Test summary report

Door: Mickey Krekels , Matthijs Kotterink

inhoudsopgave

[Tamago Project 3](#_Toc61012829)

[TestMethodes 3](#_Toc61012830)

[FrontEnd 3](#_Toc61012831)

[Angular 3](#_Toc61012832)

[BackEnd 5](#_Toc61012833)

[UserApi 5](#_Toc61012834)

[AnimalApi 5](#_Toc61012835)

[InventoryApi 6](#_Toc61012836)

[ShopApi 6](#_Toc61012837)

[BankApi 7](#_Toc61012838)

[ClockApi 7](#_Toc61012839)

[Shared 8](#_Toc61012840)

[Sonarqube 9](#_Toc61012841)

[Microservices\_Matthijs 9](#_Toc61012842)

[Microservices\_Mickey 9](#_Toc61012843)

[FrontEnd 10](#_Toc61012844)

[TestCoverage 11](#_Toc61012845)

[Microservices\_Matthijs 11](#_Toc61012846)

[Microservices\_Mickey 11](#_Toc61012847)

[Requirements 12](#_Toc61012848)

[Must Have: 12](#_Toc61012849)

[Should have: 12](#_Toc61012850)

[Could Have: 12](#_Toc61012851)

[Would Have: 12](#_Toc61012852)

[Koppelen 12](#_Toc61012853)

[Account aanmaken (Must) 01 13](#_Toc61012854)

[Geld verdienen (Must) 02 13](#_Toc61012855)

[Spullen uit winkel kopen (Must) 03 13](#_Toc61012856)

[gegevens huisdier (Must) 04 13](#_Toc61012857)

[Voeding geven huisdier (Must) 05 14](#_Toc61012858)

[Huisdier sterven (Must) 06 14](#_Toc61012859)

[Spelletjes doen met huisdier (Should) 07 14](#_Toc61012860)

[Huisdier koppelen aan het account (Could) 08 15](#_Toc61012861)

[groeien van het huisdier (Could) 09 15](#_Toc61012862)

[Meerdere spellen spelen (Would) 10 16](#_Toc61012863)

[balans opwaarderen (Would) 11 16](#_Toc61012864)

[naam geven huisdier (Would) 12 16](#_Toc61012865)

# Tamago Project

Ons project is een microservice project met meerdere api’s die communiceren. We hebben een tamagotchi achtig project waarvan de backend extern is dus een aantal tests zijn essentieel.

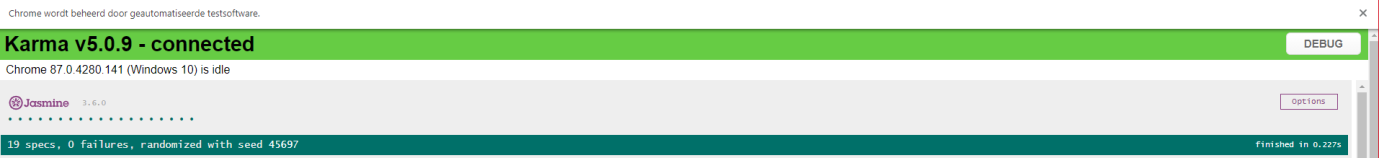
## TestMethodes

De testmethodes die we hebben gebruikt verschillen per aspect van ons project maar sommige tools hebben we veel gebruikt, voornamelijk MsTest, Jasmine, Protractor en Sonarqube zijn van toepassing geweest. De eerste drie zijn methodes om te testen en Sonarqube hebben we veel gebruikt om naar code smells, fouten en security problemen te kijken.

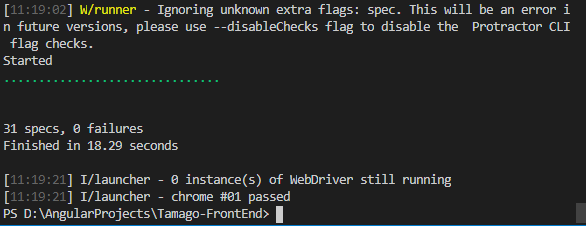
# FrontEnd

## Angular

In de frontend hebben we ‘UI’ tests en ‘End to end’ tests gebruikt om de applicatie te testen. Hier hebben we de volgende resultaten op behaalt :



Dit is het resultaat van de UI tests waarvan: **19/19 succesvol zijn**

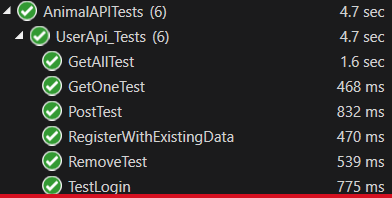


En dit zijn de resultaten voor de ‘End-to-end’ tests waarvan: **31/31 geslaagd zijn**

