

# INSPECT MOBILE APP QA POROČILO

Marko Delić, Marko Češnjaj

# 1. Statična analiza kode z SonarQube

Z sonarqube smo izvedli statično analizo kode pri čemer smo analizirali približno 3200 vrstic. Tako smo dobili naslednje vrednosti metričnih sklopov:

Sklop metrik	Vrednost / Opis
Reliability Rating	C
Maintainability	A
Security Rating	A
Security Review	E
Duplications	< 3%
Technical Debt	5h 20min

## 1.1. Reliability Rating

Ta metrika ocenjuje tveganje za pojav napak med izvajanjem aplikacije. Dosegli smo stopnjo C, kar pomeni, da je trenutno stopnja takšnega tveganje zmerna. Zaznane so bile 3 napake, ki lahko vodijo do težav pri delovanju aplikacije in sicer:

- Convert the conditional to a boolean to avoid leaked value
- React Hook "useRef" cannot be called inside a callback. React Hooks must be called in a React function component or a custom React Hook function
- 'komentar' is assigned to itself.

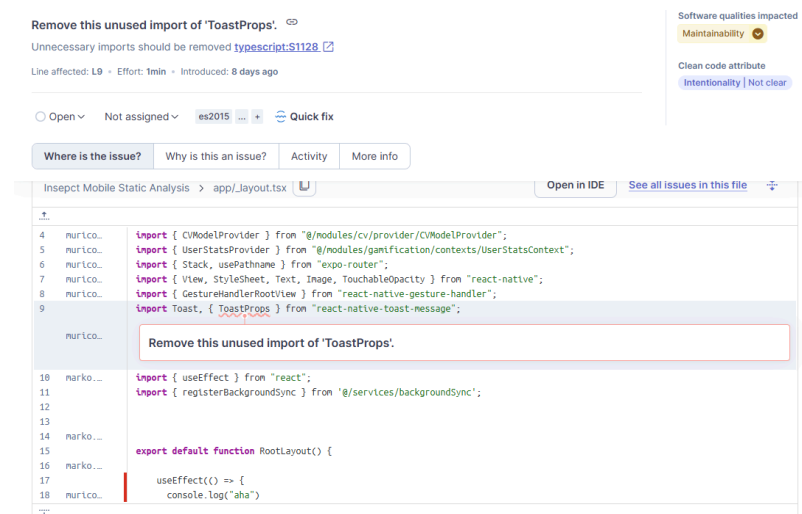
Vse tri napake so v kategoriji kritičnosti 'Medium' skupen tehnični dolg pa je ocenjen na 18 minut. To pomeni, da lahko zanesljivost naše aplikacije hitro izboljšamo z malo truda.

The screenshot displays the SonarQube web interface. On the left, a sidebar shows filters for 'My Issues' (All), 'Software Quality' (1), 'Severity' (Medium: 3), and 'Clean Code Attribute' (0). The main panel shows three issues under the heading 'Inspect Mobile Static Analysis / app/observation/details.jsx'. Each issue is a checkbox with a description, a 'Reliability' tag, and an 'Intentionality' dropdown. The issues are: 1. 'Convert the conditional to a boolean to avoid leaked value' (Reliability: type-dependent, Intentionality: react), 2. 'React Hook "useRef" cannot be called inside a callback. React Hooks must be called in a React function component or a custom React Hook function.' (Reliability: LS, Intentionality: 15), and 3. ''komentar' is assigned to itself.' (Reliability: No tags, Intentionality: 172). The bottom of the panel shows '3 of 3 shown'.

## 1.2. Maintainability

Ta metrika ocenjuje, kako berljiva je koda ter kako lahko jo je spreminjati in/ali razširjati, ne da bi pri tem povzročili napake. Dosegli smo stopnjo A, kar pomeni, da je koda v večji meri skalabilna in berljiva.

Za ta metrični sklop so bile zaznane 103 napake oziroma 'code smell-i', od tega so bile 4 ocenjene z 'High' kritičnostjo, 25 jih je bilo ocenjenih z 'Medium' kritičnostjo ter 74 z 'Low' kritičnostjo. Tehnični dolg je bil ocenjen na 5 ur in 2 minuti. Iz teh podatkov lahko sklepamo, da je koda razmeroma dobro strukturirana glede na obseg, a bi bila lahko z nekaj truda še bolj optimalna za vzdrževanje.



