Student: Chris Neven
Studentnummer: 500674423
Docent: Tim Visser
Klas: SE201
Datum: 25 december 2018

Practicum 5: Static Code Analysis & Defect Reporting

Practicum 5 van het vak Testing om de kennis en vaardigheden van statische code-analyse en opvolging daarvan te verbreden en te verdiepen. Voor het practicum wordt SonarQube Community Edition gebruikt om een applicatie te analyseren.

Document historie

#	Versie	Datum	Aanpassingen
1	1.0	22-12-2018	Structuur van practicum opgezet
2	1.1	25-12-2018	Eerste en tweede iteratie SonarQube voor front-end
3	1.2	29-12-2018	Derde, verder iteratie SonarQube voor frontend en eerste iteratie voor back-end

Inleiding

Zoals eerder vermeld is dit practicum 5 van het vak testen in de vorm van een verslag. Voor dit verslag heb ik gebruik gemaakt van SonarQube Community Edition om een applicatie statisch te kunnen testen. Als case voor het practicum heb ik mijn huidige afstudeerproject gebruikt genaamd de leden-applicatie dat deel uitmaakt van de Texel Online Reserveringssysteem. Het rapport bestaat uit een meertal hoofdstukken die als resultaat dienen van het gemaakte practicum. In het eerste hoofdstuk wordt er ingegaan op wat de applicatie die getest wordt inhoud. Vervolgens wordt er in hoofdstuk twee geanalyseerd door middel van een eerste iteratie met SonarQube die de huidige bugs weergeven.

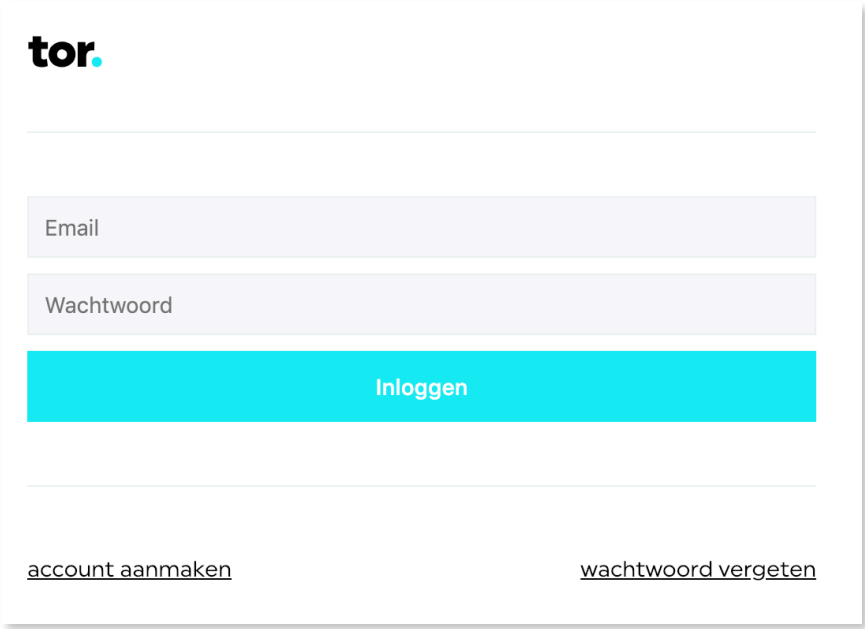
1. TOR leden-applicatie

Texels Online Reserveringssysteem (TOR) is een op maat gemaakt online boekingssysteem gemaakt door het bedrijf Oberon Medialab B.V. (Oberon). Het werd elf jaar geleden voor het eerst in gebruik genomen bij de Vereniging voor Vreemdelingenverkeer (VVV) op Texel. Inmiddels wordt het niet meer alleen op Texel gebruikt, maar ook op de overige Waddeneilanden. Het boekingssysteem beschikt over verschillende opties van accommodatie verhuur zoals bijvoorbeeld huizen, appartementen en hotels. TOR bestaat uit vier componenten. Een website waarop de consument accommodaties kan zoeken en boeken, een leden-applicatie waar verhuurders hun accommodatie online beschikbaar stellen, een backoffice applicatie waar de administratie van huur en verhuur wordt bijgehouden en een database voor het persisteren van de data van de drie opgenoemde componenten. De accommodatie die in de leden-applicatie door de verhuurder beschikbaar wordt gesteld kan door de consument worden geboekt op de website. De context van het practicum betreft de leden-applicatie en zoals eerder is vermeld is het bouwen van de leden-applicatie nog niet klaar.

