Web aplikacija za provjeru složenosti broja

Mihalj, Andrija

Undergraduate thesis / Završni rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:415812

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij računarstva

WEB APLIKACIJA ZA PROVJERU SLOŽENOSTI BROJA

Završni rad

Andrija Mihalj

Osijek, 2017.

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 1

2. ALATI 2

2.1. WAMP 2

2.2. Atom 3

2.3. HTML 3

2.4. Javascript 4

2.5. CSS 4

3. TEORIJSKA PODLOGA 6

3.1. Teorija složenih i prostih brojeva 6

3.2. Metode izračuna 7

3.2.1. Eratostenovo sito 7

3.3. Moderne metode izračuna (superračunala) 10

4. STRUKTURA I DIZAJN WEB APLIKACIJE 12

4.1. Kalkulator 13

4.2. Rezultat 17

4.3. Dizajn 18

5. ZAKLJUČAK 20

LITERATURA 21

SAŽETAK 22

ABSTRACT 23

ŽIVOTOPIS 24

PRILOZI 25

1. UVOD

Zadatak završnog rada je izraditi web aplikaciju za provjeru složenosti unesenog broja koristeći JavaScript programski jezik. Web aplikacija sastoji se od kalkulatora u koji se unosi željeni pozitivan cijeli broj koji prolazi kroz skriptirani jezik i izbacuje se rezultat provjere.

Provjera može imati četiri rezultata. Ako je broj složen, ispiše se da je takav te se ispiše njegov rastav na proste faktore kao objašnjenje. Ako je broj prost, ispiše se da je takav. Ako unesena vrijednost nije pozitivan cijeli broj, ispiše se da takav unos nije kompatibilan za ovaj kalkulator. Posljednje, ako je uneseni broj prevelik za predviđeni raspon ovog koda, ispiše se da je unesen prevelik broj.

Web aplikacija je otvorenog pristupa. U radu će se obraditi sama struktura cijele web aplikacije, uključujući kalkulator, rezultat i objašnjenje (u slučaju složenosti broja), njen dizajn te teorijska podloga iza tih izračuna i izračuna složenosti brojeva općenito.

1.1. Zadatak završnog rada

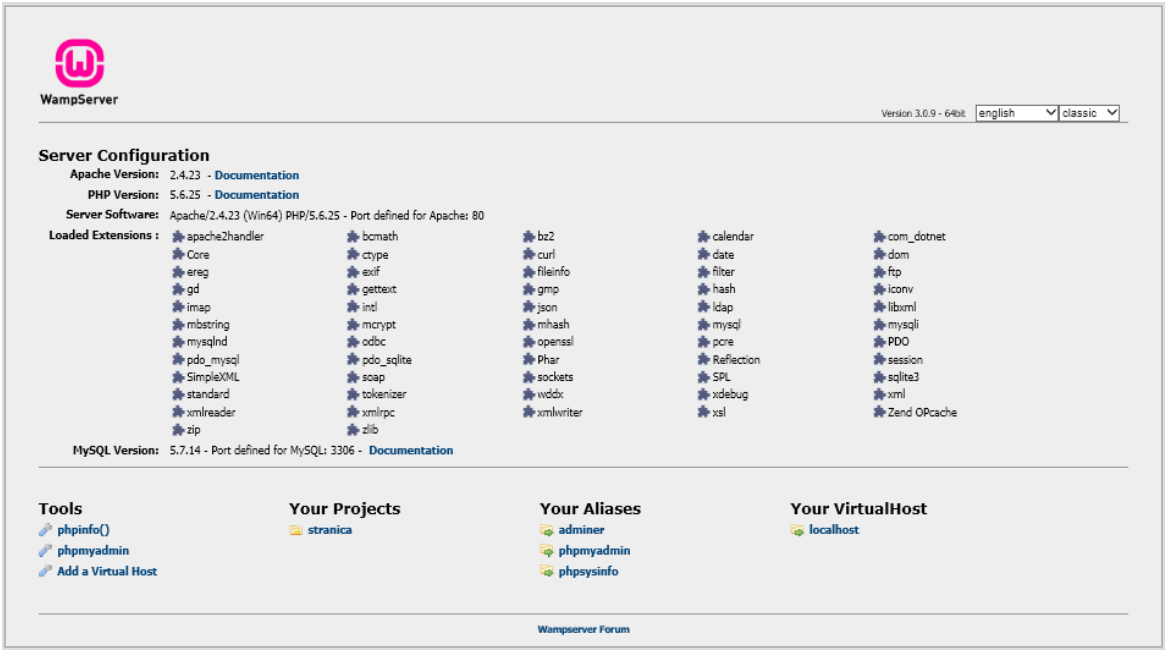
Potrebno je izraditi web aplikaciju u kojoj se za određeni pozitivan cijeli broj ispisuje je li prost ili složen. Zatim ako je složen se ispisuje objašnjenje složenosti rastavom na proste faktore.

2. ALATI

U ovom dijelu završnog rada predstavljaju se alati koji su korišteni uz njihovu sintaksu. Također, bit će prikazan i način upotrebe spomenutih alata na osnovnoj razini. Sam programski jezik upotrijebljen u stvaranju završnog rada detaljnije će se prikazati u nastavku rada.

2.1. WAMP

WAMP („*Windows, Apache, MySQL, PHP*“ [1]) je program koji se koristi za umjetno stvaranje lokalnog servera u svrhu pokretanja server-side skripti potrebnih za računalne jezike poput PHP-a i SQL-a. Prikazan na slici 2.1., osmišljen je za razvijanje i pokretanje dinamičkih web aplikacija, ali isključivo na Windows operativnom sustavu, kao što ime naslućuje. Apache je program koji oponaša server na lokalnom računalu namijenjenom za razvijanje web aplikacija te da se na njemu pokreću PHP skripte. Izvan pokretanja server-side PHP skripti, WAMP omogućuje stvaranje i upravljanje SQL bazama podataka uz program phpMyAdmin. phpMyAdmin programom upravlja se MySQL poslužiteljem baza podataka koji dopušta rad s bazama podataka uz ograničen pristup.



