ESLint - alat za statičku analizu koda u JavaScriptu

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Jovana Pejkić, 1089/2020
jovanadpejkic@gmail.com

23. decembar 2020

Sažetak

Praćenje i ispravljanje grešaka u softverskom kodu postaje sve teže zbog sve veće složenosti softvera koji se razvija. Tehnika, koja se pokazala najefikasnijom u održavanju složenog koda, jeste statička verifikacija. U ovom radu ta tehnika je obrađena upotrebom alata ESLint u jeziku JavaScript, osiguravajući punu funkcionalnost i optimalne performanse.

Sadržaj

1 Uvod

Važan deo u procesu razvoja softvera je održavanje koda čistim (eng. *clean code*) i bez grešaka. Upravo zbog toga je **statičko testiranje**[1] bitan deo ovog procesa. Glavni cilj statičkog tipa testiranja je da se defekti otkriju u ranoj fazi razvoja, kada je potrebno uložiti manje napora da bi se oni ispravili. Statičko testiranje koristi skup tehnika koje omogućavaju poboljšanje kvaliteta, efikasnosti i produktivnosti samog procesa razvoja softvera, a može se vršiti bilo ručno, bilo putem alata. Zbog svojih benefita, alati za statičku analizu se redovno koriste u razvoju softvera. Postoji mnogo različitih alata koji koriste različite funkcionalnosti, različite pristupe za statičku analizu, i mogu se koristiti za različite programske jezike. Neki alati se fokusiraju na stil kodiranja (eng. *coding styles*)[2], dok drugi pokušavaju da otkriju greške u kodu[3] [5, 1, 9].

Jedan tip alata za statičku analizu je **linter**. Njegov značaj se ogleda u tome što proverava stil (eng. *style-checking*) pisanja koda i daje sugestije kako ga unaprediti. Proveravanjem stila, mogu se pronaći i bagovi (eng. *bugs*), kao što su nepravilno otkucani nazivi promenljivih ili funkcija. Linter, kao i ostali alati za statičku verifikaciju, prolazi kroz kôd pre prevođenja, tako da se potencijalne greške u njemu mogu ispraviti još u ovoj fazi. Primeri nekih lintera su: *Pylint* i *Mypy* - za programski jezik Python, *ESLint* i *JSHint* - za programski jezik JavaScript i *CSSLint* - za stilski jezik CSS [12, 15, 2, 13, 8, 10].

Ovaj rad se bavi statičkom analizom JavaScript koda kroz upotrebu alata ESLint, jednog od lintera za kôd pisan u ovom jeziku. To je alat koji pomaže u otkrivanju neispravne sintakse i neadekvatne strukture koda. ESLint je potpuno fleksibilan linter koji radi tako što nad kodom primenjuje pravila, koja sam programer određuje. Iako je fleksibilnost pozitivna odlika ovog alata, tu se može uočiti i jedan nedostatak. Naime, ESLint nekada navodi programera na to da ispravi delove koda koji su već ispravni (odnosno korektni). Ovo se dešava zato što alat jednostavno prati datu konfiguraciju. ESLint ne može da pomogne programeru u odabiru pravila koja treba da konfiguriše (za određeni projekat), niti kako da ih konfiguriše. Samim tim, on data pravila i ne proverava. Uloga ESLinta je samo da pomogne u otkrivanju problematičnih delova koda, i to onih koje mu programer zada da treba da posmatra. Zato je na programeru da pravila adekvatno i smisleno konfiguriše [15].

Rad je podeljen u osam sekcija. Prva (sekcija 1) sadrži uvodni deo. U sekciji 2 je opisan značaj ESLinta, kao i njegova primena. Zatim se, u sekciji 3, govori o nastanku ESLinta. U sekciji 4 je opisana arhitektura ovog alata, kao i to kako ovaj alat funkcioniše. U nastavku, u sekciji 5 je demonstrirana instalacija i konfiguracija alata ESLint. Nakon toga, u sekciji 6 je opisano kako se pravila kreiraju. U sekciji 7 je prikazano kako programeri koriste ovaj alat i stilski vodič *Airbnb*, i sa kojim izazovima se tom prilikom susrećeu. Na kraju, u sekciji 8, sledi zaključak.

[1] Statičko testiranje je vrsta testiranja softvera zasnovana na odsustvu izvršavanja celog, ili delova sistema [1].

[2] Stil kodiranja se odnosi na broj belina, nedostatak zagrada, završavanje linija karakterom ";" i drugo, što ne dovodi do bagova u kodu, ali ga čini manje čitljivim i slabo održivim (eng. *maintainable*) [1].

[3] Ove greške nije lako naći i mogu dovesti do ozbiljnih bagova. U ove greške se ubrajaju: pokušaj korišćenja neinicijalizovanih promenljivih, pokušaj korišćenja nedefinisanih funkcija, loša upotreba operatora i slično [1].

2 Značaj i primena

Linter je tip alata koji je posebno značajan za dinamičke jezike koji nisu strogo tipizirani i koji se ne kompajliraju, jer su kao takvi posebno podložni greškama. Kako je JavaScript, jezik kome odgovaraju prethodno navedene karakteristike, postao jezik široke upotrebe, još je važnije imati podršku alata koji pomaže programerima u održavanju koda pisanog u ovom jeziku [3].

ESLint, kao specijalan tip lintera, je alat koji se primenjuje za detektovanje i izveštavanje o šablonima pronađenim u ECMAScript/JavaScript kodu, sa ciljem da učini kôd konzistentnim (odnosno doslednim) kako bi se izbegle potencijalne greške. Ovaj alat je otvorenog koda, a napisan je uz pomoć Node.js-a kako bi mogao da pruži okruženje za brzo izvršavanje i laku instalaciju. Iako je ESLint pomoćni JavaScript alat, postoje i linteri za druge programske jezike (neki od njih su nabrojani u sekciji 1), a neretko i kompajleri objedinjuju procese lintovanja i kompilacije [15].