## 1.3. Security Rating

Ta metrični sklop ocenjuje tveganja ranljivosti v kodi, ki bi lahko bile zlorabljene (XSS, SQL injection, razkriti API ključi...). Dosegli smo stopnjo A, kar pomeni, da v kodi ni bilo zaznanih varnostnih ranljivosti. Uporabili smo namreč dobre varnostne prakse, kot je parametriziran SQL, .env datoteka z ključi itd., kljub temu pa orodje ne more prepoznati vseh tveganj zato, bomo v nadaljevanju še posebej testirali aplikacijo iz varnostnega vidika.

▼ Software Quality ?

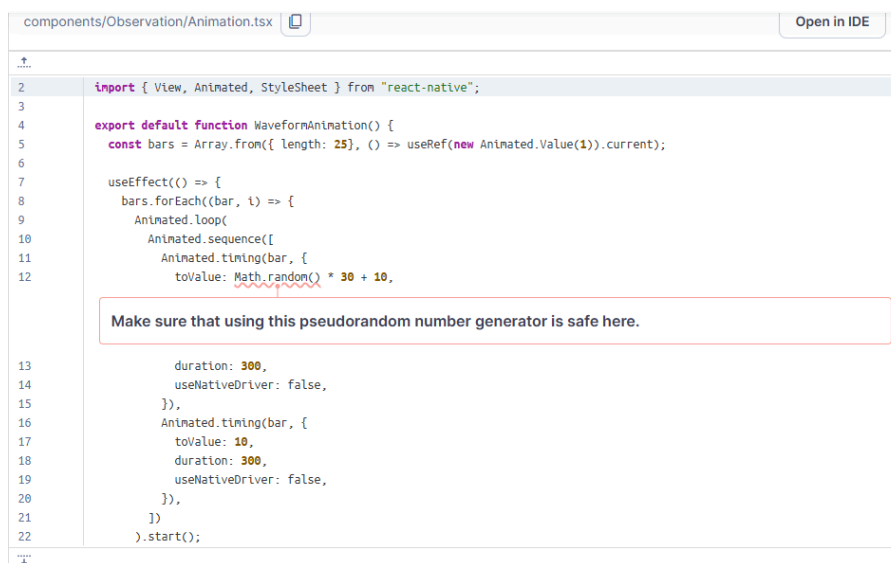
Security

0

## 1.4. Security Review

Ta sklop metrik prikazuje dele kode ("vroče točke"/"hotspots"), ki so potencialno nevarni, vendar niso klasificirani kot ranljivosti. Dosegli smo oceno E, kar pomeni, da nobenega od takšnih delov še nismo pregledali. To pomeni, da moramo te "hotspote" ročno preveriti ter jih po tem v SonarQube označiti z "reviewed".

SonarQube je z analizo samo 1 del v pregledani kodi označil kot vročo točko in sicer gre za naključno oziroma random vrednost uporabljeno pri animaciji grafike zvočnega posnetka. To pomeni, da lahko z gotovostjo samo označimo "hotspot" kot "reviewed".



The screenshot shows a code editor window titled 'components/Observation/Animation.tsx' with a 'Open in IDE' button. The code is as follows:

```
2 import { View, Animated, StyleSheet } from "react-native";
3
4 export default function WaveformAnimation() {
5   const bars = Array.from({ length: 25 }, () => useRef(new Animated.Value(1)).current);
6
7   useEffect(() => {
8     bars.forEach((bar, i) => {
9       Animated.loop(
10        Animated.sequence([
11          Animated.timing(bar, {
12            toValue: Math.random() * 30 + 10,
13
14            duration: 300,
15            useNativeDriver: false,
16          }),
17          Animated.timing(bar, {
18            toValue: 10,
19            duration: 300,
20            useNativeDriver: false,
21          })
22        ]).start();
23      });
24    });
25  });
26 }
```

A red box highlights the line `Math.random() * 30 + 10` with the message: "Make sure that using this pseudorandom number generator is safe here."

## 1.5. Duplications

Ta metrični sklop meri delež podvojenih vrstic kode skozi projekt. Podvajanje zmanjšuje vzdrževanost, saj moramo ob spremembi ene kopije spremeniti tudi druge. Dosegli smo 0% podvajanja, kar pomeni, da SonarQube ni zaznal nobene podvojene kode. To nakazuje, da je naša koda dobro modularizirana ter sledi DRY principu (Dont Repeat Yourself).

