

Univerzitet Crne Gore
Prirodno–matematički fakultet
Smjer: Računarske nauke



Izrada npm biblioteke za prioritetni red

Mentor:
Prof. dr Aleksandar Popović

Student:
Dejan Todorović, 4/18

Sadržaj

1 Uvod.....	3
2 Tehnologije i alati.....	5
2.1 JavaScript.....	5
2.2 TypeScript.....	5
2.3 Node.js.....	6
2.4 Node package manager.....	6
2.5 Travis.....	6
2.6 Docker.....	7
3 Prioritetni red.....	8
3.1 Hrpa (heap).....	8
4 Koraci u izradi biblioteke.....	11
4.1 Priprema.....	11
4.2 Instalacija node-a.....	11
4.3 Inicijalizacija biblioteke.....	11
4.4 Inicijalizacija git-a i objavljivanje na github.....	12
4.5 Instalacija typescripta.....	14
4.6 Formatiranje i lintovanje koda.....	15
4.7 Kreiranje interface-a i tipova.....	17
4.8 Implementacija funkcija.....	18
4.9 Buildovanje biblioteke.....	25
4.10 Dodavanje LICENSE fajla.....	30
4.11 Kreiranje primjera.....	31
4.11.1 Kreiranje primera u JavaScript-u.....	31
4.11.2 Kreiranje primjera u TypeScript-u.....	35
4.12 Pisanje testova i pokrivenost testovima.....	37
4.13 Pisanje README fajla.....	47
4.14 Upotreba Travis-a za automatsko izvršavanje testova.....	54
4.15 Objavljivanje biblioteke.....	55
5 Zaključak.....	58
6 Bibliografija.....	59

1 Uvod

JavaScript kao jezik koji koriste internet pretraživači već dugo je veoma popularan, a u tome mu se pridružuje i Node.js, čija je popularnost značajno porasla poslednjih godina zbog izuzetno male težine i velike fleksibilnosti. Node.js i JavaScript koriste milioni programera širom svijeta. Prilikom izrade projekata koji se baziraju na ovim tehnologijama koristimo različite pakete koji olakšavaju razvoj. Ovim paketima upravlja Node Package Manager, i karakteriše ih što su razvijeni kako od strane velikih firmi, tako i od strane laika. Na npm registru postoji više od milion i osam stotina hiljada paketa.

Rad je nastao iz potrebe autora za bibliotekom koja nudi korištenje prioritnog reda, takvog da korisnik biblioteke može sam da preda komparator koji će se koristiti prilikom poređenja elemenata. U vrijeme početka izrade rada, pretragom npm registara se vidjelo da takva biblioteka ne postoji, pa je autor odlučio sam da je razvije.

Razvijena biblioteka se zove `priority-queue-with-custom-comparator` i može se naći na sajtu npmjs-a (<https://www.npmjs.com/package/priority-queue-with-custom-comparator>), ali i drugim paket menadžerima koji koriste npm registar, pod istim imenom.

Biblioteka pruža podršku i za JavaScript i za TypeScript, ima primjere primjene za oba kao i primjer projekte, napisane testove za sve javne funkcije, dokumentaciju same biblioteke ali i svih javnih funkcija, napisane skripte za npm, kao i integraciju sa Travis-om koji automatski izvršava testove prilikom push-ovanja koda na GitHub.

Pisana je u TypeScript-u, radi lakšeg razvoja i održavanja, kao i da ne bi bilo potrebe za odvojeno naknadno pisanje tipova za korisnike ove biblioteke, koji će je koristiti uz TypeScript.

Kod je open-source, a koristi se lint-ovanje¹ i formatiranje kako bi svako ko želi da pomogne dalji razvoj biblioteke mogao to da uradi bez narušavanja formata koda, kao i nekih eslint² pravila.

Primjer projekti sadrže Dockerfile-ove kojim se može kreirati docker image sa svim alatima neophodnim za pokretanje istih.

U pozadini leži hrpa (heap) implementirana nad nizom. U ovom radu približena je hrpa kao struktura podataka, ali tema ovoga rada nije izvođenje dokaza njene korektnosti, niti dokaz vremesko/prostorne složenosti operacija.

Ova biblioteka je objavljena na GitHub-u, na lokaciji:

<https://github.com/dejtor/priority-queue-with-custom-comparator>

1 Lintovanje je automatska provjera izvornog koda radi programskih i stilskih grešaka. Ovo se radi pomoću alata za lintovanje (inače poznatog kao linter).

2 ESLint je alat za statičku analizu koda (analiza koda bez njegovog izvršavanja) za identifikaciju problematičnih obrazaca koji se nalaze u JavaScript kodu. Kreirao ga je Nicholas C. Zakas 2013. godine.

Ovaj rad takođe može da služi kao uputstvo svima koji žele da kreiraju svoju biblioteku i objave je na npm registar.

Rad osim poglavlja Uvod i Zaključak i bibliografije sadrži još 3 poglavlja.

U drugom poglavlju, „Tehnologije i alati“, ukratko su opisane tehnologije i alati korišteni za izradu rada.

U trećem poglavlju, „Prioritetni red“, predstavljen je apstraktni tip podataka koji je implementiran u mnogim jezicima, prioritetni red. Takođe je opisana hrpa (heap), jer se nalazi u pozadini implementiranog prioritetnog reda.

U četvrtom poglavlju, „Koraci u izradi biblioteke“, su detaljno opisani svi koraci koje je autor napravio da bi kreirao biblioteku.

2 Tehnologije i alati

2.1 JavaScript

JavaScript (JS) je interpreterski, objektno orijentisan jezik sa funkcijama kao centralnim konceptom. Najpoznatiji je kao skriptni jezik za veb stranice, ali se koristi i u mnogim okruženjima koja nisu pretraživači. Podržava objektno orijentisane, imperativne i funkcionalne stilove programiranja. Ukratko, JavaScript je dinamički skriptni jezik koji podržava konstrukciju objekata zasnovanih na prototipima. Osnovna sintaksa je namjerno slična i Javi i C++-u kako bi se smanjio broj novih koncepata potrebnih za učenje jezika. Jezičke konstrukcije, kao što su if naredbe, for i while petlje, i switch i try - catch blokovi funkcionišu isto kao u ovim jezicima (ili približno tako).

Karakteriše ga to što je on ustvari definisan kao standard, ECMAScript standard, a na pretraživačima i drugim alatima je da izvrše implementaciju funkcionalnosti. Tako su najpopulariniji Google-ov V8 engine (koji koriste Google Chrome, novije verzije pretraživa Opera, Node.js), JavaScriptCore (Safari), Chakra (Internet Explorer), ali postoje i druge implementacije.

2.2 TypeScript

TypeScript je programski jezik koji je razvio i održava Microsoft. To je strogi sintaksički nadskup JavaScript-a i dodaje opcionalno statičko dodavanje tipova u jezik. Besplatan je i open-source. Dizajniran je za razvoj velikih aplikacija i translaciju u JavaScript. Svi postojeći JavaScript programi su takođe ispravni TypeScript programi. Može se koristiti i na serverskoj i na klijentskoj strani. Poznatiji front-end frejmvorci podrazumijevaju njegovo korištenje, dok Node.js pored standardnog JavaScript-a podržava i TypeScript.

TypeScript podržava datoteke definicija koje mogu sadržati informacije o tipu postojećih JavaScript biblioteka, slično kao što datoteke zaglavlja C ++ mogu opisati strukturu postojećih objektnih datoteka. Ovo omogućava drugim programima da koriste vrijednosti definisane u datotekama kao da su u pitanju statički otkucani TypeScript entiteti. Postoje datoteke zaglavlja nezavisnih proizvođača za popularne biblioteke kao što su jQuery, MongoDB i D3.js. Zaglavlja TypeScript za osnovne module Node.js su takođe dostupna, što omogućava razvoj programa Node.js unutar TypeScript -a.

Prevodilac TypeScript-a je sam napisan u TypeScript-u i preveden u JavaScript. Licenciran je pod Apache licencom 2.0. TypeScript je uključen kao programski jezik u Microsoft Visual Studio 2013 Update 2 i novije verzije, pored C# i drugih Microsoft jezika.

Programeri TypeScript-a tražili su rješenje koje ne bi narušilo kompatibilnost sa standardom i njegovu podršku za više platformi. Znajući da trenutni standard ECMAScript obećava podršku programiranju zasnovanom na klasama, TypeScript je zasnovan na tom predlogu. To je dovelo do JavaScript kompajlera sa skupom sintaktičkih jezičkih proširenja, nadskupom zasnovanim na predlogu, koji pretvara proširenja u običan JavaScript.

2.3 Node.js

Node.js je open-source, cross-platform, back-end JavaScript sistem izvršavanja (runtime) koji radi na V8 engine-u i izvršava JavaScript kod izvan veb pregledača. Node.js omogućava programerima da koriste JavaScript za pisanje alata komandne linije i za skriptiranje na strani servera. Node.js predstavlja paradigmu "JavaScript svuda", objedinjavajući razvoj veb aplikacija oko jednog programskog jezika, umjesto različitih jezika za skripte na strani servera i klijenta.

Node.js ima arhitekturu upravljanja događajima sposobnu za asinhroni I/O. Ovi izbori dizajna imaju za cilj optimizaciju propusnosti i skalabilnost u veb aplikacijama sa mnogim ulazno/izlaznim operacijama, kao i za veb aplikacije u realnom vremenu (npr. Komunikacioni programi u realnom vremenu i igre za pretraživač).

2.4 Node package manager

npm je podrazumijevani menadžer paketa za JavaScript runtime okruženje Node.js. Sastoji se od klijenta komandne linije, koji se naziva i npm, i mrežne baze podataka javnih i plaćenih privatnih paketa, nazvane npm registar. Registru se pristupa putem klijenta, a dostupni paketi se mogu pretraživati i putem veb stranice npm. Menadžerom paketa i registrom upravlja npm, Inc.. npm je uključen kao preporučena funkcija u instalacioni program Node.js. Registar nema nikakav postupak provjere za podnošenje, što znači da tamo pronađeni paketi mogu biti nekvalitetni, nesigurni ili zlonamjerni. Umjesto toga, npm se oslanja na korisničke izveštaje za uklanjanje paketa ako krše smjernice tako što su lošeg kvaliteta, nesigurni ili zlonamjerni. npm izlaže statistiku uključujući broj preuzimanja i broj zavisnih paketa kako bi pomogao programerima u procjeni kvaliteta paketa. Postoji niz alternativa otvorenog koda za npm za instaliranje modularnog JavaScript-a, uključujući ied, pnpm, npmd, Yarn i Yarn 2 (Berry). Svi su kompatibilni sa javnim npm registrom (ali Berry ne stopostotno) i koriste ga podrazumijevano, ali pružaju različita iskustva na strani klijenta, obično usredsređena na poboljšanje performansi i determinizma u poređenju sa npm klijentom.