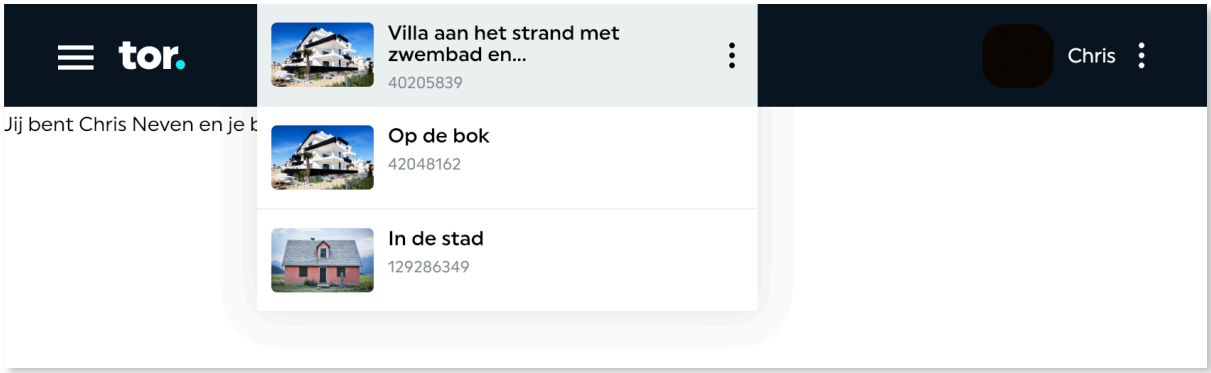
De leden-applicatie bestaat technisch gezien uit een front-end en een back-end die samen het geheel vormen. De front-end is een project geschreven in TypeScript en maakt gebruik van de library ReactJS. TypeScript is een strikte laag die compileert naar JavaScript. TypeScript voegt de mogelijkheid tot het gebruiken van strikte typen voor variabelen en het gebruik van object georiënteerd programmeren. Het laatste wordt in dit geval niet gebruikt, omdat de applicatie met het functioneel programmeer paradigma is opgezet. Verder wordt er in de beide projecten gebruik gemaakt van TSLint waardoor al veel code in real-time geanalyseerd wordt tijdens het programmeren. Dit zorgt er tevens voor dat er errors en bugs al worden aangekaart en meteen verholpen moeten worden. SonarQube maakt ook gebruik van deze TSLint configuraties tijdens het analyseren van de applicatie.

1.1 Leden-applicatie front-end

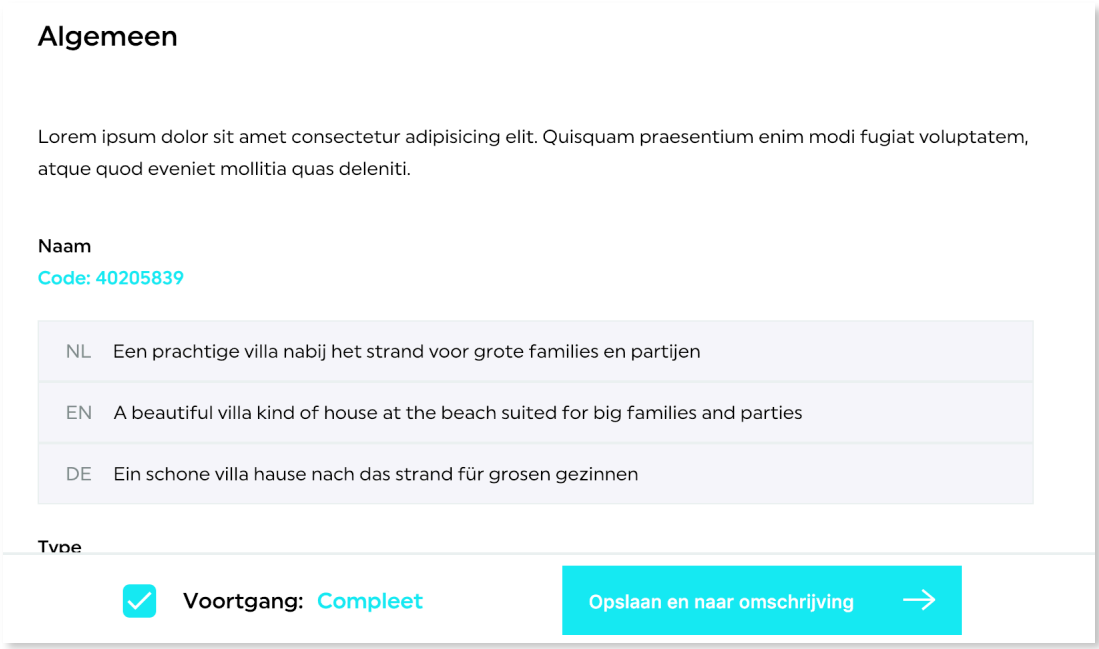
Hieronder kunnen een aantal screenshots van de applicatie worden gevonden die het uiterlijk van de front-end van de applicatie weergeven.



