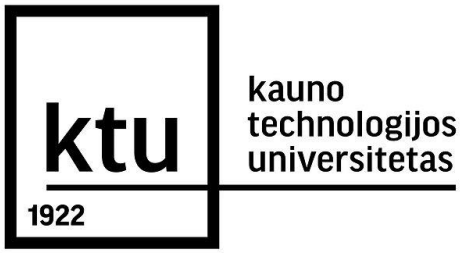


INFORMATIKOS FAKULTETAS

T120B162 Programų sistemų testavimas

3 laboratorinis darbas

Studentas: Mangirdas Kazlauskas, IFF-4/1

Dėstytojas: doc. Šarūnas Packevičius

KAUNAS 2017

Turinys

1. Įvadas

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

Darbo uždaviniai:

1) Pritaikyti statinę projekto kodo analizę be jokio įrankio – peržiūrėti programos 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 ezultatus pateikti laboratorinio darbo ataskaitoje.

2. Programos kodo peržiūros eiga

Pirmiausia yra peržiūrimas programos kodas nenaudojant jokių įrankių. Šis procesas vadinamas kodo peržiūra. Tam, kad kodo peržiūra būtų nuosekli, vadovaujamasi šiais kodo peržiūros punktais:

1) Kodo formatavimas – ar kodas rašomas nuosekliai, ar tinkami kodo eilučių atitraukimai nuo kairiojo krašto, ar yra taikomos teisingos kodo rašymo taisyklės (pvz., naudojamas CamelCase);
2) Kodo architektūra – ar kodas yra išskaidytas į komponentus arba failus (pvz., stiliaus ir scenarijų kalbų failai nėra saugomi viename faile);
3) Kodo gerosios praktikos – ar nėra kietai įsiuto kodo, ar naudojamos konstantos vietoj įkoduotų bereikšmių skaičių (pvz., if (role == 1)), ar rašomi tikslingi komentarai, ar vengiama didelių if/else blokų, ar tinkamai išnaudojamos projekte naudojamų karkasų galimybės ir pan;
4) Kodo saugumas – ar kodas yra apsaugotas nuo SQL injekcijų, cross-side scripting, ar tie programos duomenys/rezultatai, kurie neturėtų būti matomi, nėra išvedami į konsolę/naršyklės langą, ar tikrinamos įvedimo laukų reikšmės (ar yra taikoma laukų validacija).

Taip pat tam, kad būtų išlaikomas nuoseklumas, programos kodo peržiūros metu rastos klaidos bus dokumentuojamos tokiomis lentelėmis (1 lentelė):

1 lentelė. Kodo peržiūros metu rastos klaidos dokumentavimo lentelės šablonas

Failo pavadinimas	
Programos kodo fragmento nuotrauka	
Failo kodo eilutė(s)	
Klaidos kategorija (iš sąrašo)	
Klaidos apibūdinimas	
Galimas sprendimas/komentaras	

3. Programos kodo peržiūra

Atlikus kodo peržiūrą, programos kode buvo rasta keletas klaidų/pastebėjimų. Šios klaidos pateiktos žemiau esančiose 2 – 10 lentelėse:

2 lentelė. Kodo peržiūros klaida Nr. 1

Failo pavadinimas	Order.vue
Programos kodo fragmento nuotrauka	

```
<template>
<v-card class="order-card">
  <v-card-actions :class="[{'error' : orderStatusIsPaid}, {'warning' : orderStatusIsStarted}, {'success' : orderStatusIsFinished}]">
    <v-spacer></v-spacer>
    <timeago class="white--text subheading pr-1" :since="order.orderCreatedAt*1000" :auto-update="60"></timeago>
  </v-card-actions>
  <v-list class="order-items-list" v-show="this.order.orderStatusID == 2 || this.order.orderStatusID == 4" two-line dense>
    <v-divider></v-divider>
    <template v-for="(item, i) in order.orderItems">
```

Failo kodo eilutė(s)	7
Klaidos kategorija (iš sąrašo)	Kodo gerosios praktikos
Klaidos apibūdinimas	Šioje eilutėje tikrinamas užsakymo statusas. Iš kodo nėra aišku, kokį užsakymo statusą atitinka pateiktos reikšmės (2 ir 4).
Galimas sprendimas/komentaras	Reikia sukurti užsakymo statuso reikšmių konstantas ir jas naudoti kode, pvz., vietoj *this.orderStatusID == 2* naudoti *this.orderStatusID === constants.ORDER_PAID*. Taip pat reikėtų naudoti palyginimo operatorių su tipo tikrinimu, t.y., vietoj == naudoti ===.

3 lentelė. Kodo peržiūros klaida Nr. 2

Failo pavadinimas	Order.vue
Programos kodo fragmento nuotrauka	

```
31      </v-card-actions>
32      <v-container fluid grid-list-lg v-show="showCustomerInfo || this.order.orderStatusID == 4" :class="{'success' : this.order.orderStatusID == 4}">
33        <v-layout row wrap>
34          <v-flex xs2>
35            <v-avatar><img :src="order.clientPhotoUrl" /></v-avatar>
36          </v-flex>
37          <v-flex xs10>
38            <div :class="[{'white--text' : this.order.orderStatusID == 4}, 'headline']">{{order.clientFullName}}</div>
```

Failo kodo eilutė(s)	32 ir 38
Klaidos kategorija (iš sąrašo)	Kodo gerosios praktikos
Klaidos apibūdinimas	Šioje eilutėje tikrinamas užsakymo statusas. Iš kodo nėra aišku, kokį užsakymo statusą atitinka reikšmė 4. Taip pat šiose vietose yra naudojamas identiškas kodas *this.order.orderStatusID == 4*,
Galimas sprendimas/komentaras	Reikia sukurti užsakymo statuso reikšmių konstantas ir jas naudoti kode, pvz., vietoj *this.orderStatusID == 4* naudoti *this.orderStatusID === constants.ORDER_PAID*. Taip pat reikėtų naudoti palyginimo operatorių su tipo tikrinimu, t.y., vietoj == naudoti ===. Programos kodo išraišką *this.order.orderStatusID == 4* reikia iškelti į naujai sukurtą funkciją, kuri per savo vardą grąžintų loginį šios išraiškos rezultatą