2.5 Travis

Travis CI je hostovana usluga kontinuirane integracije koja se koristi za izradu i testiranje softverskih projekata hostovanih na GitHub -u i Bitbucket -u.

Travis CI je prvi CI servis koji je besplatno pruža usluge projektima otvorenog koda.

Kontinuirana integracija je praksa spajanja malih izmjena koda često - umjesto spajanja velikih promjena na kraju razvojnog ciklusa. Cilj je izgraditi zdraviji softver razvijanjem i testiranjem u manjim koracima.

Kao platforma za kontinuiranu integraciju, Travis CI podržava razvojni proces tako što automatski testira promjene koda, pružajući trenutno povratne informacije o uspjehu promjene.

2.6 Docker

Docker je skup platformi kao usluga (PaaS) proizvoda koji koriste virtuelizaciju na nivou OS za isporuku softvera u paketima koji se zovu kontejneri. Kontejneri su izolovani jedan od drugog i povezuju sopstveni softver, biblioteke i konfiguracijske datoteke. Mogu međusobno komunicirati putem dobro definisanih kanala. Pošto svi kontejneri dijele usluge jednog jezgra operativnog sistema, oni koriste manje resursa od virtuelnih mašina.

Kontejneri se pokreću od image-a (slika) za Docker, koje definišemo u Dockerfile-ovima.

Usluga ima i besplatne i premium nivoe. Softver koji hostuje kontejnere naziva se Docker Engine. Prvi put je pokrenut 2013. godine, a razvija ga Docker, Inc.

3 Prioritetni red

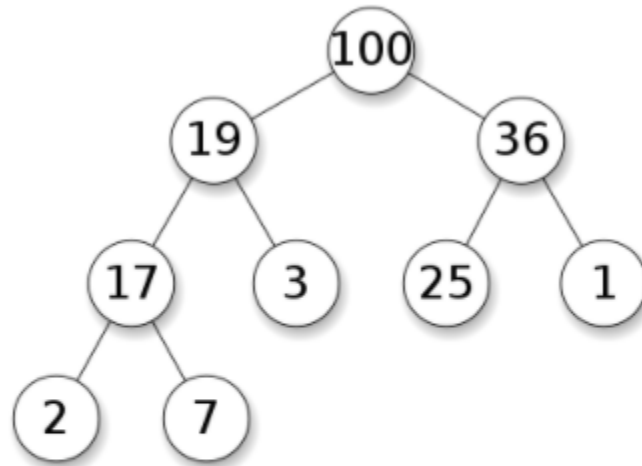
U računarstvu, prioritetni red je apstraktan tip podataka, koji je sličan regularnom redu ili steku, ali koji dodatno ima pridružen prioritet svakom elementu. U redu sa prioritetom, element sa najvećim prioritetom se uzima prije elementa sa nižim prioritetom. Ako dva elementa imaju isti prioritet, onda se uzimaju na osnovu njegovog položaja u listi.

Prioritetni red se uglavnom implementira preko hipa, ali se konceptualno razlikuje od njega. Prioritetni red je apstraktan koncept kao "lista" ili "mapa". Kao što lista može biti implementirana kao povezana lista ili kao niz, red sa prioritetom može biti implementiran preko hipa ili preko drugih metoda kao što je neuređen niz.

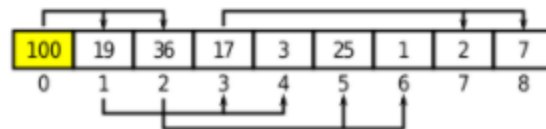
3.1 Hrpa (heap)

U kompjuterskim naukama, hrpa je struktura podataka organizovana po principu stabla koja mora da zadovoljava uslov hrpe: ključ svakog čvora je veći ili jednak od ključeva njihovih sinova. Kada je maksimalan broj djece jednog čvora dva, to je binarna hrpa.

Prikaz drveta



Prikaz niza



Slika 1 - hrpa

Hrpa je izuzetno bitna u nekim algoritmima, kao što je Dijkstrin algoritam i heapsort. Hrpa je veoma korisna struktura podataka kada treba da se ukloni čvor najvišeg ili najnižeg prioriteta.

Čvor je jedna memorijska jedinica i sadrži ključ tj. podatak. Svaki čvor ima svog pretka, sem korijena stabla koji predstavlja vrh hijerarhije. Ako je čvor A predak čvora B, onda je B potomak od A. Čvor bez potomaka se zove list, a čvor koji nije list se zove unutrašnji čvor. Ključ svakog čvora je uvek veći ili jednaki od ključeva njegovih potomaka i najveći ključ je u korjenu stabla. Ne postoji posebna veza između čvorova na istom nivou ili braće. Kako je hrpa binarno stablo, ima najmanju moguću visinu - $O(\log n)$ za hrpu sa n čvorova.

Element se dodaje u hrpu tako što se napravi da bude list, najdesniji na poslednjem nivou, odnosno takav da zauzima poslednju poziciju u nizu. Zatim se podiže ka korijenu, sve dok je vrijednost tog čvora veća od vrijednosti njegovog roditelja, zamjenjuju im se pozicije. Ova operacija se izvršava u logaritamskom vremenu.

Uklanjanje najprioritetnijeg elementa (korijena) se vrši tako što mu se zamijeni mjesto sa najdesnijim listom, a potom se novi korijen spušta nadolje. Sve dok vrijednost korijena je manja od vrijednosti

jednog od njegovih sinova, on se zamjenjuje sa vrijednijim sinom. Ova operacija se takođe izvršava u logaritamskom vremenu.

Prilikom kreiranja hrpe od niza, može se koristiti algoritam kao kod dodavanja elemenata u hrpu. Za svaki element koji nije list, pokreće se proces njegovog spuštanja niz drvo, sve dok je manje prioritetan od nekog od njegovih sinova. Ovim postupkom hrpa se kreira u linearnom vremenu.

4 Koraci u izradi biblioteke

4.1 Priprema

Za početak neophodno je izabrati ime biblioteke. Imena svih biblioteka u npm registru moraju biti jedinstvena, tako da je dovoljno provjeriti na sajtu npm-a (<https://www.npmjs.com/>) da li već postoji biblioteka sa istim imenom. Prije izrade ovog projekta na npm registru nije postojala biblioteka sa imenom `priority-queue-with-custom-comparator` pa je autor izabrao to ime.

4.2 Instalacija node-a

Preporuka autora ovoga rada je da se preuzme poslednja stabilna verzija (LTS) Node.js-a. Node se može instalirati samostalno ili preko nvm-a (Node Version Manager), alata koji nudi lakše prebacivanje sa jedne verzije node-a na drugu, i samim tim lakši razvoj.

Instaler će podrazumijevanim opcijama instalirati Node.js runtime, npm package manager, prečice za dokumentaciju, i registrovati Node.js i npm u path, kako bi mogli biti pozivani iz cmd-a. Nije neophodno instalirati alate za nativne module.

U vrijeme izrade projekta aktuelna verzija node-a je bila 14.

4.3 Inicijalizacija biblioteke

Potrebno je bilo napraviti folder koji će se zvati isto kao i biblioteka. Unutar foldera pokrenuti cmd ili PowerShell i pokrenuti proces inicijalizacije komandom „`npm init -y`“. npm onda genereše `package.json` fajl.

```
{
  "name": "priority-queue-with-custom-comparator",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Tabela 1 – inicijalni package.json

Fajl package.json je srce svakog Node projekta, sadrži podatke o projektu, zavisnostima od drugih paketa, skripte, itd..

Manjom modifikacijom ovog fajla prikladnije se opisuje ovaj projekat.

```
{
  "name": "priority-queue-with-custom-comparator",
  "version": "0.0.1",
  "description": "Priority queue data structure where you are able to set your own compare function.",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "priority-queue"
  ],
  "author": "Dejan Todorovic",
  "license": "ISC"
}
```

Tabela 2 – package.json nakon modifikacije

4.4 Inicijalizacija git-a i objavljivanje na github

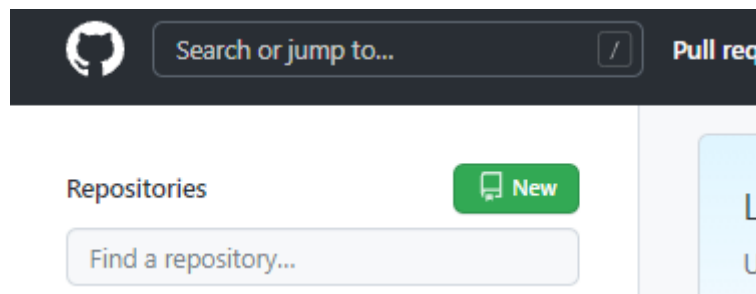
Git je alat za kontrolu verzija. Alati za kontrolu verzija se koriste na svakom ozbiljnom projektu, kako radi praćenja istorije izmjena, tako i zbog lakšeg vraćanje na prethodno stanje koda i lakši rad više ljudi na istom projektu.

GitHub je onlajn hosting servis za git.

Instalator za git se može skinuti sa <https://git-scm.com/downloads> .

Zatim je bilo potrebno kreirati nalog na GitHub-u ili se ulogovati na postojeći, na <https://github.com/> .

U sljedećem koraku neophodno je bilo krenuti na kreiranje repozitorij na GitHubu-u. Repozitorij predstavlja skup fajlova, podešavanja i istorije vezan za jedan projekat.





Slika 2 - dio početne stranice na GitHub-u

Na početnoj strani se nalazi link za kreiranje novog repozitorija. Klikom na dugme „New“ korisnik biva preusmjeren na stranicu za kreiranje novog repozitorija.

Create a new repository


A repository contains all project files, including the revision history. Already have a repository elsewhere?


Owner * Repository name *

 dejtor / priority-queue-with-custom-comparator 

Great repository names are short and memorable. Need inspiration? How about [ul](#)

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

[Create repository](#)

Slika 3 - kreiranje repozitorija

Repozitoriju je dato isto ime kao i našem projektu, izabrano da on bude javan (kako bi biblioteka bila open-source) i kreiran repozitorij klikom na „Create repository“.