## 1.6. Technical Debt

Tehnični dolg predstavlja oceno časa, potrebnega za odpravo vseh zaznanih vzdrževalnih in varnostnih težav in pomankljivosti ter težav in pomanjkljivosti zanesljivosti. Tehnični dolg je glede na obseg projekta zmerno nizek (5h 20min ob 3200 pregledanih vrsticah). Večino dolga povzroča slaba koda oziroma "code smells" (94%), kar ni kritično za funkcionalno delovanje aplikacije, je pa pomembno za njeno skalabilnost in vzdrževanost. Popolna odprava vseh težav bi toraj terjala manj kot en delovni dan.



## 2. Enotsko testiranje

Uporabljena orodja: **Jest** za izvajanje testov, **@testing-library/react-native** za testiranje React komponent, **Jest mocking** za mockanje SQLite ter Async Storage-a

### 2.1. Testiranje konteksta igrifikacije - datoteka UserStatsContext.tsx

Testiranje funkcionalnosti posodabljanja konteksta preko setStats, kjer se preveri ali se pravilno posodobijo vrednosti v contextu ter testiranje ali so pravilno prebrane začetne vrednosti "xp" in "level" iz AsyncStorage-a ob zagonu aplikacije

```
UserStatsProvider
  ✓ posodobi context pravilno prek setStats (204 ms)

  ✓ prebere začetne vrednosti iz AsyncStorage (66 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time: 1.021 s
```

Preverjamo, ali nam izpiše napako, če useUserStats hook ni znotraj providerja.

```
PASS tests/unit/UserStatsContextHookError.test.tsx
  UserStatsContext hook
    ✓ vrže napako, če hook ni znotraj providerja (36 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time: 1.377 s
```

## 2.2. Testiranje delovanja napredovanja uporabnika in dodelitve XP-ja - datoteka LevelService.ts

Testiranje ali se spremeni level, če XP ne presega praga ter ali se level posodobi, če XP preseže določen prag. Preverjanje ali se XP pravilno povečuje ter shranjuje.

```
PASS tests/LevelService.test.tsx
  checkLevel
    ✓ ne spremeni levela, če XP ne presega praga (4 ms)
    ✓ posodobi level, če XP preseže prag (1 ms)
  addXp
    ✓ poveča XP in shrani podatke (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.69 s, estimated 2 s
```

Preverjanje, kaj se zgodi, če presežemo maksimalno vrednost XP-ja (sepravi presežemo level 4) - zaželeno je, da se maksimalnega levela ne more presežti.

```
PASS tests/unit/LevelServiceMaxLvl.test.ts
  ✓ ne preseže maksimalnega levela (4) (4 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.269 s
```

## 2.3. Testiranje dodeljevanja dosežkov - datoteka AchievementService.ts

Testiranje ali se pravilno dodeli več dosežkov hkrati glede na izpolnjene pogoje.

```
PASS tests/AchievementService.test.tsx
checkAchievements - kombiniran test
  ✓ dodeli več dosežkov hkrati glede na pogoje (4 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        1.266 s, estimated 2 s
```

Testiranje, kaj se zgodi če se doda v bazo neveljaven ID dosežka - ciljno obnašanje je, da se ta ID ignorira in izpiše opozorilo v console.log

```
PASS tests/AchievementWrongId.test.tsx
  ✓ ignorira neveljaven ID dosežka in izpiše opozorilo (4 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
```

## 2.4. Coverage podatki

All files  
66.27% Statements 334/337 51.92% Branches 27/52 60.6% Functions 26/33 66.87% Lines 345/337

Press n or j to go to the next uncovered block, b, p or k for the previous block.

Filter:

File	Statements	Branches	Functions	Lines
app/profile	100%	3/3	50%	1/2
assets/styles/Profile	0%	0/0	0%	0/0
modules/gamification/contexts	100%	29/29	71.42%	10/14
modules/gamification/services	58.57%	82/140	44.44%	16/36

Testna pokritost se nanaša predvsem na **service in module**, kjer je implementirana ključna poslovna logika aplikacije. Pokritost vključuje le datoteke, ki so bile dejansko vključene v testne primere (uvožene/importane).

Rezultati pokritosti prikazujejo stanje **testirane kode**, ne celotnega projekta. To pomeni, da npr. uporabniški vmesnik (UI) ali manj pomembne pomožne komponente, ki niso bile neposredno testirane, niso upoštevane v procentih pokritosti.