# BackEnd

## UserApi

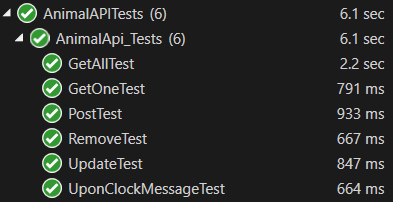
Om de gebruikersdata op te slaan hebben we een user api. Om deze te testen hebben we Unit tests uitgevoerd die zowel de connectie tussen activemq test als de api calls, hierbij hebben we de volgede resultaten behaalt:



Hier hebben we **6/6** succesvol.

## AnimalApi

Deze api is voor de data van het dier, hier hebben we ook voornamelijk unit tests gemaakt en het volgende behaalt:



Hier hebben we **6/6** succesvol.

## InventoryApi

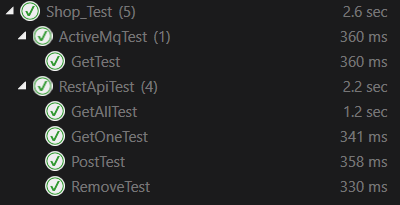
Deze api is voor de data van het Inventory van de user, we test hier vervolgens de CRUD functies . Hier kregen we deze output:



Hier hebben we **6/6** succesvol waarvan 1 voor de activemq is en 5 voor de restapi.Voor deze api is.

## ShopApi

De Shop API bevat de data van de items die koopbaar zijn in de game , we test hier vervolgens de CRUD functies . Hier kregen we deze output:



Hier hebben we **5/5** succesvol waarvan 1 voor de activemq is en 4 voor de restapi.

## BankApi

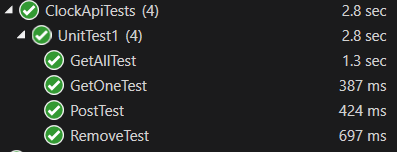
De Bank API bevat de data van de currency van de users, we test hier vervolgens de CRUD functies . Hier kregen we deze output:



Hier hebben we **6/6** succesvol waarvan 1 voor de activemq is en 5 voor de restapi.

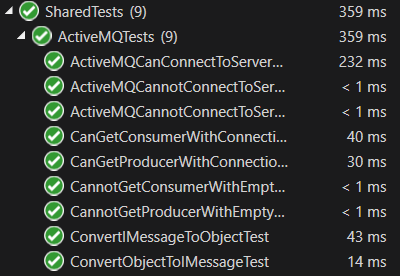
## ClockApi

Deze api is iets anders dan de rest, deze maakt namelijk gebruik van websockets om uit zichzelf data te kunnen versturen. Hierbij hebben we dus ook voor de websockets tests moeten maken. Dit was het resultaat:



Hier hebben we **4/4** succesvol

## Shared

De Shared library bevat componenten die we in elke API gebruiken , Hierin word voornamelijk de ActiveMQ getest. we kregen vervolgens deze output:

Hier hebben we **9/9** succesvol

# Sonarqube

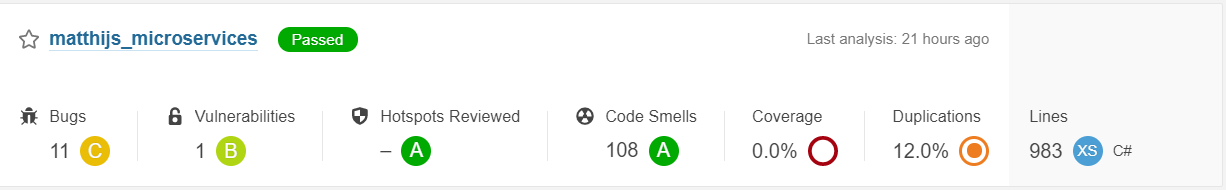
Zoals eerder beschreven hebben we veel gebruik gemaakt van sonarqube om kleine fouten te ontdekken, om deze reden willen we ook graag deze output laten zien, zodat u een beeld krijgt over de algemene codequaliteit.