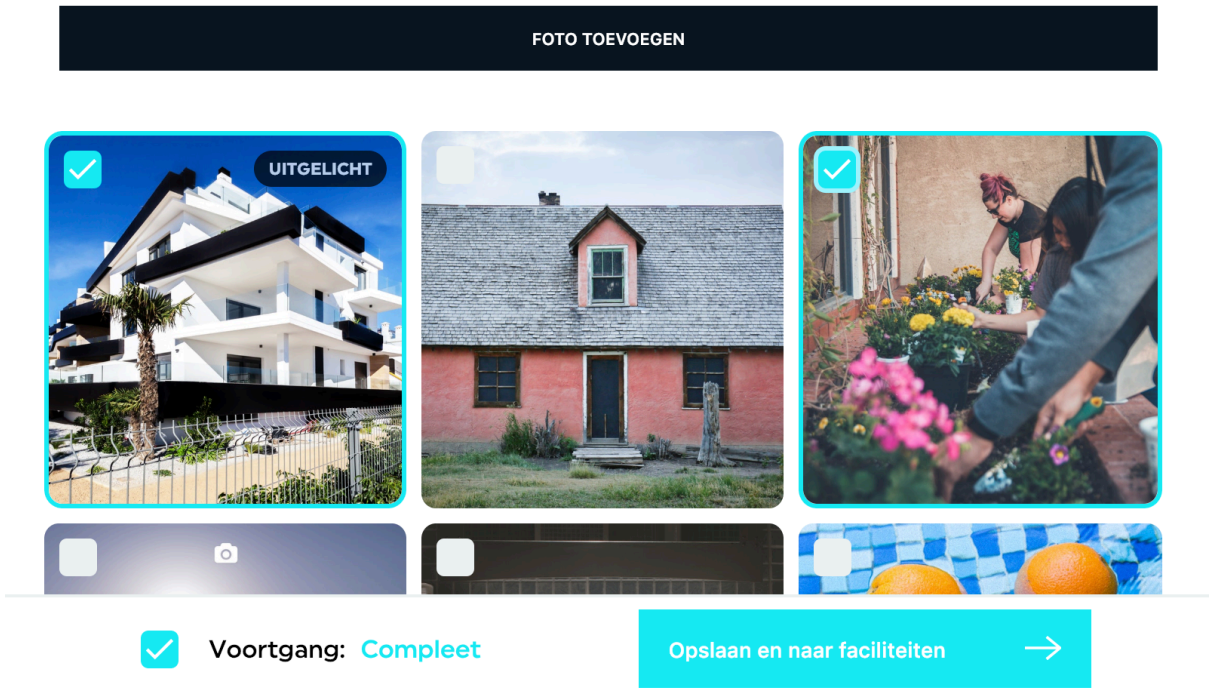
Figuur 1: Inlogscherm



Figuur 2: Accommodatie dropdown



Figuur 3: Weergeven & bewerken van accommodatie info

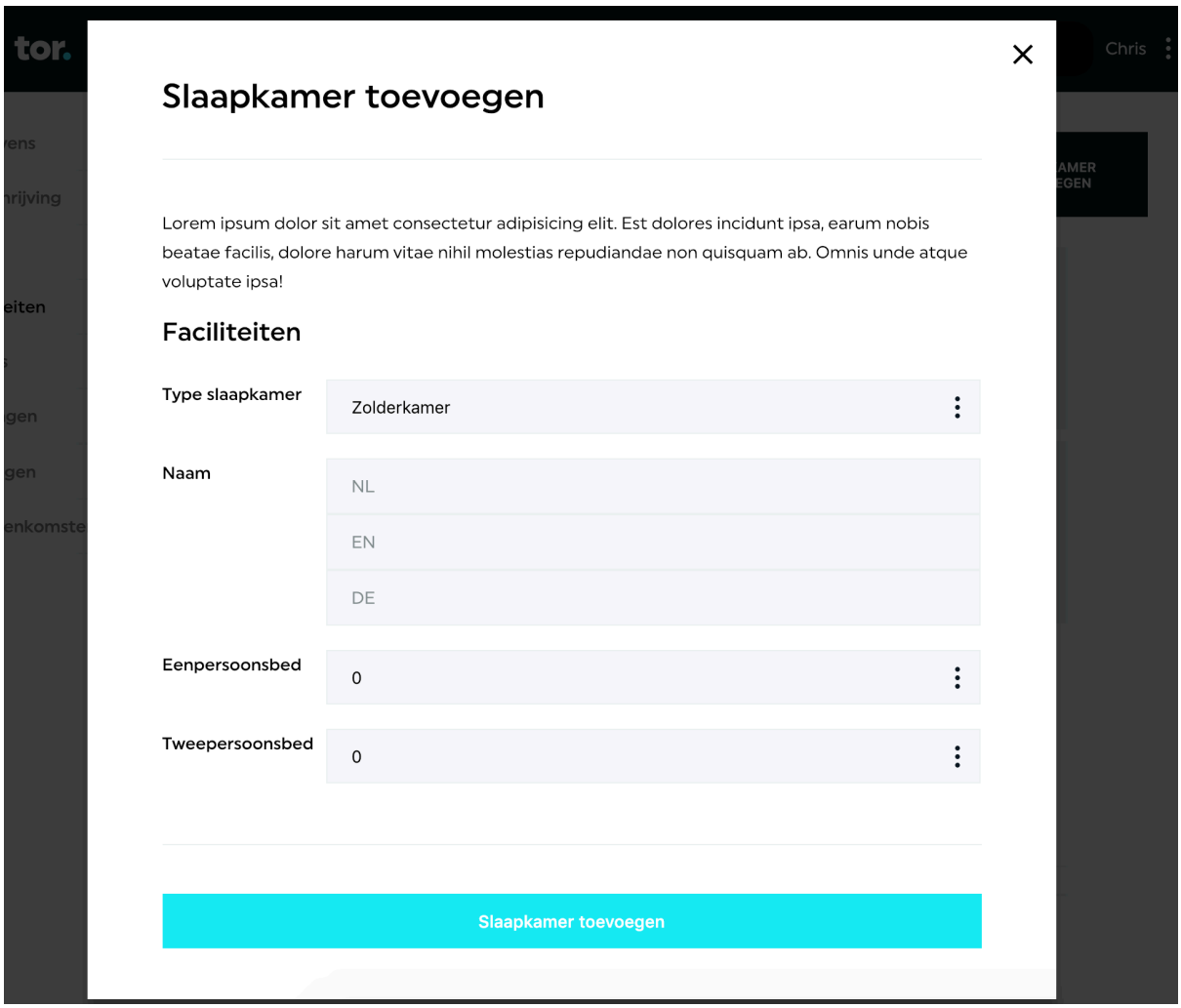


Figuur 4: Weergeven & bewerken van accommodatie foto's

Slaapkamers

Lorem ipsum dolor sit amet consectetur adipisicing elit. Quisquam praesentium enim modi fugiat voluptatem, atque quod eveniet mollitia quas deleniti.

Figuur 5: Weergeven van kamers



Figuur 6: Slaapkamer toevoegen

2.2 Leden-applicatie back-end

De back-end is ook een typescript project maar dan in plaats van dat er gebruik gemaakt wordt van ReactJS wordt er voor de back-end gebruik gemaakt van Prisma. Zie figuur 7 voor een abstract overzicht van de Prisma back-end. De back-end beschikt over meerdere entiteiten zoals accommodation, booking, user, room en nog meer. Voor het persisteren van data wordt Postgres gebruikt.