Pokritost ključnih servisov (npr. LevelService, UserStatsContext) dosega tudi 100%, medtem ko so kompleksnejši moduli, kot je AchievementService, delno pokriti (okoli 52%).



### 3. Integracijsko testiranje

Integracijsko testiranje je vrsta testiranja programske opreme, pri katerem združimo različne komponente ali module aplikacije ter preverimo njihovo skupno delovanje. V našem primeru smo imeli integracije s sistemi: Firebase Auth, Firebase Firestore, Firebase Storage ter integracijo LLM modela llama-3.3-70b-versatile.

#### 3.1. Firebase Auth testiranje

Testirali smo delovanje funkcionalnosti prijava in s tem namenom spisali integracijski test, ki simulira postopek prijave uporabnika. Test vključuje korake; 1. Avtentikacija z Firebase Auth, 2. Preverjanje v lokalni SQLite bazi, 3. Shranjevanje podatkov v AsyncStorage.

V testu smo zunanje storitve nadomestili z ustreznimi mocki ter vključili primer negativnega izida, kjer email uporabnika še ni potrjen.

```
PASS tests/integration/auth.test.ts
Integration test: Login flow
  ✓ prijavi uporabnika, shrani AsyncStorage in SQLite (12 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.378 s, estimated 4 s
Ran all test suites matching /tests\\integration\\i.
```

#### 3.2. Firebase Firestore

Za namen takšnega testa smo vzpostavili Firebase tools okolje ter Firestore Emulator, da smo simulirali delovanje Firebase Firestore sistema. V testni zbirki smo pripravili 4 testne primerke, vsakega za eno izmed CRUD operacij, da smo zajeli vse firestore funkcionalnosti, ki jih v aplikaciji potrebujemo.

```
PASS tests/integration/firestore.test.ts
Integration test: Firestore CRUD
  ✓ dodajanje dokumenta (1220 ms)
  ✓ posodabljanje dokumenta (43 ms)
  ✓ brisanje dokumenta (31 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        4.256 s, estimated 18 s
Ran all test suites matching /tests\\integration\\firestore.test.ts/i.
```



### 3.3. Firebase Storage

Za namen takšnega testa smo vzpostavili še Storage Emulator, da smo simulirali delovanje Firebase Storage sistema. Pripravili smo testni primerek, ki v Firebase Storage vstavi sliko ter preveri njeno pot ter ali se je dejansko vstavila. Po tem pa smo še pridobili url slike shranjene v Firebase Storage ter tako zajeli vse storage funkcionalnosti, ki jih v aplikaciji potrebujemo.

```
PASS tests/integration/storage.test.ts (9.915 s)
  Integration test: Firebase Storage Image save
    ✓ vstavljanje slike (199 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        11.281 s
Ran all test suites matching /tests\\integration\\storage.test.ts/i.
```

### 3.4. LLM Model

Za namen takšnega testa smo uvozili našo metodo getModelResponse, ki pošlje "chat completion" API klic na ponudnika, ki ga uporabljamo. Ustvarili smo testni tovor, ki modelu pove naj odgovori z besedo "test". Preverili smo tip dolžino, tip in vsebino odgovora.

```
PASS tests/integration/llm.test.ts (5.761 s)
  Integration test: LLM response
    ✓ llm response (319 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        7.607 s
Ran all test suites matching /tests\\integration\\llm.test.ts/i.
```

## 4. Varnostna Analiza

Zato, ker orodje za statično analizo kode SonarQube ne more z gotovostjo zaznati vseh ranljivosti smo izvedli še dodatno varnostno analizo.

### 4.1. Audit Fix

Z ukazom npm audit fix smo, kjer je možno, posodobili uporabljene pakete v projektu na stabilne verzije, ki ne vsebujejo ranljivosti. Kljub temu sta ostali 2 ranljivosti kritičnosti 'Moderate', povezani s paketom "markdown-it". Ker trenutno za ta paket ni na voljo varne posodobitve ("no fix available"), bomo v prihodnje zamenjali paket ali pa spremljali posodobitve ter, ko bo možno naložili stabilno verzijo.