```
git init
git commit -m "first commit"
git branch -M master
git remote add origin https://github.com/dejtor/priority-queue-with-custom-comparator.git
git push -u origin master
```

Tabela 3 - komande za git

Potrebno je bilo izvršiti komande iz tabele 3. na lokaciji gdje se nalazi naš projekat. Ovim komandama će se inicijalizovati git repozitorijum, staviti svi fajlovi u prvi commit, koji će ići na master granu, staviti remote adresu sa github-a i gurnuti commit na repozitorijum.

Commit predstavlja najmanju jedinicu, odnosno najmanju izmjenu u kodu koja se pravi. Svaki commit bi trebao da predstavlja jednu cjelinu. Dobra praksa je da posle njega projekat može normalno da se pokrene, odnosno da nema izmjena koje će „slomiti“ projekat.

Commit-i se radi organizacije mogu stavljati na različite grane (branches), i tako razdvajati razvoj različitih funkcionalnosti. Grane koje su stvorene za razvoj različitih funkcionalnosti, se na kraju razvoja te funkcionalnosti, spajaju na glavnu granu, koja sadrži stabilan projekat sa svim do kraja razvijenim funkcionalnosti. Ukoliko na jednom jednostavnom projektu radi samo jedna osoba, ona može za sve koristiti glavnu granu. Glavna grana se uglavnom naziva master, ali se iz razloga političke korektnosti u zadnje vrijeme uvela mogućnost njenog preimenovanja.

Remote predstavlja adresu gdje je hostovan repozitorij onlajn.

Push-ovanje ili guranje koda na neku granu predstavlja slanje određenog broja commit-a sa našeg lokalnog repozitorija na hostovani repozitorij.

Preporuka autora ovog rada je da se za lakši rad sa git-om umjesto cmd-a koristi integrisani alat editora koda ili neki pomoćni alat za rad sa git-om.

Određeni fajlovi se ne žele stavljati na git, i to se naglašava u .gitignore fajlu koji se kreira u početnom folderu projekta.

```
node_modules
lib
coverage
sample-projects\**\node_modules
sample-projects\**\dist
```

Tabela 4 - .gitignore

U gitignore-u je naglašeno da se ne želi na git-u držati node_modules folder koji sadrži instalacije svih naših zavisnosti, lib folder koji sadrži transpajlovane fajlove i coverage folder koji sadrži podatke o pokrivenosti projekta testovima.

U nastavku rada autor se neće obazirati koje izmjene ide u koji commit i kada bi trebalo odraditi guranje koda na neku granu.

4.5 Instalacija typescripta

```
npm install --save-dev typescript
```

Tabela 5 - komanda za instalaciju TypeScript-a

Za instalaciju typescripta dovoljno je bilo odraditi komandu in tabele 5.. --save-dev zastavica označava da će se ovaj paket koristiti samo na razvojnim okruženjima. Takođe, paket se može instalirati i globalno korištenjem zastavice -g, ako je u planu da typescript bude korišten i u drugim projektima. Mogu se izvršiti i obje komande, pa će se za ovaj projekat koristiti verzija koja je instalirana za ovaj projekat. Nakon instalacije može se naći typescript kao devDependency (zavisnost neophodna za

razvoj projekta) u package.json fajlu, a kreiraće se i package-lock.json, fajl koji za svaki paket sadrži verzije njegovih zavisnosti.

```
{
  "compilerOptions": {
    "target": "es2017",
    "module": "commonjs",
    "declaration": true,
    "outDir": "./lib",
    "esModuleInterop": true
  }
}
```

Tabela 6 - tsconfig.json

```
{
  "extends": "./tsconfig.json",
  "include": [
    "src"
  ],
  "exclude": [
    "node_modules",
    "**/__tests__/*",
    "lib"
  ]
}
```

Tabela 7- tsconfig.build.json

Kreirani su i tsconfig.json i tsconfig.build.json fajlovi. U tsconfig.build.json opisane su opcije neophodne za kompajlaciju projekta. tsconfig.json će biti njegov nadskup i sadržaće i konfiguraciju neophodnu za testove i lintovanje koda.

Korišćena je verzija ECMAScripta es2017, a moduli su kompajlirani po commonjs (implementacija modula po stilu Node.js-a) standardu koji je preporučen od strane zvanične typescript dokumentacije. declaration podešavanje stavljeno je na true, da bi prilikom transpajliranja u JavaScript kod dobili i fajl sa definicijama.

4.6 Formatiranje i lintovanje koda

Jako je bitno da kod prati neke standarde, posebno ako želimo da timski radimo na njemu. Da bismo to ostvarili djelimično možemo da se pomognemo formatiranjem i lintovanjem koda. Lintovanje koda predstavlja statičku analizu koda, odnosno analizu koda bez njegovog izvršavanja, i prati ne samo da li je kod pisan u očekivanom formatu već i da li se pridržavamo svih pravila.

Za formatiranje koda korišćen je Prettier, a za lintovanje eslint, linter za javascript.

```
npm install --save-dev @typescript-eslint/eslint-plugin @typescript-eslint/parser eslint eslint-config-prettier eslint-plugin-prettier prettier
```

Tabela 8 - komande za instalaciju lintera i formatera

Instalirani su zajedno sa svim neophodnim paketima, a potom kreirani i neophodni konfiguracioni fajlovi.

```
{  
  "singleQuote": true,  
  "trailingComma": "all"  
}
```

Tabela 9 - .prettierrc

Za konfiguraciju Prettier-a korišćen je .prettierrc fajl, gdje je napomenuto da posle poslednjeg elementa niza u listi ide zarez, ako su elementi razdvojeni novim redom. Takođe je napomenuto da se koriste jednostruki navodnici umjesto dvostrukih gdje god je to moguće.

```
module.exports = {  
  parser: '@typescript-eslint/parser',  
  parserOptions: {  
    project: 'tsconfig.json',  
    sourceType: 'module',  
  },  
  plugins: ['@typescript-eslint/eslint-plugin'],  
  extends: [  
    'plugin:@typescript-eslint/recommended',  
    'plugin:prettier/recommended',  
  ],  
  root: true,  
  env: {  
    node: true,  
    jest: true,  
  },  
  ignorePatterns: ['.eslintrc.js'],  
  rules: {  
    '@typescript-eslint/interface-name-prefix': 'off',  
    '@typescript-eslint/explicit-function-return-type': 'off',  
    '@typescript-eslint/explicit-module-boundary-types': 'off',  
    '@typescript-eslint/no-explicit-any': 'off',  
  },  
};
```

Tabela 10 - .eslintrc.js

Za konfiguraciju eslint-a korišćen je .eslintrc.js fajl. U njemu je naglašeno da koristi podrazumjevana podešavanja, i da gleda i Prettier-ova pravila. Ako želimo da dodamo i druga eslint pravila, to možemo

da uradimo dodavanjem rules opcije, i predavanjem objekta koji sadrži dodatna eslint pravila. U ovom projektu dodata su pravila za imenovanje interfejsa, naglašavanje koji tip vraća funkcija, ali i kojeg su tipa argumenti za javne funkcije, te korišćenje tipa any.

Potrebno je bilo još napisati skripte za lintovanje i formatiranje, i dodati ih u package.json.

```
...
  "scripts": {
    ...
    "format": "prettier --write \"src/**/*.*.ts\" \"__tests__/*.*.ts\" \"sample-projects/**/*.*.ts\"",
    "lint": "eslint \"{src,sample-projects,__tests__}/**/*.*.ts\" --fix",
    ...
  },
  ...
}
```

Tabela 11 - skripte za formatiranje i lintovanje u package.json

Skripta za Prettier određuje u kojim direktorijumima formatiramo fajlove, i napominje Prettier da može slobodno da sam ispravi fajl gdje god je u mogućnosti. Slično radi i skripta za eslint.

Pokretanje definisanih skripti je vrlo lako.

```
npm run ime_skripte
```

Tabela 12 - pokretanje skripti

Kasnije je podešeno da se ove skripte same pokreću prilikom određenih događaja.

4.7 Kreiranje interface-a i tipova

Radi urednosti koda i mogućnosti korištenja tipova za TypeScript, razdvojeni su interfejsi i tipovi, i implementacija ove biblioteke. Tako u src folderu, koja sadrži glavne izvorne kodove, postoje dva fajla, queueInterfaces.ts i queue.ts.

```
export type PriorityQueueComparator<T> = (a: T, b: T) => boolean;

export interface IPriorityQueueOptions<T> {
  comparator: PriorityQueueComparator<T>;
  initialElements?: T[];
}

export interface IPriorityQueue<T> {
  size(): number;
  isEmpty(): boolean;
  peek(): T;
}
```

```

push(value: T): void;
pushMany(values: T[]): void;
pop(): T;
clear(): void;
has(value: T): boolean;
values(): T[];
}

```

Tabela 13 - queueInterfaces.ts

U ovom fajlu postoje tip PriorityQueueComparator i interfejsi IPriorityQueueOptions i IPriorityQueue. Svi su generički i izvezeni (eksportovani).

PriorityQueueComparator pokazuje kakvog oblika želimo da bude compare funkcija koja je neophodna za inicijalizaciju prioritetnog reda.

IPriorityQueueOptions predstavlja objekat koji konstruktor prioritetnog reda prima. Osim komparatora, omogućeno je i opciono slanje elemenata koji će se naći u prioritetnom redu odmah nakon inicijalizacije.

IPriorityQueue predstavlja izgled našeg prioritetnog reda, pokazuje nam koje javne funkcije sadrži prioritetni red, koje argumente primaju te funkcije i šta vraćaju.

4.8 Implementacija funkcija

Potom je implementiran prioritetni red u fajlu src/queue.ts . Funkcije nisu prikazane redom kojim su implementirane, niti kako se nalaze u fajlu, već tako da svaka funkcija koja poziva neke druge funkcije, bude objašnjena tek nakon objašnjenja funkcija koje se unutar nje pozivaju.

```

import {
  IPriorityQueue,
  PriorityQueueComparator,
  IPriorityQueueOptions,
} from './queueInterfaces';

export default class PriorityQueue<T> implements IPriorityQueue<T> {
  private heap: T[];
  private comparator: PriorityQueueComparator<T>;
  ...
}

```

Tabela 14 - src/queue.ts (1. dio)

Prioritetni red implementira interfejs koji mu je namijenjen. Njemu su pored polja iz interfejsa dodata i privatna polja. Tako se tu nalazi generički niz, koji se koristiti za implementaciju heap-a, i komparator.

```
...
```

```

private getParent(index: number) {
  return ((index + 1) >>> 1) - 1;
}

private getLeftChild(index: number) {
  return (index << 1) + 1;
}

private getRightChild(index: number) {
  return (index + 1) << 1;
}

private compareByIndex(i: number, j: number) {
  return this.comparator(this.heap[i], this.heap[j]);
}

private swap(i: number, j: number) {
  [this.heap[i], this.heap[j]] = [this.heap[j], this.heap[i]];
}
...

```

Tabela 15 - src/queue.ts (2. dio)

Potrebne su neke pomoćne funkcije. Tako su napravljene funkcije koje će za dati čvor, računati indekse njegovog roditelja, kao i lijevog i desnog sina. Da bi ove funkcije brže izvršavale korišćene su binarne operacije.

Pored toga napravljena je funkcija koja za dva indeksa vraća koji je prioritetniji, kao i funkcija koja zamjenjuje vrijednosti dva čvora.