Ovaj linter omogućava konfigurisanje osnovnih pravila (eng. *base rules*), grupisanih u osam kategorija, a to su:

- *Possible Errors* - Pravila vezana za sintaksne ili logičke greške u kodu.
- *Best Practices* - Pravila koja podstiču da se nešto uradi na bolji način, što pomaže da se izbegnu greške.
- *Strict Mode* - Pravilo koje (ne) dopušta direktive strict mode-a (eng. *strict mode*).
- *Variables* - Pravila vezana za deklaraciju promenljivih.
- *Stylistic Issues* - Pravila vezana za stilske smernice (eng. *style guidelines*).
- *ECMAScript 6* - Pravila vezana za *ES6*, takođe poznata i kao *ES2015*.
- *Deprecated* - Zastarela pravila koja su zamenjena novim pravilima.
- *Removed* - Pravila iz ranijih verzija ESLinta, takođe zamenjena novim pravilima [15].

Umesto uključivanja željenih prvila, programeri mogu da koriste unapred podešene postavke (eng. *presets).* Postavke, koje se još nazivaju i **stilski vodiči** (eng. *style guides*)[4], su javno dostupni skupovi pravila. Primeri ovih vodiča su: *Airbnb*, *Standard*, pa čak i podrazumevana podešavanja ESLinta [15].

3 Istorija

Pojam koji se vezuje za lintere je **lintovanje** (eng. *code linting*) - tip statičke analize koja se koristi za pronalaženje problematičnih delova koda ili koda koji se ne pridržava određenog stila. Svrha lintovanja nije pronalaženje grešaka, već pronalaženje potencijalnih grešaka. Cilj ovog procesa je identifikovanje delova koda koji mogu biti podložni greškama, ili konstrukcija koje mogu dovesti do grešaka u budućnosti [3, 2, 14].

Proces lintovanja i alati koji tome služe, primenjuju se već dugi niz godina unazad. Još je Stiv Džonson (*Steve Johnson*), naučnik koji je radio u Belovim laboratorijama (eng. *Bell Labs*), razvio alat Lint, u isto vreme kada i portabilni C kompilator (1970-ih godina). Svrha ovog alata je bila

[4] Stilski vodiči su konvencije koje treba poštovati prilikom pisanja koda [14].

da pronalazi problematične delove C koda. Kasnije su usledile naprednije verzije Linta, koje su se koristile kod većine C i C++ kompilatora [4].

Originalni JavaScript linter bio je JSLint kojeg je kreirao Daglas Krokford (*Douglas Crockford*). Nešto kasnije, Anton Kovaljev (*Anton Kovalyov*) je uvideo da je JSLint suviše "tvrdoglav" (eng. *opinionated*), u tom smislu da nije bilo moguće prilagoditi ga sopstvenim potrebama, jer nije bilo nikakvih opcija za podešavanje. Upravo zbog toga, Kovaljev je želeo da napravi fleksibilniji JavaScript linter, koji je moguće lako prilagoditi okruženju u kome se koristi (odnosno u kome se kôd izvršava). Zahvaljujući njemu, 2011. godine je nastao novi linter - JSHint. Par godina kasnije, 2013. godine, nastao je ESLint kreiran od strane Nikolasa Zakasa (*Nicholas Zakas*). ESLint je postao, i do danas je, najzastupljeniji izbor lintera. Iako je ESLint po mnogo čemu sličan JSLintu i JSHintu, značajne razlike među njima, a ujedno i prednosti ESLinta, su:

- ESLint je najfleksibilniji linter.
- Funkcionalnost ESLinta se može proširiti raznim pluginovima.
- ESLint koristi ***Espree***[5] za parsiranje JavaScripta.
- ESLint koristi **AST**[6] da dođe do (eng. *evaluate*) šablona u kodu.
- U ESLintu je svako pravilo poseban dodatak, i moguće je dodavati, odnosno priključivati (još) pravila tokom izvršavanja [3].

4 Funkcionisanje i arhitektura

ESLint funkcioniše tako što parsira dati kôd i kreira AST. Nakon toga kreće sa obilaskom tog stabla i primenom pravila. Ovaj proces je opisan u podsekciji 4.1. Sama arhitektura ESLinta, odnosno opisi komponenti od kojih se ovaj alat sastoji, dati su u podsekciji 4.2.

4.1 Obilazak stabla

Nakon instalacije i konfiguracije, ESLint se može pokrenuti nad izvornim fajlovima u komandnoj liniji, kao što je to ranije opisano, ili unutar nekog IDEa (eng. *Integrated development environment, IDE*). Parametre koje ESLint uzima su izvorni fajl, koji treba da lintuje, i konfiguraciju, u kojoj je određeno koja pravila treba da koristi. Najpre parser, kojeg ESLint koristi, parsira izvorni fajl kako bi od njega dobio AST. Svaki čvor u rezultujućem ASTu, između ostalog, sadrži tip sintaksnog elementa (eng. *syntactic element*) kojeg predstavlja (na primer *"VariableDeclaration"*, *"Identifier"* ili *"FunctionDeclaration"*) i informacije o lokaciji sintaksnog elementa u izvornom fajlu. Zatim ESLint obilazi AST, počevši od vrha ka dnu, i izvršava, odnosno primenjuje pravila. Svaki čvor u ESLintu je predstavljen kao objekat koji sadrži svoje stanje i eksportuje metode koje ESLint poziva za vreme obilaženja ASTa. Tokom ovog obilaska, ESLint emituje događaje (poziva metode) čija su imena (eng. *event name*) jednaka tipu obiđenih čvorova (kao što su *"Identifier"* ili *"FunctionExpression"*). Na primer, moguće je pretraživati samo određene tipove čvorova i na njih reagovati. Sama pravila nisu ništa drugo, do

