# Wordt de software kwaliteit gewaarborgd binnen mijn applicatie door middel van geautomatiseerde testen?

Luc van den Acker

Semester 3 Software Engineering

## Inleiding

In dit document wordt beschreven welke geautomatiseerde testen binnen mijn project toegepast worden, of in de toekomst geïmplementeerd worden om de software kwaliteit te waarborgen. Ook worden de keuzes achter de testen beschreven.

De reden van het testen is om te bewijzen dat de code doet wat het moet doen, maar ook om te bewijzen dat de code van de juiste kwaliteit is. Het doel van geautomatiseerde testen is het waarborgen van de kwaliteit van de code.

## Hoe verbeteren Unit Testen de software kwaliteit van mijn applicatie?

Unittesting is een methode om stukjes code of modules afzonderlijk te testen. Voor iedere unit hoort er dus één of meerdere testen ontwikkeld worden, en meerdere testcases. Het beste is om iedere testcase onafhankelijk te laten functioneren van andere testen. Op die manier heeft het wijzigen van een test geen invloed om de kwaliteit van andere testen. Binnen mijn project bestaan er enkel Unit Testen voor de backend. Front-end unit testen zijn ook mogelijk, en hieronder wordt beschreven hoe deze geïmplementeerd zouden worden, voor zowel front- als backend.

### Front-end Unit Testen

Om in React-JS unit testen te implementeren, wordt Jest toegepast. Jest is een open-source testframework, ontwikkeld door Facebook, dat gebruikt kan worden om verschillende JavaScript-frameworks te testen. Het Jest-framework maakt het simpel om objecten buiten de testscope te ‘mocken’, oftewel het nabootsen van andere afhankelijkheden. Jest biedt een hoop verschillende ‘helpers’ aan, met als doel om heel specifiek onderscheid te maken tussen verschillende type waardes, zoals het verschil tussen ‘null’ en ‘undefined’. Hierdoor biedt Jest de mogelijheid om heel specifiek een unit te testen. Door specifieke units te testen, wordt duidelijk of de units individueel werken, maar wordt niet bewezen of de units ook samen kunnen functioneren.

### Backend Unit Testen

Om in een Java-springboot applicatie te kunnen Unit Testen, wordt Mockito gebruikt, een Java testframework. Mockito maakt het mogelijk om klassen te mocken, wat de mogelijkheid biedt om de functionaliteit te testen, zonder een database aan te roepen. Door een repository te mocken, en aan te geven welke data deze terug moet geven op basis van de meegegeven parameters, wordt getest of data goed doorgegeven wordt vanaf de controller tot aan de repository, en weer terug. Net zoals bij front-end Unit Testen, bewijzen de testen dat de units individueel naar behoren werken, maar niet dat ze samen ook goed functioneren.

## Hoe dragen integratietesten bij aan de software kwaliteit?

Unit testen waarborgen dus de kwaliteit van individuele code units, maar niet de kwaliteit van het geheel. Daarvoor worden integratietesten gebruikt. Integratietesten worden de individuele modules verbonden om als geheel getest te worden. Met een integratietest wordt getest hoe verschillende modules zich gedragen ten opzichte van elkaar. Zo is het bijvoorbeeld mogelijk om met integratietesten te testen wat er gebeurt wanneer data vanuit te backend naar de front-end verstuurd wordt. Hierbij wordt gecontroleerd of data juist opgehaald wordt, en of deze op de juiste manier ontvangen wordt. Hiervoor worden stubs gebruikt, waardoor geen echte data opgehaald hoeft te worden uit de database, maar fake data gebruikt kan worden.

## Hoe verbetert de software kwaliteit door end-to-end testen toe te passen?

Met end-to-end testen(EoE testen) wordt het hele systeem geëvalueerd aan de hand van gespecificeerde eisen. Een EoE test omvat dus de complete applicatieomgeving in een situatie die het gebruik van de applicatie in de echte wereld simuleert. Denk bijvoorbeeld aan interactie met een database, interacties met andere systemen of het gebruik van netwerkcommunicatie.

EoE testen kunnen zowel handmatig als automatisch uitgevoerd worden, maar binnen mijn project ligt de focus op automatische testen. Automatische EoE testen horen de volledige flow van de applicatie te testen, waardoor zowel de front- als de backend meegenomen worden. Bij een EoE test wordt bijvoorbeeld een detailpagina geopend. Op deze pagina hoort natuurlijk wat data weergegeven te worden, die opgehaald hoort te worden uit de database. Voor de opgehaalde data wordt niet gecontroleerd of de verschillende waardes correct zijn, maar enkel of er data opgehaald kan worden. De waardes worden niet gecontroleerd omdat deze natuurlijk kunnen veranderen tijdens de lifespan van de applicatie, wat zou betekenen dat de data verandert. Om een goede EoE test te schrijven, die alle specifieke waardes controleert, zou de test iedere keer bijgewerkt moeten worden wanneer de data bijgewerkt wordt, wat natuurlijk heel veel werk is, en niet veel meerwaarde heeft. Er wordt dus voornamelijk getest of een request vanuit de front-end verwerkt kan worden door de backend, en of er daadwerkelijk een response body binnenkomt op de front-end.

