Usporedba alata za upravljanje Javascript modulima

Teklić, Matea

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split, Faculty of Science / Sveučilište u Splitu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:166:646927

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-24**

Repository / Repozitorij:

Repository of Faculty of Science

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

USPOREDBA ALATA ZA UPRAVLJANJE JAVASCRIPT MODULIMA

Matea Teklić

Split, rujan 2023.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

USPOREDBA ALATA ZA UPRAVLJANJE JAVASCRIPT MODULIMA

Matea Teklić

SAŽETAK

Rad se bavi usporedbom alata za upravljanje Javascript modulima. Opisana je arhitektura web aplikacija te su prikazani alati kao rješenje problema koji se pojavljuju tijekom izrade web aplikacije. Istaknuti su neki od alata za upravljanje modulima te detaljno obrađeni kako bi se uočila vrijednost svakog od njih kao i prednosti odnosno mane. Kroz praktični dio su primijenjeni alati zatim izrađena usporedba na temelju koje se donose zaključci za daljnje obrađivanje. Aplikacija koja je korištena za praktični dio izrađena je Javascript jezikom.

Ključne riječi: Web aplikacija, Javascript, alati

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 29 stranica, 22 grafička prikaza, 1 tablica i 14 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Doc. dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Doc. dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Doc. dr. sc. Divna Krpan, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Doc. dr. sc. Monika Mladenović, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **Rujan, 2023.**

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Informatics
Ruđera Boškovića 33, 21000 Split, Croatia

COMPARISON OF JAVASCRIPT MODULE BUNDLERS

Matea Teklić

ABSTRACT

This paper is concentrated comparison of tools for managing Javascript modules. The architecture of web applications is described and tools are presented as a solution to problems that arise during the creation of a web application. Some of the module tools are highlighted and elaborated to see the value of each of them as well as the advantages and disadvantages. Through the practical part, the tools were applied, then a comparison was made, on the basis of which conclusions are drawn for further processing. The application that was used for the practical part was created using the Javascript language.

Key words: Web application, Javascript, tools

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 29 pages, 22 figures, 1 table and 14 references

Original language: Croatian

Mentor: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Reviewers: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Divna Krpan, Ph.D. *Assistant Professor of Faculty of Science, University of Split*

Monika Mladenović, Ph.D. *Assistant Professor of Faculty of Science, University of Split*

Thesis accepted: **September, 2023.**

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam završni rad s naslovom USPOREDBA ALATA ZA UPRAVLJANJE JAVASCRIPT MODULIMA izradila samostalno pod voditeljstvom doc. dr. sc. Gorana Zaharije. U radu sam primijenila metodologiju znanstvenoistraživačkog rada i koristila literaturu koja je navedena na kraju završnog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući navela u završnom radu na uobičajen, standardan način citirala sam i povezala s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Studentica

Matea Teklić

ZAHVALA

Prije svega, želim podijeliti zahvalnost svima koji su mi pružili podršku da uspješno privedem kraju svoj prijediplomski studij.

Zahvaljujem se mentoru i profesoru ,doc.dr.sc. Goranu Zahariji, na pomoći, trudu, jasnim smjernicama prilikom pisanja rada te neumornog prenošenja znanja kroz brojne kolegije koje ste predavali.

Veliko hvala upućujem svojim roditeljima i braći, koji su mi omogućili da se fokusiram na svoje akademske obaveze te mi pružili neprestanu podršku tijekom studiranja.

Također, zahvalila bih se svojim kumama, prijateljicama i prijateljima, kolegama i kolegicama koji su mi pomogli da uspješno prijeđem brojne prepreke. Vaše riječi potpore su me motivirale za jaču borbu do kraja.

Na kraju bih se zahvalila svim profesorima koji su pridonijeli da steknem znanje za daljnje napredovanje.

Sadržaj

Uvod

U digitalno doba web aplikacije igraju ključnu ulogu u svakodnevnom životu. JavaScript kao nezaobilazni jezik za izradu klijentske strane, omogućuje interaktivnost i bogatstvo sadržaja na webu. S rastom kompleksnosti web aplikacija, pojavila se potreba za efikasnim načinima upravljanja JavaScript modula. U ovom radu veliku važnost imaju alati koji upravljaju modulima (engl. *bundlers*) te su ponuđeni kao rješenje problema s kojim se većina današnjih developera suočava. Jedan od važnih zadataka koje alati imaju je da grupiraju više Javascript modula zatim ih spoje u jednu ili više datoteka. Razlog zbog kojeg se upotrebljavaju alati jest da se umanje slanje velike količine zahtjeva te da skrate vrijeme učitavanja određene web aplikacije.

Ovom završnom radu cilj je istražiti i usporediti različite JavaScript alate u kontekstu web aplikacija. Prije početka detaljnog analiziranja alata, važno je razumjeti arhitekturu modernih web aplikacija. Web arhitektura, u širem smislu, predstavlja strukturu i organizaciju sustava za dostavljanje sadržaja na internetu. Ona se sastoji od različitih slojeva i komponenti koje međusobno surađuju kako bi osigurale optimalno korisničko iskustvo.

Prvo je prikazana analiza za alate pojedinačno poput Webpack, Rollup , Parcel, Esbuild i Browserify te istraživanje njihovih značajki i primjene u praksi. Nakon toga slijedi usporedba njihovih performansi u različitim scenarijima kako bi se identificirale prednosti i mane svakog od alata. Dalje, prikazana je analiza fleksibilnosti konfiguracije kako bi se saznali rezultati prilagođavanja promjena zahtjeva u razvoju aplikacija. Na kraju se donose zaključak i preporuke za odabir najboljeg alata za određeni tip web aplikacija.

Ovo istraživanje ima ključnu važnost za razvoj web aplikacija jer pruža programerima i timovima za razvoj uvid u najnovije alate i tehnike za upravljanje JavaScript modulima.

1. Arhitektura web aplikacija

Web aplikacije su softverski programi koji su pohranjeni na udaljenim poslužiteljima, a korisnici im mogu pristupiti korištenjem softvera poznatih kao web preglednici. Jedan od zadataka im je omogućiti korisnicima da obavljaju različite interakcije putem interneta.

Kada korisnik unese web adresu ili URL u preglednik, preglednik zatim šalje zahtjev poslužitelju koristeći HTTP (eng. HyperText Markup Language) protokol kako bi se pristupilo resursu koji odgovara web adresi koja se navela prethodno. Web poslužitelj je odgovoran za lociranje adrese traženog izvora koji može bit jedna web stranica (engl. *single web page*) ili početna stranica web aplikacije, te se pristupa bazi podataka ako su potrebne njezine informacije za obradu zahtjeva. Kao odgovor na poslani zahtjev šalje se HTML kod. HTML je jezik koji se koristi za formatiranje sadržaja web stranice. Odgovor se šalje pregledniku te on pomoću koda prikazuje sadržaj na zaslonu kao što su slike, izbornici, kontrole i sve informacije o traženoj stranici s izgledom i programiranoj funkcionalnosti na traženoj stranici ili web aplikaciji. Prema zadanim postavkama preglednik prikazuje HTML elemente s izvornim stilom kao što su boja pozadine, oblikovanje i ostale postavljenje postavke.

Za prilagodbu izgleda i dojma svih elemenata HTML dopušta uključivanje CSS (eng. Cascading Style Sheets), jezik grafičkog dizajna koji se koristi za definiranje stila dokumenata. CSS se razvio na takav način da dopušta uključivanje animacija i drugih naprednih efekata na web aplikacijama.