4 lentelė. Kodo peržiūros klaida Nr. 3

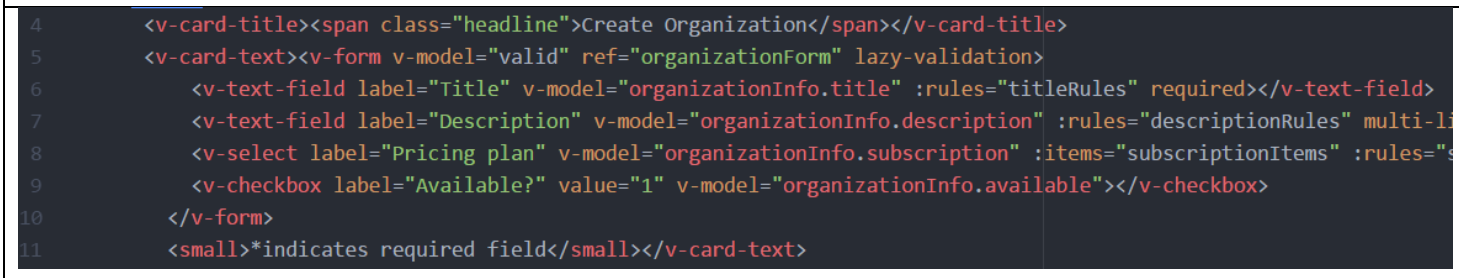
Failo pavadinimas	OrganizationCard.vue

Programos kodo fragmento nuotrauka

```
1   <template>
2   <v-card class="organization-card">
3     <v-card-title class="cyan darken-3" primary-title>
4       <div class="title white--text" v-html="organization.organizationTitle"></div>
5     </v-card-title>
```

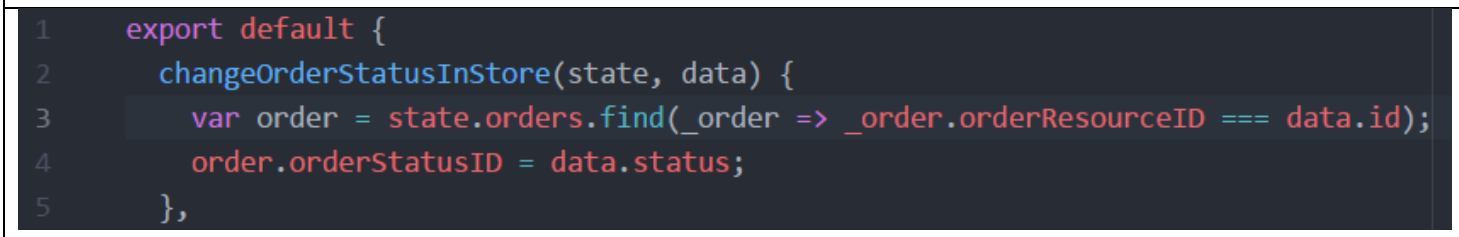
Failo kodo eilutė(s)	4
Klaidos kategorija (iš sąrašo)	Kodo saugumas
Klaidos apibūdinimas	Kintamasis *organization.organizationTitle* į div elementą yra įterpiamas paprastu tekstu (nepakeitus specialiųjų simbolių).
Galimas sprendimas/komentaras	Vietoj *v-html* atributo reikia naudoti {{ }} sintaksę, kuri padeda kintamuosius ar programos kodo gabalus įterpti pirma juos pavertus į eilutės duomenų tipą.

5 lentelė. Kodo peržiūros klaida Nr. 4

Failo pavadinimas	OrganizationDialog.vue
Programos kodo fragmento nuotrauka	
	
Failo kodo eilutė(s)	4-11
Klaidos kategorija (iš sąrašo)	Kodo formatavimas
Klaidos apibūdinimas	Turi būti palaikoma hierarchinė HTML kodo struktūra, t.y., toje pačioje eilutėje negali būti dviejų atidaromųjų/uždaromųjų HTML žymių.
Galimas sprendimas/komentaras	Kiekvienas HTML kodo vidinis elementas turi būti naujoje eilutėje nei tėvinis, atitrauktas toliau nuo kairiojo krašto. Reikia pataisyti, kad būtų pagerintas programos kodo skaitomumas.

```
4        <v-card-title><span class="headline">Create Organization</span></v-card-title>
5        <v-card-text><v-form v-model="valid" ref="organizationForm" lazy-validation>
6            <v-text-field label="Title" v-model="organizationInfo.title" :rules="titleRules" required></v-text-field>
7            <v-text-field label="Description" v-model="organizationInfo.description" :rules="descriptionRules" multi-li
8            <v-select label="Pricing plan" v-model="organizationInfo.subscription" :items="subscriptionItems" :rules="s
9            <v-checkbox label="Available?" value="1" v-model="organizationInfo.available"></v-checkbox>
10         </v-form>
11         <small>*indicates required field</small></v-card-text>
```

6 lentelė. Kodo peržiūros klaida Nr. 5

Failo pavadinimas	mutations.js
Programos kodo fragmento nuotrauka	
	
Failo kodo eilutė(s)	3
Klaidos kategorija (iš sąrašo)	Kodo gerosios praktikos
Klaidos apibūdinimas	Netinkamas kintamojo sukūrimas (naudojamas *var*).
Galimas sprendimas/komentaras	Remiantis ES6 JavaScript standartu, rekomenduojama vietoj *var* naudoti *let* arba *const*. Šiuo atveju, kadangi kintamasis programos kodo fragmente nesikeičia, kintamojo sukūrimui patartina naudoti *const*.

```
1    export default {
2      changeOrderStatusInStore(state, data) {
3        var order = state.orders.find(_order => _order.orderResourceID === data.id);
4        order.orderStatusID = data.status;
5      },
```