## Hoe beïnvloeden Performance testen de stabiliteit van mijn applicatie?

Performance testen zijn bedoeld voor het testen van de stabiliteit en snelheid van een applicatie. Met stabiliteit wordt het vermogen van de applicatie om bestand te zijn tegen belasting. Met snelheid wordt de reactiesnelheid van de applicatie bedoeld, dus hoelang het duurt voordat een nieuwe pagina geladen wordt, of hoelang het duurt tot een API-call uitgevoerd is. Performance testen controleren dus hoe groot het vermogen van een applicatie is, en wat de reactiesnelheid is.

## Hoe waarborg ik de veiligheid van mijn applicatie met behulp van automatische testen?

Om te testen of een applicatie beveiligd is tegen interne en externe bedreigingen, worden Security Testen geïmplementeerd. Deze testen of de software beveiligd is tegen kwaadaardige programma’s en virussen. Ook testen ze hoe veilig de autorisatie- en authenticatieprocessen van een applicatie zijn. Security testen controleren hoe de software zich gedraagt ten opzichte van aanvallen van hackers, en ten opzichte van kwaadaardige programma’s. Met Security testen is het ook mogelijk om te controleren hoe de software onderhouden wordt voor gegevensbeveiliging. Voor Security testen wordt SonarCloud gebruikt. SonarCloud geeft een duidelijk beeld van de kwetsbaarheden van een applicatie, maar ook van hoe deze onstaan. SonarCloud biedt volledige begeleiding bij het oplossen van codekwetsbaarheden, inclusief voorbeelden van een juiste implementatie.

## Hoe wordt de software kwaliteit in beeld gebracht?

Statische code analyse is een analyse van de software die uitgevoerd wordt zonder dat het project draait. Hiervoor wordt bijvoorbeeld SonarCloud gebruikt. Door SonarCloud aan een CI/CD pipeline toe te voegen, checkt deze de onderhoudbaarheid, betrouwbaarheid en veiligheid van de code iedere keer dat pipeline uitgevoerd wordt. SonarCloud controleert ook de code coverage, oftewel de hoeveelheid code die daadwerkelijk gebruikt wordt, en code duplications. Net zoals bij de Security testen, biedt SonarCloud voor statische code analyse ook hulp aan bij gevonden fouten, zoals uitleg en voorbeelden van hoe het wel moet.

## Wat gebeurt er na de geautomatiseerde testfase?

Nadat alle automatische testen uitgevoerd zijn, en allemaal geslaagd zijn, worden acceptatietesten uitgevoerd. Acceptatietesten zijn handmatige testen, waarbij de gebruiker door alle functionaliteiten heenloopt. De focus ligt hierbij niet alleen op de kwaliteit van de code, maar ook op de flow van de applicatie. Er wordt een lijst met vragen opgesteld waarmee een gebruiker op een logische manier door alle functionaliteiten heen gaat. De gebruiker geeft vervolgens aan of de applicatie goedgekeurd wordt, of dat er nog wat aangepast moet worden. Indien de gebruiker tevreden is over de applicatie, en deze dus ‘geaccepteerd’ wordt, kan de applicatie uitgegeven worden.

## Welke testen zullen geïmplementeerd worden per user story?

**US-04:** Als admin, wil ik nieuwe clubs toe kunnen voegen aan een competitie.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement | Unit test | Integration test | End-to-end test | Acceptatie test |
| FR-01 | X | X |  | X |

### Unit testen

Aangezien Unit Testen goedkoop en relatief eenvoudig zijn, is het een goed idee om deze toe te passen. Voor deze User Story is een Unit Test geschreven, om te testen of het Club-object, dat aan de Controller meegegeven wordt, goed doorgegeven wordt, vanaf de Controller tot aan een mock van de Repository. De echte Repository wordt niet aangeroepen omdat ik geen data wil aanmaken in deze test. Zolang de mock-Repo hetzelfde Club-object ontvangt als degene die vanuit te Controller doorgegeven wordt, zal de mock-Repo dit Club-object terugsturen. Vervolgens wordt getest of het meegegeven Club-object gelijk is aan het teruggegeven Club-object. Indien dit het geval is, slaagt de test, en is aangetoond dat de code binnen deze Unit werkt.