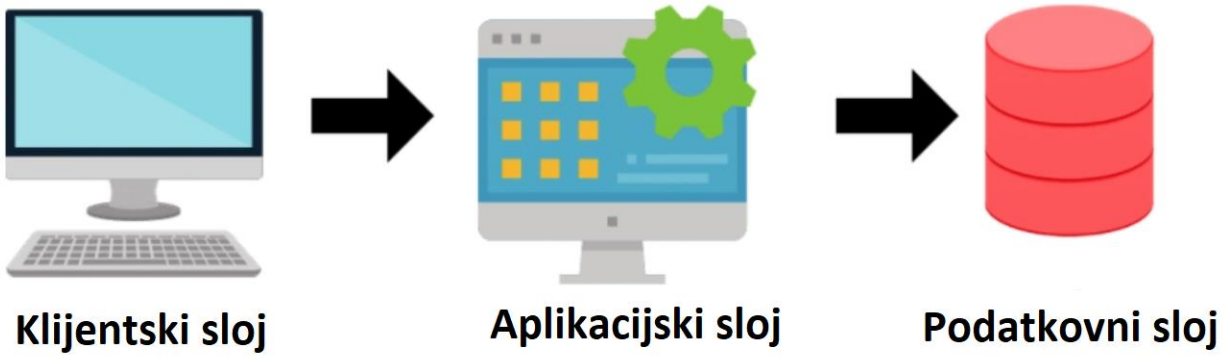
Kako web stranica prikazana u pregledniku ne bi bila statična, koristi se Javascript kod koji je uključen u HTML dokumente. Javascript je programski jezik koji omogućuje web stranicama traženje podataka, odgovaranje na korisničke događaje, prikazivanje ažuriranja sadržaja.

Postoji više vrsta web arhitektura kao što su troslojna web arhitektura, arhitektura aplikacije s jednom stranom (eng. *Single page application-SPA*) , arhitektura mikroservisa, arhitektura bez poslužitelja. U nastavku će biti opisana troslojna web arhitektura zato što je najviše korištena kad su u pitanju moderne web aplikacije.

1.1. Troslojna web arhitektura

Troslojna web arhitektura je arhitektura koja organizira aplikacije kako bi se olakšalo održavanje, modularnost i skalabilnost.

Većina modernih web aplikacija koristi troslojnu arhitekturu. U takvoj arhitekturi razlikujemo tri sloja kao što su: klijentski sloj, aplikacijski sloj i podatkovni sloj.



Slika 1.1 Prikaz troslojne web arhitekture

Klijentski sloj ili *frontend* - korisničko sučelje koji ima ulogu izravno komunicirati sa korisnikom. Uključuje sve elemente koji su prikazani na ekranu kao što su grafički elementi, navigacija te bilo koja druga interakcijska komponenta. Njegova glavna svrha je prikazivanje i prikupljanje informacija te prosljeđivanje potrebnih podataka aplikacijskom sloju.

Ovo sučelje se uglavnom sastoji od dijelova kao što su HTML, CSS i JavaScript. Ako želimo postići dinamičnost sučelja, možemo koristiti okvire poput React-a, Angular-a ,Vue-a, Bootstrap-a ali možemo koristiti i Javascript kojim možemo implementirati određenu dinamičnost.

Aplikacijski sloj ili *backend* – sloj koji je glavni dio i sadrži funkcionalnost web aplikacije. Ovaj sloj obrađuje zahtjeve koje zatraži klijentski sloj, izvodi poslovnu logiku i isporučuje potrebne podatke u obliku odgovora traženog zahtjeva. Za razliku od klijentskog sloja ,aplikacijski sloj nije vidljiv korisnicima te podržava sve operacije koje

omogućuju da sve pravilno funkcionira. Programi koji se najčešće koriste su C#, Python, Java, PHP i sl.

Podatkovni sloj – odgovoran je za upravljanje podacima i omogućuje komunikaciju s različitim bazama podataka. Pomaže u pohranjivanju i upravljanju podacima. Glavna svrha ovom sloju je izdvajanje logike pristupa podacima od ostalih slojeva. S tim se omogućuje jasna i efikasna komunikacija između aplikacijskog sloja i izvora podataka.

Pristup i upravljanje podacima ostvaruje se putem različitih programa, kao što su MongoDB, Oracle, MySQL i Microsoft SQL Server.

1.2. Izazovi u klijentskom sloju web aplikacija

Klijentski sloj web aplikacija donosi nekoliko izazova s kojima se razvojni timovi suočavaju. Ti izazovi proizlaze iz kompleksnosti modernih web aplikacija, brojnih uređaja i preglednika na koje se mora prilagoditi, te brzim učitavanjem stranica.

Neki od izazova s kojima se suočava klijentski sloj su: veliki broj Javascript datoteka, ovisnosti među modulima te performanse i optimizacija.

Kako bi prevladali izazove velikog broja JavaScript datoteka, preporuka je koristiti JavaScript bundlere. Bundleri mogu spojiti sve JavaScript datoteke u jedan ili više optimiziranih dokumenata, smanjujući broj HTTP zahtjeva i optimizirajući performanse aplikacije. Redovito testiranje i praćenje performansi također su ključni kako bi osigurali da aplikacija radi brzo i učinkovito, bez obzira na veličinu JavaScript koda.

2. Pregled popularnih JavaScript bundler-a

JavaScript bundleri su alati koji imaju zadatak grupiranja dijelova JavaScript koda i njihovih ovisnosti u jednu ili više datoteka. Njihova glavna svrha je smanjiti broj HTTP zahtjeva koje preglednik mora napraviti kako bi učitao web aplikaciju, te optimizirati performanse, brzinu učitavanja i korisničko iskustvo.

Kroz primjer će se implementirati potreba uvođenja bundler modula. Svaka aplikacija sastoji se od više Javascript datoteka, te se iste uključuju u <html> oznaku:

Slika 2.1. Primjer ulaznih Javascript datoteka u html oznakama

Svakoj od navedenih Javascript datoteka potrebna je komunikacija prema serveru i povratak odgovora za prikaz sadržaja na zaslonu. Kada postoji više navedenih Javascript datoteka, znatno se povećava vrijeme učitavanja web aplikacije.

Kako bi optimizirali učitavanje aplikacije, koriste se bundleri. Nakon primjenjivanja bundler-a u aplikaciji prikazani su slijedeći rezultati:

Slika 2.2 Rezultat nakon uporabe bundler-a

U prikazanim rezultatima vidljivo je da su sadržaji svih datoteka sa ekstenzijom *.js* spojeni u jednu. S tim je postignuto kraće učitavanje aplikacije jer prema poslužitelju ide samo jedan zahtjev koji se obrađuje.

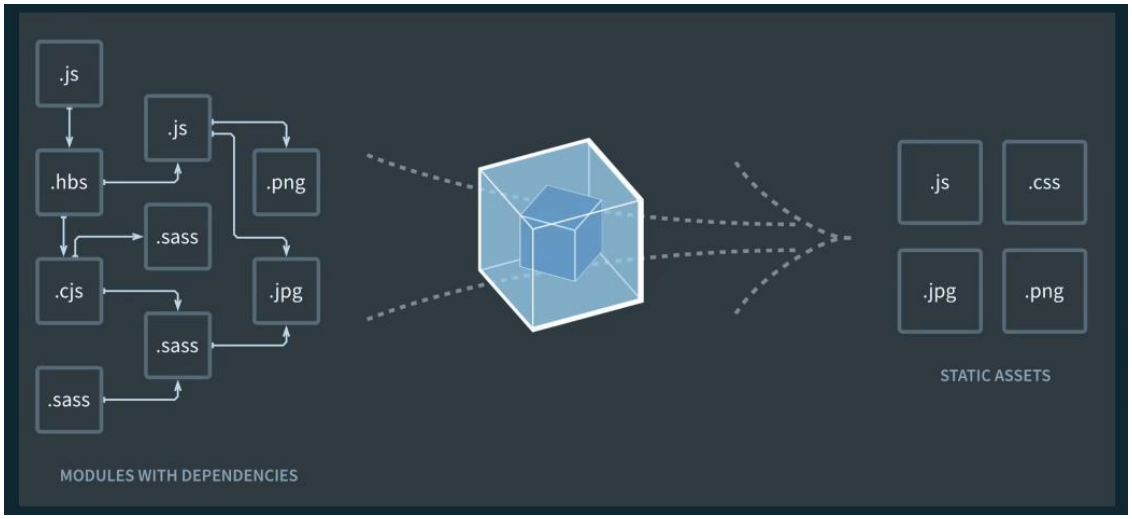
Bundleri se razlikuju po tome koliko su sposobni obraditi različite vrste datoteka koje nisu JavaScript (poput CSS-a, JSON-a, PNG-a, JPEG-a, XML-a itd.), jesu li u mogućnosti isporučiti npm pakete ili kompletnu aplikaciju te koliko podržavaju prilagodljivih konfiguracija.