## Microservices\_Matthijs

Deze microservices zijn degene die door Matthijs geschreven zijn. Ze staan allemaal in 1 project omdat dit gemakkelijker was dus de volgende dingen gingen hierbij door de sonarqube heen:

* ShopApi
* BankApi
* InventoryApi
* Gezamenlijke ‘shared’ folder

Hier is het resultaat:



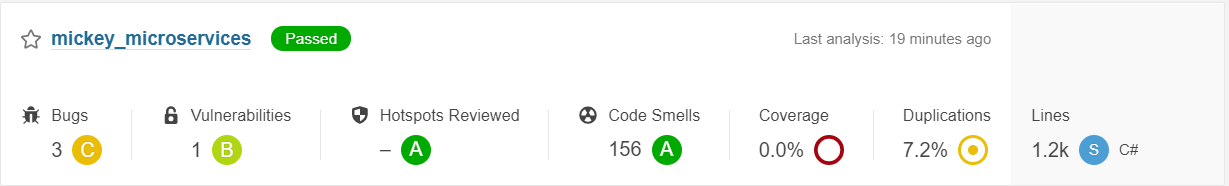
Zoals je meteen al opgemerkt is staat er 12% duplication, dit komt door de rest api die over het algemeen hetzelfde is bij elk project.

## Microservices\_Mickey

Dit zijn de microservices die door Mickey zijn geschreven. Ook hier staat alles weer in 1 project en bevat het het volgende:

* AnimalApi
* UserApi
* ClockApi
* Gateway
* Gezamelijke ‘shared’ folder

Waarvan dit het resultaat is:

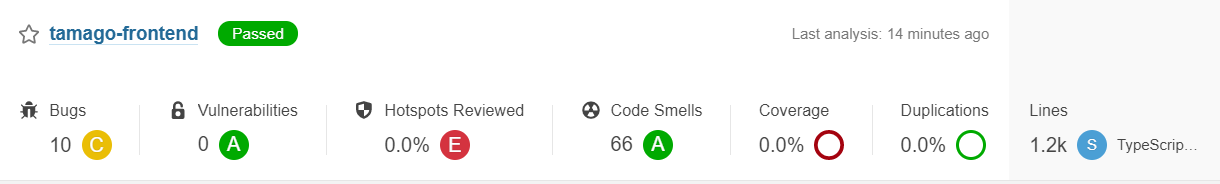


Ook hier zit weer wat duplication door het aantal projecten maar verder ziet het er goed uit.

## FrontEnd

Als laatste hebben we de frontend die met Angular is gemaakt, ook deze hebben we door sonar qube gehaalt om te zien of we nog dingen konder verbeteren.

Dit kwam eruit:



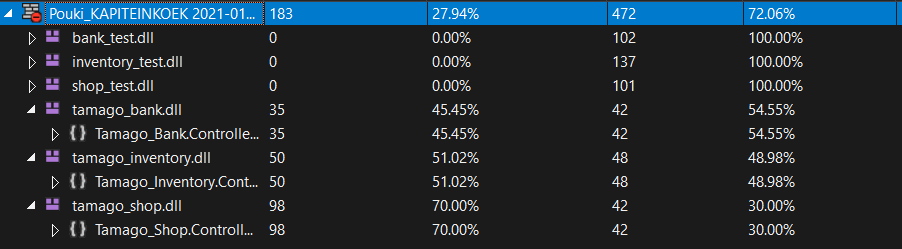
Zoals je ziet zit hier geen duplication in en verder qua fouten valt het ook heel erg mee.

# TestCoverage

Het punt testcoverage is belangrijk om te laten zien hoeveel procent we hebben getest en dus hoe zeker we kunnen zijn dat het project werkt.

## Microservices\_Matthijs

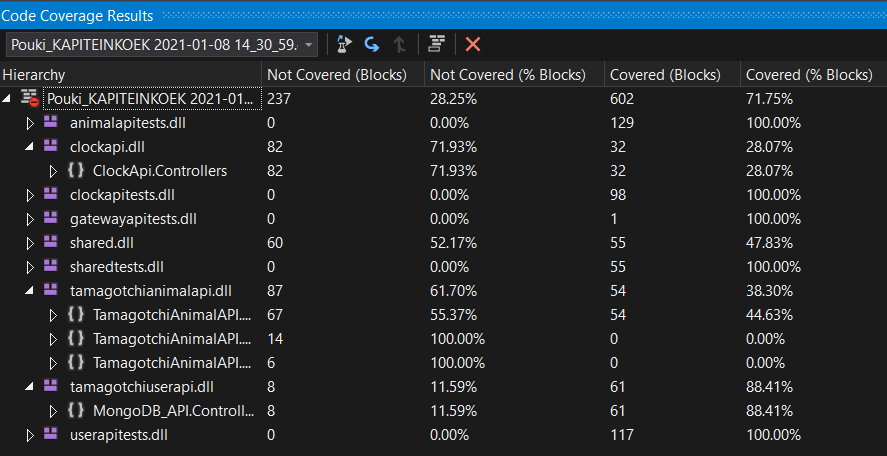
Zoals voorheen verteld heeft Matthijs meerde projecten in een solution zitten voor gemak, om de code coverage eruit te krijgen hebben we visual studio zelf gebruikt, hierbij heeft Matthijs op zijn project 72.92% coverage behaalt, dit is niet 100% omdat er een aantal functies bijzitten die private zijn en dus op andere manieren getest zijn. Dit houdt bijvoorbeeld de Activemq functies in die niet aangeroepen mogen worden via de frontend omdat alleen de backend hiermee praat.