### Intergration test

In de front-end bestaan er twee pagina’s, die samen de functionaliteit van US-04 aanbieden. Op de eerste pagina kiest de admin aan welke competitie de club toegevoegd moet worden, en op de tweede pagina moet de admin de gegevens van de club invoeren. Om te testen of deze samen goed functioneren, heb ik integration testen geschreven. Eerst worden alle competities opgehaald, vervolgens wordt een van deze competities geselecteerd. Daarna wordt de volgende pagina geladen waarop de gegevens van de club ingevuld horen te worden. Vervolgens wordt gekeken of de ingevoerde waardes, inclusief het ID van de competitie, overeenkomen met de waardes in de API-call. Op deze manier test ik beide functionaliteiten voor zowel de front- als de backend.

### End-to-end testen

Voor deze User Story zullen geen end-to-end testen toegepast worden. Iedere keer dat een end-to-end test uitgevoerd zou worden voor deze functionaliteit, zou er een nieuwe club toegevoegd worden aan de database. Om de database zo netjes mogelijk te houden, is het dus niet heel handig om end-to-end testen uit te voeren voor POST-methods, tenzij deze vervolgens direct weer verwijderd wordt in dezelfde test.

## Acceptatietesten

Acceptatietesten zijn voor iedere functionaliteit zinvol, want ze geven een beeld van hoe goed je applicatie valt bij de doelgroep. Daarom is het verstandig om ook voor deze functionaliteit acceptatietesten uit te voeren.

**US-07:** Als gebruiker, wil ik alle competities in kunnen zien.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement | Unit test | Integration test | End-to-end test | Acceptatie test |
| FR-01 | X |  | X | X |

### Unit testen

Net zoals voor US-04 is Unit Testing een goed idee. Hierbij wordt wederom een mock aangemaakt van de Repository. Tegen deze mock wordt gezegd dat deze een lijstje met competities terug moet geven. Vervolgens wordt getAllLeagues aangeroepen in de Controller. Vanaf hier wordt, volgens dezelfde flow als hierboven, de request doorgegeven aan de mock-Repository, en vervolgens teruggestuurd naar de Controller. De test controleert dan of het juiste aantal competities teruggegeven worden, en of de properties, zoals bijvoorbeeld het maximale aantal toegestane clubs, correct zijn.

### Integration testen

Voor deze user story acht ik het niet nodig om integration testen te schrijven. Omdat de end-to-end testen al controleren of er data opgehaald kan worden, heeft het geen meerwaarde om dit met integration testen nog een keer aan te tonen. Deze user story is niet afhankelijk van andere onderdelen van mijn project, en dus heeft het geen meerwaarde om de samenhang met de andere onderdelen te testen.

### End-to-end testen

Zoals hierboven vermeld, worden er end-to-end testen geïmplementeerd voor deze user story. Met deze testen wordt vastgesteld of de flow van de applicatie correct is. Er wordt gekeken of er een lijst met competities opgehaald kan worden uit de database. De testen controleren niet of daadwerkelijk alle competities opgehaald worden, omdat er natuurlijk altijd een nieuwe competitie toegevoegd of verwijderd kan worden, wat betekent dat de test iedere keer bijgewerkt moet worden als de data verandert. De end-to-end testen voor deze user story testen dus de API-call, dus of er data opgehaald kan worden.

### Acceptatietesten

Aangezien de end-to-end testen niet controleren of de ontvangen data correct is, zal dit handmatig gecontroleerd moeten worden. Dit gebeurt door acceptatietesten. Ook hier is niet te controleren of echt alle competities opgehaald worden, maar het is wel mogelijk om te controleren of de opgehaalde data juist weergegeven wordt. De tester zal een tabel zien, met daarin de specifieke gegevens van een aantal competities, en controleert of deze data op een correcte/logische manier weergegeven wordt.

## Conclusie

Om de software kwaliteit van mijn applicatie te waarborgen, worden een heel aantal verschillende soorten testen uitgevoerd. Unit testen controleren of de individuele units in de backend functioneren, waar integratietesten controleren of deze units ook samen kunnen functioneren in zowel de front- als in de backend. Met performance testen toon ik aan dat mijn applicatie snel en stabiel is. Security testen tonen aan dat mijn applicatie bestand is tegen veiligheidsrisico’s. In hoeverre tonen deze testen aan dat mijn applicatie doet wat het moet doen? Via SonarCloud wordt in beeld gebracht in hoeverre mijn testen het project controleren, en wat nog toegevoegd moet worden om de overige code te testen. Wanneer al deze testen uitgevoerd en geslaagd zijn, gaat een tester handmatig controleren of de applicatie doet wat ervan verwacht wordt, en of de applicatie klaar is voor gebruik. De kwaliteit van de software wordt dus eerst getest, en vervolgens wordt de gebruikersvriendelijkheid getest. Zo worden zowel de technische, als de menselijke risico’s van mijn applicatie afgedekt, en is dus de concluderen dat de software kwaliteit van mijn applicatie gewaarborgd wordt.