```

...
private siftUp() {
  let node = this.size() - 1;
  while (node > 0 && this.compareByIndex(node, this.getParent(node))) {
    const parentNode = this.getParent(node);
    this.swap(node, parentNode);
    node = parentNode;
  }
}

/**
 *
 * @param value element to be added to heap, adds it in O(log n) operations, n is size of heap
 * @returns size of heap
 */
push(value: T) {
  this.heap.push(value);
  this.siftUp();
  return this.size();
}
...

```

Tabela 16 - src/queue.ts (3. dio)

U radu je implementirana funkcija za dodavanje na prioritetni red, push funkcija. Nakon dodavanja na hrpu ona poziva siftUp (ispravi nagore) funkciju i vrati veličinu prioritetnog reda. Ovo dodavanje se izvršava u logaritamskog vremenu.

siftUp je funkcija koja kreće od novodatog elementa, koji se nalazi u listu drveta, i penje na gore sve dok je njegov prioritet veći od prioriteta njegovog roditelja, ili dok se taj element nije popeo na vrh stabla.

```

...
/**
 *
 * @param values elements to be added to heap, adds it in O(k * log n) operations, n is size of heap,
 * k is number of elements added
 * @returns size of heap
 */
pushMany(values: T[]) {
  values.forEach((value) => {
    this.push(value);
  });
  return this.size();
}
...

```

Tabela 17 - src/queue.ts (4. dio)

Tu je i funkcija `pushMany`, koja dozvoljava dodavanje niza elemenata odjednom. Ona prije nego vrati novu veličinu prioritetnog reda, za svaki element pozove `push` funkciju. Stoga je složenost ove funkcije $O(k * \log n)$, gdje je n veličina prioritetnog reda, a k veličina predatog niza, koji treba da bude unesen u prioritetni red.

```
...
private siftDown(node = 0) {
  let leftChild = this.getLeftChild(node);
  let rightChild = this.getRightChild(node);
  while (
    (leftChild < this.size() && this.compareByIndex(leftChild, node)) ||
    (rightChild < this.size() && this.compareByIndex(rightChild, node))
  ) {
    const maxChild =
      rightChild < this.size() && this.compareByIndex(rightChild, leftChild)
        ? rightChild
        : leftChild;

    this.swap(node, maxChild);

    node = maxChild;
    leftChild = this.getLeftChild(node);
    rightChild = this.getRightChild(node);
  }
}
...
```

Tabela 18 - src/queue.ts (5. dio)

Još jedna privatna funkcija koja je neophodna je `siftDown` (ispravi nadolje). Njen zadatak je da krene od nekog čvora, i sve dok je njegov prioritet manji od nekoga od njegovih sinova, da ga zamijeni sa prioritetnijim sinom. Ukoliko ne predamo od kojeg čvora želimo da krenemo, podrazumijevaće se da želimo da krenemo od glave drveta.

Ova funkcija svoj posao radi u logaritamskom vremenu a neophodna je prilikom uklanjanja elemenata sa prioritetnog reda ali i prilikom kreiranja prioritetnog reda sa inicijalnim elementima.

```
...
/**
 *
 * @returns top of priority queue and removes it from priority queue in  $O(\log n)$ , if priority queue is
empty returns undefined
 */
pop() {
  if (this.isEmpty()) {
```

```

    return undefined;
}

const returnValue = this.peak();
const lastIndexOfHeapArray = this.size() - 1;
if (lastIndexOfHeapArray > 0) {
    this.swap(0, lastIndexOfHeapArray);
}
this.heap.pop();
this.siftDown();
return returnValue;
}
...

```

Tabela 19 - src/queue.ts (6. dio)

Sledeća od osnovnih funkcija koja se morala implementirati je pop funkcija, čija je svrha skidanje elemenata sa prioritetnog reda.

Ova funkcije u logaritamskog vremenu, vrati najprioritetniji element iz prioritetnog reda, ali ga i ukloni, i rekonstruiše red. Ukoliko je red prazan ova funkcija vrati undefined. Undefined u TypeScript-u obično je dodijeljeno promjenjivima koje su deklarirane ali im nije još dodijeljena vrijednost.

```

...
private buildHeap(array: T[]) {
    this.heap = JSON.parse(JSON.stringify(array)) as T[];
    for (let i = Math.floor(array.length / 2) - 1; i >= 0; i--) {
        this.siftDown(i);
    }
}
}
...

```

Tabela 20 - src/queue.ts (7. dio)

Privatni metod buildHeap, koristiće se prilikom izgradnje heap-a, u slučaju da se inicijalizuje sa nekim početnim elementima. Ona za svaki element koji nije list poziva siftDown funkciju. Ovo se izvršava u linearnom vremenu.

```

...
/**
 *
 * @param options
 * options.comparator: function used to compare elements;
 * options.initialElements: elements to be put in priority queue initially in O(n) time
 */
constructor(options: IPriorityQueueOptions<T>) {
    this.heap = [];
    this.comparator = options.comparator;
}

```

```

    if (options.initialElements) this.buildHeap(options.initialElements);
  }
  ...

```

Tabela 21 - src/queue.ts (8. dio)

Potreban je i konstruktor. On prima obavezan komparator i opcionalni niz početnih elemenata i kreira prioritetni red. U slučaju da su poslani početni elementi ova funkcija radi u linearnom, a inače u konstantnom vremenu.

```

...
/**
 *
 * @returns size of priority queue in O(1)
 */
size() {
  return this.heap.length;
}

/**
 *
 * @returns is priority queue empty in O(1)
 */
isEmpty() {
  return this.size() === 0;
}
...

```

Tabela 22 - src/queue.ts (9. dio)

Implementirane su i funkcije koje vraćaju veličinu niza i da li je niz prazan. Stoga su kreirane funkcije `size` i `isEmpty`, obje sa konstantnom složenosti.

```

...
/**
 *
 * @returns top of priority queue in O(1), if priority queue is empty returns undefined
 */
peek(): T {
  return this.heap[0] ? JSON.parse(JSON.stringify(this.heap[0])) : undefined;
}

/**
 * clears priority queue in O(1)
 */
clear(): void {
  this.heap = [];
}

```

...

Tabela 23 - src/queue.ts (10. dio)

Potrebne su i funkcije koje vraćaju vrijednost najprioritetnijeg elementa iz prioritetnog reda, kao i funkcija koja će da isprazni niz.

Funkcija peek će vratiti vrijednost najprioritetnijeg elementa, u konstantnom vremenu, kopiranjem vrijednosti najprioritetnijeg elementa. Kopiranje se radi kako bi se izbjeglo da ukoliko je u pitanju objekat, kasnijom manipulacijom sa rezultatom ove funkcije, dođe do ugrožavanja korektnost strukture.

Funkcija clear će heap zamijeniti praznim nizom, a brisanje elemenata iz memorije prepustiće garbage collector-u, procesu koji u pozadini uklanja iz memorije sve nekorišćene objekte.

```
...
/**
 * checks if value exists in priority queue in O(n)
 */
has(value: T) {
  return !!this.heap.find((ele) => ele === value);
}

/**
 *
 * @returns all values of priority queue in O(n)
 */
values() {
  return JSON.parse(JSON.stringify(this.heap)) as T[];
}
...
```

Tabela 24 - src/queue.ts (11. dio)

Na kraju su implementirane i funkcije has i values.

has funkcija provjera da li neki element postoji u prioritetnom redu. Ova provjera se izvršava u linearnom vremenu i ukoliko su elementi reda referenti tipovi, ova provjera se vršiti po referenci.

values vraća vrijednosti svih elemenata iz niza, tako što kopira vrijednost svakoga, slično kao peek funkcija. Ova funkcija ne vraća elemente u sortiranom redosledu, stoga radi u linearnoj složenosti. Ukoliko bi željeli da ovakva funkcija vraća elemente u sortiranom redosljedu, može se pozivati pop funkcija, sve dok prioriteni red ne postane prazan. Ovaj proces bi bio identičan radu heap sort algoritma.

4.9 Buildovanje biblioteke

```
...
"main": "lib/queue.js",
"types": "lib/queue.d.ts",
"scripts": {
  ...
  "build": "tsc --project tsconfig.build.json",
  ...
},
...
```

Tabela 25 - promjene u package.json

U package.json dodata je skripta za build-ovanje (izgradnju) projekta. Ova skripta će prepustiti tsc-u (TypeScript kompajleru) da fajlove prevede u JavaScript fajlove.

Takođe, u package.json fajlu naznačeno je gdje se nalazi glavni fajl koji sadrži funkcije koje će moći da koriste korisnici biblioteke, i gdje se nalaze tipovi za TypeScript. Ove vrijednosti su stavljene pod poljima main i types.

```
npm run build
```

Tabela 26 - pokretanje build komande

Sada je dovoljno pokrenuti build skriptu i dobija se lib folder i u njemu fajlovi: queue.d.ts, queue.js, queueInterfaces.d.ts i queueInterfaces.js.

.d.ts fajlovi su deklaracioni fajlovi za TypeScript. Oni sadrže popise dostupnih tipova.

```
"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
class PriorityQueue {
  /**
   *
   * @param options
   * options.comparator: function used to compare elements;
   * options.initialElements: elements to be put in priority queue initially in O(n) time
   */
  constructor(options) {
    this.heap = [];
    this.comparator = options.comparator;
    if (options.initialElements)
      this.buildHeap(options.initialElements);
  }
  /**
   *
   * @returns size of priority queue in O(1)
   */
}
```

```

size() {
  return this.heap.length;
}
/**
 *
 * @returns is priority queue empty in O(1)
 */
isEmpty() {
  return this.size() === 0;
}
/**
 *
 * @returns top of priority queue in O(1), if priority queue is empty returns undefined
 */
peek() {
  return this.heap[0] ? JSON.parse(JSON.stringify(this.heap[0])) : undefined;
}
/**
 * clears priority queue in O(1)
 */
clear() {
  this.heap = [];
}
/**
 * checks if value exists in priority queue in O(n)
 */
has(value) {
  return !!this.heap.find((ele) => ele === value);
}
/**
 *
 * @returns all values of priority queue in O(n)
 */
values() {
  return JSON.parse(JSON.stringify(this.heap));
}
buildHeap(array) {
  this.heap = JSON.parse(JSON.stringify(array));
  for (let i = Math.floor(array.length / 2) - 1; i >= 0; i--) {
    this.siftDown(i);
  }
}
/**
 *
 * @param value element to be added to heap, adds it in O(log n) operations, n is size of heap
 * @returns size of heap

```