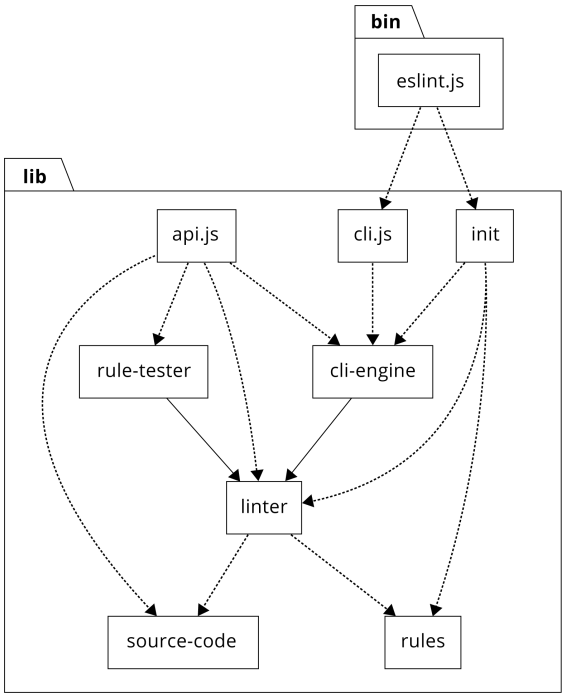
[5] *Espree* je parser kojeg ESLint podrazumevano koristi [15].

[6] Stablo apstraktne sintakse ili apstraktno sintaksno stablo (eng. *Abstract syntax tree, AST*) je način za reprezentaciju izvornog koda. Sintaksički analizator, odnosno parser (deo programskog prevodioca) koji se bavi sintaksom programa kao rezultat proizvodi AST, koji onda isporučuje daljim fazama u procesu kompilacije [6].

set instrukcija koje se izvršavaju nad ASTom, odnosno nad određenim čvorovima. Primarni cilj upotrebe pravila je da prijave upozorenja onda kada "pronađu" problematične delove koda, na osnovu čega ESLint kreira objekat pravila (eng. *the rule object*) [15].

4.2 Arhitektura

U ovoj podsekciji predstavljene su najvažnije komponente koje čine arhitekturu ESLinta. Komponente su najpre prikazane na slici 1, a onda je i svaka od njih opisana u tabeli 1.



Slika 1: Arhitektura ESLinta

bin/eslint.js	Ovo je omotač (eng. *wrapper)* koji ne radi ništa više od pokretanja ESLinta, prosleđujući argumente komandne linije objektu *cli* [15].
lib/api.js	Ovaj fajl predstavlja objekat koji sadrži javne klase (eng. *public classes*): *Linter*, *CLIEngine*, *RuleTester*, i *SourceCode* [15].
lib/cli.js	Ovo je srž ESLinta. *Cli* objekat predstavlja *API* interfejsa komandne linije. Glavni metod koji se koristi je *cli.execute()*, koji prihvata niz stringova koji predstavlja opcije komandne linije. *Cli* objekat vrši sva čitanja iz fajlova, prolaz kroz direktorijume, ulaz i izlaz [15].
lib/init/	Ovaj modul omogućava krajnjim korisnicima podešavanje konfiguracionog fajla uz pomoć flega (eng. *flag*) $--init$ [15].
lib/cli-engine/	Ovo je modul koji je ustvari klasa *CLIEngine*, koja pronalazi i čita konfiguraciju (odnosno, konfiguracione fajlove) i izvorne fajlove i onda zajedno sa klasom *Linter* vrši verifikaciju koda. Glavni metod klase *CLIEngine* je metod *executeOnFiles()*, koji prihvata niz imena fajlova i direktorijuma nad kojima pokreće linter [15].

lib/linter/	Ovaj modul je osnovna (eng. *core*) *Linter* klasa koja vrši verifikaciju koda u skladu sa opcijama u konfiguraciji. Glavni metod klase *Linter* je metod *verify()*, koji prima dva argumenta: izvorni kôd, kojeg treba verifikovati, i konfiguracioni objekat. Metod prvo parsira dati kôd uz pomoć podrazumevanog parsera *Espree* (ili nekog drugog definisanog parsera) i vraća AST. Takođe, ovaj modul ne vrši nikakav ulaz/izlaz (eng. *I/O*) niti vrši interakciju sa konzolom [15].
lib/rule-tester/	Ovaj modul je *RuleTester* klasa koja je omotač oko *Mocha* okvira za testiranje (eng. *testing framework*), te obezbeđuje da pravila mogu biti propuštena kroz jedinične (eng. *unit*) testove. Ova klasa dopušta pisanje konzistentno formatiranih testova za svako pravilo koje je implementirano, čime pruža sigursnost da će svako od tih pravila raditi. Interfejs klase *RuleTester* radi sa globalnim metodama za testiranje *Mocha* okvira, ali se može modifikovati tako da radi i sa drugim okvirima za testiranje (eng. *testing frameworks*) [15].
lib/source-code/	Ovaj modul je *SourceCode* klasa čija je uloga da prikaže parsiran izvorni kôd. Ona prima izvorni kôd i čvor ASTa "Program" (eng. *Program node*) koji predstavlja kôd [15].
lib/rules/	Sadrži ugrađena pravila koja verifikuju izvorni kôd [15].

