1485 ms)
    ✓ Table Heading (1402 ms)
    ✓ Table Body First 2 Elements (1315 ms)
    ✓ Pagination (1268 ms)
    ✓ Screenshot (1483 ms)
  Categories
    ✓ Heading (1452 ms)
    ✓ Table Heading (1373 ms)
    ✓ Elements (1439 ms)
    ✓ Screenshot (1544 ms)
  View Book
    ✓ View Button (1473 ms)
    ✓ View Inputs (1433 ms)
    ✓ Back Button (1502 ms)
    ✓ Screenshot Inputs (1492 ms)
    ✓ Screenshot Buttons (1412 ms)
  Book Print
    ✓ Table Heading (1470 ms)
    ✓ Elements (1399 ms)
    ✓ Mark As Taken/Returned (7619 ms)
    ✓ Add/Delete Book Print (7609 ms)
    ✓ Screenshot (1536 ms)
  Edit Book
    ✓ Edit Button (1480 ms)
    ✓ Back Button (1468 ms)
    ✓ Screenshot Inputs (1567 ms)
    ✓ Screenshot Back Button (1444 ms)
    ✓ Submit Edit Data (25605 ms)
  Add/Delete Book
    ✓ Add Button (1408 ms)
    ✓ Add Inputs (1399 ms)
    ✓ Back Button (1468 ms)
    ✓ Screenshot Inputs (1490 ms)
    ✓ Screenshot Back Button (1496 ms)
    ✓ Submit and Delete New Data (15715 ms)

Test Suites: 1 passed, 1 total
Tests:       34 passed, 34 total
Snapshots:   0 total
Time:        102.074 s
Ran all test suites.

```

Дискусија

Puppeteer и Jest рамките ни дозволуваат на лесен и брз начин да се тестира голем дел од функционалностите на апликациите и како такви се значаен дел во креирањето на една апликацијата. Можноста со во било кое време да се провери на автоматизиран начин корисничкиот интерфејс на дадена апликација овозможува побрзо и поефикасно развивање на дадена замисла.

Од технички аспект, тестовите за корисничкиот интерфејс на дадена frontend апликација се лесни за имплементирање со помош на големиот број рамки за тестирање. Останува на развивачот на апликацијата да одреди кои рамки и кој вид на тестирање е најдобро.

Се надевам дека ова истражување ќе Ви помогне да добиете подобра слика за Puppeteer и Jest, за предностите кои ги нудат како и за нивното користење. Сметам дека во време на развој на различни видови апликации, голема предност е да имаме основни познавања за можностите што тестирањето ни ги нуди.

Користена литература

1. <https://geekflare.com/software-testing-tools/>
2. <https://help.apify.com/en/articles/3195646-when-to-use-puppeteer-scraper>
3. <https://lambdageeks.com/puppeteer-automation/>
4. <https://itnext.io/getting-started-using-puppeteer-headless-chrome-for-end-to-end-testing-8487718e4d97>
5. <https://flaviocopes.com/puppeteer/>
6. <https://www.softwaretestinghelp.com/jest-testing-tutorial/>
7. <https://jestjs.io/docs/api>
8. <https://medium.com/touch4it/end-to-end-testing-with-puppeteer-and-jest-ec8198145321>
9. <https://egghead.io/lessons/puppeteer-course-introduction-testing-with-puppeteer-and-jest>
10. <https://www.udemy.com/course/automated-headless-browser-testing-with-puppeteer/>
11. <https://www.loginradius.com/blog/async/e2e-testing-with-jest-puppeteer/>
12. <https://blog.logrocket.com/react-end-to-end-testing-jest-puppeteer/#whatispuppeteer>