Postoje različite vrste bundlera, kao što su Webpack, Parcel, Rollup, Esbuild, Vite, Browserify i FuseBox.

2.1. Webpack

Webpack je modul bundler za Javascript aplikacije. Zadatak mu je da umjesto učitavanja mnogo pojedinačnih datoteka na klijentskoj strani, sjedinjuje u jedan ili više paketa te ih poslužuje pregledniku.

Također prolazi kroz paket i stvara graf ovisnosti koji se sastoji od različitih modula koji su potrebni web aplikaciji da bi funkcionirala kako se očekuje.



Slika 2.3 Prikaz rada Webpack bundler-a

Na slici 2. prikazana je ideja rada Webpack-a. Na lijevoj strani slike poredani su različiti moduli s ovisnostima koje se koriste u aplikaciji. Kada se koristi bundler, datoteke istog

tipa grupiraju se u jedan paket na izlazu. To znači da više datoteka iste vrste koje su bile na ulazu, sada su spojene i nalaze se u jednoj datoteci na izlazu. Svrha takvog procesa je manje nepotrebno učitavanje jedne po jedne datoteke te kao ušteda brzine i veličine samog procesa pakiranja.

2.1.1. Prednosti

Pristup modulima – Webpack omogućuje razdvajanje aplikacije na manje, samostalne module. Tim pristupom postoji mogućnost organizacije koda i jasnija struktura projekta.

Loaderi – Pomoću loadera može se automatski prevesti JavaScript kod novije verzije u starije verzije kako bi bili podržani u starijim preglednicima koji ne podržavaju nove značajke.

Podrška za različite vrste datoteka – Webpack podržava razne tipove datoteka, uključujući JavaScript, CSS, slike, fontove itd. Takva podrška omogućuje fleksibilnost u radu s različitim tehnologijama i resursima.

Hot Module Replacement (HMR) – HMR omogućuje učinkovito ažuriranje modula tijekom razvoja. Kada se napravi promjenu u kodu, Webpack će automatski zamijeniti samo taj dio koda bez potrebe za ručnim osvježavanjem cijele stranice.

Optimizacija performansi – Webpack automatski obavlja optimizacije poput minimizacije koda (učinkovitije pakiranje i uklanjanje nepotrebnih dijelova), što rezultira manjom veličinom datoteka. Manje datoteke znače brže učitavanje stranica i bolju izvedbu web aplikacija.

2.1.2. Mane

Složenost konfiguracije - Webpack može biti dosta složen i teže razumljiv za početnike zbog mnogobrojnih opcija i postavki. Konfiguriranje alata za određene potrebe može biti složeno, a pogrešno podešavanje može dovesti do problema u izgradnji aplikacije.

Vrijeme izgradnje - Vrijeme izgradnje može biti dugotrajno za velike aplikacije koje u sebi sadrže dosta datoteka. Složenost aplikacije može uvelike usporiti sam proces izvršavanja.

2.2. Rollup

Rollup predstavlja bundler JavaScript modula nove generacije koji omogućuje konfiguriranje malih dijelova JavaScript koda u veće i složenije cjeline, poput biblioteka ili aplikacija. Njegova funkcionalnost temelji se na novom standardiziranom formatu modula koji je uključen u ES6 reviziju JavaScripta. Umjesto korištenja prethodnih rješenja poput CommonJS-a i AMD-a, Rollup koristi ES module, što donosi brojne prednosti. Sličan je alatima Browserify i Webpack.

Ono što Rollup čini posebno zanimljivim je njegova sposobnost stvaranja malih datoteka. U usporedbi s drugim alatima za izradu JavaScript paketa, Rollup gotovo uvijek generira manje i brže pakete.

2.2.1. Prednosti

Tree-shaking - oblik eliminacije neiskorištenog koda iz izlaznog modula. Ovo je posebno korisno u situacijama gdje koristite samo određene dijelove biblioteke, jer smanjuje veličinu koda koji se isporučuje korisnicima.

Brza izgradnja - Rollup je poznat po brzini izgradnje projekata te njegova efikasnost i optimizacija omogućuju brzo generiranje izlaznog koda.

2.2.2. Mane

Potreba za modernim preglednicima – poznato je da Rollup koristi ES module, konačni kod koji se generira zahtijeva podršku modernih preglednika. Ako je potrebno koristiti starije verzije preglednika, možda će se trebati tražiti dodatno rješenje kako bi se osigurala usklađenost.

2.3. Parcel

Parcel je popularan među programerima početnicima jer pruža jednostavan i intuitivan način za pakiranje web aplikacija. Njegova izvedba omogućava brzu izgradnju projekta. Posebno je popularan zbog svoje sposobnosti automatskog otkrivanja zavisnosti i uključivanja istih u izlazni paket, što znači da programeri ne moraju ručno upravljati

uvozima i ovisnostima. Sve to olakšava pokretanje i razvoj web aplikacija bez potrebe za daljnjim razumijevanjem složenih konfiguracija alata.

2.3.1. Prednosti

Jednostavnost konfiguracije - Parcel ne zahtijeva složene konfiguracije. Automatski prepoznaje datoteke u projektu i generira pakete bez potrebe za ručnim postavljanjem konfiguracijskih datoteka.

Niska prepreka za učenje - ističe se svojom jednostavnom i intuitivnom sintaksom koja olakšava razumijevanje i upotrebu, posebno za početnike u web razvoju.

2.3.2. Mane

Smanjena prilagodljivost u odnosu na druge alate - izgrađen je kako bi pružio jednostavnost i brzinu, što može značiti da nema toliko prilagodljivosti ili naprednih mogućnosti kao neki drugi alati za pakiranje kao što je Webpack.

Relativno novi alat - Parcel je relativno novi alat u usporedbi s nekim drugim alatima za pakiranje koji su bili na tržištu duže vrijeme. Iako je već dobro podržan, možda će biti potrebno neko vrijeme za provjerene metode i dokazane prakse.

2.4. Esbuild

Glavna značajka ovog alata je njegova izuzetna brzina izgradnje. Iako Esbuild ne pruža mnoge pogodnosti za razvojne programere koje se mogu pronaći u alatima poput create-react-app, sve više se pojavljuje podrška za Esbuild koja popunjava te praznine.

Važno je napomenuti da je Esbuild relativno nov alat. Još uvijek nije dosegao verziju 1.0 i nije u potpunosti spreman za produkcijsku upotrebu, iako je vrlo blizu. Pruža intuitivno sučelje za JavaScript i naredbeni redak s pametnim zadanim postavkama.