Tabela 1: Opis arhitekture

5 Instalacija i konfiguracija

U ovoj sekciji biće demonstrirano kako instalirati i konfigurisati alat ESLint. Najpre će, u podsekciji 5.1 biti opisana instalacija ovog alata iz terminala, na *Linux* operativnom sistemu. Zatim će, u podsekciji 5.2 biti opisan proces inicijalne konfiguracije, i na kraju pokretanje ESLinta, kao i upotreba opcije $--fix$, u podsekciji 5.3.

5.1 Instalacija

Pre same instalacije, potrebno je obezbediti Node.js (verzija 10.12.0 ili verzije počev od 12.0.0) sa *SSL* podrškom. ESLint može biti instaliran lokalno, kao u listingu 1 ili globalno, kao u listingu 2. Razlika je u tome što se lokalnom instalacijom ESLint instalira u tekućem folderu (folderu u kome je programer pozicioniran), i može se koristiti samo u okviru tog projekta (unutar koga je instaliran), dok se globalnom instalacijom ESLint instalira samo jednom, a onda se može koristiti bilo gde. Međutim, ako se ESLint instalira globalno i ažurira često, zbog nastalih promena (eng. *breaking changes*) možda neće lintovati dobro kod starih projekata. Tada je potrebno uraditi dodatni posao, odnosno ažurirati projekat. U oba slučaja, instalacija se može izvršiti preko komandne linije koristeći **npm** ili **yarn**. U listingu 1 je prikazana instalacija uz pomoć oba [15].

```
$ npm install eslint --save-dev
# ili
$ yarn add eslint --dev
```

Listing 1: Lokalna instalacija

```
npm install eslint --global
```

Listing 2: Globalna instalacija

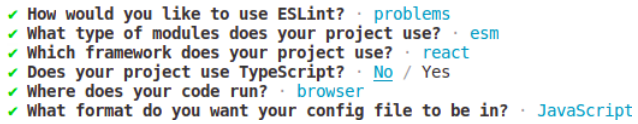
5.2 Konfiguracija

Korak nakon instalacije obuhvata podešavanje inicijalne konfiguracije (konfiguracionog fajla). Jedan način da se to uradi je uz pomoć flega $--$***init***[7], kao što je to prikazano u listingu 3 [15].

```
$ npx eslint --init
# ili
$ yarn run eslint --init
```

Listing 3: Podešavanje konfiguracije

Ukoliko su sve prethodne komande uspešno izvršene, sledi serija pitanja na osnovu čijih odgovora se pravi odgovarajuća konfiguracija, odnosno fajl **.eslintrc.{js, yml, json}**. Primer pitanja i odgovora dat je na slici 2 [15].



Slika 2: Serija pitanja pri podešavanju konfiguracije

Nakon davanja odgovora na ova pitanja, ukoliko je sve prošlo u redu, dobija se poruka o uspešnosti kreiranja konfiguracionog fajla. Sadržaj samog fajla je moguće izmeniti i to je prikazano u podsekciji 6.2 [15].

5.3 Pokretanje

Konačno, pokretanje ESLinta vrši se komandom prikazanom u listingu 4, gde se pored komande *eslint* zadaje putanja do fajla nad kojim treba pokrenuti linter. Nakon ovoga, izlistavaju se sve greške i upozorenja pronađena u datom fajlu, ukoliko postoje. Neke od njih moguće je otkloniti upotrebom opcije $--$***fix***[8], kao što je to prikazano u listingu 5. Na ovaj način, rešavaju se samo određene greške, i to u zavisnosti od konfiguracije. To mogu biti: ukljanjanje viška belina, dodavanje ";" na mestima na kojima nedostaje, zamena dvostrukih navodnika jednostrukim i drugo. Spisak svih pravila, čije probleme opcija $--$*fix* automatski rešava, nalazi se u zvaničnoj dokumentaciji [15].

```
$ npx eslint imeFajla.js
# ili
$ yarn run eslint imeFajla.js
```

Listing 4: Pokretanje ESLinta

```
$ eslint --fix imeFajla.js
```

Listing 5: Upotreba opcije $--$*fix*

[7] Fleg $--$*init* podrazumeva da postoji fajl *package.json*. Ako ne postoji, u terminalu prethodno treba pokrenuti komandu *npm init* ili *yarn init* [15].

[8] Probleme koje opcija $--$*fix* rešava odnose se na pravila iz dokumentacije koja su označena ključem ispred svog naziva [15].

6 Zadavanje pravila i konfigurisanje

Sadržaj ove sekcije produbljuje priču o procesu konfiguracije alata ESLint. Naime, u podsekciji 5.2 je kreiran isključivo inicijalni konfiguracioni fajl. Kako bi se on izmenio, potrebno je najpre naučiti kako se zadaju pravila. Ovo je obrađeno u podsekciji 6.1. Zatim su, u podsekcijama 6.2 i 6.3 demonstrirana dva osnovna načina za konfiguraciju, i to:

1. **Konfiguracioni fajlovi** - Podrazumevaju korišćenje JavaScript, YAML ili JSON fajla (odnosno fajla *.eslintrc.{js, yml, json}*, koji je opisan u podsekciji 5.2) za specifikaciju informacija o konfiguraciji za ceo direktorijum i sve njegove poddirektorijume. Ova specifikacija može biti izvedena i unutar polja *"eslintConfig"* u fajlu *package.json*. Šta god da se odabere, ESLint će to automatski naći i pročitati. Takođe, moguće je i odrediti željeni konfiguracioni fajl (koji će ESLint da koristi, odnosno čita) u komandnoj liniji. U nastavku, u podsekciji 6.2 je detaljno opisan kofiguracioni fajl kreiran u poglavlju 5.2. Takođe, na slici 3 je prikazan primer *.eslintrc* fajla sa ekstenzijom *.js*, dok je u sekciji 7, na slici 4 predstavljen primer fajla sa ekstenzijom *.json*.
2. **Konfiguracioni komentari** - Korišćenjem ovih komentara moguće je umetnuti informacije o konfiguraciji unutar samog JavaScript fajla. U sekciji 6.3 je prikazana upotreba ovih komentara [15].