7 lentelė. Kodo peržiūros klaida Nr. 6

Failo pavadinimas	actions.js
Programos kodo fragmento nuotrauka	

```
21      getBranchesForUser() {
22        return new Promise((resolve, reject) => {
23          Vue.axios.get('https://localhost:8080/api/branches').then(response => resolve(response)).catch(error => reject(error));
24        });
25      },
26      getOrganizationsWithBranches() {
27        return new Promise((resolve, reject) => {
28          Vue.axios.get('https://localhost:8080/api/organizations').then(response => resolve(response)).catch(error => reject(error));
29        });
30      },
```

Failo kodo eilutė(s)	23 ir 28
Klaidos kategorija (iš sąrašo)	Kodo gerosios praktikos
Klaidos apibūdinimas	Funkcijose naudojama kietai įsiūtas API domenas.
Galimas sprendimas/komentaras	Reikėtų API domeną iškelti kaip konstantą arba naudoti projekto konfigūracijos failą, kuriame būtų nurodyta, į kokį domeną yra kreipiamasi. Tai yra reikalinga tam, kad pereinant iš vienos programavimo aplinkos į kitą (pvz., iš kūrimo į sistemos diegimo aplinką), visose funkcijose reikėtų pakeisti domeno pavadinimą.

8 lentelė. Kodo peržiūros klaida Nr. 7

Failo pavadinimas	store.js
Programos kodo fragmento nuotrauka	

```
12    const store = new Vuex.Store({
13      state: {
14        authUserData: {
15          name: '',
16          token: '',
17          photoURL: '',
18          selectedBranchResourceID: '',
19          roleID: '',
20        },
21        orders: [],
22      },
23      plugins: [
24        createPersistedState({
25          paths: ['authUserData'],
26          getState: key => Cookies.getJSON(key),
27          setState: (key, state) => Cookies.set(key, state, {
28            expires: 1,
29          }),
30        }),
31      ],
32      getters,
33      mutations,
34      actions,
35    });
```

Failo kodo eilutė(s)	13-22
Klaidos kategorija (iš sąrašo)	Kodo architektūra.
Klaidos apibūdinimas	*State* objektas yra per stipriai susietas su *store* objektu.
Galimas sprendimas/komentaras	Reikėtų *state* objektą iškelti į atskirą failą (taip pat, kaip iškelti *getters*, *mutations*, *actions*). Taip būtų atskirta ši kodo dalis į atskirą komponentą. Tai būtų naudinga programos testavimo procese, kai, pvz., prieš testą į *store* objektą būtų tiesiog paduodamas atskiras *state* objekto failas testavimui, nereikėtų perrašinėti kiekvienos reikšmės *store* objekto viduje.

9 lentelė. Kodo peržiūros klaida Nr. 8

Failo pavadinimas	actions.js
Programos kodo fragmento nuotrauka	

```
12      changeOrderStatus(context, data) {
13        Vue.axios.put(`${constants.CURRENT_API_DOMAIN}/api/orders/${data.id}`, {
14          orderStatusID: data.status,
15        }).then(() => {
16          context.commit('changeOrderStatusInStore', data);
17        }).catch((error) => {
18          console.log(error);
19        });
20      },
```

Failo kodo eilutė(s)	18
Klaidos kategorija (iš sąrašo)	Kodo saugumas
Klaidos apibūdinimas	Į naršyklės konsolės langą yra išvedamas klaidos pranešimas.
Galimas sprendimas/komentaras	Reikia naudoti vidinius projekto *log* failus, į kuriuos būtų išvedama informacija apie atsiradusias klaidas.

10 lentelė. Kodo peržiūros klaida Nr. 9

Failo pavadinimas	BranchesView.vue
Programos kodo fragmento nuotrauka	

```
96        closeOrganizationDialog(closedOnSave, organizations, closedOnCreate) {
97          var showSnackbar = true;
98          if (closedOnSave) {
99            this.organizations = organizations;
100           if (closedOnCreate) {
101             this.snackbarText = 'Organization created successfully!';
102           } else {
103             this.snackbarText = 'Organization updated successfully!';
104           }
105           this.snackbar = true;
106         }
107         this.organizationDialogOpen = false;
108       },
109     },
```

Failo kodo eilutė(s)	97
Klaidos kategorija (iš sąrašo)	Kodo gerosios praktikos
Klaidos apibūdinimas	Kintamasis *showSnackbar* kode nėra naudojamas.
Galimas sprendimas/komentaras	Reikėtų kintamąjį panaikinti arba panaudoti vietoj kietai įsiūtos *true* reikšmės 105 eilutėje.

