

Informatikos fakultetas

**T120B162 Programų sistemų testavimas**

**3 laboratorinio darbo ataskaita**

|  |  |  |
| --- | --- | --- |
|  | Studentas: | Monika Kaltenytė, IFF-5/3 |
| Dėstytojas: | doc. Šarūnas Packevičius |

Kaunas 2018

# Įvadas

3 laboratorinio darbo tikslas – išmokti atlikti tikslingą programinio kodo peržiūrą bei pritaikyti statinės kodo analizės įrankius savo projekte.

Darbo uždaviniai:

1. Peržiūrėti kodą ir rasti potencialias klaidas.
2. Susirasti tinkamus statinės kodo analizės įrankius kuriamam projektui.
3. Pritaikyti statinės kodo analizės įrankius programos kodui ir identifikuoti programos kodo klaidas.
4. Parašyti naują taisyklę, kuri būti pritaikoma statiškai nagrinėjant kodą.
5. Statinės kodo analizės rezultatus pateikti laboratorinio darbo ataskaitoje.

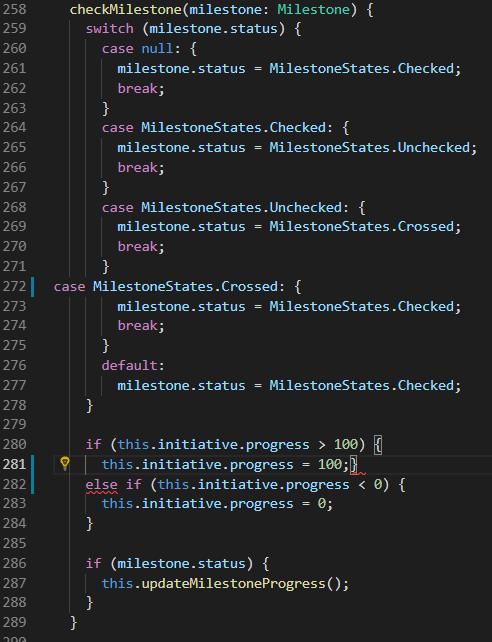
# Programos koda peržiūra

Peržiūrint kodą vadovaujamasi šiomis kodo rašymo taisyklėmis:

1. Architektūra – ar skirtingo kodo tipai skaidomi į skirtingus failus. Pavyzdžiui testuojamoje programoje tikrinima ar visur yra atskiri failai typescript‘ui, css‘ui ir html‘ui.
2. Formatavimas – ar kodas rašomas tvarkingai, tai yra ar kodo eilutės teisingai atitrauktos nuo kairiojo krašto, nepaliekami nereikalingi tarpai ar laikomasi CamelCase taisyklių ir panašiai.
3. Ar tikslingai rašomi komentarai, ar kodas tvarkingai paskirtomas į atskiras funkcijas išvengiant neskaitomo kodo, ar nėra kietai įsiūto kodo, ar naudojamos constantos kai reikšmė nesikeičia, ar išnaudojamos naudojamo karkaso ( šiuo atveju Angular ) galimybės ir panašiai.
4. Saugumas – ar vartotojui nėra pateikiama duomenų, kurių jis neturėtų matyti.

Toliau pateikiami pastebėjimai iš kodo, atsižvelgiant į anksčiau paminėtas taisykles.

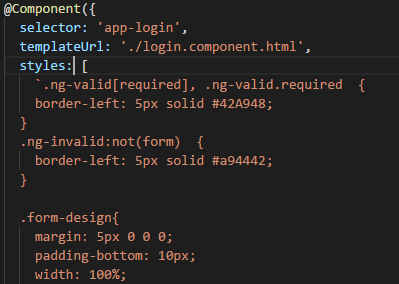
* Programoje pastebėtas neformatuotas kodas typescript faile – [initiative-editor.component.ts](http://home.component.ts) ( pav 1.):



pav. 1 Neteisingas kodo atitraukimas

Matoma, kad 272-282 eilutėse nesilaikoma kodo atitraukimo formatavimo. Sprendimas būtų arba pačiam arba naudojantis naudojamu IDE kodą suformatuoti, tai yra reikiamose vietose ( funkcijų viduje, if‘uose ir pan. ) kodą atitraukti toliau nuo kairės paraštės, skliaustus dėti pagal gerąsias programavimo praktikos taisykles.