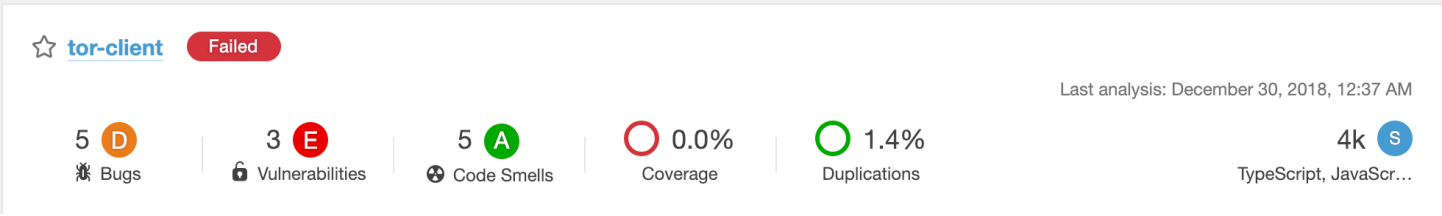
Figuur 7: Overzicht back-end

2. SonarQube analyse

Om het analyseren van de applicatie zo compleet uit te voeren moeten zowel de back-end als front-end meegenomen worden. Voor de beide projecten is een sonar-project.properties bestand opgezet voor sonar-scanner om de projecten te kunnen analyseren. De node packages die zijn uitgesloten van de analyse omdat dat buiten de scope gaat van de leden-applicatie. Er kan onderscheid gemaakt worden tussen de verschillende niveaus van prioriteiten: low, medium, high, critical en blocker. Voor meer informatie zie de volgende tabel.

Low	Lage prioriteit voor het verhelpen. In productie stage noodzakelijk eruit, in development stage kan het eventueel van toepassing zijn.
Medium	Medium prioriteit voor het verhelpen.
High	Hoge prioriteit voor het verhelpen.
Critical	Kritieke prioriteit voor het verhelpen. Blockt de applicatie niet, maar is op het gebied van security zeer noodzakelijk dat de issue verholpen wordt.
Blocker	Kritieke prioriteit die de applicatie zou kunnen blocken in productie. Moet gelijk verholpen worden

2.1 Eerste Front-end analyse



Figuur 8: Overzicht back-end

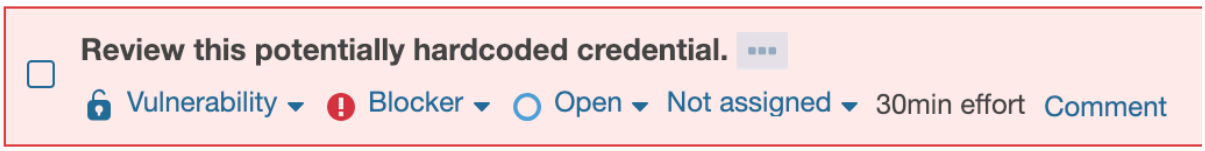
Zoals gezien kan worden in figuur 8 is na het analyseren van de leden-applicatie een negatieve satus: failed. Het project beschikt over vijf bugs, drie security issues en vijf code smells. Dit samen leidt er toe dat de status failed is. In de eerste iteratie van de analyses worden de vulnerabilities aangepakt.

2.1.1 Debugger issue

Figuur 9: Debugger issue

Het prioriteitsniveau is high ondanks dat het geen directe bij-effect op de werking van het systeem. Echter, is het noodzakelijk dat de issue verholpen wordt omdat het de kwetsbaarheid voor aanvallen verhoogt.

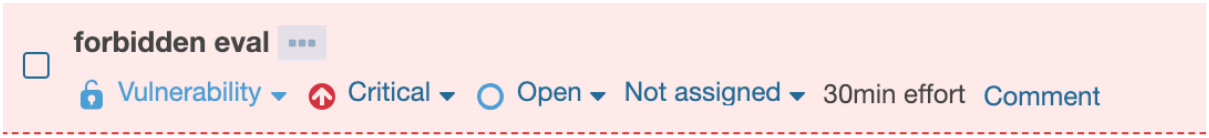
2.1.2 Hardcoded password issue



Figuur 10: Hardcoded password issue

Het prioriteitsniveau is blocker omdat er een wachtwoord in de code is gezet. Dit was om te testen of het inloggen goed werkte. Dit moet dus nog verholpen worden en is zeer belangrijk. Ongewenste gebruikers kunnen achter het wachtwoord komen en zo authenticatie verwerven.