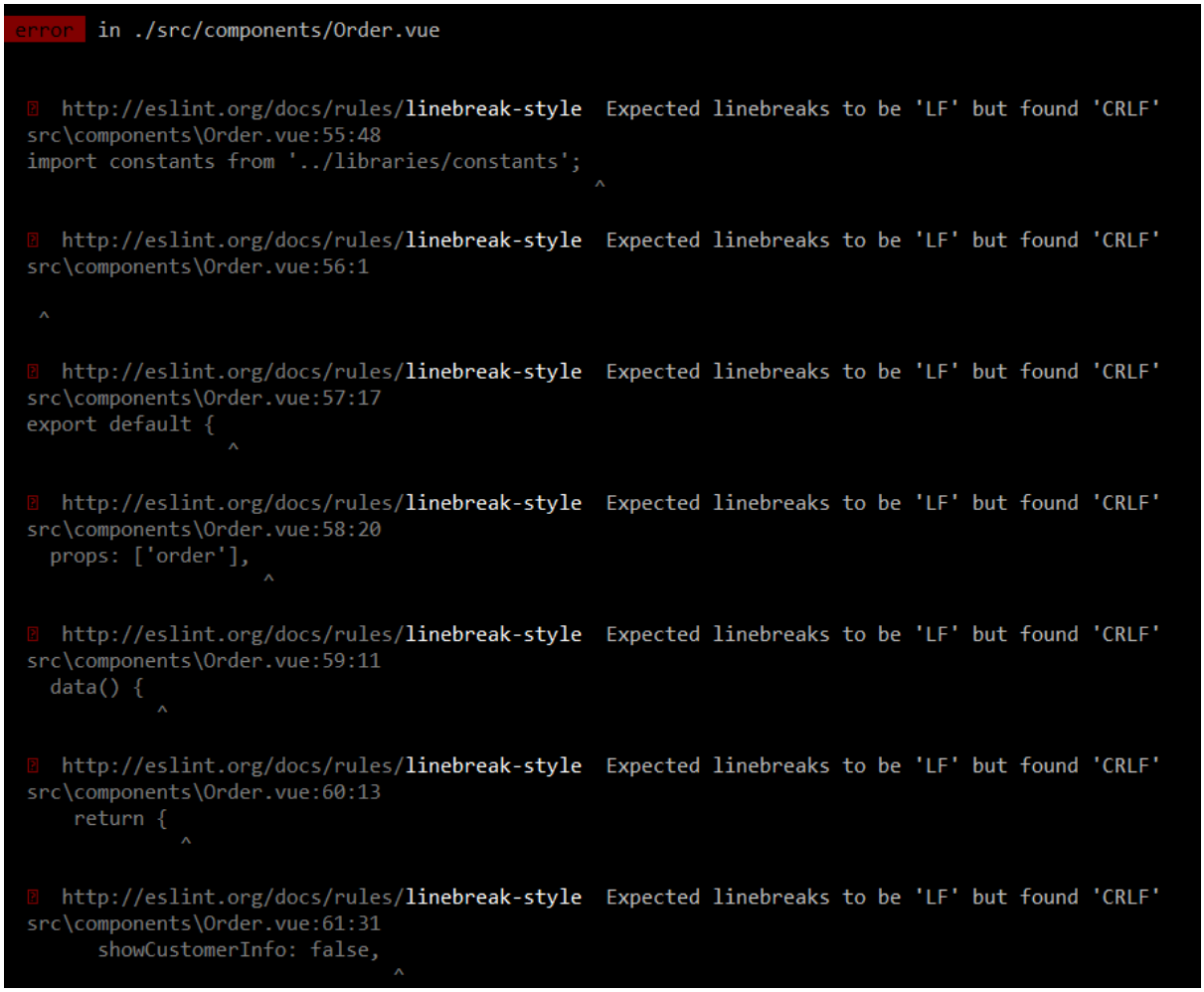
4. Priemonės kodo statinei analizei atlikti

Statinės kodo analizės įrankiu yra testuojama internetinė aplikacija, kuris kodo pagrindą sudaro Vue.js karkasas, todėl statinei kodo analizei pasirinktas tinkamas įrankis – ESLint. Šis įrankis yra skirtas JavaScript scenarijų rašymo kalbos statinei kodo analizei. Taip pat įrankis yra nesunkiai integruojamas į programavimo aplinką, kuriamo projekto kodą, taip pat aiški dokumentacija leidžia nesunkiai identifikuoti klaidų/įspėjimų pranešimus, kurie nurodo, kaip reikia pakeisti programos kodą, kad jis atitiktų reikalavimus bei būtų teisingas.

Taip pat kartu su ESLint pasirinktas ir Airbnb-base taisyklių rinkinio pagrindas.

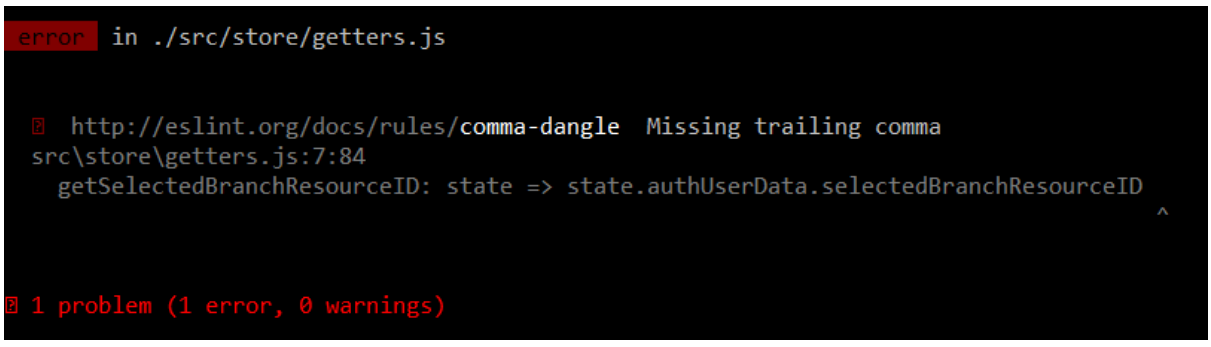
5. Programos kodo statinė analizė

Panaudojus statinės kodo analizės įrankį ESLint, buvo rasta 61 klaida ir 2 įspėjimai. Klaidų pranešimai kartu su eilutėmis, kuriose rasto klaidos, pateikiami konsolės lange. Didžiąją dalį klaidų sudarė tai, kad Order.vue failo eilučių pabaigos buvo CRLF tipo, o ne LF – iš viso rastos 45 šio tipo klaidos (1 pav.).