Naša aplikacija uporablja markdown pretvorbo iz uporabniškega vnosa pri pogovoru z LLM asistentom, zato je pomembno, da ranljivost res spremljamo ter jo takoj ob možnosti odpravimo.

```
{
  "auditReportVersion": 2,
  "vulnerabilities": {
    "markdown-it": {
      "name": "markdown-it",
      "severity": "moderate",
      "isDirect": false,
      "via": [
        {
          "source": 1092663,
          "name": "markdown-it",
          "dependency": "markdown-it",
          "title": "Uncontrolled Resource Consumption in markdown-it",
          "url": "https://github.com/advisories/GHSA-6vfc-qv3f-vr6c",
          "severity": "moderate",
          "cwe": [
            "CWE-400",
            "CWE-1333"
          ],
          "cvss": {
            "score": 5.3,
            "vectorString":
"CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L"
          },
          "range": "<12.3.2"
        }
      ]
    }
  }
}
```

```
    ],
    "effects": [
      "react-native-markdown-display"
    ],
    "range": "<12.3.2",
    "nodes": [
      "node_modules/markdown-it"
    ],
    "fixAvailable": false
  },
  "react-native-markdown-display": {
    "name": "react-native-markdown-display",
    "severity": "moderate",
    "isDirect": true,
    "via": [
      "markdown-it"
    ],
    "effects": [],
    "range": "*",
    "nodes": [
      "node_modules/react-native-markdown-display"
    ],
    "fixAvailable": false
  }
},
"metadata": {
  "vulnerabilities": {
    "info": 0,
    "low": 0,
    "moderate": 2,
    "high": 0,
    "critical": 0,
    "total": 2
  },
  "dependencies": {
    "prod": 829,
    "dev": 616,
    "optional": 52,
    "peer": 39,
    "peerOptional": 0,
    "total": 1449
  }
}
```

## 4.2. Eslint Security

Dodatno, da preverimo še lastno kodo poleg vključenih paketov smo naložili eslint ter eslint-plugin-security paketa ter pripravili konfiguracijo. Po analizi smo tako kot z SonarQube ugotovili, da ni bilo zaznanih nobenih varnostnih ranljivosti temveč samo "code smells", od katerih se da 3 napake in 5 opozoril potencialno avtomatsko odpraviti z `-fix` ukazom.

## 5. Analiza kode

### 5.1. Kompleksnost

Z VS Code pluginom CodeMetrics ročni pregled kompleksnosti raznih funkcij.

Ugotovitve po pregledu so, da kompleksnost niha med 3-5, kar je nizka vrednost in nakazuje, da spisane funkcije niso preveč kompleksne. So pa tudi izjeme, kjer je kompleksnost višja od 5, kar pa nakazuje, da bi lahko določene funkcije spisali z manjšo kompleksnostjo.

```
Complexity is 6 It's time to do something...
export async function insertImageInFirestore(observationId: number, imagePath: string) {
  const netInfo = (await NetInfo.fetch()).isConnected;
  if (!netInfo) return;

  try {
    const response = await fetch(imagePath);
    const blob = await response.blob();

    const fileName = imagePath.split("/").pop() || `photo_${Date.now()}.jpg`;
    const path = `observations/${Date.now()}${fileName}`;

    const storageRef = ref(storage, path);
    await uploadBytes(storageRef, blob);

    const imageUrl = await getDownloadURL(storageRef);

    const firebaseUid = await AsyncStorage.getItem("user_firebase_id");

    const observationRef = doc(firebaseDb, "users", firebaseUid, "observations", observationId.toString());
    await setDoc(observationRef, {image_path: imageUrl}, {merge: true});
  } catch (error) {
    console.error(error);
  }
}
```

```
Complexity is 3 Everything is cool!
export const useXpUpdater = () => {
  const { setStats } = useUserStats();

  const gainXp = async (amount: number) => {
    const result = await addXp(amount);
    setStats(result);
  };

  return gainXp;
};
```

```
Complexity is 6 It's time to do something...
const handleScan = async () => {
  if (!model) {
    console.warn("Model še ni naložen");
    return;
  }
  setIsLoading(true);
  if (cameraRef.current) {
    try {
      const photo = await cameraRef.current.takePictureAsync();
      const modelInput = await preprocessImage(photo.uri);
      const output = await model.run(modelInput);
      const scores = Array.from(output[0] as Float32Array);
      const topClass = scores.indexOf(Math.max(...scores));
      router.push({pathname: '/observation/details', params: {prediction: topClass, photoUri: photo.uri}});
    } catch (error) {
      console.error("Napaka pri zajemu slike: ", error);
    }
  }
}
```

```
Complexity is 5 Everything is cool!
const setStats = (stats: Partial<UserStats>) => {
  if (stats.xp !== undefined) setXp(stats.xp);
  if (stats.level !== undefined) setLevel(stats.level);
  if (stats.progress !== undefined) setProgress(stats.progress);
  if (stats.levelUp !== undefined) setLevelUp(stats.levelUp);
};

Complexity is 4 Everything is cool!
useEffect(() => {
  Complexity is 3 Everything is cool!
  (async () => {
    const storedXp = await AsyncStorage.getItem("user_xp");
    const storedLevel = await AsyncStorage.getItem("user_level");
    if (storedXp !== null) setXp(Number(storedXp));
    if (storedLevel !== null) setLevel(Number(storedLevel));
  })();
}, []);
```