Sl. 2.1. *WAMP "Localhost".*

2.2. Atom

Atom je besplatan uređivač koda otvorenog koda. Podržava razne računalne jezike i kodove, uključujući: C/C++, C#, CSS, Git, HTML, JavaScript, Java, PHP, Python, Ruby on Rails, SQL, XML itd. [2]. Atom podržava više operativnih sustava. Temeljen je na *Electron* programu. Pisan je u *CoffeScript*-u i *Less*-u. Može se koristiti i kao integrirano okruženje za razvijanje (tzv. IDE (eng. *Integrated Development Environment*).

2.3. HTML

HTML (eng. *Hyper Text Markup Language*) je opisni jezik za izradu web aplikacija i stranica. Primarna sredstva pri stvaranju koda su oznake (eng. *tags*) pomoću kojih se Internet pregledniku daje informacija o vrsti i tipu elemenata koji se koriste. Neki od HTML elemenata su: <*button*>, <*body*>, <*title*>, <*form*> i sl. Postoje dvije verzije HTML elemenata, odnosno otvarajuće i zatvarajuće oznake. Otvarajuće počinju znakom <, a završavaju > (npr. <*button*>), a zatvarajuće započinju znakom </ te završavaju > (npr. </*button*>). Slično tome, postoje i samozatvarajući elementi kojima nije potrebna zatvarajuća oznaka (npr. <*link* ... />). Svaki element može imati i atribute kojima se pobliže definira taj element te se tako mogu definirati ime elementa (*name*=*"... "*), klasa (*class*=*"... "*), ID (*id*=*"... "*) i drugo. Svi atributi se stavljaju unutar znakova < i > otvarajućih oznaka. HTML nije programski jezik što znači da se njime ne izvršavaju nikakvi zadaci, nego se samo opisuje sadržaj web stranice pa se tako njime mogu stvarati tablice, mijenjati boje i fontovi pojedinih elemenata, mogu se odvajati odlomci itd. Također se mogu odrediti stilovi, odnosno hoće li tekst biti podebljan, ukošen ili podvučen, te se mogu zadati i različiti tipovi naslova, ovisno o važnosti naslova. „Temeljna zadaća HTML jezika jest uputiti web preglednik kako prikazati hipertekst dokument. Pri tome se nastoji da taj dokument izgleda jednako bez obzira o kojemu je web pregledniku, računalu i operacijskom sustavu riječ. HTML datoteke su zapravo obične tekstualne datoteke, ekstenzija im je .html ili .htm. Poveznice unutar HTML dokumenata povezuju dokumente u uređenu hijerarhijsku strukturu i time određuju način na koji posjetitelj doživljava sadržaj stranica.“ [3]

2.4. Javascript

Javascript (slika 2.2.) je programski jezik kojeg izvršava web preglednik umjesto servera odnosno klijent umjesto poslužitelja [4]. Izvorno je razvijen kao alat za dodavanje interaktivnosti i grafika web stranicama. Kroz utjecaj programskih jezika Java i C, postaje vodeći jezik za izradu svega od jednostavnih animacija i grafika do igara i naprednih dinamičkih elemenata na web stranicama. Moguće ga je smjestiti bilo gdje unutar HTML koda i radi bez komunikacije sa poslužiteljem. Prednosti su mu velika brzina, jednostavnost i svestranost uz žrtvu sigurnosti i kompatibilnosti sa klijentom.

```
function provjeriProst(num) {
    if (num < 0) {
        num *= -1;
    }
    var pola = num / 2;
    for (var i = 0; i <= pola; i++) {
        if (num % i === 0) {
            broj_faktora++;
            if (broj_faktora > 1) {
                // ako je broj djeljiv s nekim drugim brojem osim 1, nije prost
                return false;
            }
        }
    }
    // ako broj nije djeljiv ni s jednim brojem manjim od svoje polovice, prost
    return true;
}
```

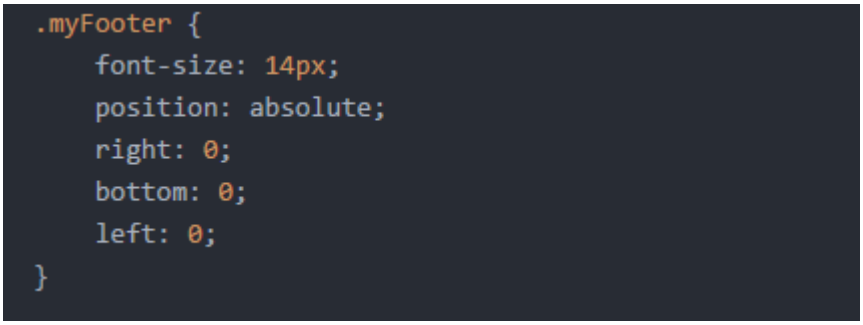
Sl. 2.2. *Javascript okruženje, unutar Atom alata.*

2.5. CSS

CSS je „stilski“ opisni jezik koji se koristi za oblikovanje web stranica [5].

Definira stil i veličinu teksta, stil pozadine, poravnanja i granica, ali i tablica, slika i drugih objekata. Koristeći CSS, web dizajner ima puno veću kontrolu nad izgledom stranice te znatno više opcija nego da web stranicu stilizira unutar samog HTML koda, kako je prije bila norma. Osim HTML

dokumenata, kompatibilan je i sa XML dokumentima. Uz HTML i Javascript čini temeljne jezike za izradu web stranica i aplikacija. Primjer CSS kôda prikazan na slici 2.3.