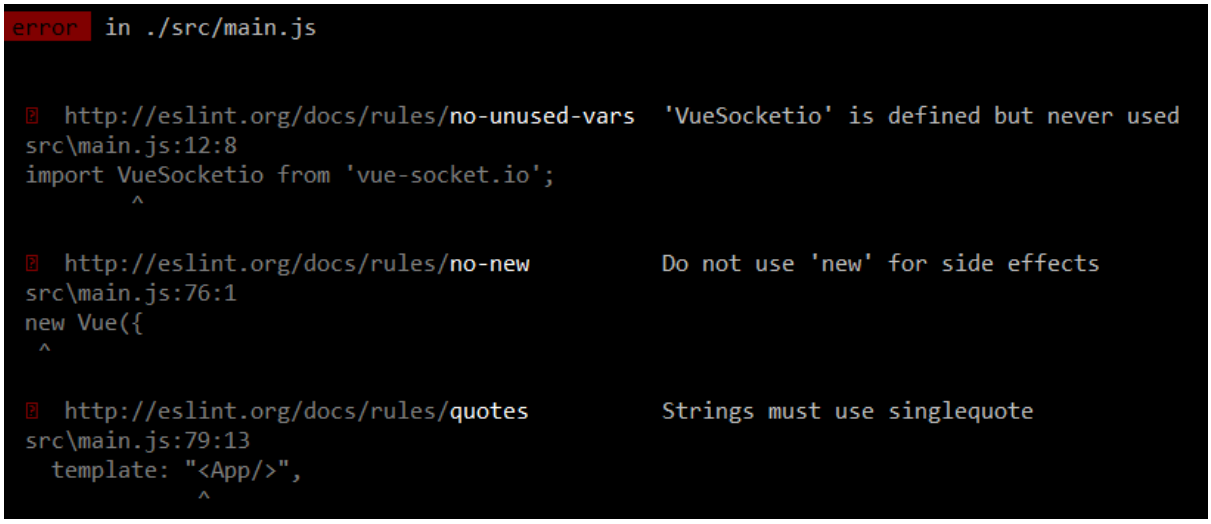
1 pav. Failo Order.vue klados

Faile getters.js rasta trūkstamo kablelio klaida (2 pav.):



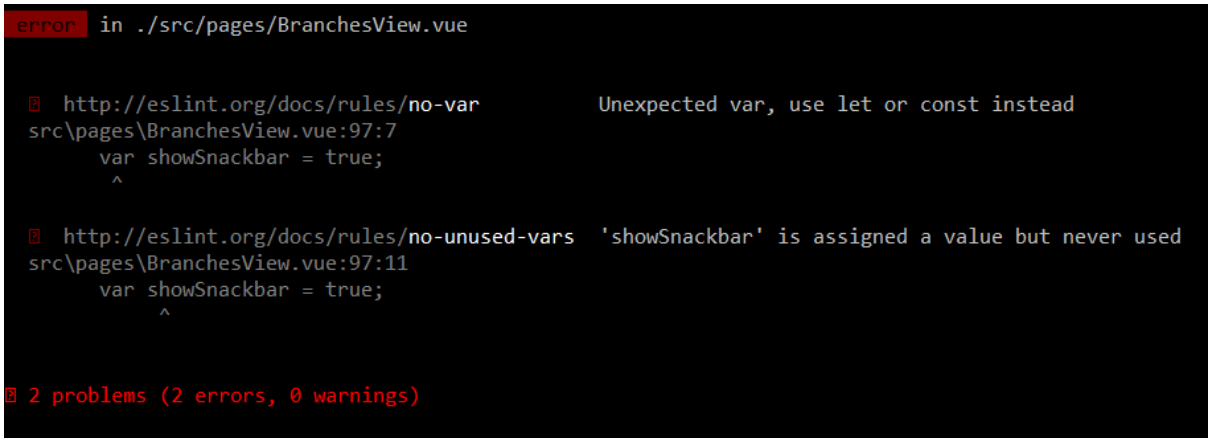
2 pav. Failo getters.js klaidos

Comma-dangle taisyklė nustato tai, kad po parametrų reikšmių objektuose ar elementų masyvuose visada turi būti kablelis, kas supaprastina naujų objektų/parametrų pridėjimą į duomenų objektą ir šalinimą iš jo.