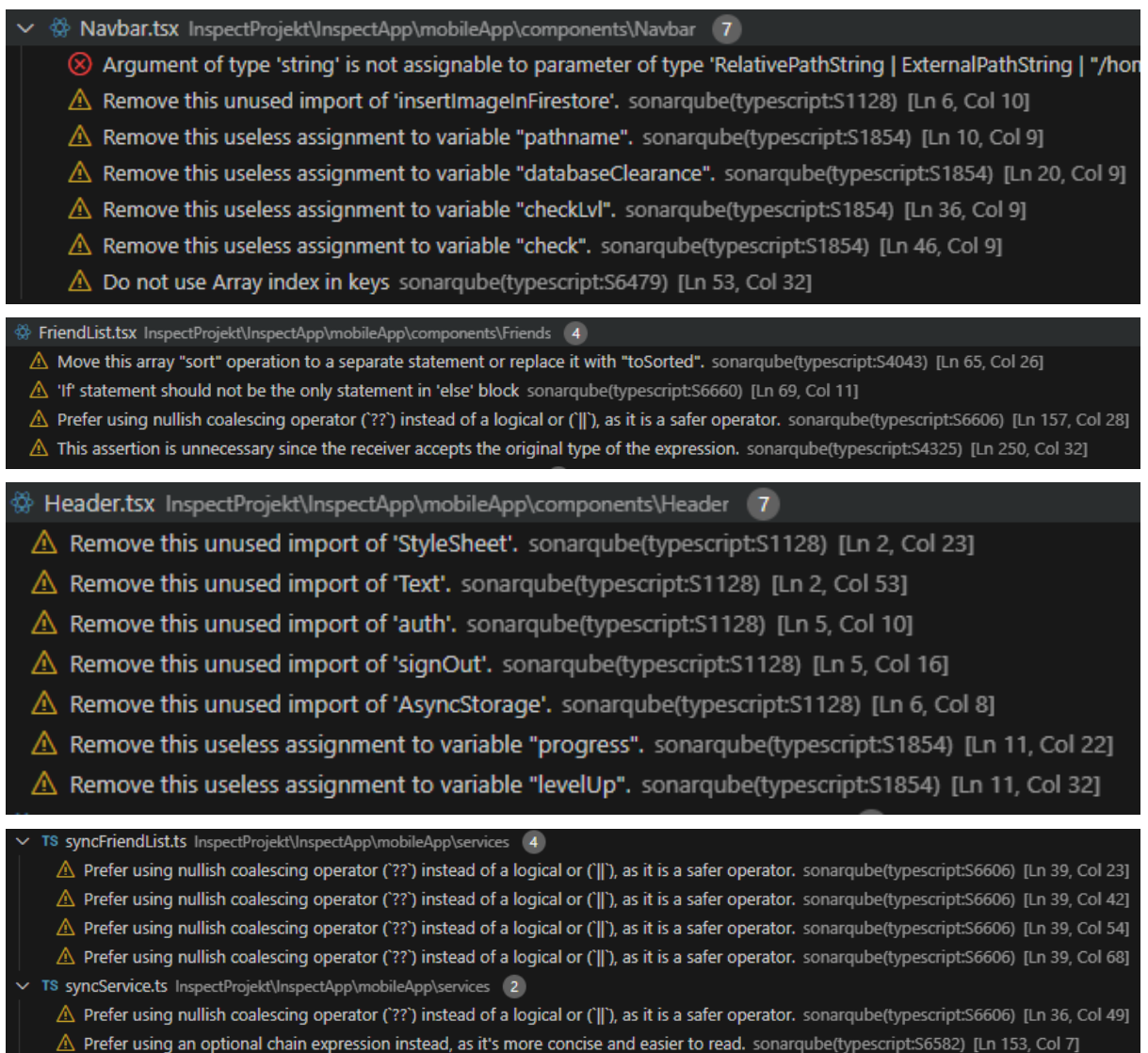
```
Complexity is 4 Everything is cool!
function processData(data: InsectsOfRedRow[]) {
  const uniqueDruzine = [...new Set(data.map((item) => item.naziv_druzine))];
  const preprocessed = uniqueDruzine.map((druzina) => ({
    druzina,
    insects: data.filter((insect) => insect.naziv_druzine == druzina),
  }));
  setRedData(preprocessed);
}
```

