Upravljanje pogreškama

Dragi polaznici, ova prezentacija nije primarno namijenjena za učenje već služi kao **pomoćni materijal**. Kao takva, ne može zamijeniti predavanja, literaturu kao niti vašu (pro)aktivnost na nastavi i konzultiranje s predavačem u slučaju potencijalnih pitanja i/ili nejasnoća.

Upravljanje kvalitetom

2 sata predavanja, 5 sati vježbi

Ključne riječi

Cjeline

Kvaliteta software-a

Otkrivanje pogrešaka

Optimizacija

Testiranje sa Jest-om

Vježba

Upravljanje pogreškama

Kontrola kvalitete

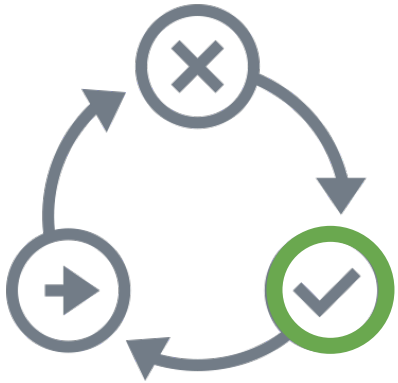
Za one koji žele znati više

https://smartbear.com/learn/automated-testing/best-practices-for-automation
http://www.bryntum.com/blog/javascript-quality-assurance-pt-2-finding-bugs/
https://www.jslint.com/
https://jestjs.io/

Zašto testirati kod?

Zato što želimo svojim klijentima dostaviti **kvalitetan** i **profesionalan** kod. Kao i sa svim u developmentu, što je ranije pronađen bug, to je manje sredstava, vremena i novca potrebno za njezino popravljanje.

--> https://five.agency/how-we-work/



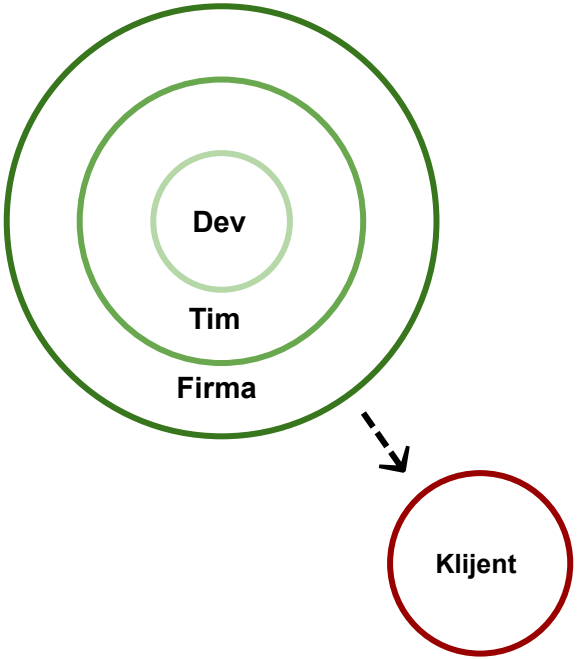
Tko sve može detektirati bug?

Developer - kod pisanja koda, kod testiranja aplikacije

Development tim - kod testiranja aplikacije, kod provjere koda ili pomoću CI paketa

QA tim u firmi - kod ručnog ili automatskog testiranja aplikacije

Klijent ili krajnji korisnik - najgori scenarij, bug u produkciji šteti ugledu firme i košta više za popraviti



Pojmovi

Quality control (QC) - kontrola kvalitete skup je aktivnosti koje kontroliraju kvalitetu proizvoda koji se razvija identificiranjem eventualnih grešaka.

Quality assurance (QA) - osiguranje kvalitete softwarea, postupak QC-a je podskup koji spada pod QA.

Test-driven development (TDD) - razvoj softwarea vođen testiranjem pristup je razvoju koji se sastoji od pisanja testova, praćenog proizvodnim kodom i ponovnog faktoriranja po potrebi.

