Zadatak - Step by Step – Github reporting alat

Kratak opis zadatka : Comand prompt aplikacija koja na prosleđeno ime Github korisnika poziva github rest API i vraća korisničke informacije i top repozitorije za tog korisnika koji se sortirani u opadajućem redosledu

1.Napraviti NPM projekat - npm init

2.Napraviti tsconfig.json – tsc –init

3.Napraviti src folder gde ćemo smeštati source fajlove

4. U tsconfig podesiti rootDir na src lokaciju „./src"

5.Napraviti js folder

6.U tsconfig podesiti outDir na js lokaciju

7.U src folderu napraviti index.ts fajl

8.Za test ukucati console.log("Cao");

9.U package.json definisati start atribut u scripts I ukucati konandu tsc –p tsconfig.json && node js/index.js

10.Pokrenimo npm start

11.Dodajmo dependencies tj biblioteke , prva je request sa kojom ćemo pozivati REST API (možemo da koristimo HTTP API koji dolazi sa node.js ali radimo sa request) – npm install request --save
12.Dodajemo lodash biblioteku sa rad sa objektima (sortiranje objekata) - npm install loadash –save

13.Dodajmo types definitions za razvoj u TS kodu za request i za lodash – npm install @types/lodash @types/request –save-dev

14.Pogledajmo Github api koji ćemo da koristimo :

https://api.github.com/users/typescript

15.Pogledajmo repozitorije

https://api.github.com/users/typescript/repos

16.Napravićemo dve klase u koje ćemo smestiti podatke iz repozitorija. Jedna klasa će se zvati User ,a druga Repo.

17.User.ts treba da izgleda

```
1  export class User {
2      login: string;
3      fullName: string;
4      repoCount: number;
5      followerCount: number;
6      repos: Repo[];
7  }
```

```
import { Repo } from "./Repo";

export class User {
    login: string;
    fullName: string;
    repoCount: number;
    followerCount: number;
    repos: Repo[];
}
```

18.Repo.ts treba da izgleda

```
export class Repo {
    name: string;
    description: string;
    url: string;
    size: number;
    forkCount: number;
}
```

19.Napravimo novu klasu GitHubApiService.ts koja će pozivati Rest API i sadržavaće metod getUserInfo() I metod getRepos(). getUserInfo će pozivati na osnovu korisničkog imena podatke za korisnika sa GitHub servisa. Ovde ćemo importovati request biblioteku sa kojom ćemo pozvati servis GitHub.

Ovde imamo tri argumenta , prvi pozivamo servis i treci argument je callback funckija koja ima argument response i zatim će se izvršiti funkcija koja će vratiti vrednost koje dobije iz APIja

Moramo da koristimo tri argumenta callback funkciji

Moramo da koristimo I opcije da postavimo headers I to nam je drugi argument u get funckiji, ako nemamo opcije dodate dobicemo greku prilikom poziva API servisa da nam request nema User-Agent header

Koristimo treci parametar body za ispis rezulata API servisa

```
import * as request from 'request';

export class GithubApiService {
    getUserInfo(userName: string) {
        let options: any = {
            headers: {
                'User-Agent': 'request'
            }
        }
        request.get('https://api.github.com/users/' + userName, options, (error: any, response: any, body: any) => {
            console.log(body);
        })
    }

    getRepos() {

    }
}
```

20. Napraviti instancu servisa u index.ts , pre toga importovati GithubApi Service

```
1  import {GithubApiService} from './GithubApiService';
2
3  let svc = new GithubApiService();
```

21. Pozvati metod getUserInfo I proslediti mu parametar typescript

22.. Prvo napravimo konstruktor u User klasi koji prima parametar userResponse. Napravimo repos promenljivu opcionom

```
import { Repo } from "./Repo";

export class User {
    login: string;
    fullName: string;
    repoCount: number;
    followerCount: number;
    repos?: Repo[];

    constructor(userResponse: any) {
        this.login = userResponse.login;
        this.fullName = userResponse.name;
        this.repoCount = userResponse.public_repos;
        this.followerCount = userResponse.followers;
        // this.login = userResponse.login;

    }
}
```