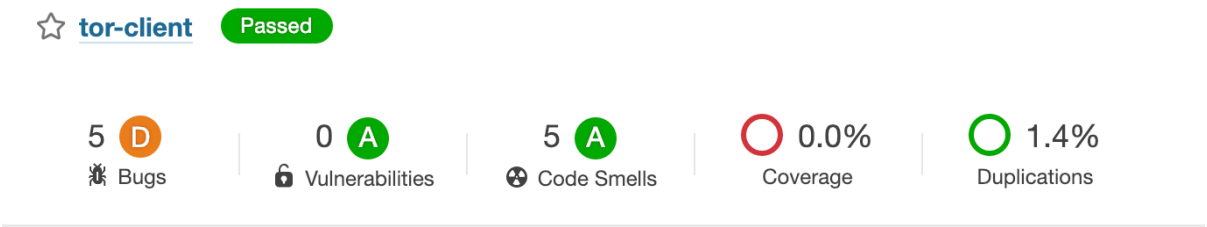
2.1.3 Forbidden eval issue



Figuur 10: Forbidden eval issue

Het prioriteitsniveau is critical omdat er gebruik gemaakt wordt van eval. Dit zorgt ervoor dat er willekeurige code uitgevoerd kan worden in real-time. Ondanks dat het de applicatie niet stopt van runnen is het noodzakelijk dat de issue spoedig wordt verholpen.

2.2 Tweede Front-end analyse



Figuur 11: Overzicht tweede analyse

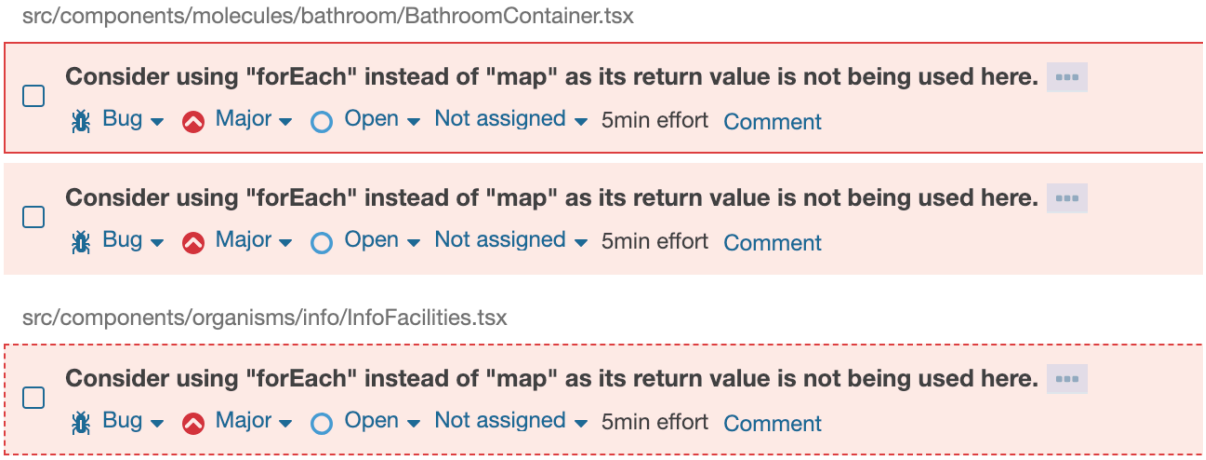
In de tweede iteratie van de analyses zijn de vulnerabilities die voorheen in figuur 8 te zien waren verholpen. Er zijn 5 bugs die aangepakt moeten worden om de kwaliteit van de applicatie te verhogen en het cijfer richting een A te sturen.

2.2.1 Type without members bug

Figuur 12: Type without members bug

Het prioriteitsniveau is critical, simpelweg omdat het een error kan veroorzaken doordat het type automatisch een any of never kan zijn. Dit betekent foute boel doordat het doel van TypeScript om types te geven voorbij geschoten wordt. Want bijvoorbeeld een type combineren met een any betekent dat het type automatisch any is.

2.2.2 Map issue



Figuur 13: Map issue

Er zijn zoals gezien kan worden in figuur 12 drie bugs met eenzelfde beschrijving. Het aanroepen van map op een array vergt een value dat teruggegeven wordt. Ondanks dat het werkt zal het moeten worden omgeschreven naar een forEach loop in plaats van dus een map loop. Het prioriteitsniveau is medium omdat het de output niet verandert.