## Microservices\_Mickey

We hebben voor flexibel opzet gekozen bij maken van onze C# solutions. Namelijk door middel van meerde project hebben binnen de zelfde solution. Hierdoor kunnen we makkelijk componeten die we in ander projecten ook gebruiken recyclen. hierbij heeft Mickey op zijn project 71.75% coverage behaalt. Dit is echter niet 100% omdat er een aantal onderdelen zijn die niet test baar zijn

(bijv. de startup of de program.cs).



# Requirements

Als laatste willen we de voorheen beschreven tests koppelen aan de requirements zodat we kunnen laten zien wat we goed werkend hebben kunnen krijgen. Eerst laten we een plaatje zien met alle requirements die we hadden om deze vervolgens te koppelen aan de tests en onderdelen van het project.

We hadden de volgende requirements volgens de MoSCoW methode opgezet:

### Must Have:

**FR-01** Een account moet aangemaakt kunnen worden

**\*FR-03** De gebruiker moet geld kunnen verdienen

**FR-04** Een Account moet spullen uit de winkel kunnen kopen

**FR-05** De details van het huisdier moeten altijd zichtbaar zijn voor een account (blijheid, voeding, energie)

**\*FR-06** De gebruiker moet zijn huisdier voeding kunnen geven

**\*FR-07** Het huisdier moet kunnen sterven na een te lange tijd van inactiviteit

### Should have:

**\*FR-09**De gebruiker moet activiteiten kunnen doen om zijn huisdier blij te kunnen maken (spelletjes)

### Could Have:

**FR-10** Een account moet een huisdier kunnen koppelen aan zijn account

**\*FR-11** De gebruiker zijn huisdier moet kunnen groeien

### Would Have:

**\*FR-12** De gebruiker moet meerdere spellen kunnen spelen

**FR-13** Een account moet zijn balans kunnen opwaarderen via paypal

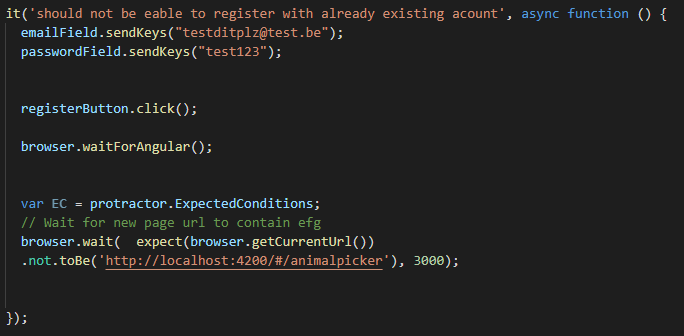
**FR-14** Een account kan zijn huisdier een naam geven

## Koppelen

Om de requirements aan te kunnen tonen via tests hebben we voornamelijk de ‘End to end’ tests geschreven. Omdat deze werken in de frontend en onze requirements 9/10 keer iets met de frontend te maken hebben.

### Account aanmaken (Must) 01

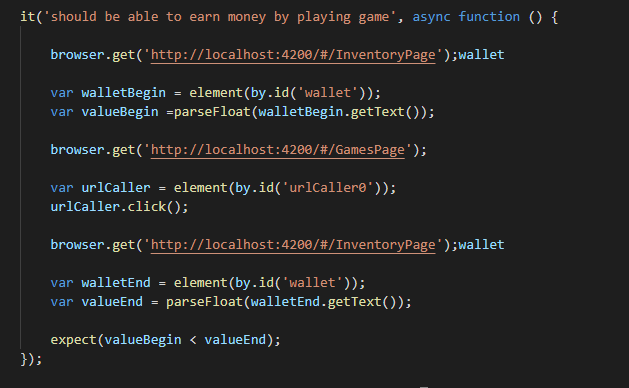
Het account aanmaken hebben we een protractor test, dit gaat samen met de user backend waar een account aangemaakt wordt.

Protractor test:

### Geld verdienen (Must) 02

Voor het geld verdienen hebben we een simpele test gemaakt die naar een spel gaat en terug waarop er geld gestord zou moeten worden.