Sl. 2.3. *CSS jezik, unutar Atom alata.*

3. TEORIJSKA PODLOGA

Prirodan broj *x* djeljiv je prirodnim brojem *y* ako i samo ako postoji prirodan broj *z* takav da je *x* rezultat množenja brojeva *y* i *z* kao što je vidljivo u općoj formuli djeljivosti prirodnih brojeva (3-1).

$$x{:}\, y \;\Leftrightarrow x = y * z;\; x, y, z \in N \tag{3-1}$$

3.1. Teorija složenih i prostih brojeva

Svi prirodni brojevi koji su djeljivi isključivo sa brojem 1 i sa samim sobom, a veći od broja 1, nazivaju se prostim brojevima. Svi prirodni brojevi veći od broja 1 koji nisu prosti nazivaju se složenim brojevima. Točnije, prost broj je svaki prirodan broj veći od 1 koji ne može biti zapisan kao produkt dva manja prirodna broja, a svi ostali prirodni brojevi su složeni i sačinjeni od prostih faktora. Time se svi prirodni brojevi dijele na broj 1, proste i složene kao što je opisano među osnovnim teoremima aritmetike [6].

Euklidov teorem opisuje kako prostih brojeva ima beskonačno mnogo. Teorem to dokazuje koristeći se argumentom *reductio ad absurdum* [7] odnosno indirektnim matematičkim dokazom:

> Pretpostavlja se postojanje konačno mnogo prostih brojeva: $p_1, p_2, p_3, \dots, p_n$
> Pomnože li se svi prosti brojevi (kojih je, po pretpostavci, konačno mnogo) dobije se broj *N* koji je očito veći od svih prostih brojeva. Budući da se *N* sastoji od svih prostih brojeva, on ne može biti prost tj. složen je. Nadalje, ako se broju *N* doda broj 1, taj novi broj *M* (3-2)., bi morao biti djeljiv sa ijednim postojećim prostim brojem u konačnom nizu (kao složeni broj) ili sam biti prost. U slučaju da je *M* prost, dokazan je novi prosti broj što krši pretpostavku, a u slučaju da je složen, pri dijeljenju broja *M* s nekim prostim brojem p_m (gdje je $1 \leq m \leq n$) dobije se ostatak 1, što krši pretpostavku zato što je onda potrebno definirati novi prosti broj kojim će M biti djeljiv bez ostatka u skladu s osnovnim teoremima aritmetike.
>
> $$M = p_1 * p_2 * p_3 * \dots * p_n + 1;\;\; (\text{npr. } M = 2 * 3 * 5 + 1 = 31) \tag{3-2}$$

Euklidov teorem o beskonačnosti prostih brojeva pruža temelj za traženje istih raznim matematičkim metodama.

3.2. Metode izračuna

Prosti brojevi, iako beskonačni, nisu potpuno nasumični. Na primjer, svi prosti brojevi su ujedno i neparni (uz iznimku broja 2), odnosno završavaju s 1, 3, 7 ili 9. Slijedeći te uzorke rađaju se algoritamske metode traženja prostih brojeva. Postoje različite metode za izračun tj. traženje prostih brojeva. Najjednostavnija i najmanje efikasna metoda je tzv. *brute force* metoda tj. dijeljenje nekog nasumičnog broja manjim, poznatim, prostim brojevima ne bi li se pokazao prostim. U matematičkoj teoriji brojeva prosti brojevi traže se algoritmima čija se složenost mjeri na temelju vremena potrebnog da se izvrše, to vrijeme označavamo kao *O(N)* gdje N predstavlja veličinu skupa. Ti algoritmi se nazivaju sita. Ime su dobili zbog načina rada jer filtriraju složene brojeve, a ostaju samo prosti.

Najjednostavnije i najstarije je tzv. Eratostenovo sito (250. g. pr. Krista), ali postoje i Sundaramovo sito (1934. g.), Atkinovo sito (2004. g.) i različite varijacije istih za složenije izračune. Ovisno o veličini skupa N unutar kojeg tražimo proste brojeve, koriste se sita više odnosno niže složenosti i brzine.

3.2.1. Eratostenovo sito

Eratostenovo sito korišteno je u izradi ove web stranice. Ono nalazi sve proste brojeve unutar skupa prirodnih brojeva veličine N unutar vremena T opisanog u formuli (3-3) koja može varirati.

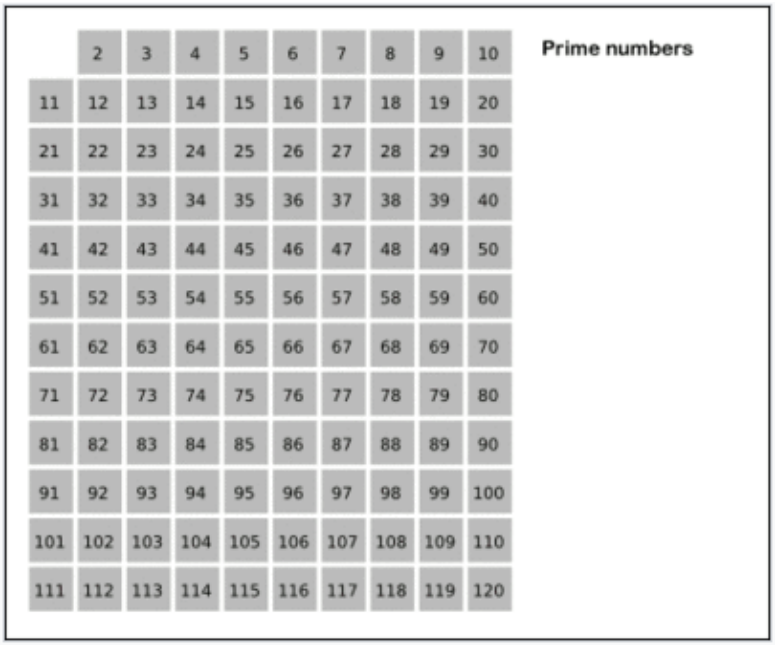
$$T(N) = O(N \log\log N) \tag{3-3}$$

Zna se da je to vremenska složenost klasičnog Eratostenovog sita pogledom u sam kôd. Analizom *for* petlje sita vidi se da ona ima složenost kao u (3-4), gdje je *i* skup svih prostih brojeva manjih od *N* počevši od broja 2.