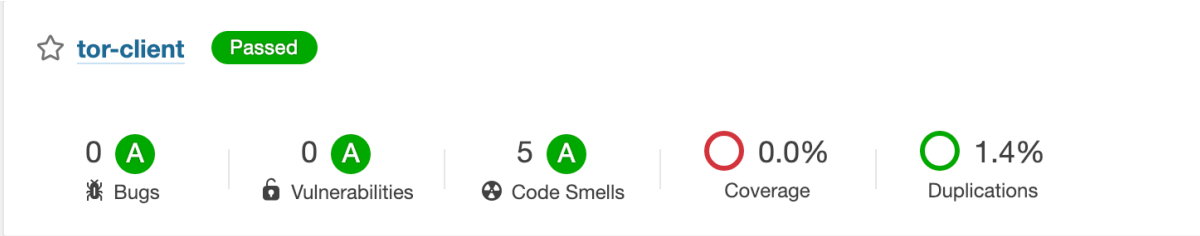
2.2.3 No return issue

Figuur 14: No return issue

De bovenstaande issue was oorspronkelijk als Major beoordeeld en in mijn ogen dus minimaal een prioriteitsniveau van medium. Echter, ben ik het hier niet mee eens omdat het door middel van een simpele refactor kan worden gefixt. Het zal in dit geval undefined worden maar dat maakt niet uit omdat het een void functie is en

dus sowieso al geen return value heeft. Ik neem deze bug aan als een false positive. Ondanks dat, zal deze issue overlegd moeten worden met een collega.

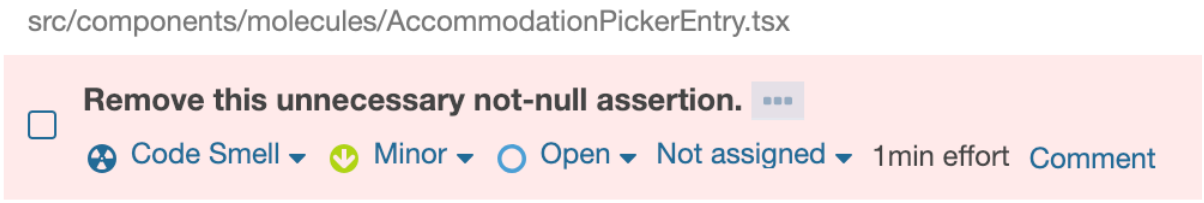
2.3 Derde Front-end analyse



Figuur 15: Overzicht verder iteratie

In de tweede iteratie van de analyses zijn de bugs die voorheen in figuur 8 te zien waren verholpen. Er zijn 5 code smells die aangepakt moeten worden om de kwaliteit van de applicatie te verhogen.

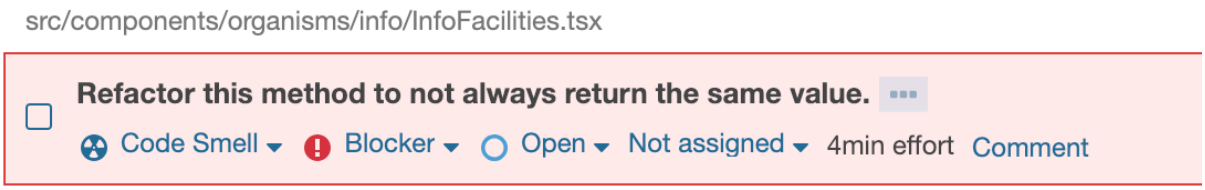
2.3.1 Unnecessary not-null assertion



Figuur 16: Unnecessary not-null assertion

De smell uit figuur 16 is van lage prioriteitsniveau omdat het een not-null assertion betreft. Dit betekent dat door middel van een ! aan de compiler kan worden verteld dat deze niet null kan zijn. Het heeft nu verder geen invloed op de rest van de applicatie en daarom is het niveau dus laag.