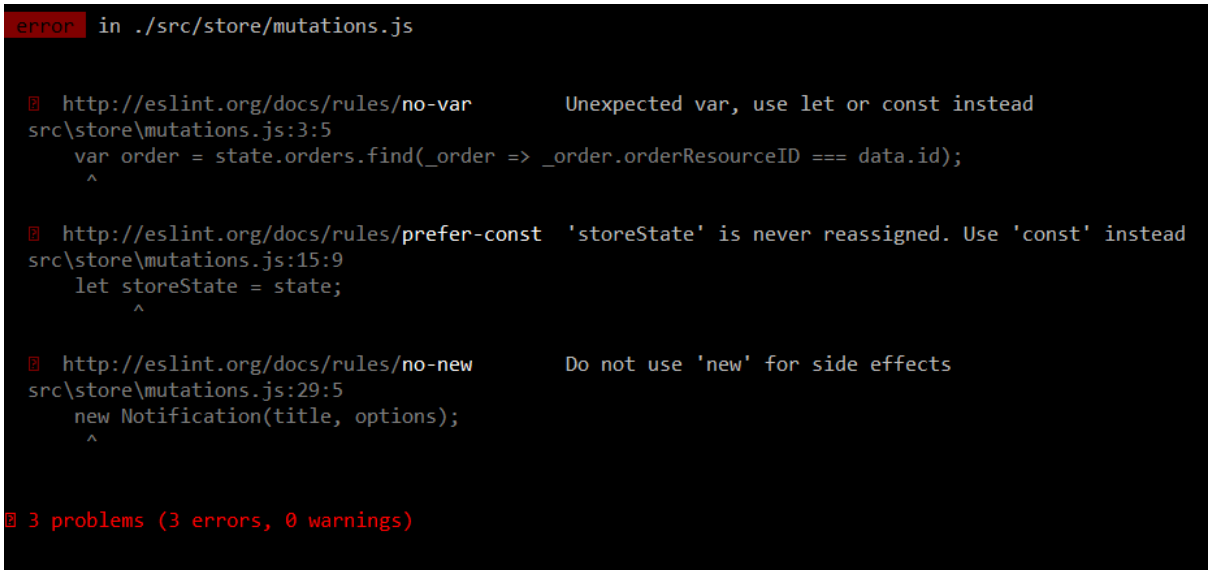
3 pav. Failo main.js klaidos

No-unused-vars taisyklė nurodo, kad programos kode negali būti jokių nepanaudotų kintamųjų. Taisyklė *no-new* nurodo tai, kad jei objektas yra sukuriamas su raktažodžiu *new*, tai jis turi būti išsaugomas kažkokiame kintamajame, *new* negali tiesiog sukurti objekto ir jo niekur neišsaugoti. Taisyklė *quotes* nurodo, kad simbolių eilutės turi būti tipo *singlequote* (taip yra nustatyta Airbnb-base taisyklių pagrinde).



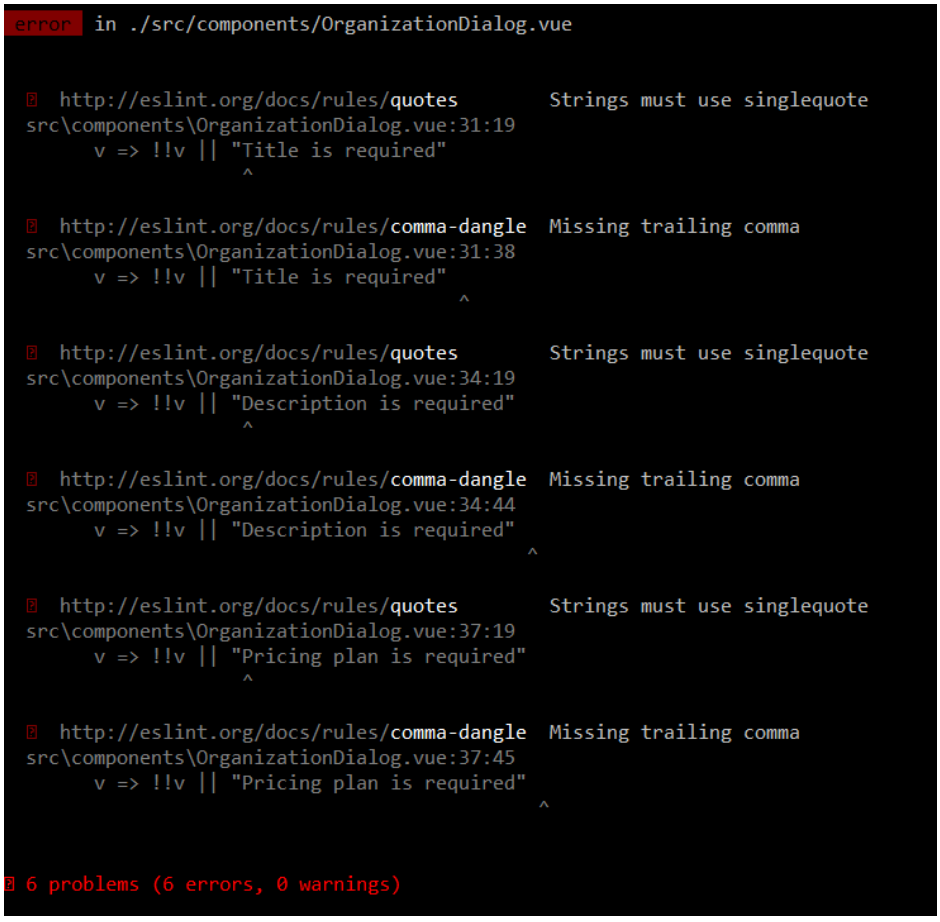
4 pav. Failo BranchesView.vue klaidos

Taisyklė *no-var* apibrėžia ES6 JavaScript standartą, kad vietoj kintamųjų sukūrimui naudojamo raktažodžio *var* turi būti naudojamas *let* arba *const*. Taip pat šiame faile nebuvo panaudotas kintamasis *showSnackbar* (kas jau buvo pastebėta kodo peržiūros metu).



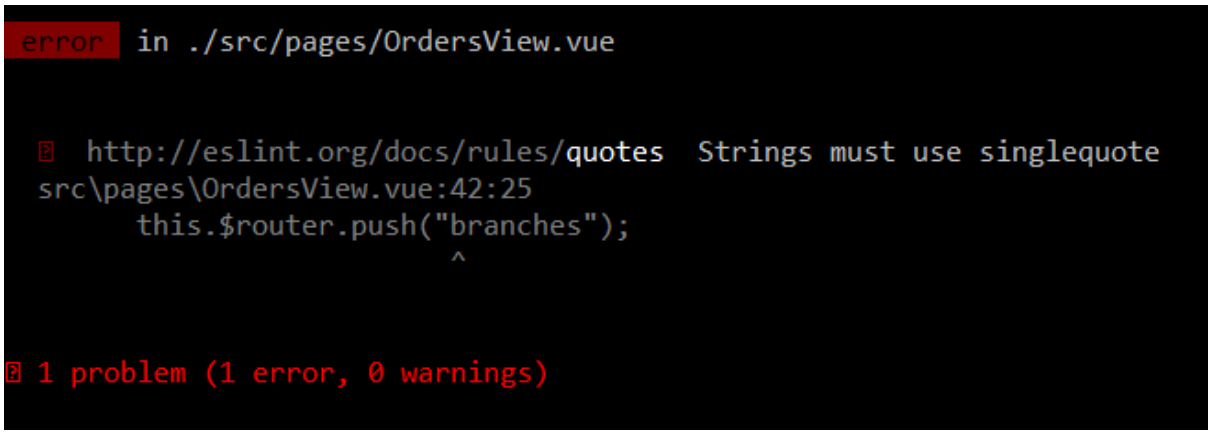
5 pav. Failo mutations.js klaidos

Pažeista *no-var* ir *no-new* taisyklės. Taisyklė *prefer-const* apibrėžia tai, kad jei kintamasis tolimesniame kode nėra daugiau priskiriamas (t.y., kintamojo reikšmė nėra perrašoma), tikslinga yra naudoti ne *let*, bet *const*.