$$T(N) = O\left(\sum_{i=2}^{N} \frac{N}{i}\right) \tag{3-4}$$

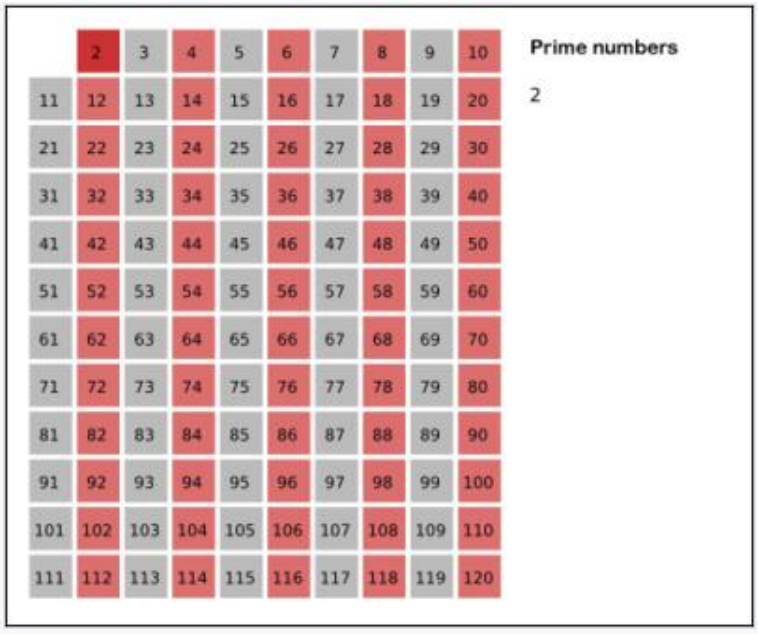
Sito funkcionira tako da redom označava brojeve koji su složeni tj. sadrže više od jednog prostog faktora, počevši s prostim brojem 2, dok ne ostanu samo neoznačeni, prosti brojevi.

Postupak je jednostavan, odabere se nekakav skup brojeva *N* (npr. broj 120) i u polje brojeva se ispišu svi brojevi od 2 do 120 kao na slici 3.1.



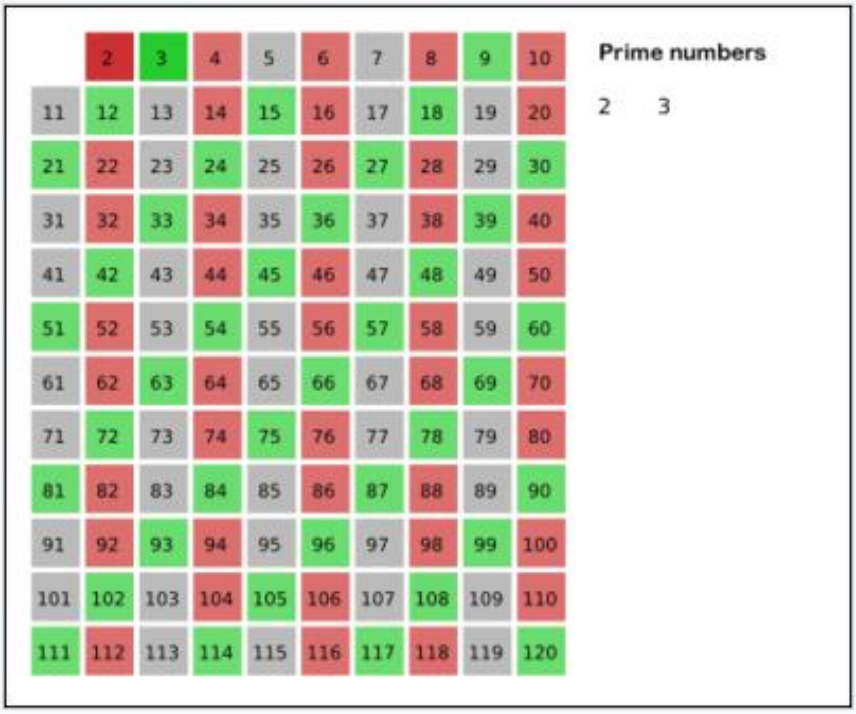
Sl. 3.1. *Ispis polja N [9].*

Zatim se odabere prvi broj u polju, broj 2, i označe svi brojevi u polju N kojima je broj 2 jedan od prostih faktora kao na slici 3.2.



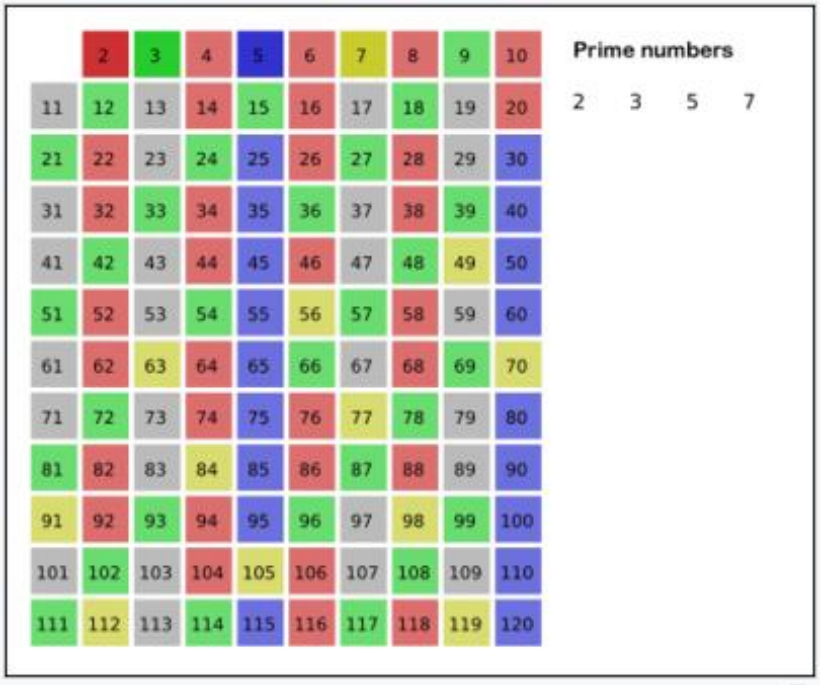
Sl. 3.2*. Označavanje svih višekratnika broja 2 [9].*

Odabere se idući prosti broj, broj 3, i označe svi brojevi u polju N kojima je broj 3 jedan od prostih faktora kao na slici 3.3.