```

    */
    push(value) {
        this.heap.push(value);
        this.siftUp();
        return this.size();
    }
    /**
     *
     * @param values elements to be added to heap, adds it in  $O(k * \log n)$  operations, n is size of heap,
    k is number of elements added
     * @returns size of heap
     */
    pushMany(values) {
        values.forEach((value) => {
            this.push(value);
        });
        return this.size();
    }
    /**
     *
     * @returns top of priority queue and removes it from priority queue in  $O(\log n)$ , if priority queue is
    empty returns undefined
     */
    pop() {
        if (this.isEmpty()) {
            return undefined;
        }
        const returnValue = this.peek();
        const lastIndexOfHeapArray = this.size() - 1;
        if (lastIndexOfHeapArray > 0) {
            this.swap(0, lastIndexOfHeapArray);
        }
        this.heap.pop();
        this.siftDown();
        return returnValue;
    }
    getParent(index) {
        return ((index + 1) >>> 1) - 1;
    }
    getLeftChild(index) {
        return (index << 1) + 1;
    }
    getRightChild(index) {
        return (index + 1) << 1;
    }
    compareByIndex(i, j) {

```

```

    return this.comparator(this.heap[i], this.heap[j]);
  }
  swap(i, j) {
    [this.heap[i], this.heap[j]] = [this.heap[j], this.heap[i]];
  }
  siftUp() {
    let node = this.size() - 1;
    while (node > 0 && this.compareByIndex(node, this.getParent(node))) {
      const parentNode = this.getParent(node);
      this.swap(node, parentNode);
      node = parentNode;
    }
  }
  siftDown(node = 0) {
    let leftChild = this.getLeftChild(node);
    let rightChild = this.getRightChild(node);
    while ((leftChild < this.size() && this.compareByIndex(leftChild, node)) ||
      (rightChild < this.size() && this.compareByIndex(rightChild, node))) {
      const maxChild = rightChild < this.size() && this.compareByIndex(rightChild, leftChild) ?
rightChild : leftChild;
      this.swap(node, maxChild);
      node = maxChild;
      leftChild = this.getLeftChild(node);
      rightChild = this.getRightChild(node);
    }
  }
}
exports.default = PriorityQueue;

```

Tabela 27 - lib\queue.js

U queue.js se nalazi implementacija funkcija, prevedenih u JavaScript.

```

import { IPriorityQueue, IPriorityQueueOptions } from './queueInterfaces';
export default class PriorityQueue<T> implements IPriorityQueue<T> {
  private heap;
  private comparator;
  /**
   *
   * @param options
   * options.comparator: function used to compare elements;
   * options.initialElements: elements to be put in priority queue initially in O(n) time
   */
  constructor(options: IPriorityQueueOptions<T>);
  /**
   *
   * @returns size of priority queue in O(1)
   */

```

```

size(): number;
/**
 *
 * @returns is priority queue empty in O(1)
 */
isEmpty(): boolean;
/**
 *
 * @returns top of priority queue in O(1), if priority queue is empty returns undefined
 */
peek(): T;
/**
 * clears priority queue in O(1)
 */
clear(): void;
/**
 * checks if value exists in priority queue in O(n)
 */
has(value: T): boolean;
/**
 *
 * @returns all values of priority queue in O(n)
 */
values(): T[];
private buildHeap;
/**
 *
 * @param value element to be added to heap, adds it in O(log n) operations, n is size of heap
 * @returns size of heap
 */
push(value: T): number;
/**
 *
 * @param values elements to be added to heap, adds it in O(k * log n) operations, n is size of heap,
k is number of elements added
 * @returns size of heap
 */
pushMany(values: T[]): number;
/**
 *
 * @returns top of priority queue and removes it from priority queue in O(log n), if priority queue is
empty returns undefined
 */
pop(): T;
private getParent;
private getLeftChild;

```

```

private getRightChild;
private compareByIndex;
private swap;
private siftUp;
private siftDown;
}

```

Tabela 28 - lib\queue.d.ts

Fajl queue.d.ts je deklaracioni fajl i sadrži tip za prioritetni red, koji će koristiti TypeScript korisnici ove biblioteke.

```

export declare type PriorityQueueComparator<T> = (a: T, b: T) => boolean;
export interface IPriorityQueueOptions<T> {
    comparator: PriorityQueueComparator<T>;
    initialElements?: T[];
}
export interface IPriorityQueue<T> {
    size(): number;
    isEmpty(): boolean;
    peek(): T;
    push(value: T): void;
    pushMany(values: T[]): void;
    pop(): T;
    clear(): void;
    has(value: T): boolean;
    values(): T[];
}

```

Tabela 29 - lib\queueInterfaces.d.ts

Fajl queueInterfaces.d.ts sadrži tipove i interfejsse formirane za TypeScript od queueInterfaces.ts fajla.

```

"use strict";
Object.defineProperty(exports, "__esModule", { value: true });

```

Tabela 30 - lib\queueInterfaces.js

queueInterfaces.js je praktično prazan i sadrži samo sintaksnu za export-ovanje među različitim modulima. Da želimo da omogućimo korištenje u različitim vrstama modula već je ranije označeno opcijom „esModuleInterop“ u tsconfig fajlu.

4.10 Dodavanje LICENSE fajla

Računarski softver često prate licence, koje govore šta se može raditi sa tim softverom. Open source kod prate licence koje dozvoljavaju slobodnu upotrebu, modifikaciju i podjelu koda. Najčeće licence za otvoren kod su ISC, MIT, BSD, ali postoje i druge. U ovom projektu izabrana je ISC, koja je suštinski slična kao i ostale, ali napisana jednostavnijim jezikom.

Pri kreiranju biblioteke uz npm bio je ponuđen izbor licenciranja. U ovom koraku bi se samo naziv licence dodao u package.json fajla, ali ne bi dobili i generisan LICENSE fajl. Ukoliko tada nije izabrana licenca, može se naknadno jednostavno dodati u package.json.

```
...  
"license": "ISC",  
...
```

Tabela 31 - licenca u package.json

Zatim je neophodno kreirati LICENSE fajl. Za to se može iskoristiti open-source biblioteka sa npm-a generate-license (<https://www.npmjs.com/package/generate-license>), verzija 1.0.0.

```
npm install --global generate generate-license
```

Tabela 32 - komanda za instalaciju paketa za licenciranje

Prvo je neophodno globalno instalirati pakete generate i generate-license. Paket generate je projekat istih autora koji nudi alat koji se koristi preko komande linije, i koristi generatore (kao što je generate-license) za generisanje određenih vrsta fajlova. Pored generatora za licencu postoje i razni generatori za README fajlove, testne fajlove, fajlove za integraciju sa Travisom ali i razne druge vrste fajlova.

```
gen license:isc
```

Tabela 33 - komanda za generisanje licencnog fajla

Komandom iz tabele 33. kreiran je LICENSE fajl za projekat, sa ISC deklaracijom.

Ulaskom u ovaj fajl preko GitHub-a može se u jednostavnim crtama vidjeti da autor dozvoljava komercijanu upotrebu, distribuciju, modifikaciju i privatnu upotrebu ovoga koda. Takođe primjećuje se da licenca sadrži ograničenja kojim ne pruža bilo kakve garancije, uključujući garanciju pouzdanosti.

4.11 Kreiranje primjera

Kako bi korisnici biblioteke mogli lakše da je upotrijebe, dodati su primjeri za njenu upotrebu. Napravljeni su primjeri za upotrebu i u JavaScript-u i u TypeScript-u.

4.11.1 Kreiranje primera u JavaScript-u

U root direktoriju biblioteke, u sample-projects folderu, koji je kreiran kako bi u njemu bili smješteni testni projekti, kreiran je folder sa imenom primjer projekta. U ovom slučaju folder je nazvan „javascript“.

```
npm init -y
```

Tabela 34 - komanda za inicijalizaciju JavaScript projekta

Komandom iz tabele 34. je pokrenuto kreiranje aplikacije. Zastavicom „-y“ kaže se npm-u da se prihvataju sva podrazumijevana podešavanja. Potom se dobije kreiran package.json za primjer projekat.

```
{
  "name": "javascript",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Tabela 35 - package.json js primjer projekta

```
npm i priority-queue-with-custom-comparator
```

Tabela 36 – komanda za instalaciju ranije kreiranog paketa

```
{
  "name": "test-library-with-js",
  "version": "1.0.0",
  "description": "",
  "type": "module",
  "main": "index.js",
  "scripts": {
    "start": "node ./index.js",
  }
}
```



```

    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "priority-queue-with-custom-comparator": "^1.0.2"
  }
}

```

Tabela 37 - package.json js primjer projekta nakon instalacije paketa

Instalirana je kreirana biblioteka i ona se automatski dodaje u package.json fajl. Pored toga odrađene su određene modifikacije na package.json fajlu kao što su izmjena imena projekta i dodavanje skripte za pokretanje projekta.

```

import PriorityQueue from 'priority-queue-with-custom-comparator';

const q = new PriorityQueue.default({
  comparator: (a, b) => {
    return a - b < 0;
  },
});
q.pushMany([-1, 4, 8, -9]);
q.push(2);

console.log('Queue size: ', q.size());

```

Tabela 38 - index.js

Potom je dodat index.js fajl koji sadržati osnovni rad sa bibliotekom. To je u ovom slučaju kreiranje prioritenog reda, dodavanje na njega, i štampanje njegove veličine.

Da bi u JavaScriptu sa ECMA Sript modulima koristili ovu biblioteku mora se koristiti `PriorityQueue.default` sintaksa pri import-u, jer je kreirana biblioteka rađena za CommonJS module, što je Node.js-ov standard.

```
FROM node:14-alpine
WORKDIR /usr/src/app
COPY ["package.json", "package-lock.json*", "./"]
RUN npm install --silent && mv node_modules ../
COPY . .
RUN chown -R node /usr/src/app
USER node
CMD ["node", "index.js"]
```

Tabela 39 - Dockerfile js primjer projekta

```
**/.dockerignore
**/.vs
**/.vscode
**/docker-compose*
**/compose*
**/Dockerfile*
**/node_modules
README.md
```

Tabela 40 - .dockerignore js primjer projekta

Kako bi korisnici kreirane biblioteke mogli da lakše pokrenu primjer projekat, dodat je Dockerfile iz kojeg se može kreirati Docker slika (Docker image). `.dockerignore` fajl sadrži fajlove koje nije potrebno koristiti prilikom kreiranja slike. Ovi fajlovi se mogu kreirati ručno, ali je lakša opcija da ukoliko se koristi VS Code, da se iskoristi Docker ekstenzija.

Tako će prilikom kreiranja slike, Docker skinuti kompresovanu sliku Node.js, verzije 14, i podesiti radni folder u okviru slike. Zatim kopira `package.json` i `package-lock.json` i instalira sve zavisnosti, i pomjera `node_modules` folder (Dockerova preporuka kako bi se izbjegli razni OS vezani problemi). Kopiraju se ostali fajlovi, mijenja se vlasnik svih fajlova, prebacuje se na drugog korisnika i pokreće projekat.

4.11.2 Kreiranje primjera u TypeScript-u

Za kreiranje primjer projekta u TypeScript-u korišćen je NestJS, jedan od najpopularnijih Node.js frejmworka, koji pruža potpunu podršku za TypeScript.