2.3.2 Function with if-statement to return same value



Figuur 17: Function always returns same value

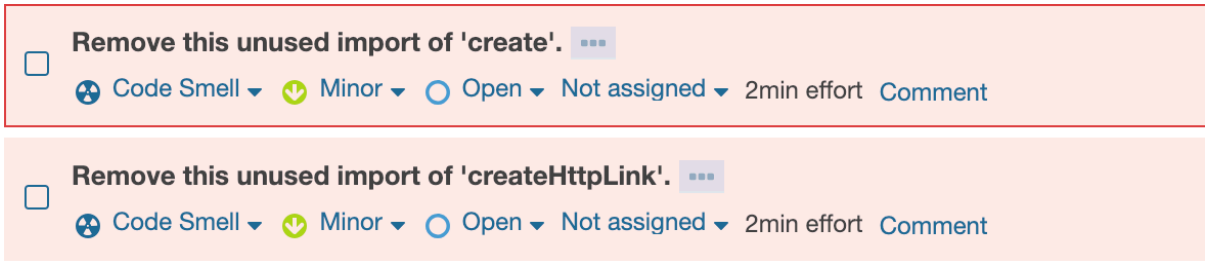
Figuur 17 betreft een functie die een if-statement bevat waarbij blijkbaar beide wegen dezelfde waarde geven. Dit is daarom beoordeeld als een blocker en moet aangepakt worden met een blocker prioriteitsniveau

2.3.3 Await on non-promise

Figuur 18: Await on non-promise issue

Await in JavaScript en TypeScript wordt gebruikt om de return value van een promise synchroon te maken. Echter, in het geval van figuur 18 wordt het gebruikt op een void functie. Na onderzoek blijkt dit ook de issue van 2.2.3 te veroorzaken, en is het dus geen false-positive. De issue heeft prioriteitsniveau critical omdat het twee bug/smelss betreft en moet met urgentie worden opgelost.

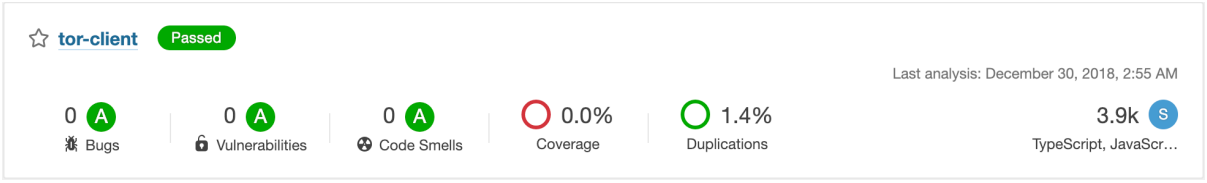
2.3.4 Unused import code smells



Figuur 19: Await on non-promise issue

Zoals in figuur 19 kan worden gezien zijn er twee code smells waarbij er ongebruikte elementen worden geïmporteerd. Dit geldt als een code smell en zal in productie verholpen moeten worden. Maar het heeft verder geen invloed op de werking van de applicatie en verhoogt niet de kwetsbaarheid en daarom is het prioriteitsniveau low gegeven.

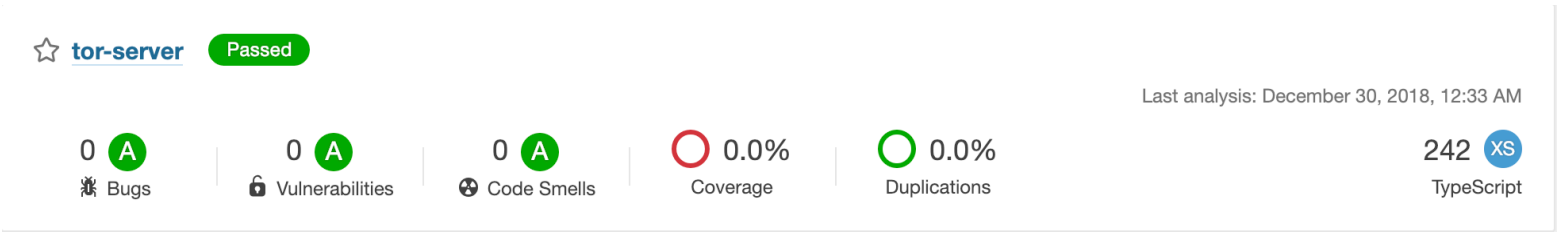
2.4 Vierde Front-end analyse



Figuur 20 geeft weer dat er geen issues meer zijn in het tor-client project, leden-applicatie project. Er kan worden geconcludeerd dat de issues succesvol verholpen zijn dankzij het statisch analyseren met SonarQube.

Figuur 20: Overzicht verder iteratie

2.2 Back-end analyse



Figuur 20: Overzicht back-end

In de tor-server applicatie bevinden zich geen issues. Dit komt mede doordat het een zeer jong project is en zal de aankomende tijd nog flink worden uitgebreid. Verder wordt er gebruik gemaakt van Prisma en dat brengt veel out-of-the-box functies met zich mee.

Referenties

https://docs.sonarqube.org/latest/