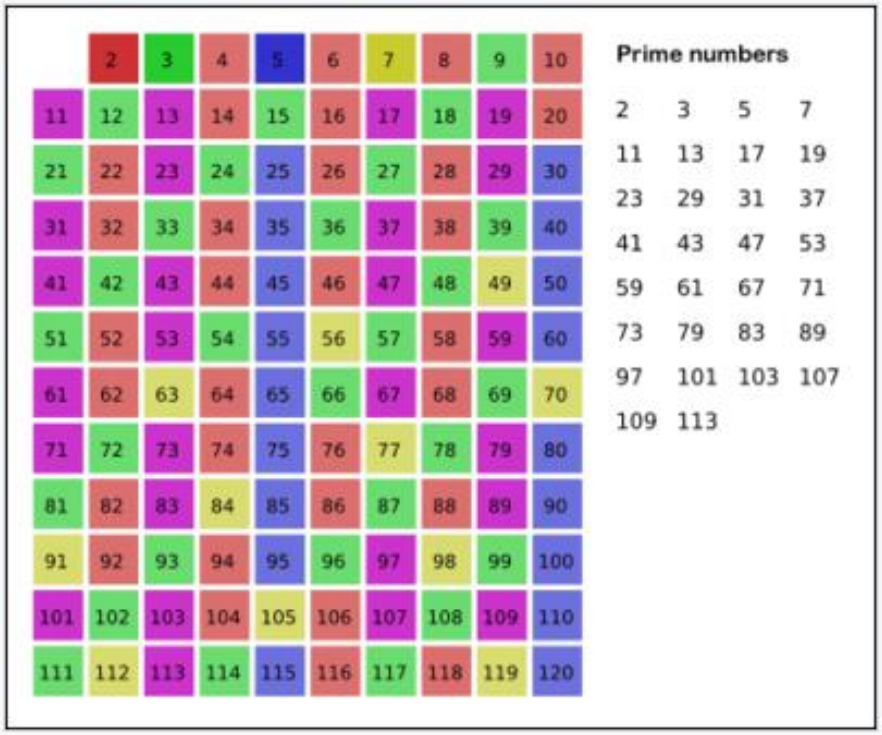
Sl. 3.3. *Označavanje svih višekratnika broja 2 i 3 [9].*

Slijede prosti brojevi 5 i 7 odnosno svi brojevi kojima su oni višekratnici, označeni na slici 3.4.



Sl. 3.4. *Označavanje svih višekratnika brojeva 5 i 7 [9].*

Naposljetku ostaju isključivo prosti brojevi koji se onda ispišu u niz pronađenih prostih brojeva kao što je vidljivo na slici 3.5.



Sl. 3.5. *Označavanje svih prostih brojeva [9].*

Eratostenovo sito je zbog svoje jednostavnosti ograničeno na manje veličine polja budući da ima vremenski i memorijski efikasnijih sita.

3.3. Moderne metode izračuna (superračunala)

Superračunala su glavna metoda izračuna novih prostih brojeva u 21. stoljeću. Koristeći moćne procesore industrijske snage povezane u podzemna mora računala, znanstvenici i matematičari mogu obaviti najsloženije izračune koje je moguće zamisliti ali i najmučnije. Sa visokim razinama vremenske složenosti (*O(N)*) algoritmi za izračun prostih brojeva su najbolje svladani mrežom stotina računala koji obavljaju milijune izračuna u isto vrijeme, filtrirajući brojeve kroz algoritamska sita. Primjer jednog takvog superračunala je „Bura“ koja pripada analitičkom odjelu riječkog sveučilišta, vidljiva na slici 3.6.



***Sl. 3.6**. Superračunalo riječkog sveučilišta - „Bura“ [10].*

Superračunala najčešće pronalaze tzv. Mersenneove proste brojeve, odnosno proste brojeve koji su opisani jednadžbom (3-5).

$$M_n = 2^n - 1, n \in N \tag{3-5}$$

Za vrijeme pisanja ovog rada najveći pronađeni prosti broj je $M_n = 2^{74,207,281} - 1$, pronađen je u kolovozu 2017. i gotovo je 5 milijuna znamenki veći od prošlog najvećeg prostog broja [11].

4. STRUKTURA I DIZAJN WEB APLIKACIJE

Ideja rada je izraditi robusnu web aplikaciju za računanje stupnja složenosti nekog broja uz prikaz rastava na proste faktore. Web aplikacija je pisana Javascript programskim jezikom kako bi se oslonila na računalnu snagu klijenta umjesto da složene izračune šalje preko servera. Budući da je Javascript jezik sa ograničenom maksimalnom veličinom broja od $n = 2^{53} - 1$, bilo bi moguće napraviti bazu podatka svih znanih prostih brojeva od 2 do $2^{53} - 1$ i tako imati pripremljene sve moguće odgovore na upite u kalkulatoru. Međutim, takav pristup je izuzetno neefikasan i stoga je za ovaj rad korišteno Eratostenovo sito, kao što je navedeno ranije.

Logika web aplikacije stvorena je u Atom razvojnom okruženju koristeći se knjižnicama podataka i predložaka za Javascript poput jQuery i Vue.js koje su uokvirene HTML i CSS opisnim jezicima.

Sučelje web stranice je minimalističko radi dostizanja najbržih mogućih rezultata izračuna. Sastoji se od kalkulatora, gumba za potvrdu unosa i polja s rezultatima što je vidljivo na slici 4.1. Također sadrži kratki opis rada na *O stranici* odjeljku.