U root direktoriju biblioteke, u sample-projects folderu, koji je kreiran kako bi u njemu bili smješteni testni projekti, kreiran je folder sa imenom primjer projekta. U ovom slučaju folder je nazvan „nestjs-typescript“.

```
npm i -g @nestjs/cli
```

Tabela 41 - komanda za instalaciju nest-a

Da bi koristili ovaj frejmwork neophodno ga je instalirati globalno na razvojnoj mašini.

```
nest new nestjs-typescript
```

Tabela 42 - komanda za kreiranje novog nest projekta

Zatim je kreiran primjer projekat preko NestJS-a. Prilikom kreiranja ponuđen je izbor paket menadžera koji bi se koristio. Kreirana biblioteka podržava rad sa svim paket menadžerima, ali u ovom radu je izabrano da se radi sa npm-om.

NestJS kreira konfiguracijone fajlove za TypeScript, eslint, nest, kreira git fajlove, README fajl, package.json i package-lock.json, src folder koji sadrži kodove, test folder za testove ali i node_modules. U src folderu kreira main.ts fajl iz koga se pokreće projekat, kao i jedan module, app module koji se pokreće iz main fajla. Za app module kreira module fajl, fajl za kontroler, fajl za servis i fajl za testove.

Da ne bi imali konfikata sa sistemom za kontrolu verzije biblioteke, uklonjen je kreirani folder .git i fajl .gitignore. Da ne bi imali problema sa lintovanjem biblioteke uklonjeni su .prettierrc i .eslintrc.js.

Uklonjen je takođe i tesni fajl za app module, jer se ne koristiti u ovom primjeru. Primjer je stavljen direktno u main.ts fajl.

```
npm i priority-queue-with-custom-comparator
```

Tabela 43 - komanda za instalaciju ranije kreiranog paketa

Potom je instalirana kreirana biblioteka.

```
/*import { NestFactory } from '@nestjs/core';  
import { AppModule } from './app.module';*/  
import PriorityQueue from 'priority-queue-with-custom-comparator';  
async function bootstrap() {
```

```

/* const app = await NestFactory.create(AppModule);

await app.listen(3000);*/

const q = new PriorityQueue<number>({
  comparator: (a, b) => {
    return a - b < 0;
  },
});

q.pushMany([-1, 4, 8, -9]);

q.push(2);

console.log('Queue size: ', q.size());
}

bootstrap();

```

Tabela 44 - main.ts

Fajl main.ts sadržati osnovni rad sa bibliotekom. To je u ovom slučaju kreiranje prioritenog reda, dodavanje na njega, i štampanje njegove veličine. Sintaksa za kreiranje i pokretanje app modula je zakomentarisana. Ovo nije pravilno korištenje NestJS frejmworka ali je dovoljno dobar primjer da pokaže kako raditi sa ovom bibliotekom u TypeScript-u.

```

FROM node:14-alpine

WORKDIR /usr/src/app

COPY ["package.json", "package-lock.json*", "./*"]

RUN npm i -g --silent @nestjs/cli@8.1 && npm install --silent && mv node_modules ../

COPY . .

RUN chown -R node /usr/src/app

USER node

CMD ["npm", "start"]

```

Tabela 45 - Dockerfile ts primjer projekta

```

**/.dockerignore

**/.vs

```

```
*/.vscode
*/docker-compose*
*/compose*
*/Dockerfile*
*/node_modules
README.md
```

Tabela 46 - .dockerignore ts primjer projekta

Na kraju su dodati i fajlovi za Docker. Dockerfajl za ovaj projekat će biti sličan onome za javascript, jedina razlika će biti što će se instalirati i NestJS.

4.12 Pisanje testova i pokrivenost testovima

Pisanje testova je popularna praksa u razvoju softvera koja uzima sve više maha. Time pravimo kod koji je pouzdaniji, čitljiviji i lakše izmjenjiv. Najpopularnije vrste testova su unit, integracioni, end-to-end, smoke, ali postoje i mnoge druge. Developeri najčešće pišu unit, integracione i ent-to-end testove, dok i drugi timovi pišu mnoge vrste testova.

U okviru ove biblioteke implementirani su unit testovi, jer jedino njih i ima smisla implementirati u ovakvom projektu. Unit testovi su testovi koji testiraju sitne jedinice koda, najčešće testiraju da li funkcije za pravi ulaz vraćaju pravi izlaz i izazivaju li prave sporedne efekte.

Ovaj projekat ima stopostotnu pokrivenost unit testovima testiranih djelova koda, odnosno queue.ts fajla. Iako je što veća pokrivenost testovima poželjna, ona nije garant da u testovima ne postoji greška, jer testovi mogu i dalje preskočiti neki bitan slučaj, ili očekivati pogrešan rezultat neke operacije.

Postoji više frejmworka za testiranje za JavaScript, a u ovoj projektu korišćen je Jest.

```
{
  "transform": {
    "^.+\\. (tj) sx?$": "ts-jest"
  },
  "testRegex": "(/__tests___/.*(\\.) (spec))\\. (js|ts)$",
  "moduleFileExtensions": [
    "ts",
```

```

    "tsx",
    "js",
    "jsx",
    "json",
    "node"
  ],
  "collectCoverageFrom": [
    "src/queue.ts",
    "!**/__tests__/**",
    "!**/node_modules/**"
  ]
}

```

Tabela 47 - jestconfig.json

U jestconfig.json definisana su pravila koja će koristiti Jest. Rečeno je Jest-u koje fajlove da koristi za testiranje, gdje su testovi, te za koje fajlove da mjeri pokrivenost testovima.

Pošto su testovi manje više slični, pokazan je samo dio testova u radu, dok ostatak se može naći u git repozitorijumu navedenom na početku rada.

```

export const defaultMaxComparator = (a: number, b: number): boolean => {
  return a > b;
};

```

Tabela 48 - __tests__/test.helper.ts

Napravljen je pomoćni fajl za testove, iz koga se eksportuje compare funkcija za cijele brojeve, koja ih sortira od manjeg ka većem.

```

import PriorityQueue from '../src/queue';
import { defaultMaxComparator } from './test.helper';

test('initial state (created with initialElements)', () => {
  const numberPriorityQueue = new PriorityQueue<number>({

```

```

    comparator: defaultMaxComparator,
    initialElements: [2, 3, 1],
  });

  expect(numberPriorityQueue.size()).toBe(3);

  numberPriorityQueue.clear();

  expect(numberPriorityQueue.size()).toBe(0);
});

test('initial state (created without initialElements)', () => {
  const numberPriorityQueue = new PriorityQueue<number>({
    comparator: defaultMaxComparator,
  });

  expect(numberPriorityQueue.size()).toBe(0);

  numberPriorityQueue.clear();
  vr
  expect(numberPriorityQueue.size()).toBe(0);
});

```

Tabela 49 - __tests__/clear.spec.ts

Testovi za clear funkciju su krajnje jednostavni. Testira se clear funkcija nad kreiranim prioritetnim redom, kreiranim sa i bez inicijalnih elemenata. U ovim testovima se „očekuje“ da funkcija size nakon pozvane funkcije clear vrati nulu.

```

import PriorityQueue from '../src/queue';
import { defaultMaxComparator } from './test.helper';

```

```

afterEach(() => {
  jest.clearAllMocks();
});

test('initial state (created with initialElements)', () => {
  const numberPriorityQueue = new PriorityQueue<number>({
    comparator: defaultMaxComparator,
    initialElements: [2, 6, 7],
  });
  expect(numberPriorityQueue.values().toString()).toBe([7, 6, 2].toString());
});

test('initial state (created without initialElements)', () => {
  const numberPriorityQueue = new PriorityQueue<number>({
    comparator: defaultMaxComparator,
  });
  expect(numberPriorityQueue.values().toString()).toBe([].toString());
});

```

Tabela 50 - __tests__/constructor.spec.ts

Da bi testirali konstruktor, jednostavno se nakon kreiranja prioritenog reda pozove funkcija values, koja bi trebala da vrati niz, koji sadrži iste elemente kao onaj koji je predat za initial values. Ako ga nije bilo, onda očekujemo da će values da vrati prazan niz.

```

import PriorityQueue from '../src/queue';
import { defaultMaxComparator } from './test.helper';

test('initial state (created without initialElements) and some added', () => {
  const numberPriorityQueue = new PriorityQueue<number>({