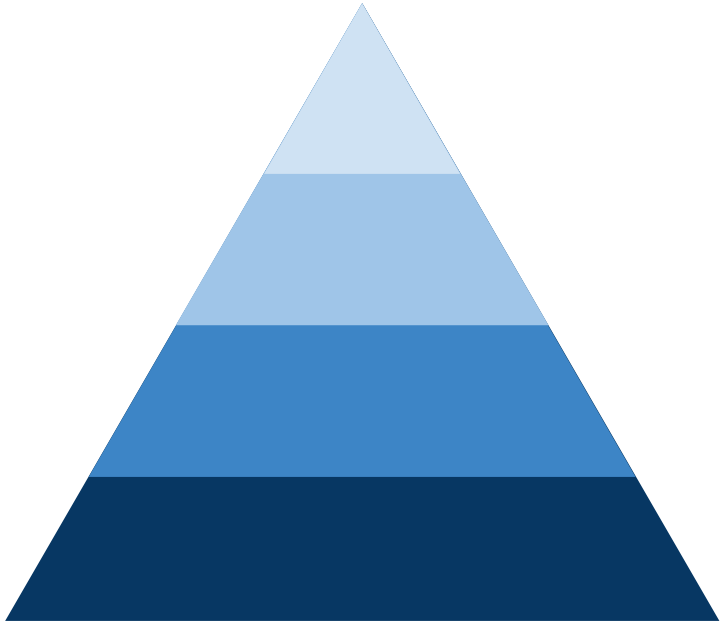
Razine testiranja

E2E - provjera aplikacije kroz sve sustave, od backenda do frontenda

INTEGRACIJA - provjera da više modula ispravno rade zajedno

UNIT - provjera da individualni dijelovi (moduli) funkcioniraju (u izolaciji), alat: test framework

STATIČNO - detekcija bugova i zatipaka dok se piše kod, alat: JSLint



Automatski testovi

Najviše koristi možete dobiti automatizacijom:

- repetitivnih testova
- testova koji imaju tendenciju biti uzrokovani ljudskom pogreškom
- često korištenih funkcija visokog rizika
- testova koji zahtijevaju puno vremena kad se izvode ručno

Kontrola kvalitete - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 25min

Upravljanje pogreškama

Otkrivanje pogrešaka

Za one koji žele znati više

https://en.wikipedia.org/wiki/Lint_%28software%29
https://en.wikipedia.org/wiki/Breakpoint
https://developers.google.com/web/tools/chrome-devtools/javascript/reference
https://www.jslint.com/

Linting

Linting je postupak provjere izvornog koda za programske, kao i stilističke pogreške.
Linter je program koji podržava linting (provjeru kvalitete koda).

1. Linter uzima izvor JavaScript i skenira ga
2. Ako pronađe sintaksni ili strukturni problem, vraća poruku koja opisuje problem i približnu lokaciju unutar izvora

Linteri

Linter	Prednosti	Nedostatci
JSLint	Možete odmah početi rad s linterom	Nema konfiguracijski file
JSHint	Može se konfigurirati, samo bazični ES6	Ne možemo dodati vlastito pravilo
JSCS	Može se posve customizirati po potrebi	Ne detektira sve sintaksne bugove
ESLint	Najbolja ES6 podrška, vrlo fleksibilan	Zahtijeva konfiguraciju, spor

JSHint

Koristi **.jshintrc** file (JSON) za konfiguraciju pravila linanja.
Također se može koristiti inline.

```
{
    "maxerr"        : 50,
    "bitwise"       : true,
    "camelcase"     : false,
    "curly"         : true,
    "eqeqeq"        : true,
    "forin"         : true,
    "immed"         : true,
    "indent"        : 4
}
```

Chrome DevTools

Chrome DevTools skup je alata za web programere ugrađenih izravno u preglednik Google Chrome i najčešće je korišten web developer alat.

Tab	**Uloga**
Elements	za HTML i CSS
Console	za debuggiranje grešaka i JS koda
Sources	Pregled svih izvora stranice i za postavljanje breakpointova
Performance	Praćenje redosljeda izvođenja i utjecaja na brzinu stranice
Network	Prikaz redosljeda loadanja resursa i detalja requesta

Točke prekida (breakpoints)

Točka prekida je namjerno mjesto zaustavljanja u programu, postavljeno u svrhu uklanjanja pogrešaka.