Sl. 4.1. *Sučelje web aplikacije.*

4.1. Kalkulator

Kalkulator prostih faktora pisan je u Javascript programskom jeziku jer je on izvrstan za matematičke izračune pogotovo zato što sve izračune odrađuje računalo klijenta umjesto server poslužitelja, što daje brže rezultate.

Kalkulator za izračun prostih faktora se sastoji od više funkcija pisanih u Javascriptu. Glavna funkcija poziva na sebe sve sporedne funkcije u kôdu i diktira ispis u polje s rezultatima, prikazano na slici 4.2.

```
// glavna funkcija
function main() {
    $(idIspis).empty();
    broj_faktora = 0;

    var unesenaVrijednost = $(idUnos).val();
    var pravaVrijednost = provjeriVrijednost(unesenaVrijednost);
    ispisi("<br/>"); ispisi("<hr/>"); ispisi("<br/>");
    switch (pravaVrijednost) {
        case "x":
            ispisi("Unesena vrijednost nije ispravna.");
            break;
        case 0:
            ispisi("0 nije prost ni slozen broj.");
            break;
        default:
            // provjeri da nije unesen prevelik broj
            if (pravaVrijednost > Number.MAX_SAFE_INTEGER) {
                ispisi("Unesen prevelik broj.")
                ispisi("<br />")
                ispisi("Javascript podrzava brojeve do (2 ^ 53) - 1=9007199254740991.");
                return;
            }
            // provjeri da broj nije prost
            if (provjeriProst(pravaVrijednost)) {
                ispisi("Broj je prost.");
                return;
            }
            ispisi(pronadjiProsteFaktore(pravaVrijednost));
            break;
    }


}
```

Sl. 4.2. *Glavna (eng. main) funkcija Javascript koda.*

Glavna funkcija zapravo je veliko *switch-case* odnosno *if-else* grananje koji na sebe poziva sve sporedne funkcije koje provjeravaju status kalkulatora u odnosu na uneseni broj kojeg prima.

Prva sporedna funkcija je *provjeriVrijednost()* koja provjerava je li unesena vrijednost uopće pozitivan prirodan broj. Funkcija vraća *x* ako vrijednost nije prirodan broj, npr. ako je *-1* ili *Osijek* upisano u polje) ili vraća nulu ako je vrijednost nula. Kod je vidljiv na slici 4.3.

```
// provjera unesene vrijednosti (negativni brojevi, neispravne vrijednosti)
function provjeriVrijednost(value) {
    try {
        var num = parseInt(value);
        // ako je unesena vrijednost koja nije broj
        if (isNaN(num)) {
            return "x";
        }
        else if (num === 0) {
            return 0;
        }
        else {
            return num;
        }
    }
    catch (e) {
        console.log(e);
    }
}
```

Sl. 4.3. *Sporedna funkcija provjeriVrijednost().*

Za ispravne vrijednosti funkcija vraća broj koji je unesen te njega ponovo prima glavna funkcija i sprema u varijablu *pravaVrijednost*. Glavna funkcija zatim provjerava je li uneseni broj koji je sada sigurno prirodan i veći od 0 prevelik za Javascript programski jezik. Uspoređuje se unesena varijablu s maksimalnom dopuštenom vrijednošću za Javascript matematičke operacije, odnosno s $n = 2^{53} - 1$ i ispisuje je li ona unesenim brojem prekoračena. Kod koji prikazuje opisano prikazan je na slici 4.4.

```
// provjeri da nije unesen prevelik broj
if (pravaVrijednost > Number.MAX_SAFE_INTEGER) {
    ispisi("Unesen prevelik broj.")
    ispisi("<br />")
    ispisi("Javascript podrzava brojeve do (2 ^ 53) - 1=9007199254740991.");
    return;
}
```

Sl. 4.4. *Provjera je li uneseni broj prevelik.*

U slučaju da je sve po pravilima unosa, glavna funkcija prepušta varijablu s prirodnim brojem sporednoj funkciji *provjeriProst()* koja provjerava je li uneseni broj prost ili složen. Za provjeru je li broj prost, kôd podijeli broj sa svakim brojem od 2 do polovice unesenog broja te, ako pronađe više od jednog faktora (ako izračuna da je broj djeljiv sa nekim drugim brojem osim broja 1), vraća vrijednost *false*. Način provjere je li broj prost tako što se dijeli sa svakim prirodnim brojem do svoje polovice vidljiv je na slici 4.5.

```
// provjerava je li broj prost
function provjeriProst(num) {
    if (num < 0) {
        num *= -1;
    }
    var pola = num / 2;
    for (var i = 0; i <= pola; i++) {
        if (num % i === 0) {
            broj_faktora++;
            if (broj_faktora > 1) {
                // ako je broj djeljiv s nekim drugim brojem osim 1, nije prost
                return false;
            }
        }
    }
    // ako broj nije djeljiv ni s jednim brojem manjim od svoje polovice, prost je
    return true;
}
```

Sl. 4.5. *Provjera je li broj prost.*

Za izračun složenosti broja u kôdu je izraženo tzv. Eratostenovo sito, aritmetički logaritam koji, procesom traženja prostih faktora i zanemarivanjem brojeva koji imaju više od jednog prostog faktora, nalazi proste brojeve unutar raspona od 2 do *n*, ali i proste faktore unesenog broja *n*.