2.4.1. Prednosti

Efikasnost - Esbuild provodi snažni tree shaking, što rezultira manjim veličinama konačnih paketa. Ovo poboljšava brzinu učitavanja web stranica, jer se prenosi samo minimalno potreban kod.

Široka podrška za moderne JS funkcionalnosti - Esbuild podržava mnoge moderne funkcionalnosti JavaScripta, kao što su moduli, generatori i druge, tako da možete koristiti najnovije značajke bez brige o podrški preglednika.

2.4.2. Mane

Nedostatak naprednih funkcija - Esbuild je relativno nov alat, pa postoji mogućnost nedostatka određenih naprednih funkcija koje se mogu naći u drugim bundlerima, poput optimizacije koda ili detaljnijih izvještaja o greškama.

Nedostatak opširne dokumentacije - Budući da je Esbuild noviji alat, njegova dokumentacija možda nije toliko dobro organizirana kao kod drugih bundlera. To može stvoriti određene poteškoće za nove korisnike prilikom učenja i integracije.

2.5. Browserify

Browserify je alat za izgradnju paketa koji omogućuje programerima korištenje modula napisanih za Node.js u okviru web preglednika. U osnovi, Browserify omogućuje modularizaciju koda u Node.js stilu, gdje se JavaScript datoteke mogu podijeliti u manje module, a zatim te module mogu koristiti u drugim datotekama.

Browserify prolazi kroz nekoliko definiranih faza tijekom procesa povezivanja modula, što je uobičajeno za većinu JavaScript bundlera. Prva faza je stvaranje grafa ovisnosti. U ovoj fazi, Browserify počinje s navedenim ulaznim datotekama i traži sve "require()" pozive unutar tih datoteka. Svaki "require()" poziv se rješava na temelju putanje datoteke, a proces se ponavlja za svaki putanju datoteke koja se dalje koristi u drugim "require()" pozivima.

Kada je cijeli graf ovisnosti aplikacije potpuno mapiran, Browserify stvara samostalni paket koji se sastoji od spojenih datoteka s jedinstvenim ID-ovima.

Konačni paket se može postaviti u jedan <script> element za učitavanje u pregledniku.

2.5.1. Prednosti

Široka podrška za module - Browserify omogućava korištenje CommonJS modula u klijentskom sloju. To omogućava strukturiranje koda u manje module, olakšava ponovno korištenje koda i poboljšava organizaciju projekta.

Razdvojeni izlaz - Browserify može generirati razdvojeni izlaz ,što omogućava da se kod podijeli na manje datoteke kako bi se poboljšala brzina učitavanja stranice, posebno kada je riječ o velikim projektima.

2.5.2. Mane

Ograničena optimizacija - U usporedbi sa nekim drugim bundlerima, Browserify možda ne pruža toliko naprednih mogućnosti optimizacije i analize koda.

Smanjena brzina generiranja bundle-a - Browserify može generirati veće bundle datoteke u odnosu na neke druge bundlere. Ovo može dovesti do povećanog vremena učitavanja za krajnje korisnike, pogotovo ako bundle sadrži puno biblioteka.

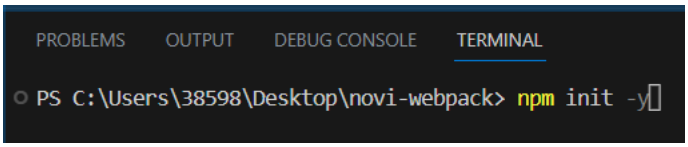
3. Primjeri korištenja bundler-a

U ovom dijelu, kroz primjere biti će prikazano korištenje bundler-a koji su opisani u prethodnom poglavlju. Opisati će se potrebna detaljna konfiguracija za svakog od bundler-a te nakon toga rezultati koji se dobiju nakon izvršavanja potrebnih naredbi.

3.1. Konfiguracije bundler-a

Webpack

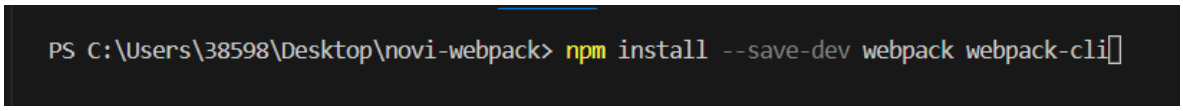
Prije početka stvaranja konfiguracije, u terminalu se stvara datoteka package.json pomoću naredbe :



Slika 3.1 Naredba za stvaranje datoteke *package.json*

U naredbi je upotrijebljen ***-y*** dodatak s kojim se automatski prihvaćaju zadane vrijednosti potrebnih podataka i generira se datoteka bez potvrde korisnika.

Nakon toga instalira se webpack i webpack CLI , a to se postiže naredbom :



Slika 3.2 Naredba za instalaciju Webpack i Webpack-CLI

Ova naredba koristi kraticu *npm* za instaliranje paketa. Webpack je paket koji sadrži sam webpack, dok je webpack-CLI paket koji pruža naredbeni redak za pokretanje webpacka putem terminala.

Opcija *--save-dev* osigurava da se paketi instaliraju kao razvojne ovisnosti i dodaju se u područje "devDependencies" u *package.json* datoteci. To znači da se ovi paketi koriste samo tijekom razvoja aplikacije, a ne za samu produkciju.

Nakon što se instalacija završi, webpack i webpack-CLI bit će dostupni za korištenje u projektu.

Sljedeći korak je stvoriti datoteku koja će se zvati *webpack.config.js*:

```
const path = require('path'); // uključivanje modula koji pruža funkcionalnosti za korištenje putanja do datoteka i direktorija na sustavu datoteka
const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin; // uključivanje Webpack plugin "BundleAnalyzerPlugin" koji omogućuje analizu i vizualizaciju veličine bundle datoteka

module.exports = {
  entry: './src/index.js', // Ulazna točka aplikacije
  output: {
    path: path.resolve(__dirname, 'dist'), // Izlazni direktorij i datoteka
    filename: 'bundle.js', //naziv generirane datoteke
  },
  mode: 'production',
  performance: {
    maxEntrypointSize: 512000, //postavljanje maksimalne ulazne veličine na 512000 bajtova
    maxAssetSize: 512000 //postavljanje maksimalne veličine izvora poput JavaScript-a, slika ili fontova na 512000 bajtova

  },
  module: {
    rules: [
    {
      test: /\.js$/, // Provjerava sve .js datoteke
      exclude: /node_modules/, // Isključuje node_modules direktorij
      use: {
        loader: 'babel-loader', // Koristi Babel loader za pretvaranje JavaScript koda
        options: {
            "presets": ["@babel/preset-env", "@babel/preset-react"] // Koristi Babel preset-ove za pretvaranje koda
        },
      },

    },
    {
      test: /\.html$/, // Provjerava sve .html datoteke
      use: "html-loader" // Koristi html-loader za obradu HTML-a
    },
    {
        test: /\.css$/, // Provjerava sve .css datoteke
        use: ['style-loader', 'css-loader'], // Koristi style-loader i css-loader za obradu CSS-a
      },
      }]},
   stats: {
    assets: true, // Prikazuje informacije o generiranim izlaznim datotekama
    modules: false, // Ne prikazuje informacije o modulima
    entrypoints: false, // Ne prikazuje informacije o ulaznim točkama aplikacije
    timings: true, // Prikazuje informacije o vremenu izvršavanja
```

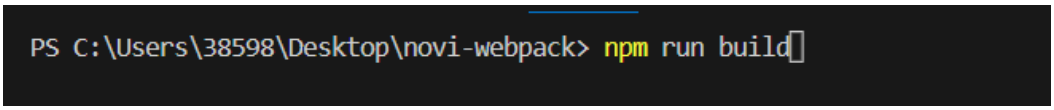
```
    performance: true, // Prikazuje upozorenja o
performansama aplikacije
  },
  plugins: [
    new BundleAnalyzerPlugin(), // Dodaje
BundleAnalyzerPlugin za generiranje vizualne analize veličine
paketa
]};
```

Kod 1. Konfiguracija Webpack

Ova konfiguracija definira kako će Webpack obraditi aplikaciju. Utvrđuje ulaznu točku *(entry)*, izlazne datoteke (*output*), način rada (*mode*), pravila za obradu različitih vrsta datoteka (*module.rules*), postavke statistika (*stats*), i dodatne plugine (*plugins*).