6 pav. Failo OrganizationDialog.vue klaidos

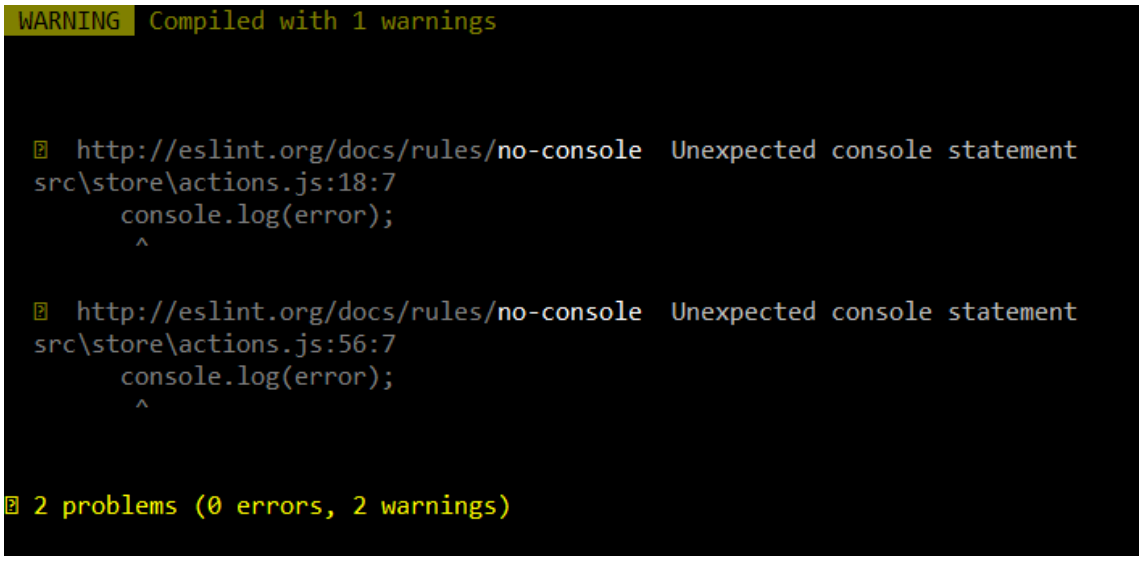
Šiame faile pažeidžiamos dvi skirtingos taisyklės – *comma-dangle* bei *quotes*. Abi taisyklės jau yra aptartos, todėl jos tiesiog ištaisomos.



7 pav. Failo OrdersView.vue klaida

Faile *OrdersView.vue* pažeidžiama kabučių klaida – simbolių eilutės turi būti išskirtos viengubomis kabutėmis.

Taip pat statinės kodo analizės įrankis pateikia ir įspėjimus – ne kritines klaidas, tačiau rekomenduojamas kodo vietas, kurias reikėtų pataisyti (8 pav.):



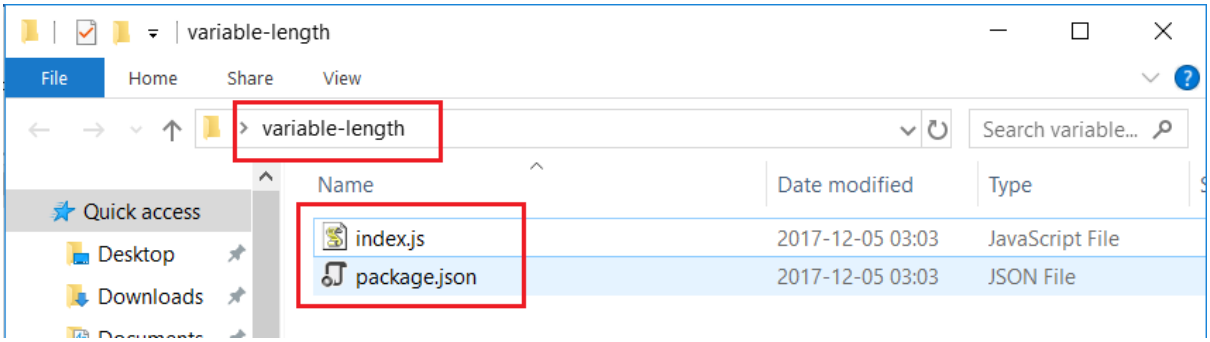
8 pav. Failo actions.js įspėjimai

Kaip matome, taisyklė *no-console* apibrėžia tai, kad nederėtų naudoti *console* komandos JavaScript kode. Ši taisyklė siejasi su jau aptarta kodo saugumo problema, kai programos duomenys yra išvedami į konsolės langą naršyklėje.