23. Napravimo novu instancu User klase gde ćemo smestiti sadržaj response body koji smo ispisivali u konzolu, ispišimo ovu instancu u konzolu

```
request.get('https://api.github.com/users/' + userName, options, (error: any, response: any, body: any) => {
    let user = new User(JSON.parse(body));
    console.log(user);
})
```

Request biblioteka vraća response body kao string pa je neophodno da ju parsujemo u JSON objekat

24.Možemo ovo da uradimo i na drugi način ako u options dodamo property json:true ,tada ne treba JSON.parse metod

```
import * as request from 'request';
import { User } from "./User";

export class GithubApiService {
    getUserInfo(userName: string) {
        let options: any = {
            headers: {
                'User-Agent': 'request'
            },
            json: true
        }
        request.get('https://api.github.com/users/' + userName, options, (error: any, response: any, body: any) => {
            let user = new User(body);
            console.log(user);
        })
    }

    getRepos() {

    }
}
```

25.Možemo da koristimo promise kako bi mogli da upravljamo asinhronim izlazom nazad do korisnika, mi ćemo da koristimo callback za ove potrebe, a to već radimo preko get metode.

Pošto je i getUserInfo asinhrona metoda prilagodićemo je za prosleđivanje podataka tako što ćemo joj dodati još jednu metodu koja će biti callback funkcija i nazvaćemo je cb, a ona vraća any tip

```
export class GithubApiService {
    getUserInfo(userName: string, cb: (user: User) => any) {
```

U istoj klasi trebamo proslediti cb funkciji user instancu

```
request.get('https://api.github.com/users/' + userName, options, (error: any, response: any, body: any) => {
    let user = new User(body);
    cb(user)
})
```

Takođe treba izmeniti u index.ts fajlu poziv funkcije getUserInfo gde ćemo kao drugi parametar dodati callback funkciju koja će primiti User objekat kao parametar i zatim ga ispisati u konzolu

```
svc.getUserInfo('typescript', (user:User) =>{
      console.log(user);
)};
```

25a.Testirajmo napisan kod

26. Napravimo konstruktor u Repo klasi za prijem podataka iz repozitorija

```
export class Repo {
    name: string;
    description: string;
    url: string;
    size: number;
    forkCount: number;

    constructor(repo: any) {
        this.name = repo.name;
        this.description = repo.description;
        this.url = repo.html_url;
        this.size = repo.size;
        this.forkCount = repo.forks;

    }

}
```

27.Napišimo sada funkciju getRepos koja će nam vratiti sve repozitorije korisnika , obratimo pažnju d se vraća niz repozitorija koje moramo da hendlujemo

Koristićemo map funkciju koja svaki element niza smešta u objekat Repo tj. uzimamo pojedinačne repozitorije u svakom prolazu.

Map funkcija inače izvršava funkciju prolaskom kroz svaki element niza I vraća novi niz sa odgovarajućim povratnim vrednostima i sve vrednosti smešta u mapu .

```
getRepos(userName: string, cb: (repos: Repo[]) => any) {
    request.get('https://api.github.com/users/' + userName + '/repos' , OPTIONS, (error: any, response: any, body:
        let repos = body.map((repo: any) => new Repo(repo));
        cb(repos);

    })

}
```

28.Sada uradimo poziv funkcije getRepos u index.ts fajlu

```
svc.getRepos('typescript', (repos:Repos[]) =>{

      console.log(repos);

)};
```

29. Testirajmo napisan kod

30. Napravimo poziv funkcije u funkciji u index.ts tj prosledićemo jedan objekat u drugu funkciju , ovo smo već predvideli I modelom kada smo napisali da jedan User može imati više repozitorija,pogledajmo

```
1  import { Repo } from "./Repo";
2
3  export class User {
4      login: string;
5      fullName: string;
6      repoCount: number;
7      followerCount: number;
8      repos: Repo[];
9  }
```

E sada je potrebno da sjedinimo da poziva funkcije (ovo možemo jer koristimo asinhrone meteode,tj.callback funkcije)

```
svc.getUserInfo('typescript', (user:User) =>{
      svc.getRepos('typescript', (repos:Repos[]) =>{
            user.repos = repos;
            console.log(user);
      )};
)};
```

U ovoj situaciji dostupni su i user i repos objekti odjednom.