## 5.2. Code smells

Analiza s SonarLint je pokazala, da se večina zaznanih težav (code smells) nahaja v komponentah in servisih za sinhronizacijo podatkov. Najpogostejši problemi vključujejo:

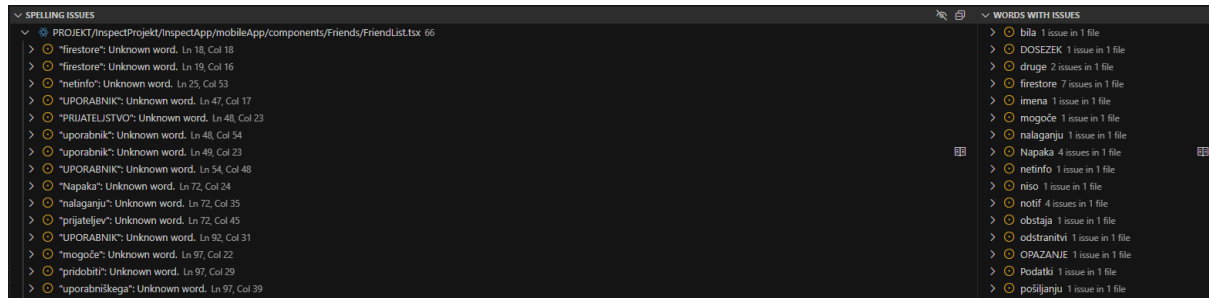
- **Neuporabljeni importi ali spremenljivke**, kar nakazuje na nepopolno refaktorizacijo ali nepotrebno kompleksnost.
- **Uporaba operatorja || namesto ??**, kar lahko povzroči neželene logične napake, ko želimo preverjati samo null/undefined in ne vseh "falsy" vrednosti.

Ti code smell-i kažejo na pomanjkanje strogega preverjanja kakovosti med razvojem (npr. s pomočjo lintanja, unit testov ali rednega refactoringa). Kljub temu, da gre večinoma za manjše težave, so ti znaki pomembni, saj se lahko postopoma kopičijo in vodijo v tehnični dolg ter otežujejo dolgoročno vzdrževanje in razširjanje aplikacije.



## 5.3. Preverjanje slovničnih/tipkarskih napak

Med analizo kode z orodjem **Code Spell Checker** ni bilo zaznanih nobenih tipkarskih napak v imenih spremenljivk, funkcij, komentarjih ali drugih besedilnih nizih.



Orodje je sicer zaznalo več izrazov kot morebitne napake (npr. “uporabnik”, “nalaganju”, “firestore”), vendar gre v vseh primerih za namenoma uporabljene slovenske izraze ali tehnične termine, ki jih črkovalnik ne prepozna kot standardne besede.

Kljub temu ni bilo ugotovljenih dejanskih tipkarskih ali slovničnih napak. To kaže na dosledno in jezikovno ustrezno poimenovanje v kodi, kar prispeva k boljši berljivosti in vzdrževanju projekta.



## 6. CI/CD QA Pipeline

Za namen avtomatskega testiranja in sprotnega zagotavljanja kakovosti smo pripravili Github Actions workflow. V njem so vsebovana varnostna eslint in npm audit preverjanja, pogon unit ter integration testov in statična analiza kode z SonarQube cloud. Nismo nastavili nobenih pravil, ki bi ob neuspehu pipeline zavrnili pull oziroma push, saj bo eslint velikokrat zavrnilo zaradi "code smells", ki pa niso kritični.