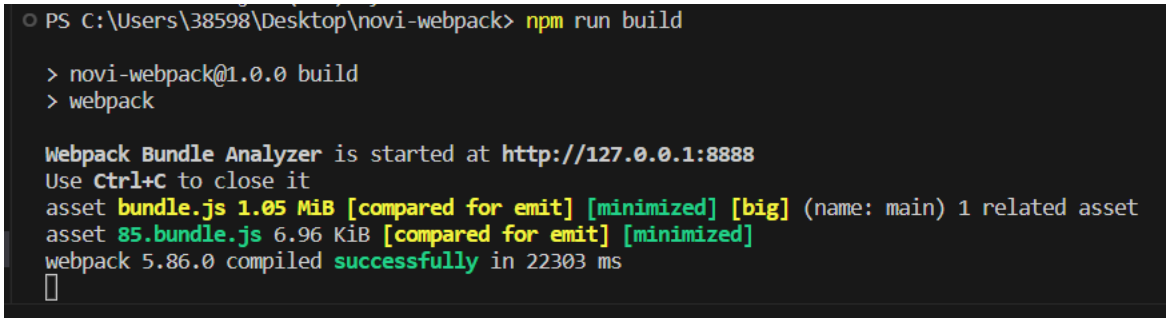
Nakon što je dovršena konfiguracija, prelazi se na dodavanje source (src) foldera. U njega se dodaju sve datoteke koje se koriste u aplikaciji, poput .js , .css, .html i drugih potrebnih datoteka.

Prije pokretanja naredbe potrebno ju je dodati u package.json. Nakon toga može se pokrenuti proces izgradnje koristeći :



Slika 3.3 Pokretanje procesa izgradnje

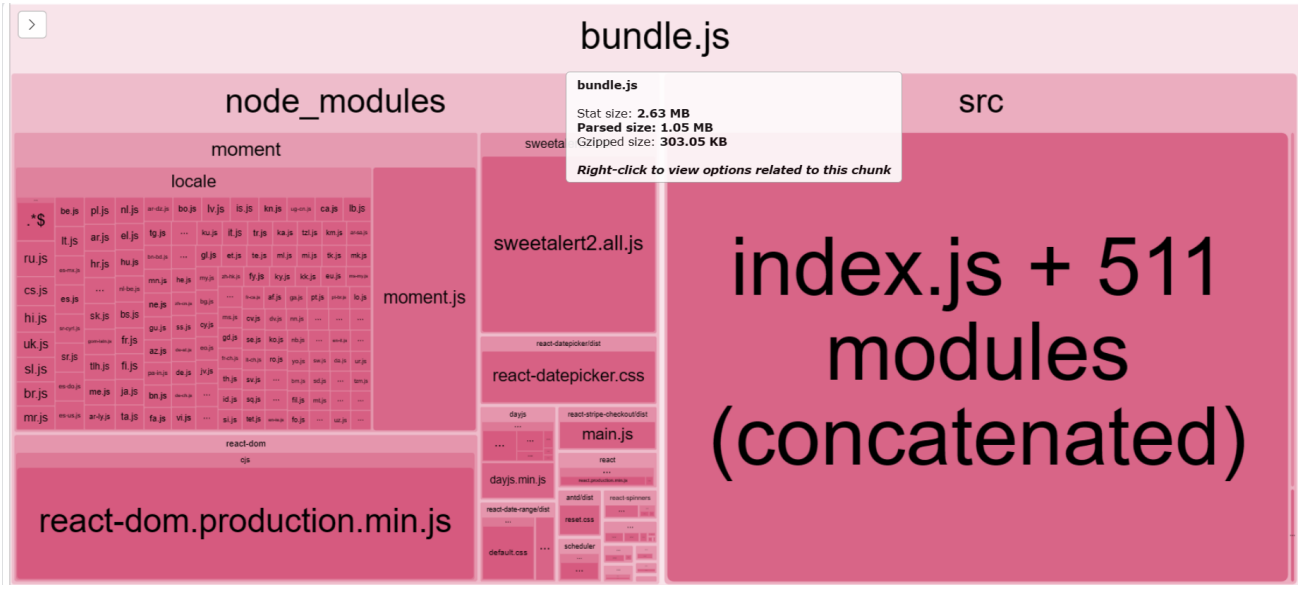
Kada se izvrši naredba, Webpack će proći kroz konfiguraciju, obraditi aplikaciju i generirati izlazne datoteke. Tijekom tog procesa, na konzoli će biti prikazano vrijeme izvršavanja i informacije o veličini generiranog paketa.



Slika 3.4 Prikaz rezultata izvršene naredbe u terminalu

Nakon završetka izgradnje, generirane datoteke bit će smještene u izlaznom direktoriju koji je definiran u konfiguraciji. Koristeći BundleAnalyzerPlugin, može se vizualno analizirati veličina paketa. BundleAnalyzerPlugin je alat koji se koristi u razvoju web aplikacija

pomoću Webpack-a kako bi se analizirala veličina paketa generiranih tijekom procesa izgradnje.



Slika 3.5 Vizualna analiza veličine paketa koristeći BundleAnalyzerPlugin-a

Iz slike 7, može se uočiti vizualna analiza koja pruža detaljan prikaz stanja trenutnih paketa. U ovom detaljnom prikazu, posebno su istaknute tri vrijednosti: veličina datoteke, veličina analiziranog koda, te rezultati analize nakon primjene gzip kompresije.

Veličina datoteke (Stat size)- odnosi se na stvarnu veličinu datoteke koja predstavlja paket na disku. Stat size predstavlja fizički prostor koji će paket zauzimati na disku.

Veličina analiziranog koda (Parsed size)- označava veličinu paketa nakon prolaska kroz postupak analize koda. To uključuje samo korisni kod i ignorira sve dijelove koji nisu korišteni ili su eliminirani tijekom procesa analize. Parsed size daje uvid koliko će zapravo korisnog koda biti uključeno u paket.

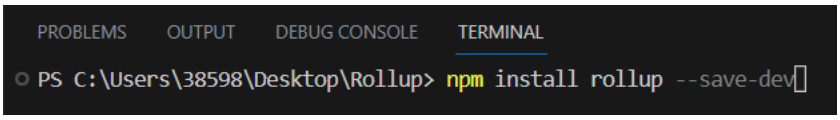
Rezultat analize nakon primjene gzip kompresije (Gzipped size) - prikazuje kolika će biti veličina paketa nakon što se primijeni gzip kompresija. Gzip je popularna metoda kompresije koja može značajno smanjiti veličinu tekstualnih datoteka poput JavaScript i CSS datoteka. Uobičajeno je koristiti gzip na poslužitelju kako bi se paketi poslali do preglednika u komprimiranom obliku, što smanjuje vrijeme preuzimanja i poboljšava brzinu učitavanja web stranice.

Ove tri vrijednosti koriste za bolje razumijevanje smanjivanja veličina paketa ili povećavanja tijekom različitih faza učitavanja, od fizičke veličine datoteke, preko veličine

optimiziranog koda, do konačne veličine nakon primjene kompresije. Na temelju ovih informacija, mogu se optimizirati paketi za potrebe postizanja boljih performansi web aplikacije.