6.1 Pravila

Pravila se mogu dodavati i menjati unutar komentara u nekom JavaScript fajlu ili unutar sekcije *"rules"* u konfiguracionom fajlu. Ukoliko ova sekcija inicijalno ne postoji, potrebno je napraviti je. Ne postoje propisi, niti uputstva, koji skup pravila treba koristiti. Izbor pravila zavisi od konkretnog projekta i samog programera, i od velikog je značaja napraviti dobar izbor, koji najviše odgovara potrebama [3, 15].

Pravilo u ESLintu se, pre svega, sastoji od naziva pravila, kao što su na primer *"semi"* i *"quotes"*, i vrednosti koja predstavlja nivo greške (eng. *error level*) datog pravila. Nivo greške može imati jednu od sledećih vrednosti:

- **"off" ili 0** - Isključuje pravilo.
- **"warn" ili 1** - Uključuje pravilo kao upozorenje (ne utiče na kôd izlaza (eng. *exit code*)).
- **"error" or 2** - Uključuje pravilo kao grešku (kôd izlaza (eng. *exit code*) dobija vrednost 1) [15].

Tri navedena nivoa grešaka omogućavaju programeru preciznu kontrolu nad načinom na koji ESLint primenjuje pravila. Opšti format pravila prikazan je u listinzima 6 i 7. Iako je prikazano zadavanje samo jednog pravila, oba formata omogućavaju dodavanje i više od jednog pravila [15].

```
/* eslint ime_pravila : "nivo_greske" */
```

Listing 6: Format komentara

```
"rules": {
        "ime_pravila" : "nivo_greske"
        }
```

Listing 7: Format pravila unutar konfiguracionog fajla

Nešto što se takođe može zadati upotrebom pravila jeste Parser koji će ESLint koristiti. Ovo je demonstrirano u podsekciji 6.1.1.

6.1.1 Određivanje parsera

ESLint podrazumevano koristi parser *Espree*. Međutim, parser je moguće promeniti, odnosno zadati drugi, u konfiguracionom fajlu. Primer definisanja parsera dat je u listingu 8. U njemu je prikazan deo konfiguracionog fajla, gde je kao vrednost atributa *"parser"* postavljeno *"esprima"*. Nakon ove konfiguracije umesto podrazumevanog parsera *Espree*, ESLint će koristiti parser *Esprima* [15].

```
{
    "parser": "esprima",
    "rules": {
        "semi": "error"
    }
}
```

Listing 8: Definisanje parsera *Esprima*

Nije svejedno koji se parser koristi, već je potrebno da on zadovoljava određene uslove. U nastavku su izlistani parseri koji su kompatibilni sa ESLintom. To su:

- *Esprima* - Parser napisan u jeziku ECMAScript.
- *@babel/eslint-parser* - Omotač oko parsera *Babel* koji ga čini kompatibilnim sa ESLintom.
- *@typescript-eslint/parser* - Parser koji konvertuje TypeScript u *ESTree*-kompatibilnu formu tako da može da se koristi u ESLintu [15].

6.2 Konfiguracioni fajl

Iako se, od samog kreiranja konfiguracionog fajla, u njemu već nalaze informacije, kao što je to prikazano na slici 3, one se u potpunosti mogu izmeniti i prilagoditi specifičnom projektu. Svako pravilo ESLinta se može isključiti i ESLint se može pokrenuti tako da radi samo bazičnu sintaksnu validaciju. Sam konfiguracioni fajl se sastoji iz više objekata[9] sa nekim osobinama i odgovarajućim vrednostima, i to je ono što se zapravo menja, odnosno konfiguriše. Iako ni jedno jedino pravilo na početku nije eksplicitno uključeno, korišćenjem osobine *"extends": "eslint:recommended"* uključuje se skup pravila koji obuhvata sva čekirana pravila iz liste pravila koja se nalazi u zvaničnoj dokumentaciji. Umesto ove podrazumevane, može se koristiti i konfiguracija koju je neko drugi kreirao. ESLint neće lintovati kôd dok se ne zada vrednost za atribut *"extend"* ili se u konfiguraciji, unutar objekta *"rules"*, eksplicitno ne uključe pravila [15].

Postoje nekoliko informacija koje se mogu konfigurisati. To su:

- **Environments** - U kom okruženju će se skripte izvršavati. Svako okruženje omogućava korišćenje određenih globalnih promenljivih u skriptama. Na primer, ukoliko je postavljeno *"browser" : true* (kao na slici 3), onda se ESLintu naglašava da se skripte mogu izvršavati

[9] Objekti su specifični za JavaScript fajl. Prilikom kreiranja ovog konkretnog konfiguracionog fajla, odgovor na pitanje o njegovom formatu bio je JavaScript. Da je bio izabran JSON ili YAML format, konfiguracioni fajlovi bi imali drugačiju strukturu. Svakako, sam sadržaj bi bio identičan [15].

```
module.exports = {
    "env": {
        "browser": true,
        "es2021": true,
        "node": true
    },
    "extends": [
        "eslint:recommended",
        "plugin:react/recommended"
    ],
    "parserOptions": {
        "ecmaFeatures": {
            "jsx": true
        },
        "ecmaVersion": 12,
        "sourceType": "module"
    },
    "plugins": [
        "react"
    ],
    "rules": {
    }
};
```

Slika 3: Konfiguracioni fajl *.eslintrc.js*

u pretraživaču (eng. *browser*), što dopušta skriptama da pristupe globalnim promenljivama koje postoje samo u pretraživaču (eng. *browser-only global variables*), kao što je to promenljiva *window*.