Neke od funkcija:

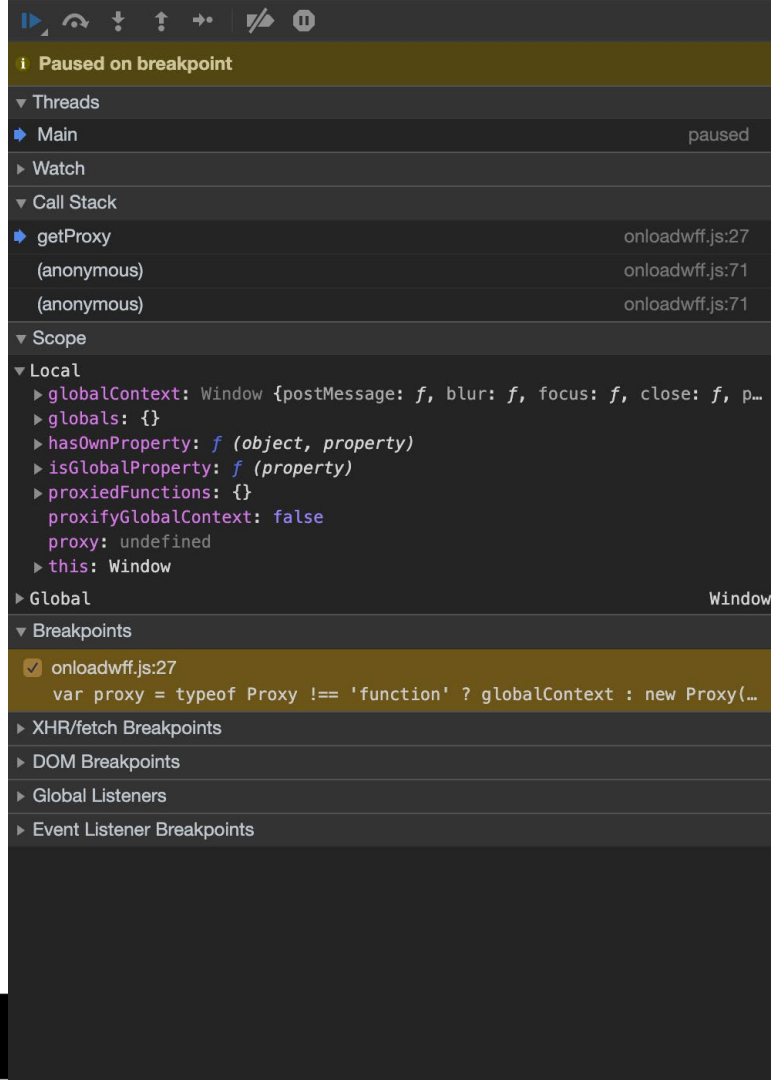
- set breakpoint
- remove breakpoint
- resume script execution
- pause script execution
- step over to next function call

Google DevTools

Sources

Prilikom breakpoint zastoja možemo dobiti informacije o:

- **threadu** na kojem se izvodi JS kod
- **call stack**-u koji je prethodio trenutnoj točci
- **scopu** varijabli



Otkrivanje pogrešaka - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 25min

Upravljanje pogreškama

Optimizacija

Za one koji žele znati više

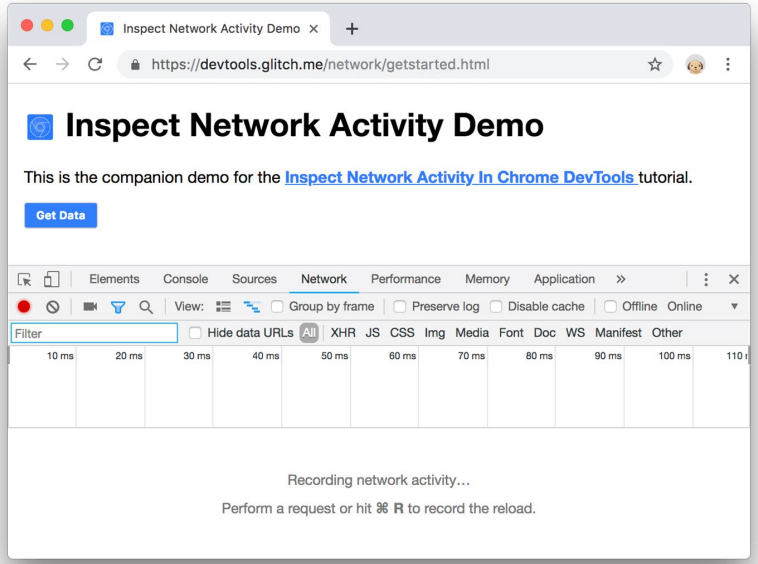
https://developers.google.com/web/tools/chrome-devtools/network/
https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/