Rollup

Da bi se uključio Rollup, u terminalu VS se upisuje naredba :



Slika 3.6 Naredba za instalaciju Rollup-a

Nakon što se izvršila naredba, spremljena je u razvojnim ovisnostima u datoteci koja se naziva *package.json*.

Slika 3.7 Provjera uspješnosti izvršavanja prethodne naredbe

U projekt se dodaje src folder u kojem će se nalaziti potrebne datoteke za razvoj aplikacije.

Kada se instalira Rollup, neophodno je stvoriti datoteku pod nazivom "rollup.config.js" u kojoj se postavljaju značajke i postavke potrebne za uspješno pokretanje aplikacije.

```
import { babel } from '@rollup/plugin-babel';
import css from 'rollup-plugin-css-only'
export default{
    input:'src/index.js', // Ulazna datoteka za Rollup
    output:{
        file:'build/bundle.js', // Izlazna datoteka koju
Rollup generira
        format: 'cjs' // Format izlazne datoteke - CommonJS
    },
    external: [ // Vanjski moduli koji se ne uključivaju u
izlaznu datoteku
        'react',
```

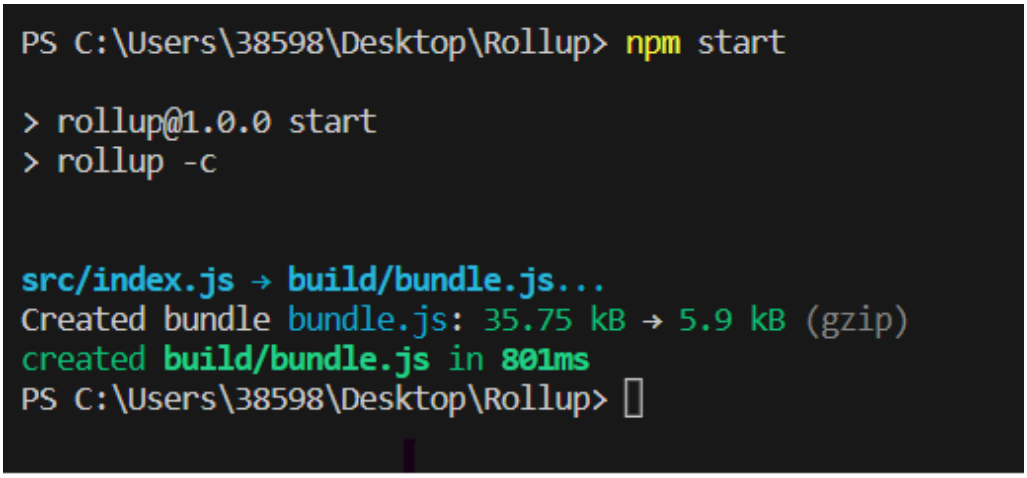
```
            'react-dom',
            'react-router-dom',
            'axios',
            'antd',
            'moment',
            'react-datepicker/dist/react-datepicker.css',
            'react-dom/client',
            'react-stripe-checkout',
            'sweetalert2',
            'antd/dist/reset.css',
            'react-spinners/ClipLoader',
            'react-date-range/dist/styles.css',
            'react-date-range/dist/theme/default.css',
            'react-bootstrap/Button',
            'react-bootstrap/Modal',
            'react-bootstrap/Carousel'
          ],
        plugins: [
            css({output:'bundle.css'}), // Rollup plugin za
obradu CSS-a
            babel({
              exclude: 'node_modules/**', // Izbjegavanje
prevođenja modula iz node_modules direktorija
              presets: ['@babel/preset-react'], // Korištenje
@babel/preset-react za prevođenje React koda
            }),
          ]}
```

Kod 3. Konfiguracija Rollup-a

U konfiguraciji Rollup-a na vrhu se uključuje *@rollup/plugin-babel* za prevođenje React koda i *rollup-plugin-css-only* za obradu CSS datoteka. Vanjski moduli navedeni u *external* polju neće biti uključeni u izlaznu datoteku jer se očekuje da budu dostupni kao vanjski resursi prilikom izvođenja izlazne datoteke.

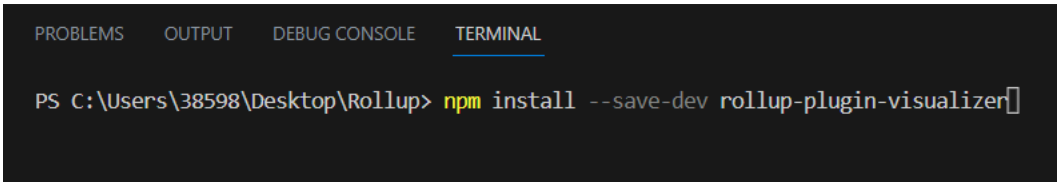
Nakon što je datoteka *rollup.config.js* konfigurirana, u skripti datoteke *package.json* navodi se naredba *rollup -c*. Zastavica '-c ' koja se šalje Rollup-u označava da se koristi konfiguracijska datoteka (*rollup.config.js).*

Pozivanjem skripte naredbom *npm start*, stvara se novi folder sa izlaznom datotekom koja je postavljena u konfiguraciji. Kao rezultat izvođenja naredbe prikazuje se veličina i brzina izvođenja.



Slika 3.8 Rezultati nakon izvođenja naredbe *npm start*

Da bi se vizualno prikazali rezultati, koristi se dodatak *Rollup Plugin Visualizer* pomoću naredbe:



Slika 3.9 Prikaz naredbe za dodatak Rollup Plugin Visualizer

Nakon instalacije, potrebno je na vrh *rollup.config.js* datoteke dodati uvoz visualizer iz biblioteke koja je prethodno opisana. To se može postići na sljedeći način:

```
import { visualizer } from "rollup-plugin-visualizer";
```

Slika 3.10 Uvoz biblioteke na vrh datoteke *rollup.config.js*

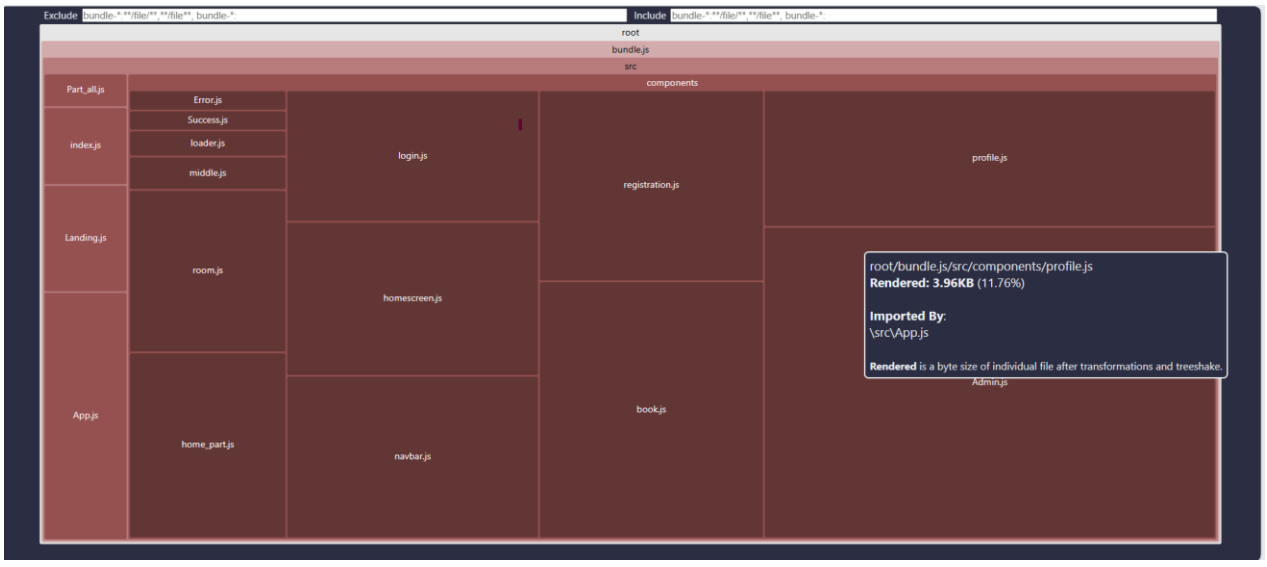
Također u dodacima (engl. plugins) se dodaje metodu koja će se pozvati prilikom pokretanja:

```
plugins: [

    css({ output: 'bundle.css' }),
    babel({
        exclude: 'node_modules/**',
        babelHelpers: 'bundled',
        presets: ['@babel/preset-react'],
    }),
    bundleSize(),
    visualizer()
],
```