- **Globals** - Dodatne globalne promenljive kojima skripte mogu da pristupaju tokom svog izvršavanja.
- **Rules** - Koja pravila su uključena i na kom nivou greške (eng. *error level*). Više reči o ovome je bilo u podsekciji 6.1 [15].

6.3 Upotreba komentara za konfiguraciju

Za konfiguraciju pravila unutar samog JavaScript fajla (gde se ne radi o konfiguracionom fajlu *.eslintrc.js*) koriste se komentari za konfiguraciju. U primerima u nastavku prikazano je kako se pišu ovakvi komentari [15].

Primer 6.1 *U ovom primeru, u listingu 9 pravilo "eqeqeq" je isključeno, a pravilo "curly" je uključeno kao greška [15].*

```
1000 /* eslint eqeqeq: "off", curly: "error" */
```

Listing 9: Stringovi za uključivanje/isključivanje pravila

Primer 6.2 *Za prethodna pravila moguće je koristiti i numerički ekvivalent. Ovaj primer je isti kao i prethodni, samo što koristi numeričke kodove (eng.* numeric codes*) umesto stringova. Pravilo "eqeqeq" je i dalje isključeno, a pravilo "curly" je postavljeno kao greška (eng.* error*). Ovo je prikazano u listingu 10 [15].*

```
1000 /* eslint eqeqeq: 0, curly: 2 */
```

Listing 10: Numerički kodovi za uključivanje/isključivanje pravila

Primer 6.3 *Ako pravilo ima dodatne opcije, one se mogu označiti korišćenjem niza literala, kao što je to učinjeno u listingu 11. U ovom komentaru se za pravilo "quotes" postavlja opcija "double". Prvi element niza je uvek nivo greške (broj ili string) [15].*

```
1000 /* eslint quotes: ["error", "double"], curly: 2 */
```

Listing 11: Dodatne opcije kao nizovi literala

Primer 6.4 *Komentari za konfiguraciju mogu da uključuju i opise koji objašnjavaju zašto je komentar bitan (odnosno neophodan). Opis se navodi nakon konfiguracije od koje je odvojen sa dva ili više uzastopnih karaktera "-", kao što je to prikazano u listingu 12 [15].*

```
1000 /* eslint eqeqeq: "off", curly: "error" -- Ovde
        sledi opis o tome zbog cega je ova konfiguracija
         neophodna. */
```

Listing 12: Opis konfiguracionog komentara - 1. način

Primer 6.5 *U ovom primeru prikazan je još jedan način pisanja opisa, dat u listingu 13 [15].*

```
1000 /* eslint eqeqeq: "off", curly: "error"
         --------
1002     Ovde sledi opis o tome zbog cega je ova
        konfiguracija neophodna. */
```

Listing 13: Opis konfiguracionog komentara - 2. način

7 Primer upotrebe

Osim što programer može sam da zadaje svoja pravila unutar konfiguracionog fajla, on ima i mogućnost da koristi već postojeće skupove pravila. Preciznije, programer može da koristi stilske vodiče koji sadrže svoja pravila za formatiranje, koja određuju kako kôd na visokom nivou treba da bude napisan i organizovan. Ova pravila mogu da obuhvataju način imenovanja promenljivih i funkcija, vrstu navodnika (na primer, negde se koriste isključivo jednostuki, a negde isključivo dvostruki navodnici), koliko se belina koristi pri uvlačenju teksta, odnosno koda, kada se prelazi u novi red (eng. *line breaks*), kao i druge odredbe koje se tiču stila pisanja koda [7].

U ovom primeru biće demonstrirana upotreba jednog takvog stilskog vodiča, kao i greške koje se tom prilikom javljaju. U zavisnosti od toga koji stilski vodič se koristi, greške se mogu javiti u različitim delovima istog koda. Radi demonstracije ovog primera upotrebe dati su fajlovi *.eslintrc.json*, prikazan na slici 4, i *index.js*, prikazan na slikama 5 i 6. Stilski vodič koji se koristi je *Airbnb* [11].

Na *Linux* operativnim sistemima, da bi *Airbnb* mogao da se koristi, potrebno je instalirati odgovarajuće verzije svih paketa koji se mogu izlistati pomoću naredbe prikazane u listingu 14. Za instalaciju se koristi naredba prikazana u listingu 15 [11].

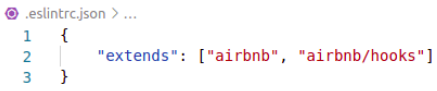
```
$ npm info "eslint-config-airbnb@latest"
    peerDependencies
```

Listing 14: Izlistavanje paketa

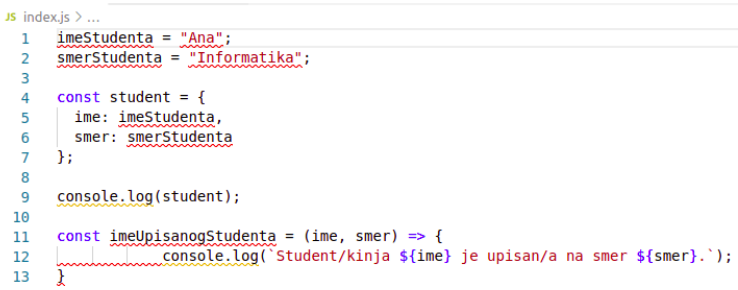
```
$ npx install-peerdeps --dev eslint-config-airbnb
```

Listing 15: Instalacija paketa

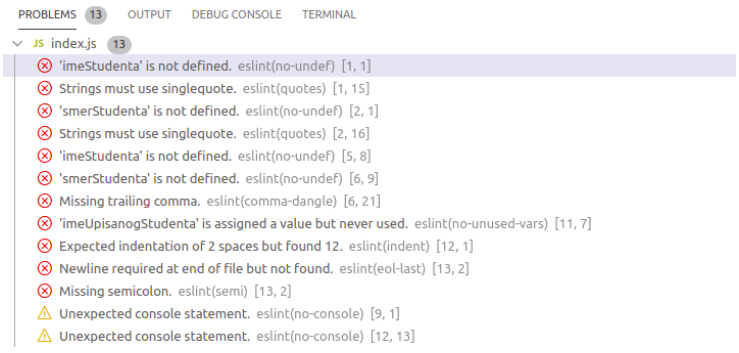
Nakon instalacije, u fajlu *.eslintrc.json*, pomoću atributa *"extends"*, uključuju se sva pravila koja ovaj stilski vodič "propisuje". Ovo se može videti na slici 4 [11].