Google DevTools

Network

Network nam daje informacije o loadanom resursu:

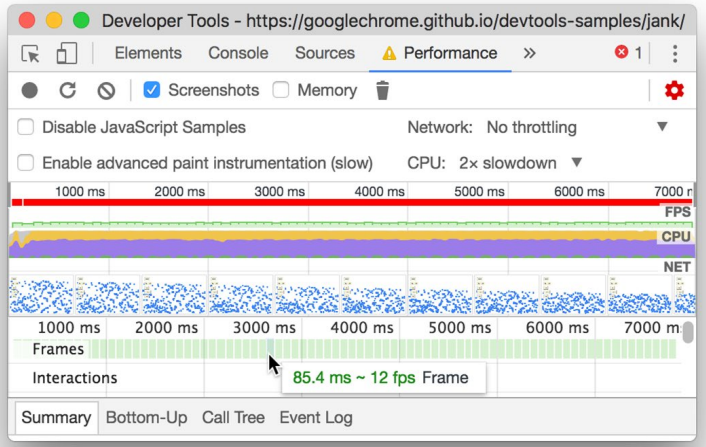
- poredak loadanja resursa
- HTTP status kod
- vrsti resursa
- što je uzrokovalo da se zatraži resurs
- koliko je trajao zahtjev



Google DevTools

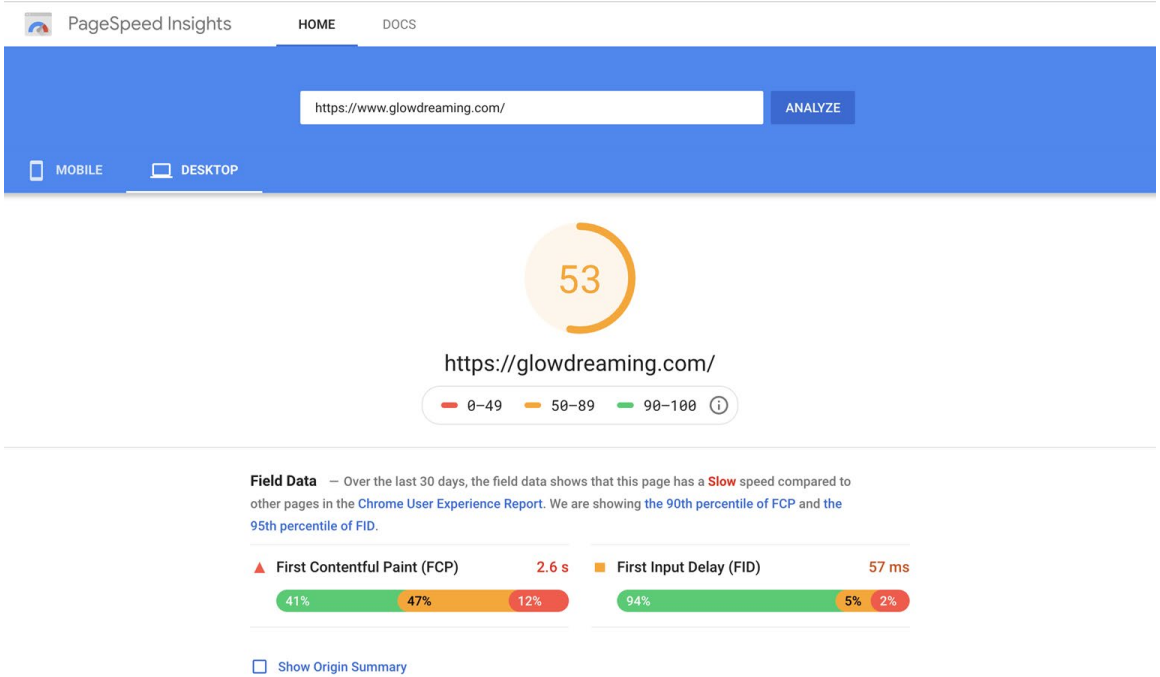
Performance

Performance tab analizira kako vaša stranica radi **dok se prikazuje** (za razliku od učitavanja).
Kompliciran je za shvatiti čak i za iskusnog developera i zahtijeva detaljno znanje o radu browsera.