Slika 3.11 Prikaz dodataka u *rollup.config.js* datoteci

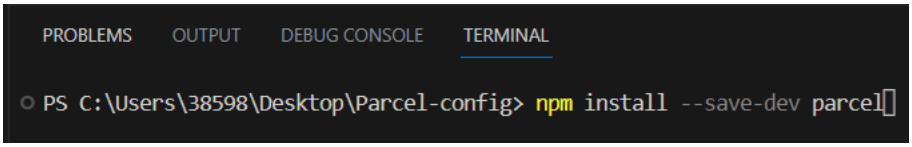
Nakon pokretanja stvara se nova datoteka, sa ekstenzijom .html ,u kojoj postoji mogućnost prikaza pojedinih datoteka. Može se vidjeti koliko memorije sadrže, postotak od ukupnih datoteka te iz kojeg dijela projekta dolaze.



Slika 3.12 Mogućnosti prikaza strukture i veličine pojedinih datoteka

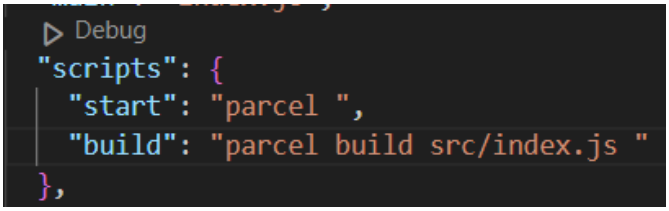
Parcel

Instalacija Parcel-a se izvršava putem terminala naredbom:



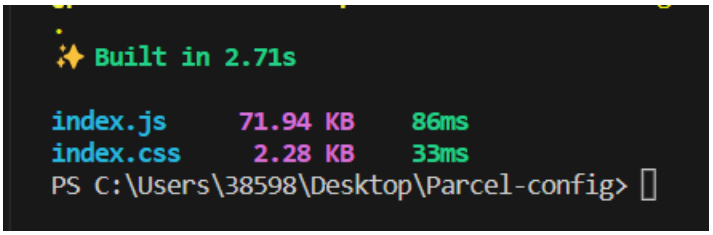
Slika 3.13 Naredba za instalaciju Parcel-a

U *package.json* postavlja se u skripti način kako će se pokretati aplikacija:



Slika 3.14 Skripta u *package.json* datoteci

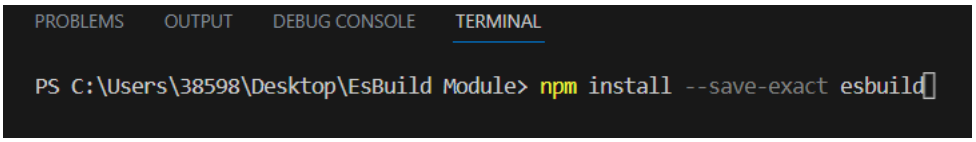
Sa naredbom *npm run build* pokreće se skripta te nakon izvršavanja postavlja se i dodatak s kojim se definira ispis veličine i vremena konfiguriranja određenih datoteka.



Slika 3.15 Rezultat nakon pokretanja naredbe *npm run build*

Esbuild

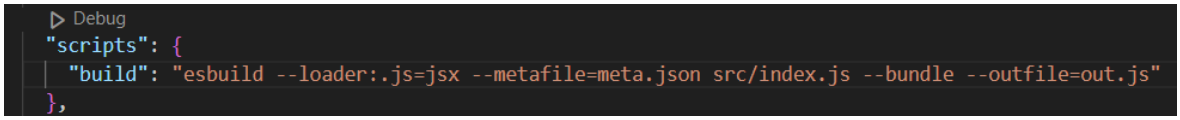
Uključivanje biblioteke Esbuild-a postiže se upisivanjem naredbe u terminal:



Slika 3.16 Naredba potrebna za instalaciju Esbuild-a

Dodatak *--save-exact* osigurava da se točna verzija biblioteke instalira te zabilježi u package.json. Ovim dodatkom stvara se nemogućnost zamjene neke druge verzije određene biblioteke.

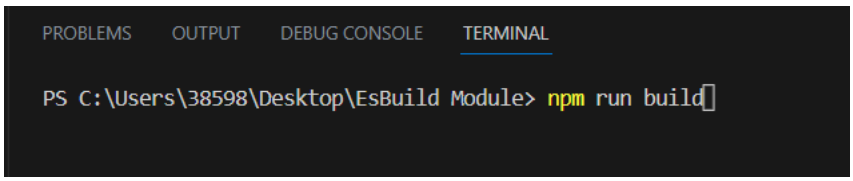
U package.json određuje se što će se pozvati prilikom pokretanja skripte. Da bi se postavile željene naredbe, u skriptama treba dodati sljedeće vrijednosti:



Slika 3.17 Prikaz izgleda skripte

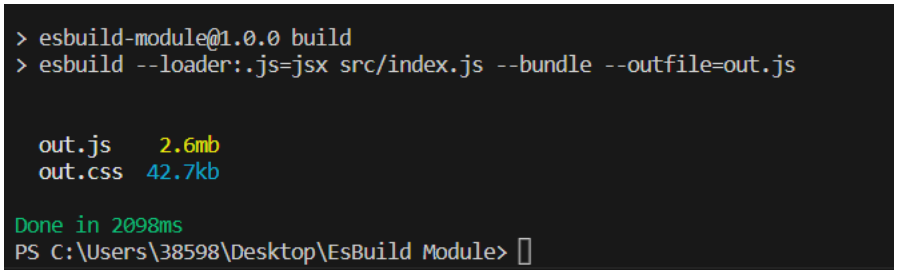
Naredba *build* koristi se za izgradnju projekta. Dodatak *--loader* specificira da se sve *.js* datoteke tretiraju kao jsx, *--metafile* sadrži metapodatke o izgrađenom projektu te služi za vizualno prikazivanje rezultata, *src/index.js* označava koja datoteka se obrađuje kao ulaz, *--bundle* označava da se sve ovisnosti datoteka trebaju pakirati zajedno kako bi se izvršio završni paket, *--outfile* može se odredit naziv izlazne datoteke.

Nakon instalacije potrebne biblioteke, pokretanje skripte izvršava se naredbom:



Slika 3.18 Naredba za pokretanje skripte

Kao rezultat pokretanja skripte prikazuju se slijedeći rezultati:



Slika 3.19 Rezultat nakon izvršavanja skripte

Rezultati se mogu prikazati i u vizualnom obliku. Koristeći meta datoteku koja je kreirana pri pokretanju skripte, na službenoj stranici se ista uvrštava u određeni dio. Nakon toga se pojavljuje vizualni prikaz veličine i strukture traženih datoteka:

Slika 3.20 Vizualni prikaz nakon izvršavanja skripte

Na vrhu vizualnog prikaza, prikazana je veličina i broj ulaznih i izlaznih datoteka. Na dnu je prikazan vizualan dio podjele izlaznih datoteka. Svaka kategorija je prikazana različitom bojom te je na taj način omogućen pregled datoteka korisnicima.