Slika 4: Konfiguracioni fajl *.eslintrc.json* u kome se koristi stilski vodič *Airbnb*



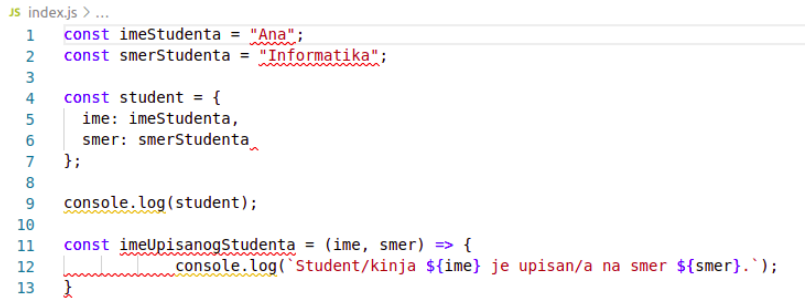
Slika 5: Primer JavaScript fajla *index.js*



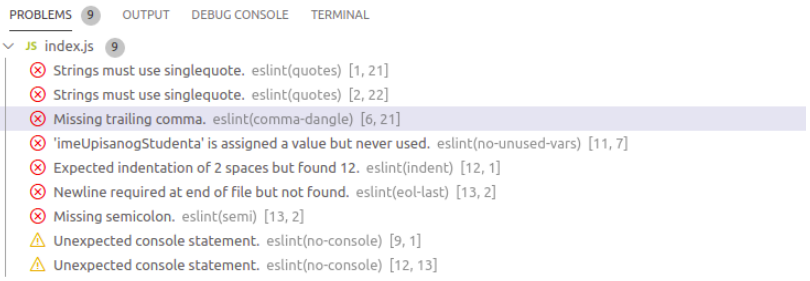
Slika 6: Kartica *"Problems"* fajla *index.js*

Primetno je da fajl *index.js*, koji je prikazan na slici 5, ima dosta grešaka. One su nabrojane u kartici *"Problems"*, prikazanoj na slici 6. Prva greška iz ove liste odnosi se na nedefinisanu promenljivu *"imeStudenta"*.

Međutim, može se uočiti da ni promenljiva *"smerStudenta"* nije definisana, te se obe ove greške rešavaju dodavanjem ključne reči *"const"* ispred naziva ovih promenljivih. Ova promena je prikazana na slici 7, a ishod na slici 8.

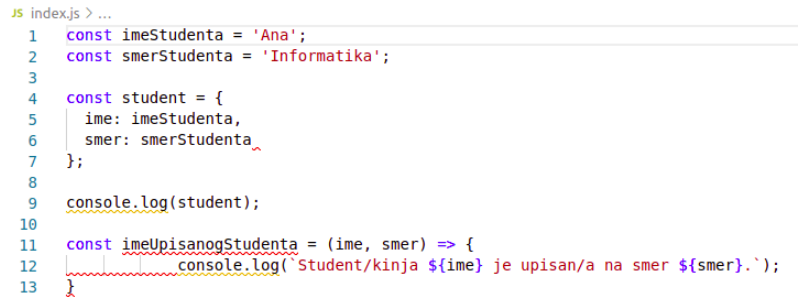


Slika 7: Problem dvostrukih navodnika



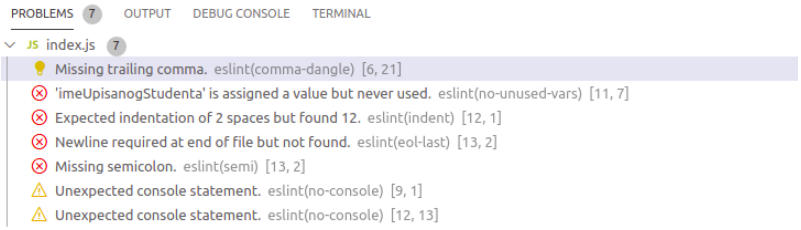
Slika 8: Problem dvostrukih navodnika - kartica "Problems"

Greška, koja je na vrhu liste (slika 8), je nastala zbog toga što stringovi koriste dvostruke navodnike. Prema konfiguraciji koja se koristi, navodnici u kodu treba da budu jednostruki. Rezultat koji se dobija nakon zamene dvostrukih navodnika jednostrukim (slika 9), prikazan je na slici 10.