31.Koristeći lodash biblioteki sa metodom sortBy sortirati repozitorije po broju forkova.

Metoda loadash izgleda ovako

```
_.sortBy(collection, [iteratees=[_.identity]])
```

source npm package

Creates an array of elements, sorted in ascending order by the results of running each element in a collection iteratee. This method performs a stable sort, that is, it preserves the original sort order of equal elements. Th invoked with one argument: *(value)*.

Since

0.1.0

Arguments

`collection` *(Array|Object)*: The collection to iterate over.
`[iteratees=[_.identity]]` *(...(Function|Function[]))*: The iteratees to sort by.

Returns

(Array): Returns the new sorted array.

Pogledati više na linku https://lodash.com/docs/4.17.11#sortBy

Kod za sortiranje izgleda ovako

```
import {GithubApiService} from './GithubApiService';
import * as _ from 'lodash';
import { User } from './User';
import { Repo } from './Repo';
svc.getUserInfo('typescript', (user:User) =>{
      svc.getRepos('typescript', (repos:Repos[]) =>{
            let sortedRepos = _.sortBy(repos,[(repo:Repo) => repo.forkCount]);
             user.repos = repos;
             console.log(user);
      )};
)};
```

Metod sortBy ima prvi argument kolekciju , a drugi argument je niz objekta Repo koji se sortiraju u rastućem obliku, ako hoćemo da ih prikažemo u opadajućem koristimo samo množenje sa -1

```
let sortedRepos = _.sortBy(repos,[(repo:Repo) => repo.forkCount * -1]);
```

32. Koristeći lodash biblioteku ograničiti prikaz repozitorija korisnika na 5

Za ove potrebe koristićemo metod take

Pogledati dokumentaciju : https://lodash.com/docs/4.17.11#take

Prvi argument je kolekcija iz koje se izvlači broj elemenata, a drugi argument je koliko se iz kolekcije izvlači elemenata

```
svc.getUserInfo('typescript', (user:User) =>{
      svc.getRepos('typescript', (repos:Repos[]) =>{
            let sortedRepos = _.sortBy(repos,[(repo:Repo) => repo.forkCount]);
            let filteredRepos = _.take(sortedRepos, 5)
            user.repos = filteredRepos;
             console.log(user);
      )};
)};
```

33. Izmeniti program da se nazivi korisnika GitHub prosleđuju preko komandne linije

Za ove potrebe NODE.js koristi process metod koji ima parametar argv koji kao treći parametar prima unos sa konzole tj. treću poziciju npr.

```
npm start typescript
```

Ovde je typescript treći argument

Dakle trebamo da uradimo proveru prilikom pokretanja programa da li postoji treći argument I njega ćemo da koristimo kao ulazni parametar u našoj aplikaciji I zameniti sa nazivom korisnika

```
import { GithubApiService } from './GithubApiService';
import * as _ from 'lodash';
import { User } from "./User";
import { Repo } from "./Repo";

let svc = new GithubApiService();
if (process.argv.length < 3) {
    console.log('Please pass the username as an argument');
}
else {
    let username = process.argv[2];
    svc.getUserInfo(username, (user: User) => {
        svc.getRepos(username, (repos: Repo[]) => {
            let sortedRepos = _.sortBy(repos, [(repo: Repo) => repo.forkCount * -1]);
            let filteredRepos = _.take(sortedRepos, 5);
            user.repos = filteredRepos;
            console.log(user);
        });

    });
}
```

34. Testirajmo program sa sledećom komandom

```
npm start typescript
```