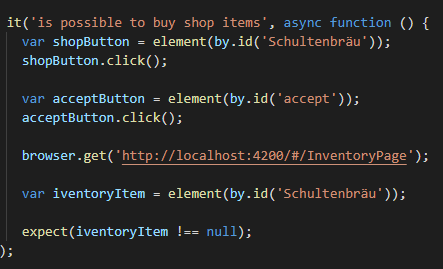
Protractor test:



### Spullen uit winkel kopen (Must) 03

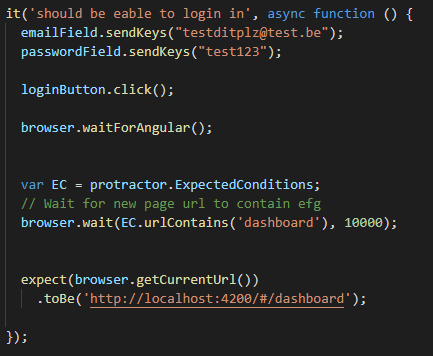
Voor het kopen van spullen uit de winkel kopen we via protractor een ‘Schultenbrau’ en kijken vervolgens of deze in de inventaris zit.

Protractor test:



### gegevens huisdier (Must) 04

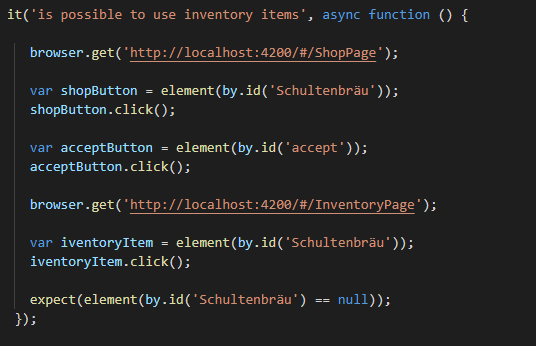
Om de gegevens van een huisdier te kunnen bekijken met je in kunnen loggen en de gegevens moeten vervolgens ingeladen worden. Hierbij hebben we ook een frontend test geschreven omdat we dan konden zien of je daadwerkelijk het dier ook te zien krijgt.



### Voeding geven huisdier (Must) 05

Voor het voeding geven van het huisdier controleren we of het voedsel uit de invetaris is gegaan. Zo weten we dat deze geconsumeerd is, dit gebeurt namelijk pas als laatste als alles is goedgekeurd.

Protractor test:



### Huisdier sterven (Must) 06

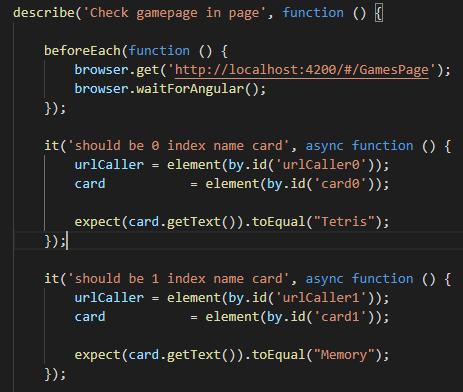
Voor het sterven van het huisdier hebben we een filmpje gemaakt. Hierin laten we aan de hand van een handmatige test zien dat het huisdier kan sterven mits alle waardes op 0 staan.

Youtube link:

<https://www.youtube.com/watch?v=n-qXaa5RjW4&feature=youtu.be>

### Spelletjes doen met huisdier (Should) 07

Voor de activiteiten die je kunt doen met het huisdier hoeft er alleen maar via de frontend gecommuniceert te worden. Om deze reden hebben we ook hier een protractor test gemaakt om dit gedeelte aan te tonen.



### Huisdier koppelen aan het account (Could) 08

Als de gebruiker een account aanmaakt moet hij ook de keuze hebben uit een aantal huisdieren. Hiervoor hebben we een kleine pagina waar de gebruiker de naam en het dier kan kiezen.

De volgende test hebben we hiervoor geschreven:



### groeien van het huisdier (Could) 09

Het groeien van het huisdier gebeurt aan de hand van een level. Dit hebben we ook via de handmatige tests getest en we hebben al meerdere huisdieren waarvan het niveau omhoog is.

### Meerdere spellen spelen (Would) 10

Het aanklikken van een spel was al mogelijk met de vorige test die beschreven was, hier hebben we vervolgens meer spellen toegevoegd en dit werkte ook succesvol.

### balans opwaarderen (Would) 11

De paypal betaal methodes hebben we niet kunnen toevoegen

### naam geven huisdier (Would) 12

Dit hebben we al gedaan bij het kiezen van een huisdier en ook die test was succesvol.