Više o tome pročitajte na Chrome DevTools referentnoj stranici

PageSpeed Insights



PageSpeed Insights

Bilo koja stranica na internetu se može analizirati kroz Google alat zvan **PageInsights**. Ove su informacije važne i za SEO.

On nam daje informacije o:

- **ocjeni** koju joj Google dodjeljuje na temelju kategorija koje prate brzinu izvođenja (za desktop i za mobile)
- primijećenim **problemima** i predloženim koracima koji se mogu poduzeti kako bi se ocjena povisila i stranica ubrzala

Don't repeat yourself

"Svako znanje mora imati jedinstven, nedvosmislen, autoritativan prikaz unutar sustava"

Princip razvoja softvera usmjerenog na smanjenje ponavljanja uzoraka softvera, zamijenivši ga apstrakcijama ili koristeći normalizaciju podataka kako bi se izbjegla suvišnost.

Minifikacija JS-a

Loadanje JavaScript koda (kao i HTML-a i CSS-a) se može smanjiti **minifikacijom skripti**, što smanjuje veličinu datoteke.

```
// return random number between 1 and 6
function dieToss() {
  return Math.floor(Math.random() * 6) + 1;
}
// function returns a promise that succeeds if a 6 is
tossed
function tossASix() {
  return new RSVP.Promise(function(fulfill, reject) {
    var number = Math.floor(Math.random() * 6) + 1;
    if (number === 6) {
      fulfill(number);
    } else {
      reject(number);
    }
  });
}
// display toss result and launch another toss
function logAndTossAgain(toss) {
  console.log("Tossed a " + toss + ", need to try
again.");
  return tossASix();
}

function logSuccess(toss) {}
```

```
function dieToss(){return Math.floor(6*Math.random())+1}function
tossASix(){return new RSVP.Promise(function(a,b){var
c=Math.floor(6*Math.random())+1;6===c?a(c):b(c)})}function
logAndTossAgain(a){return console.log("Tossed a "+a+", need to try
again."),tossASix()}function logSuccess(a){}
```

Zapamtite

Za optimizaciju JavaScripta na vašoj stranici:

1. Pišite optimiziran i DRY JS kod
2. Loadajte JS skriptu na ispravnom mjestu u HTML-u
3. Minificirajte JS skripte na produkciji
4. Kad je primjereno, loadajte JS kod preko CDN-a

Optimizacija - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 30min

Upravljanje pogreškama

Testiranje koda

Za one koji žele znati više

https://jestjs.io/
https://wanago.io/2018/08/27/testing-javascript-tutorial-types-of-tests-of-unit-testing-with-jest/

Testiranje koda

Unit testing
Functional testing
Integration testing
System testing
Usability testing
Software performance testing
Load testing
Snapshot testing

Installation testing
Regression testing
Stress testing
Acceptance testing
Beta testing
Volume testing
Recovery testing

Jest test framework

```
test(imeTesta, () => {
  expect(testnaVrijednost).funkcijaPodudaranja;
});
```

Funkcije podudaranja

toBe, toEqual, toBeNull, toBeUndefined, toMatch, toBeClose, toContain, …

Jest test framework - primjer

sum.js

```
function sum(a, b) {
  return a + b;
}
module.exports = sum;
```

sum.test.js

```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Mock funkcije

Mock je tehnika zamjene dijelova koda koje ne možemo kontrolirati (npr. requesti prema API-ju) sa nečim što možemo predvidjeti i testirati.

```
test("returns undefined by default", () => {
  var mock = jest.fn();
  var result = mock("foo");

  expect(result).toBeUndefined();
  expect(mock).toHaveBeenCalled();
  expect(mock).toHaveBeenCalledTimes(1);
  expect(mock).toHaveBeenCalledWith("foo");
});
```

Jest test framework

Koraci instalacije

1. Instaliranje Jest-a preko npm-a
2. Dodavanje test naredbe u package.json
3. Pisanje testa (defaultno se stavljaju u __tests__ direktorij)
4. Pokretanje testa nad fileom
5. Ako je test prošao, super. Ako nije, onda poravak koda!

Testiranje sa Jest-om - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 25min