```



```
    comparator: defaultMaxComparator,  
});  
  
expect(numberPriorityQueue.has(5)).toBe(false);  
expect(numberPriorityQueue.has(2)).toBe(false);  
expect(numberPriorityQueue.has(0)).toBe(false);  
expect(numberPriorityQueue.has(-1)).toBe(false);  
expect(numberPriorityQueue.has(6)).toBe(false);  
expect(numberPriorityQueue.has(-2)).toBe(false);  
expect(numberPriorityQueue.has(-9)).toBe(false);  
expect(numberPriorityQueue.has(58)).toBe(false);  
expect(numberPriorityQueue.values().toString()).toBe([].toString());  
expect(numberPriorityQueue.size()).toBe(0);
```

```
numberPriorityQueue.push(5);
```

```
expect(numberPriorityQueue.has(5)).toBe(true);  
expect(numberPriorityQueue.has(2)).toBe(false);  
expect(numberPriorityQueue.has(0)).toBe(false);  
expect(numberPriorityQueue.has(-1)).toBe(false);  
expect(numberPriorityQueue.has(6)).toBe(false);  
expect(numberPriorityQueue.has(-2)).toBe(false);  
expect(numberPriorityQueue.has(-9)).toBe(false);  
expect(numberPriorityQueue.has(58)).toBe(false);  
expect(numberPriorityQueue.has(185)).toBe(false);  
expect(numberPriorityQueue.has(201)).toBe(false);  
expect(numberPriorityQueue.values().toString()).toBe([5].toString());  
expect(numberPriorityQueue.size()).toBe(1);
```

```
numberPriorityQueue.push(-1);

expect(numberPriorityQueue.has(5)).toBe(true);
expect(numberPriorityQueue.has(2)).toBe(false);
expect(numberPriorityQueue.has(0)).toBe(false);
expect(numberPriorityQueue.has(-1)).toBe(true);
expect(numberPriorityQueue.has(6)).toBe(false);
expect(numberPriorityQueue.has(-2)).toBe(false);
expect(numberPriorityQueue.has(-9)).toBe(false);
expect(numberPriorityQueue.has(58)).toBe(false);
expect(numberPriorityQueue.has(185)).toBe(false);
expect(numberPriorityQueue.has(201)).toBe(false);
expect(numberPriorityQueue.values().toString()).toBe([5, -1].toString());
expect(numberPriorityQueue.size()).toBe(2);
```

```
numberPriorityQueue.push(2);

expect(numberPriorityQueue.has(5)).toBe(true);
expect(numberPriorityQueue.has(2)).toBe(true);
expect(numberPriorityQueue.has(0)).toBe(false);
expect(numberPriorityQueue.has(-1)).toBe(true);
expect(numberPriorityQueue.has(6)).toBe(false);
expect(numberPriorityQueue.has(-2)).toBe(false);
expect(numberPriorityQueue.has(-9)).toBe(false);
expect(numberPriorityQueue.has(58)).toBe(false);
expect(numberPriorityQueue.has(185)).toBe(false);
expect(numberPriorityQueue.has(201)).toBe(false);
```

```
expect(numberPriorityQueue.values().toString()).toBe([5, -1, 2].toString());  
expect(numberPriorityQueue.size()).toBe(3);
```

```
numberPriorityQueue.push(6);
```

```
expect(numberPriorityQueue.has(5)).toBe(true);  
expect(numberPriorityQueue.has(2)).toBe(true);  
expect(numberPriorityQueue.has(0)).toBe(false);  
expect(numberPriorityQueue.has(-1)).toBe(true);  
expect(numberPriorityQueue.has(6)).toBe(true);  
expect(numberPriorityQueue.has(-2)).toBe(false);  
expect(numberPriorityQueue.has(-9)).toBe(false);  
expect(numberPriorityQueue.has(58)).toBe(false);  
expect(numberPriorityQueue.has(185)).toBe(false);  
expect(numberPriorityQueue.has(201)).toBe(false);  
expect(numberPriorityQueue.values().toString()).toBe(  
    [6, 5, 2, -1].toString(),  
);  
expect(numberPriorityQueue.size()).toBe(4);
```

```
numberPriorityQueue.push(6);
```

```
expect(numberPriorityQueue.has(5)).toBe(true);  
expect(numberPriorityQueue.has(2)).toBe(true);  
expect(numberPriorityQueue.has(0)).toBe(false);  
expect(numberPriorityQueue.has(-1)).toBe(true);  
expect(numberPriorityQueue.has(6)).toBe(true);  
expect(numberPriorityQueue.has(-2)).toBe(false);
```

```
expect(numberPriorityQueue.has(-9)).toBe(false);
expect(numberPriorityQueue.has(58)).toBe(false);
expect(numberPriorityQueue.has(185)).toBe(false);
expect(numberPriorityQueue.has(201)).toBe(false);
expect(numberPriorityQueue.values().toString()).toBe(
  [6, 6, 2, -1, 5].toString(),
);
expect(numberPriorityQueue.size()).toBe(5);

numberPriorityQueue.push(-2);

expect(numberPriorityQueue.has(5)).toBe(true);
expect(numberPriorityQueue.has(2)).toBe(true);
expect(numberPriorityQueue.has(0)).toBe(false);
expect(numberPriorityQueue.has(-1)).toBe(true);
expect(numberPriorityQueue.has(6)).toBe(true);
expect(numberPriorityQueue.has(-2)).toBe(true);
expect(numberPriorityQueue.has(-9)).toBe(false);
expect(numberPriorityQueue.has(58)).toBe(false);
expect(numberPriorityQueue.has(185)).toBe(false);
expect(numberPriorityQueue.has(201)).toBe(false);
expect(numberPriorityQueue.values().toString()).toBe(
  [6, 6, 2, -1, 5, -2].toString(),
);
expect(numberPriorityQueue.size()).toBe(6);

numberPriorityQueue.push(-9);
```

```
expect(numberPriorityQueue.has(5)).toBe(true);
expect(numberPriorityQueue.has(2)).toBe(true);
expect(numberPriorityQueue.has(0)).toBe(false);
expect(numberPriorityQueue.has(-1)).toBe(true);
expect(numberPriorityQueue.has(6)).toBe(true);
expect(numberPriorityQueue.has(-2)).toBe(true);
expect(numberPriorityQueue.has(-9)).toBe(true);
expect(numberPriorityQueue.has(58)).toBe(false);
expect(numberPriorityQueue.has(185)).toBe(false);
expect(numberPriorityQueue.has(201)).toBe(false);
expect(numberPriorityQueue.values().toString()).toBe(
  [6, 6, 2, -1, 5, -2, -9].toString(),
);
expect(numberPriorityQueue.size()).toBe(7);
```

```
numberPriorityQueue.push(58);
```

```
expect(numberPriorityQueue.has(5)).toBe(true);
expect(numberPriorityQueue.has(2)).toBe(true);
expect(numberPriorityQueue.has(0)).toBe(false);
expect(numberPriorityQueue.has(-1)).toBe(true);
expect(numberPriorityQueue.has(6)).toBe(true);
expect(numberPriorityQueue.has(-2)).toBe(true);
expect(numberPriorityQueue.has(-9)).toBe(true);
expect(numberPriorityQueue.has(58)).toBe(true);
expect(numberPriorityQueue.has(185)).toBe(false);
expect(numberPriorityQueue.has(201)).toBe(false);
expect(numberPriorityQueue.values().toString()).toBe(
```

```
[58, 6, 2, 6, 5, -2, -9, -1].toString(),
);
expect(numberPriorityQueue.size()).toBe(8);

numberPriorityQueue.push(185);

expect(numberPriorityQueue.has(5)).toBe(true);
expect(numberPriorityQueue.has(2)).toBe(true);
expect(numberPriorityQueue.has(0)).toBe(false);
expect(numberPriorityQueue.has(-1)).toBe(true);
expect(numberPriorityQueue.has(6)).toBe(true);
expect(numberPriorityQueue.has(-2)).toBe(true);
expect(numberPriorityQueue.has(-9)).toBe(true);
expect(numberPriorityQueue.has(58)).toBe(true);
expect(numberPriorityQueue.has(185)).toBe(true);
expect(numberPriorityQueue.has(201)).toBe(false);
expect(numberPriorityQueue.values().toString()).toBe(
  [185, 58, 2, 6, 5, -2, -9, -1, 6].toString(),
);
expect(numberPriorityQueue.size()).toBe(9);

numberPriorityQueue.push(201);

expect(numberPriorityQueue.has(5)).toBe(true);
expect(numberPriorityQueue.has(2)).toBe(true);
expect(numberPriorityQueue.has(0)).toBe(false);
expect(numberPriorityQueue.has(-1)).toBe(true);
expect(numberPriorityQueue.has(6)).toBe(true);
```

```

expect(numberPriorityQueue.has(-2)).toBe(true);
expect(numberPriorityQueue.has(-9)).toBe(true);
expect(numberPriorityQueue.has(58)).toBe(true);
expect(numberPriorityQueue.has(185)).toBe(true);
expect(numberPriorityQueue.has(201)).toBe(true);
expect(numberPriorityQueue.values().toString()).toBe(
  [201, 185, 2, 6, 58, -2, -9, -1, 6, 5].toString(),
);
expect(numberPriorityQueue.size()).toBe(10);
});

```

Tabela 51 - __tests__/push.spec.ts

Funkcija push je testirana tako što se nakon svakog poziva push funkcije provjeri da li se u prioritetnom redu nalaze ispravni elementi, te da li je prioritetni red prave veličine.

```

...
"scripts": {
...
  "test": "jest --config jestconfig.json",
  "test:coverage": "jest --config jestconfig.json --coverage",
},
...

```

Tabela 52 - skripte vezane za testove u package.json

U package.json dodata je skripta za pokretanje testova, i kreiranje izvještaja o pokrivenosti testovima.

Nakon pokretanja test:coverage skripte, dobija se coverage folder, koji sadržati metapodatke o pokrivenosti testovima, ali i preglednu veb stranicu sa prikazanim podacima o tome.

4.13 Pisanje README fajla

Kako bi drugi znali šta je ova biblioteka, kako se koristi, koje su njene funkcionalnosti, ali dobili i druge informacije napravljen je README.md fajl. To je prilično popularna vrsta fajla pisana u Markdown jeziku.

```
# priority-queue-with-custom-comparator
```

```
![npm](https://img.shields.io/npm/v/priority-queue-with-custom-comparator)
![npm](https://img.shields.io/npm/dm/priority-queue-with-custom-comparator.svg))(https://
www.npmjs.com/package/priority-queue-with-custom-comparator)
![npm](https://img.shields.io/badge/node-%3E=%206.0-blue.svg))(https://www.npmjs.com/package/
priority-queue-with-custom-comparator)      ![GitHub      repo
size](https://img.shields.io/github/repo-size/dejtor/priority-queue-with-custom-comparator)      ![npm]
(https://img.shields.io/npm/l/priority-queue-with-custom-comparator)      ![GitHub      last
commit](https://img.shields.io/github/last-commit/dejtor/priority-queue-with-custom-comparator)      !
[Travis (.com)](https://img.shields.io/travis/com/dejtor/priority-queue-with-custom-comparator)
```

```

```

Priority queue implemented using Heap data structure, allows using custom comparator function.

```
# Contents
```

```
- [Example](#Example)
- [TS](#TS)
- [JS](#JS)
- [Functions](#Functions)
```

```
## Example:
```

```
### TS
```

```
...
```

```
import PriorityQueue from 'priority-queue-with-custom-comparator';
```



```

class Rand {
  num: number;
}

const numberPriorityQueue = new PriorityQueue<number>({
  comparator: (a, b) => {
    return a - b < 0;
  },
  initialElements: [3, 1],
});
numberPriorityQueue.pushMany([-1, 4, 8, -9]);
numberPriorityQueue.push(2);

console.log('top of queue', numberPriorityQueue.pop());
console.log('top of queue', numberPriorityQueue.peek());
console.log(
  'size after inserting 1, 2 and 3',
  numberPriorityQueue.pushMany([1, 2, 3]),
);

const classPriorityQueue = new PriorityQueue<Rand>({
  comparator: (a, b) => a.num > b.num,
});

classPriorityQueue.pushMany([
  { num: 5 },
  { num: 1 },

```

```

    { num: -9 },
    { num: 11 },
    { num: 15 },
    { num: 51 },
    { num: 155 },
  ]);

  console.log('classPriorityQueue: ', classPriorityQueue.values());

  const stringPriorityQueue = new PriorityQueue<string>({
    comparator: (a, b) => a.length > b.length,
  });

  stringPriorityQueue.pushMany(['abcd', 'a', 'abcdeef', 'string']);

  console.log('stringPriorityQueue: ', stringPriorityQueue.values());
  ...

  ### JS

  ...

  import PriorityQueue from 'priority-queue-with-custom-comparator'

  class Rand {
    num;
  }

  const numberPriorityQueue = new PriorityQueue.default({