6. Naujos taisyklės įtraukimas į kodo statinės analizės įrankį

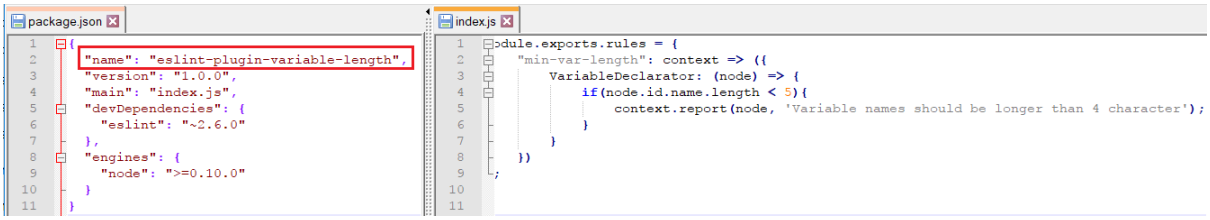
Naudojantis įrankiu ESLint, naujos taisyklės sukūrimas yra nesudėtinas. Tam, kad būtų galima sukurti naują taisyklę, pirmiausia reikia sukurti npm modulį. ESLint taisyklės sukūrimui reikia sukurti

ESLint įskiepį (plugin) bei aprašyti pačią taisyklę. Pirmiausia sukuriamas npm modulis, kuris atitiks ESlint įskiepį – sukuriamas katalogas su norimu įskiepio pavadinimu, o jame sukuriami du failai: package.json (nes ESLint įskiepis yra npm modulis, todėl šitas failas yra būtinas) bei index.js (aprašo pačios ESLint taisyklės logiką). Šio proceso rezultatas pavaizduotas 9 paveikslėlyje:



9 pav. ESLint įskiepio failų struktūros kūrimas

Po to package.json faile aprašoma npm modulio informacija, o index.js faile aprašoma pačios ESLint taisyklės logika (10 pav.).

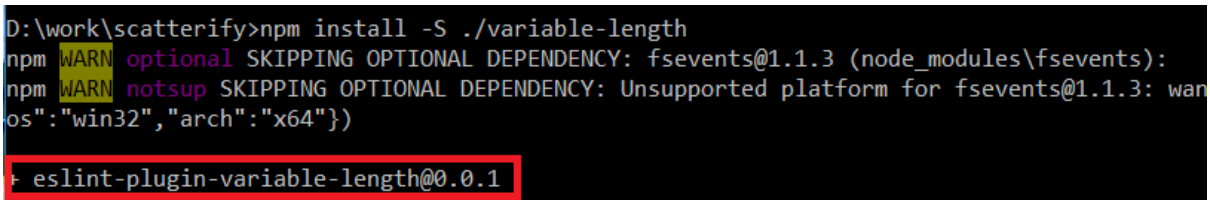


10 pav. package.json ir index.js failų turinys

Reikia paminėti, kad ESLint įskiepio pavadinimas būtinai turi prasidėti *eslint-plugin*, o toliau seka norimas įskiepio pavadinimas, kuris sutampa su sukurto katalogo pavadinimu.

Index.js faile aprašoma taisyklė: šiuo atveju aprašyta taisyklė, kuri tikrina, kad sukurtų kintamųjų vardas privalo būti ilgesnis nei 4 simboliai.

Kai failai paruošti, visas katalogas yra įkeliamas į projekto katalogą (tame pačiame lygyje, kur yra projekto package.json failas) ir paleidžiama npm komanda, kuri prijungia sukurtą įskiepį prie projekto (11 pav.).

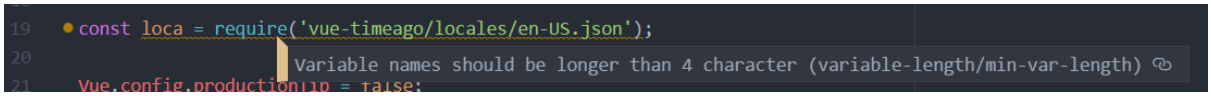


11 pav. sukurtas ESLint įskiepis prijungtas prie projekto

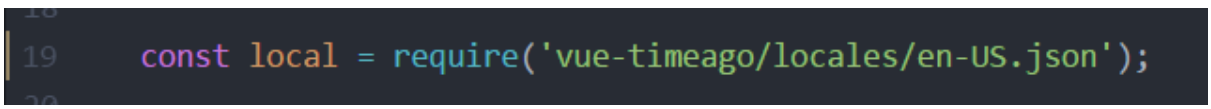
Kai įskiepis prijungtas prie projekto, jį reikia užregistruoti ESLint konfiguracijos faile .eslintrc.js (12 pav.).

12 pav. Sukurta taisyklė užregistruojama konfiguracijos faile

Kaip matome, įskiepis yra prijungiamas prie *plugins* masyvo, o pati taisyklė yra prijungiama prie *rules* objekto. Taip pat pažymima, kad taisyklė tik įspės, kad jos nėra laikomasi, tačiau kritinės klaidos neiškvies (nustatyta taisyklės svarba *warn*). Išbandžius taisyklę matome, kad ji iš karto veikia – 13 paveikslėlyje matome, kad kai kintamojo vardas yra 4 simbolių ilgio, tai rodomas įspėjimas, o kai kintamojo vardas pailginamas iki 5 ir daugiau simbolių, įspėjimas nebėra rodomas (14 pav.)



13 pav. Įspėjimas, kai kintamojo vardas yra trumpesnis nei 5 simboliai



14 pav. Kai kintamojo vardas susideda bent iš 5 simbolių, įspėjimas nebėra rodomas

7. Išvados

1) Laboratorinio darbo metu buvo peržiūrėtas programos kodas bei surastos potencialios klaidos bei rastos taisytinos kodo vietos, pritaikant gerąsias programavimo praktikas;
2) Surasti statinės kodo analizės įrankiai, kurie, kaip parodė analizės procesas, buvo tinkami kuriamam projektui;
3) Surastas statinės kodo analizės įrankis buvo pritaikytas programos kodui, buvo identifikuotos, o vėliau ir ištaisytos surastos programos kodo klaidos;
4) Parašyta nauja taisyklė, kuri vėliau buvo integruota į statinės kodo analizės įrankį;
5) Paruošta laboratorinio darbo ataskaita.