Sporedna funkcija *pronadjiProsteFaktore()* unesenog broja *n*, vidljiva na slici 4.6., zapravo koristi svoju sporednu funkciju *najmanjiFaktor()*. Funkcija *pronadjiProsteFaktore()* zapravo samo ispisuje niz (eng. *string*) umnoška svih prostih faktora koji su nađeni Eratostenovim sitom (ako je broj složen) tako što rekurzivno poziva samu sebe, ponavljajući i/ili listajući sljedeće proste faktore od najmanjeg (*minFaktor*) do najvećeg. Za iznimke uzima negativnu vrijednost broja gdje vraća prvi najmanji prosti faktor kao negativan ili da je sam broj svoj vlastiti najmanji prosti faktor (npr. broj 3) gdje samo ispiše njega.

```
// vraca string umnoska svih prostih faktora
function pronadjiProsteFaktore(n) {
    // ako je unesena vrijednost negativna:
    if (n < 0) {
        return '-' + pronadjiProsteFaktore(-n);
    }
    // pronadji najmanji faktor kojim se broj moze dijeliti
    var minFaktor = najmanjiFaktor(n);

    // ako je broj jednak najmanjem faktoru s kojim se moze dijeliti,
    // zavrsi funkciju
    if (n === minFaktor) {
        return '' + n;
    }

    // rekurzivni poziv funkcije
    return minFaktor + ' * ' + pronadjiProsteFaktore(n / minFaktor);
}
```

Sl. 4.6. *Sporedna funkcija pronadjiProsteFaktore().*

Sporedna funkcija *najmanjiFaktor()* zapravo obavlja algoritam Eratostenovog sita tako što prvo provjerava je li poslani broj konačan, brojevna vrijednost, broj 0, broj 1 ili djeljiv s jednim od osnovnih prostih brojeva (2, 3, 5). Zatim traži kvadratni korijen unesenog broja i provjerava sve vrijednosti brojeva do korijena koristeći *for* petlju s *if* operatorima koji traže ostatak dijeljenja prvotno unesenog broja. Prvotni broj prolazi kroz *for* petlju i dijeli se sa svim prostim brojevima do 31 a zatim se

povećava granica za broj 30 i tako do granične vrijednosti tj. do pronalaska svih prostih faktora. Način rada Eratostenovog algoritma u sporednoj funkciji vidljiv je na slici 4.7.

```
// pronalazi najmanji faktor
function najmanjiFaktor(n) {
    // provjerava je li poslani broj konacan ili brojevna vrijednost
    if (isNaN(n) || !isFinite(n)) return NaN;
    if (n === 0) return 0;
    // provjerava proste vrijednosti
    if (n % 1 || n * n < 2) return 1;
    if (n % 2 === 0) return 2;
    if (n % 3 === 0) return 3;
    if (n % 5 === 0) return 5;
    // pronadji kvadratni korijen broja
    var m = Math.sqrt(n);
    // provjeri vrijednosti do korijena
    for (var i = 7; i <= m; i += 30) {
        if (n % i === 0) return i;
        if (n % (i + 4) === 0) return i + 4;
        if (n % (i + 6) === 0) return i + 6;
        if (n % (i + 10) === 0) return i + 10;
        if (n % (i + 12) === 0) return i + 12;
        if (n % (i + 16) === 0) return i + 16;
        if (n % (i + 22) === 0) return i + 22;
        if (n % (i + 24) === 0) return i + 24;
    }
    return n;
}
```

Sl. 4.7. *Sporedna funkcija najmanjiFaktor() koju poziva funkcija pronadjiProsteFaktore().*

4.2. Rezultat

Sâm ispis u polje *rezultat* odrađuje sporedna funkcija ispisa opisana u Javascript datoteci i vidljiva na slici 4.8.

```
// ispis rezultata
function ispisi(string) {
    $(idIspis).append("<div>" + string + "</div>");
}
```

Sl. 4.8. *Sporedna funkcija ispisi() za ispis rezultata.*

Mogući ishodi prolaska neke unesene vrijednosti kroz ovu web aplikaciju imaju pet verzija i ispisuju se navedenom *ispisi()* funkcijom. Funkcija vraća sljedeće izraze:

- „Unesena vrijednost nije ispravna“ za unose koji nisu brojevi
- „0 nije prost ni složen broj“ za unesenu nulu
- „Unesen prevelik broj. Javascript podržava brojeve do (2^53)-1.“ za prevelik unos
- „Broj je prost“ za unesen prosti broj
- „ prosti faktor * prosti faktor * … * prosti faktor“ za složene brojeve.

4.3. Dizajn

Dizajn web aplikacije minimalističkog je tipa radi osiguravanja što efikasnijeg izračuna kalkulatora prostog ili rastava složenog broja. Dizajn je odrađen djelomično u HTML opisnom (eng. *markup*) jeziku, a djelomično u CSS opisnom jeziku. Točnije, uz Bootstrap *framework* korištena je i *style.css* datoteka u kojoj su navedene promjene boje pozadine, fonta teksta i slično. Kao utjecaj su poslužili standardni stilovi web kalkulatora s naslovom, poljem za unos, gumbom za izračun i poljem rezultata. Stranica također sadrži sporedni dio s kratkim opisom njene svrhe. Kako bi se mogao koristiti Bootstrap *framework*, korišteni su *class* operatori za svaki HTML element. Oni rade tako da se svakom zada ime i zatim se putem tog imena određuje izgled elementa. To je korišteno za gumb i polje unosa, za tekst na glavnoj i sporednoj stranici te za logo fakulteta u podnožju stranice. Pomoću HTML-a također je implementirana ikona loga fakulteta i naziv stranice za kartice, vidljivo u obliku koda na slici 4.9. Ostatak vizualnog stila, točnije: boja pozadine, pojedinih elemenata, stil fonta, širine gumba, margina i sl., pripada *style.css* datoteci. Isječak CSS kôda web aplikacije prikazan je na slici 4.9. za glavne elemente web aplikacije.