```

```

    comparator: (a, b) => {
      return a - b < 0;
    },
    initialElements: [3, 1],
  });
numberPriorityQueue.pushMany([-1, 4, 8, -9]);
numberPriorityQueue.push(2);

console.log('top of queue', numberPriorityQueue.pop());
console.log('top of queue', numberPriorityQueue.peek());
console.log('size after inserting 1, 2 and 3', numberPriorityQueue.pushMany([1, 2, 3]));

const classPriorityQueue = new PriorityQueue.default({ comparator: (a, b) => a.num > b.num });

classPriorityQueue.pushMany([
  { num: 5 },
  { num: 1 },
  { num: -9 },
  { num: 11 },
  { num: 15 },
  { num: 51 },
  { num: 155 },
]);

console.log('classPriorityQueue: ', classPriorityQueue.values());

const stringPriorityQueue = new PriorityQueue.default({ comparator: (a, b) => a.length > b.length });

stringPriorityQueue.pushMany(['abcd', 'a', 'abcdeef', 'string']);

```

```

console.log('stringPriorityQueue: ', stringPriorityQueue.values());
...

## Functions:

...

/**
 *
 * @param options
 * options.comparator: function used to compare elements;
 * options.initialElements: (optional) elements to be put in priority queue initially in O(n) time
 */
constructor(options: PriorityQueueOptions<T>);

/**
 *
 * @returns size of priority queue in O(1)
 */
size(): number;

/**
 *
 * @returns is priority queue empty in O(1)
 */
isEmpty(): boolean;

/**
 *
 * @returns top of priority queue in O(1), if priority queue is empty returns undefined
 */

```

```

peek(): T;
/**
 * clears priority queue in O(1)
 */
clear(): void;
/**
 * checks if value exists in priority queue in O(n)
 */
has(value: T): boolean;
/**
 *
 * @returns all values of priority queue in O(n)
 */
values(): T[];
/**
 *
 * @param value element to be added to heap, adds it in O(log n) operations, n is size of heap
 * @returns size of heap
 */
push(value: T): number;
/**
 *
 * @param values elements to be added to heap, adds it in O(k * log n) operations, n is size of heap,
k is number of elements added
 * @returns size of heap
 */
pushMany(values: T[]): number;
/**

```

```

*

* @returns top of priority queue and removes it from priority queue in O(log n), if priority queue is
empty returns undefined

*/

pop(): T;

...

```

Tabela 53 - README.md

README fajl sadrži ime biblioteke, štitove (shields), oznaku da podržava i JavaScript i TypeScript, kraći opis, sadržaj, primjere u JavaScript-u i TypeScript-u ali i opise i složenosti svih javnih odnosno izvezenih funkcija. Stavljeni su štitovi koje pokazuju najnoviju verziju biblioteke, broj mjesečnih preuzimanja, neophodnu verziju Node-a, veličinu repozitorije, licencu, datum zadnjeg commit-a, i da li prolaze svi testovi po Travis-u.

4.14 Upotreba Travis-a za automatsko izvršavanje testova

Kako bi nakon svakog push-a koda na GitHub bili izvršeni testovi, i javno pokazano da oni uspješno prolaze korišten je Travis. Time je korisnicima ulivena dodatna sigurnost prilikom korišćenja ove biblioteke.

```

language: node_js

node_js:

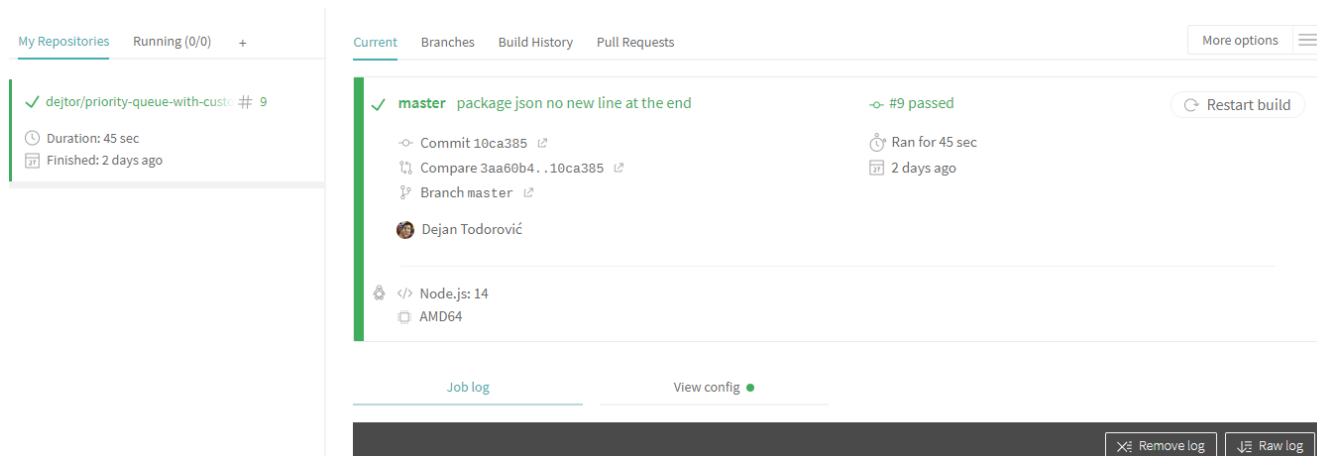
- 14

```

Tabela 54 - .travis.yml

Travis fajl se piše u yaml formatu i dovoljno je bilo napomenuti koja tehnologija i koja verzija se koristi. Takođe je potrebno da u package.son fajlu imamo test skriptu, koju smo već dodali.

Potom je neophodno bilo registrovati se na <https://www.travis-ci.com/> , preko postojećeg GitHub naloga. Na sajtu Travisa, vrši se aktivacija, i izbor koje GitHub repozitorije se žele integrisati sa Travis-om, u ovom slučaju kreirana biblioteka. Time je završena integraciju, i nakon svakog sljedećeg push-ovanog commit-a Travis izvršava testove, meglom autora obavještava o rezultatu, i shodno tome održava štit koji se koristi u README fajlu.



Slika 4 - početna stranica Travis-a

Na početnoj stranici Travis-a mogu se naći rezultati, zajedno sa detaljnim logovima, istorijom izvršavanja ali i drugim detaljima i opcijama.

4.15 Objavljivanje biblioteke

Na kraju biblioteku je bilo neophodno objaviti na npm registar.

```
npm publish
```

Tabela 55 – komanda za objavljivanje biblioteke

Komandom publish vrši se objavljivanje paketa na npm registar, nakon toga biblioteka se može naći na sajtu npmjs-a, yarn-a, ali i drugih paket menadžera za JavaScript pakete.

Ukoliko želimo da biblioteku unapređujemo i mijenjamo preporučljivo je mijenjanje njene verzije.

```
npm version <major | minor | patch>
```

Tabela 56 - komanda za verzionisanje biblioteke

Zavisno od vrste izmjena mijenja se major, minor ili patch broj biblioteke, odnosno prvi, drugi ili treći broj u verziji. Ukoliko je napravljena izmjena, takva da kod koji je radio sa prošlom verzijom biblioteke bi mogao da prestane da radi prelaskom na noviju verziju, mijenja se major. Ukoliko je samo dodata neka nova funkcionalnost, mijenja se minor, a ukoliko se rade samo neke sitnije ispravke odnosno zakrpe, mijenja se patch.

```
...
"scripts": {
...

```

```

"prepare": "npm run build",
"prepublishOnly": "npm test && npm run lint",
"preversion": "npm run lint",
"version": "npm run format && git add -A src",
"postversion": "git push && git push --tags"
},
...

```

Tabela 57 - skripte za verzionisanje u package.json

Dodate su skripte koje će se izvršavati prilikom publish-ovanja i verzionisanja biblioteke.

Tako će prije nego kod bude publish-ovan, biti isprobano da li može uspješno da se build-uje, prolaze li testovi, i baca li linter neke greške.

Prilikom verzionisanja koda, kod će takođe biti lintovati, formatiran i push-ovan na remote granu.

```

...
"repository": {
  "type": "git",
  "url": "https://github.com/dejtor/priority-queue-with-custom-comparator.git"
},
"files": [
  "lib/**/*"
],
"keywords": [
  "priority-queue",
  "heap",
  "custom-comparator",
  "comparator",
  "data-structure",
  "data-structures",
  "priority",

```



```
"queue"
```

```
],
```

```
...
```

Tabela 58 - dodaci u package.json fajlu

Kako bi ova biblioteka bila bolje opisana na GitHub-u i stranicama povezanim sa npm registrom, u package.json fajl dodat je opis gdje se nalazi remote repozitorija, kao i ključne riječi za biblioteku. Takođe, dodato je koji fajlovi trebaju da whitelisted (odobreni) u npm release-u, kako bi izbjegli dodavanje nepotrebnih fajlova za korisnike biblioteke.

5 Zaključak

U ovom radu pokazan je postupak kreiranja biblioteke za Node.js i njeno objavljivanje na npm registar. Korištene su razne dobre prakse, pisanje testova, pisanje primjera i README fajlova, Continuous Integration alati, pravljeni Docker fajlovi, licencni fajlovi, korišteni linteri i formateri. Autor je sve ovo smatrao kao neophodan, ali ne i dovoljan, uslov da se napravi dobra biblioteka za npm. Node nije jedina tehnologija koja korisnicima dozvoljava da sami prave i objavljuju svoje biblioteke, i proces za druge tehnologije može da ima dosta sličnosti.

6 Bibliografija

<https://medium.com/selleo/top-trends-in-node-js-to-watch-in-2021-d94ff38cc31e> (vrijeme pristupa 4. Oktobar 2021 19:00 CEST)

<https://docs.travis-ci.com/user/for-beginners/> (vrijeme pristupa 4. Oktobar 2021 19:15 CEST)

https://en.wikipedia.org/wiki/Travis_CI (vrijeme pristupa 4. Oktobar 2021 19:20 CEST)

[https://en.wikipedia.org/wiki/npm_\(software\)](https://en.wikipedia.org/wiki/npm_(software)) (vrijeme pristupa 4. Oktobar 2021 19:30 CEST)

<https://en.wikipedia.org/wiki/Node.js> (vrijeme pristupa 4. Oktobar 2021 19:40 CEST)

<https://sr.wikipedia.org/sr-el/JavaScript> (vrijeme pristupa 4. Oktobar 2021 19:50 CEST)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript (vrijeme pristupa 4. Oktobar 2021 19:55 CEST)

<https://en.wikipedia.org/wiki/ESLint> (vrijeme pristupa 7. Oktobar 2021 22:40 CEST)

https://sr.wikipedia.org/sr-el/%D0%A0%D0%B5%D0%B4_%D1%81%D0%B0_%D0%BF%D1%80%D0%B8%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D1%82%D0%BE%D0%BC

(vrijeme pristupa 7. Oktobar 2021 22:50 CEST)

[https://sr.wikipedia.org/wiki/Hip_\(struktura_podataka\)](https://sr.wikipedia.org/wiki/Hip_(struktura_podataka)) (vrijeme pristupa 7. Oktobar 2021 22:55 CEST)