* Pastebėtas failų architechtūros pažeidimas – css aprašytas login.component.ts :

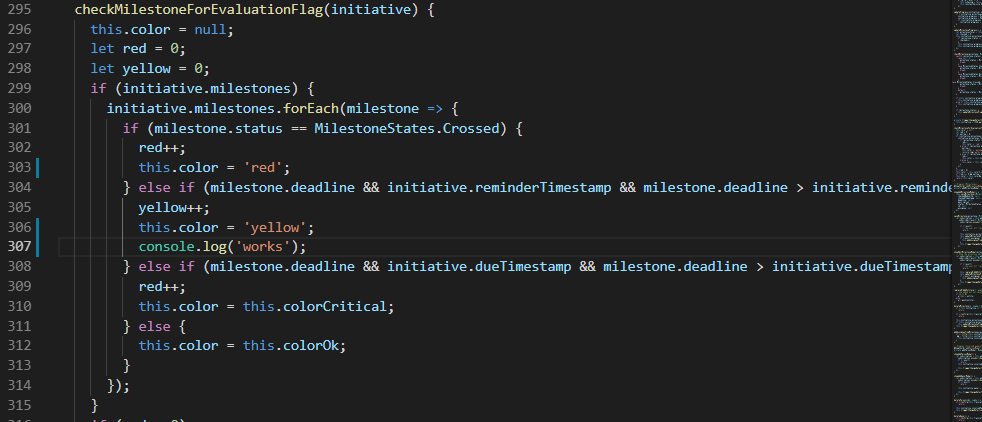


pav. 2 Neteisinga architechtūra

TypeScript tai daryti leidžia, bet rekomenduomaja didesnius stiliaus pakeitimus rašyti į atskyrus failu.

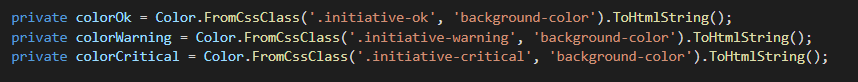
Šį css fragmentą vertėtu iškelti į login.component.css

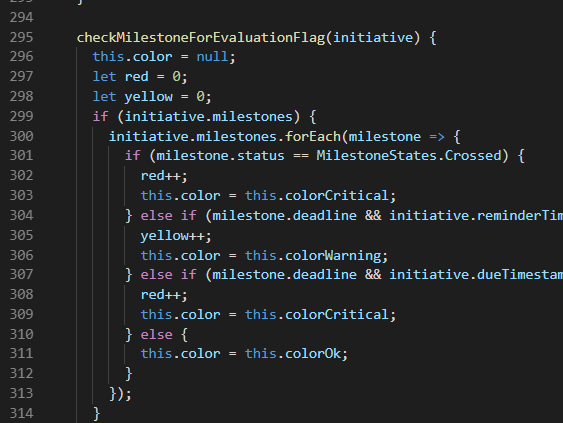
* Čia ( pav 3.) matomi net kelios klaidos :



pav. 3 “Kietai” įrašytas kelias bei nereikiama informacija varotojui

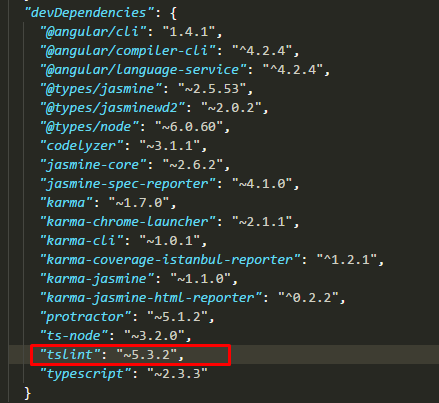
Aukčiau pateiktame paveiklėlyje galima matyti net kelias klaidas. 307 eilutėje matoma, kad į konsolę įrašoma nereikalinga informacija. Reikėtų ją pakeisti informaciniu pranešimu vartotojui. Eilutėse 303, 306 matome ,,kietai’’ įrašytas spalvas. Protingiau yra apsirašyti konstantą, tuomet nereikės ieškoti visų vietų kode, kurias reikia pakeisti, užteks pakeisti vieną konstantą. Apačioje pateikiamas pavyzdys, kaip tai būtų galima padaryti.





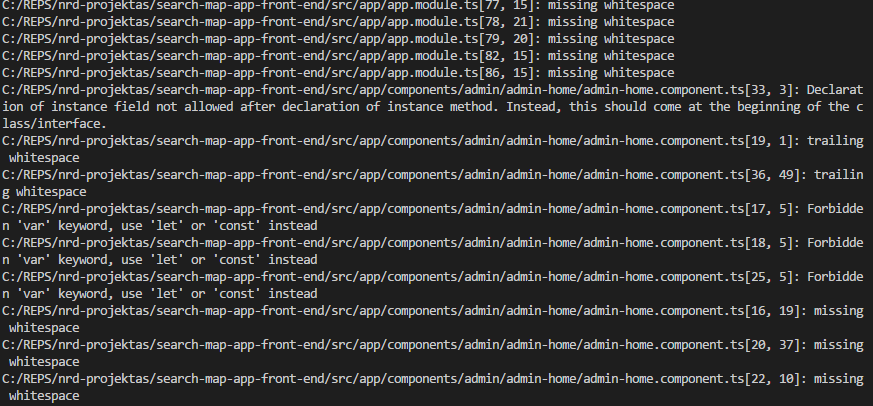
# Programos kodo statinė analizė

Skyriuje aprašoma, koks pasirinktas statinės analizės įrankis, bei pateikiama jo naudojimo pavyzdžių.