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <link rel="stylesheet" type="text/css" href="style.css">
    <link rel="icon" href="https://www.ferit.unios.hr/images/favicon.ico">
    <script src="script.js"></script>

    <title>Prost ili slozen</title>
</head>
```

Sl. 4.9. *Isječak HTML kôda koji dodaje logo i naziv stranici vidljiv na karticama web preglednika.*

```css
body {
    background-color: #cee0ff;
    margin: 10px;
    font-size: 18px;
    text-align: center;
}

.inpt {
    margin-top: 40px;
    margin-bottom: 20px;
    padding: 10px;
    font-family: sans-serif;
    text-align: center;
    background-color: #8cb6ff;
    border: none;
    font-size: 20px;
    width: 700px;
}

.btn {
    margin-bottom: 20px;
    font-size: 22px;
    border: 2px black solid;
    border-radius: 5px;
    font-family: sans-serif;
}

    .btn:hover {
        cursor: pointer;
    }

.outpt {
    text-align: center;
}
```

Sl. 4.9. *Isječak style.css kôda s opisom polja unosa, gumba i polja rezultata.*

5. ZAKLJUČAK

U ovom je radu izrađena web aplikacija za provjeru složenosti unesenog broja koja vraća rastav na faktore u slučaju složenosti unesenog broja odnosno greške u slučaju unosa vrijednosti izvan predodređenih parametara.

Kako bi izrada web aplikacije izračuna složenosti broja bila moguća, trebalo se upoznati s CSS programskim jezikom, jQuery skriptnom knjižnicom podataka za Javascript, HTML opisnim jezikom putem Bootstrap *frameworka* te WAMP aplikacijom za umjetno stvaranje servera kojim provjeravamo ispravan rad web aplikacije. Kraj tehničkih potrebitosti, moralo se detaljno proučiti osnove aritmetike, teoriju brojeva i razrade algoritama – posebno Eratostenovog sita.

Zbog korištenja Javascript programskog jezika, web aplikacija ima svoja ograničenja ali je zato iznimno efikasna unutar raspona koji joj je dodijeljen. Aplikacija se izvršava na procesoru klijenta umjesto poslužitelja i stoga radi bez Internet veze. Izrada web aplikacije bila je relativno jednostavna nakon što je temeljito proučena teorijska podloga potrebna za izradu kalkulatora.

Cilj rada bio je izraditi web aplikaciju koja provjerava složenost unesenog broja uz njegov rastav i on je postignut.

Korištenjem programskog jezika sa širim rasponom maksimalnog broja mogao bi se programirati kalkulator šireg opsega ali bi tada bilo potrebno upotrijebiti neko efikasnije algoritamsko sito, npr. Atkinovo, te uspostaviti vezu između poslužitelja i klijenta. Točnije, Javascript se izvodi na klijentu, a nekakav opširniji jezik bi zahtijevao da se aplikacija izvodi na serveru.

LITERATURA

[1] WampServer, http://www.wampserver.com/en/ (pristupljeno 17.3.2017.)

[2] Wikpedia, Atom (text editor), https://en.wikipedia.org/wiki/Atom_(text_editor) (pristupljeno 17.3.2017.)

[3] Wikipedia, HTML, https://hr.wikipedia.org/wiki/HTML (pristupljeno 22.6.2017.)

[4] Wikipedia, JavaScript, https://hr.wikipedia.org/wiki/JavaScript (pristupljeno 22.6.2017.)

[5] Wikipedia, CSS, https://hr.wikipedia.org/wiki/CSS (pristupljeno 22.6.2017.)

[6] T. Tao, Structure and randomness in the prime numbers, UCLA 2007., http://www.math.ucla.edu/~tao/preprints/Slides/primes.pdf

[7] Wikipedia, Reductio ad absurdum, https://en.wikipedia.org/wiki/Reductio_ad_absurdum

[8] Wikipedia, Generating primes, https://en.wikipedia.org/wiki/Generating_primes

[9] Wikipedia, Sieve of Eratosthenes, https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

[10] Sveučilište u Rijeci, Centar za napredno računanje i modeliranje, https://cnrm.uniri.hr/

[11] Wikipedia, Mersenne prime, https://en.wikipedia.org/wiki/Mersenne_prime

SAŽETAK

Tema završnog rada je izrada web aplikacije za provjeru složenosti broja. Aplikacija određuje je li uneseni broj prost ili složen te ispisuje rastav na proste faktore složenog broja. Web aplikacija je rađena unutar Atom razvojnog okruženja programskim jezikom JavaScript, opisnim jezikom HTML i CSS te uz pomoć aplikacije WAMP koja služi za stvaranje umjetnih servera i baza podataka.

Ključne riječi: vue.js, jquery, css, html, javascript, složenost broja, prost broj, eratostenovo sito

ABSTRACT

Web application for verifying prime numbers

The topic of this final work is the making of a web application to verify if an entered number is a prime number and if not, dissolving it into factors. The web application was made in the Atom development program using the programming language JavaScript, HTML and CSS markup language and the WAMP application for creating artificial local servers and databases.

Keywords: vue.js, jquery, css, html, javascript, prime number, complex number, sieve of erathostenes

ŽIVOTOPIS

Andrija Mihalj rođen je 1. kolovoza 1994. godine u Osijeku, Hrvatska. Pohađa Osnovnu školu Josipa Antuna Ćolnića u Đakovu do 2009. te iste godine upisuje opći smjer Gimnazije A. G. Matoša u Đakovu. Srednju školu završava s vrlo dobrim uspjehom. Po završetku srednje škole, 2013. godine, upisuje tadašnji Elektrotehnički fakultet Osijek, danas Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, smjer računarstvo.

Izvrsno se služi engleskim jezikom te ima srednju razinu znanja za rad s Microsoft Office alatima. Razina znanja za rad s programskim jezicima C, C++ i C# je osnovna, dok ima srednju razinu znanja s programskim jezicima PHP i Javascript te opisnim jezicima HTML i CSS.

Dobitnik je stipendije od strane privatnih poduzetnika za potporu obrazovanja studenata na elektrotehničkim smjerovima za akademske godine 2014./15., 2015./16. te 2016./17. Od radnog iskustva, treba spomenuti da je radio kao radnik u skladištu za HEMCO d.o.o. u Đakovu u sklopu ljetne sezone te kao uspješan agent u teleprodajnom centru za Hrvatski Telekom u Osijeku u trajanju od 2 godine.

Andrija Mihalj

_____

PRILOZI

CD:

Završni rad u .doc i .pdf formatu

Programski kôd .js formatu

HTML struktura u .html formatu

Dizajn stranice u .css formatu