4. Usporedba

U nastavku je prikazana usporedba četiri popularna alata za pakiranje JavaScript aplikacija: Webpack, Rollup, Parcel i Esbuild. Aplikacija na kojoj su izvođene usporedbe sastoji se od HTML, CSS i Javascript-a. Dio koji se odnosi na Javascript broji 21 datoteku sa *.js* ekstenzijom dok CSS sadrži 6 datoteka sa nastavkom *.css*. Datoteka koja ima glavnu ulogu prilikom izgradnje web aplikacije potječe iz HTML-a te u njoj se uključuju sve Javascript i css datoteke. Ova usporedba obuhvaća verzije alata, vrijeme izvođenja za proces pakiranja izražena u milisekundama (ms), veličinu paketa kao rezultat nakon primjene bundlera koje su prikazane u veličinama kao što su MiB (mebibyte), kB (kilobyte) i MB (megabyte), potrebnu konfiguraciju za svaki alat, mjesečni broj preuzimanja te veličinu prije početka procesa pakiranja. Ovi podaci pružit će uvid u performanse, veličinu paketa i popularnost svakog alata.

	Webpack	**Rollup**	**Parcel**	**Esbuild**
Verzija	5.88.1	3.26.0	2.9.3	0.18.11
Vrijeme izvođenja	22303 ms	801 ms	2710 ms	2098 ms
Veličina nakon primjene bundlera (.js)	1.05 MiB	35.75 kB	71.94 kB	2.6 MB
Veličina nakon primjene bundlera (.css)	/	/	2.28 kB	42.7 kB
Potrebna konfiguracija	DA	DA	NE	NE
Mjesečni broj preuzimanja	24.804.990	12.696.786	162.481	13.913.090

Veličina bez pakiranja	4.75 MB	2.5 MB	39.5 kB	131 kB

Tablica 4.1 Usporedba JavaSrcipt bundler-a

Usporedba alata kao što su Webpack, Rollup, Parcel i Esbuild prikazana je u Tablici 1. Odabrani kriteriji pomoću kojih će se uspoređivati su verzija određenog alata, vrijeme pakiranja, veličina pakiranja razdvojenih u .js i .css datoteke, potrebnu konfiguraciju, mjesečni broj preuzimanja te veličina bez pakiranja.

Postoji različiti način gledišta s kojih se može promatrati usporedba, a jedan od načina je vrijeme pakiranja. Naime ako je cilj da rezultati budu u što kraćem roku obrađeni, može se koristit alat Rollup jer je vrijeme pakiranja 801 ms za unaprijed pripremljene ulazne datoteke.

Drugi način gledišta je da pakiranje zauzima što manje prostora, što su rezultati i pokazali nakon obrade da Rollup alat zauzima najmanje prostora.

Ako se gleda kriterij verzije, Webpack se pokazao najbolji jer što je broj verzije veći, to znači da je alat nadograđivan kako bi se prilagodio nužnim promjenama. Esbuild je najnoviji među prikazanim alatima, te se nije uspio dovoljno razviti zato verzija tog alata ne prelazi 1.0.0.

Kriterij kao što je mjesečni broj preuzimanja prikazuje popularnost određenog alata. Webpack je jako popularan što prikazuju i brojke od 24.804.990 preuzimanja, drugo mjesto zauzeo je Esbuild sa 13.913.090 preuzimanja. Najmanje preuzimanja na mjesečnoj bazi je Parcel sa 162.481 preuzimanja.

U konačnici, odabir određenog alata za pakiranje ovisi o više kriterija te potrebama određenog projekta. Ako su prioriteti brzina i minimalna veličina datoteka koristit će se Rollup. Parcel je jako dobar izbor ako je važna jednostavnost i minimalna konfiguracija alata. Webpack je trenutno popularan sa velikim brojem korisnika, ali zahtjeva puno vremena i veličine pakiranja.

Zaključak

U ovom istraživanju uspoređena su četiri popularna JavaScript bundler modula poput Webpack, Rollup, Parcel i Esbuild u svrhu identifikacije njihovih prednosti i mana. Analizom performansi, vremena i veličine generiranih paketa te prilagodljivosti konfiguracije, zaključak je da se Webpack pokazao izvrsnim izborom za veće i kompleksnije projekte zahvaljujući bogatoj zajednici i fleksibilnoj konfiguraciji.

S druge strane, Rollup je pokazao izvanredne performanse kod manjih aplikacija, dok se Parcel istaknuo svojom jednostavnošću upotrebe i brzim postavljanjem projekta. Esbuild kao najnoviji bundler još čeka svoj razvoj u procesu pakiranja web aplikacija. Ovi rezultati dokazuju da odabir najboljeg bundlera ovisi o konkretnim potrebama svake aplikacije.

Preporuka je da se buduća istraživanja usmjere na analizu utjecaja različitih bundlera web aplikacija te na istraživanje novih tehnologija koje bi mogle poboljšati performanse i smanjiti veličinu generiranih paketa. Također, važno je pratiti razvoj novih bundlera i njihovu podršku zajednice kako bi se uvijek koristile najnovije i najbolje tehnike za razvoj web aplikacija.

Literatura

[1] Flanagan, D. *JavaScript:The Definitive Guide, Third Edition,* 1998.

[2] Bažant, A. i dr. *Osnove arhitekture mreža, 2.izdanje,* Zagreb 2004.

[3] Shklar L., Rosen R. *Web Application Architecture: Principles, Protocols and Practises*, 2003. http://bedford-computing.co.uk/learning/wp-content/uploads/2016/07/Web-Application-Architecture-Principles-Protocols-and-Practices.pdf

[4] *Webpack* , https://webpack.js.org/guides/getting-started/#basic-setup

[5] *Esbuild,* https://esbuild.github.io/getting-started/#your-first-bundle

[6] *Parcel,* https://parceljs.org/getting-started/webapp/

[7] *Browserify*, https://browserify.org/

[8] *Rollup*, https://rollupjs.org/introduction/

[9] Medium, Abrickis A. *Modern Approach of JavaScript Bundling With Webpack,* 2017 https://betterprogramming.pub/modern-approach-of-javascript-bundling-with-webpack-3b7b3e5f4e7

[10] Lawson N. *A brief and incomplete history of JavaScript bundlers,* 2017 https://nolanlawson.com/2017/05/22/a-brief-and-incomplete-history-of-javascript-bundlers/

[11] Haworth H. *Comparing the New Generation of Build Tools,* 2021 https://css-tricks.com/comparing-the-new-generation-of-build-tools/

[12] Ivanovs A. *The Most Popular Build Tools for Front-end Developers in 2023* https://stackdiary.com/build-tools-for-web-development/

[13] Munley C. *Demystifying Code Bundling with JavaScript Modules,* 2020 *https://www.simplethread.com/javascript-modules-and-code-bundling-explained/*

[14] Gardón D. *The Complete JavaScript Module Bundlers Guide* , 2022 https://snipcart.com/blog/javascript-module-bundler

Pregled slika

Pregled tablica

Skraćenice

CLI	Command Line Interface
Npm	Node Package Manager
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
JSON	JavaScript Object Notation
VS	Visual Studio
HTTP	The Hypertext Transfer Protocol
ES	ECMAScript
XML	Extensible Markup Language
ID	Identity document
AMD	Asynchronous Module Definition