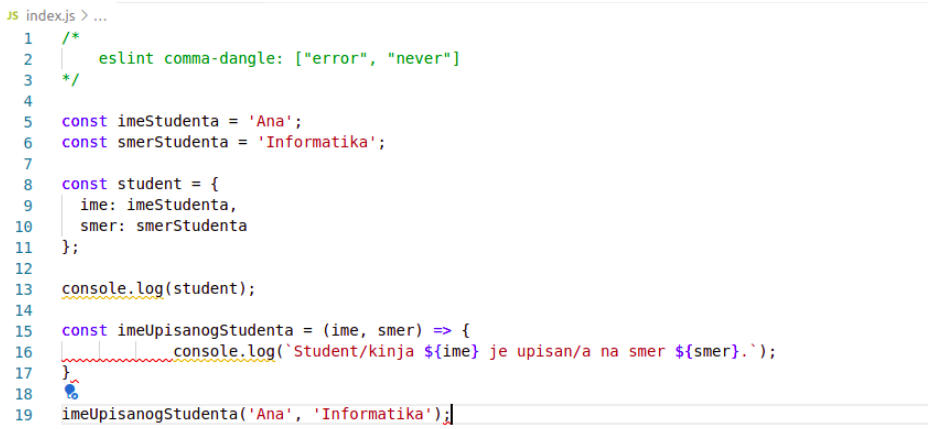
Slika 9: Problem *"Missing trailing comma"*

Pri nekoj konfiguraciji se mogu javiti i greške ili upozorenja koja nisu smislena. Primer jedne takve greške se može videti na slici 10. U pitanju je nedostajuća prateća zapeta (eng. *missing trailing comma*) nakon poslednjeg atributa u objektu *student* (slika 9). Ovakve, ali i druge greške

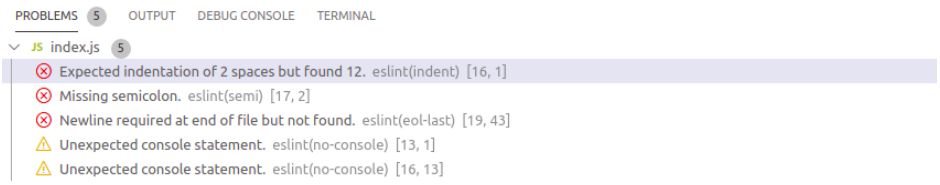


Slika 10: Problem *"Missing trailing comma"* - kartica "Problems"

i upozorenja, programer može da isključi na dva načina o kojima je bilo više reči u sekciji 6. U ovom primeru to je urađeno upotrebom komentara unutar fajla *index.js*. Takođe, funkcija *"imeUpisanogStudenta"* je nakon definicije i pozvana kako bi još jedna greška bila ispravljena. Ove izmene su prikazane na slikama 11 i 12.



Slika 11: Isključivanje pravila *"comma-dangle"*



Slika 12: Isključivanje pravila *"comma-dangle"* - kartica "Problems"

Na redu je uklanjanje viška belina (slika 12). Naime, očekivano je uvlačenje dužine dve beline, a naredba u liniji 16 je uvučena za dvanaest belina. Posle ove ispravke sledi dodavanje karaktera *";"* nakon definicije funkcije *"imeUpisanogStudenta"* i dodavanje novog reda na kraju fajla. Rezultat ovih izmena prikazan je na slici 13.

U ovom trenutku ostala su samo upozorenja zbog neočekivane naredbe *"console"*. Da bi ova naredba ostala deo koda, a upozorenja nestala, po-

Slika 13: Problem *"Unexpected console statement"*

trebno je u komentaru isključiti pravilo *"no-console"*. Nakon toga, dobija se kôd bez ikakvih grešaka i upozorenja, kao što je prikazano na slici 14.

Slika 14: Kartica *"Problems"* bez detektovanih grešaka

8 Zaključak

U ovom radu je prikazana arhitektura i rad alata ESLint, kao i njegova upotreba na konkretnom primeru. Iako je po nekim funkcionalnostima ESLint sličan drugim alatima za statičku verifikaciju, izdvajaju ga fleksibilnost i lako prilagođavanje kompletne konfiguracije specifičnom projektu, brza i laka instalacija i jednostavna integracija. ESLint je alat koji garantuje konzistentnost stila koda u timovima (eng. *developers team*) i pomaže u tome da kôd postane čistiji i čitljiviji.

Literatura

[1] Jean-Louis Boulanger. *Static Analysis of Software*. ISTE Ltd i John Wiley and Sons, Great Britain i the United States, 2012.

[2] Ethan Brown. *Learning JavaScript: JavaScript Essentials for Modern Application Development*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2016.

[3] Ethan Brown. *Web Development with Node and Express: Leveraging the JavaScript Stack*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2019.

[4] Ian F. Darwin. *Checking C Programs with Lint*. O'Reilly and Associates, Inc., 103 Morris Street, Suite A Sebastopol, CA 95472, 1988.

[5] Miodrag Živković. *Testiranje softvera*. Univerzitet Singidunum, Beograd, 2018.

[6] Barry L. Kurtz Kenneth Slonneger. *Formal Syntax and Semantics of Programming Languages*. Addison-Wesley Publishing Company, Inc., United States of America, 1995.

[7] Arie Van Deursen Kristín Fjóla Tómasdóttir, Maurício Aniche. The Adoption of JavaScript Linters in Practice: A Case Study on ESLint. *IEEE Transactions on Software Engineering*, 46:863 – 891, 2019.

[8] Jukka Lehtosalo. Mypy, 2012. on-line at: http://mypy-lang.org/.

[9] James Long. Prettier, 2017. on-line at: https://prettier.io/.

[10] Nicole Sullivan Nicholas C. Zakas. CSSLint, 2011. on-line at: http://csslint.net/.

[11] Inc. npm. Npmjs, 2014. on-line at: https://www.npmjs.com/.

[12] Valentin Kipyatkov Sergey Dmitriev. JetBrains, 2000. on-line at: https://www.jetbrains.com/.

[13] Sylvain Thénault. Pylint, 2001. on-line at: https://www.pylint.org/.

[14] Nicholas C. Zakas. *Maintainable JavaScript: Writing Readable Code*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2012.

[15] Nicholas C. Zakas. ESLint, 2013. on-line at: https://eslint.org/.