Kadangi front-end sitstemos dalis rašoma naudojant TypeScript, todėl naudojamas Tslint statinės analizės įrankis. Angular karkase šį įrankį panaudoti galima įsirašius package pavadinimu tslint.

pav. 4 Lint`nerio pridėjimas į projektą

TsLint‘neris realizuojamas cmd lange paleidus komandą - ng lint



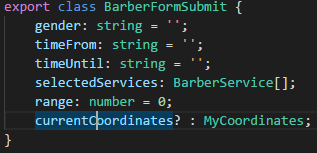
pav. 5 Lint`nerio sugeneruotos klaidos

Atlikos komandą parodomos kode rastos klaidos. Toliau nagrinėjame kiekvieną atvejį atskirai ( tik unikalius ), išanalizuojame bei pataisome.

* String naudoti ne “ bet ‘



Pataisytas kodas :

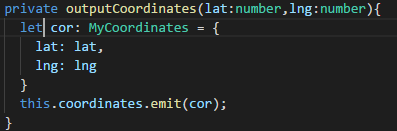


* Nereikia naudoti var - geriau naudoti let





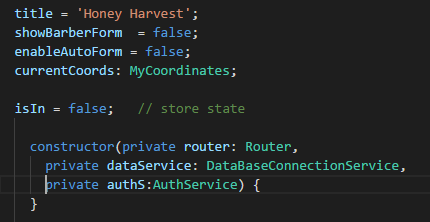
Pataisytas kodas :



* Kintamieji turėtu buti paskelbti prieš construktorių :



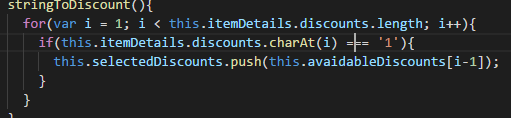
Pataisytas kodas :



* Naudoti === vietoj ==



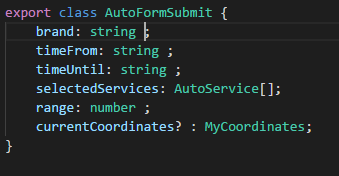
Pataisytas kodas



* Ne visada būtina nurodyti kintamojo tipą :



Pataisytas kodas :



# Naujos taisyklės įtraukimas į statinės analizės įrankį

TsLint naujų taisyklių kūrimas aprašytas: <https://palantir.github.io/tslint/develop/custom-rules/>

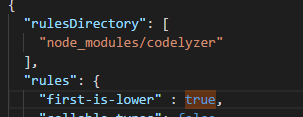
Naudojant šia pamoką, sukurta taisyklė, tikrinanti ar kintamųjų pavadinimai prasideda mažąja raide. Taisyklei sukurti reikia .js failo ( arba .ts ), kuriame ji aprašoma. Šiuo atveju sukuriamas failas firstIsLowerRule.js. Failo pavadinimas būtinai turi būti parašytas camelCase ir failo pavadinimo pabaigoje būtinai turi būti žodis Rule. Parašytos taisyklės turinys atrodo taip:



pav. 6 Taisyklės kodas

Kaip matome TsLint įrankis

Toliau reikia taisyklę įtrauki į projekte esantį failą tslint.json



pav. 7 Taisyklės įtraukimas

Taisyklė atpažįstama pagal sukurtos taisyklės failo pavadinimą. Iš naujo paleidus ng lint komanda cmd lange, matome, kad įrankio rodomų klaidų vel atsirado:



Patikrinus failus matoma, kad kintamieji ten iš tikrųjų prasideda didžiosiomis raidėmis. Pakeitus kintamųjų pirmasias raides į mažasias ir vel panaudojus šia komandą klaidų vel nebėra. Tai reiškia kad sukurtoji nauja taisyklė funkcionuoja teisingai.

# Išvados

1. Laboratorinio darbo metu buvo peržiūrėtas programos kodas, bei surastos potencialios klaidos, taip pat rastos taisytinos kodo vietos, suprasta kodo peržiūros esmė.
2. Surastas tinkamas statinės kodo analizės įrankis projektui, kuris buvo pritaikytas programos kodui. Identifikuotos, o vėliau ir ištaisytos surastos programos kodo klaidos.
3. Parašyta nauja taisyklė, kuri vėliau buvo integruota į statinės kodo analizės įrankį. Kodas pataisytas pagal naujai rastas klaidas.