# Eindverslag semester 3

Mickey en Matthijs

Inhoud

[Eindverslag semester 3 1](#_Toc61870207)

[Project 3](#_Toc61870208)

[Requirements 3](#_Toc61870209)

[*Must Have*: 3](#_Toc61870210)

[*Should have*: 3](#_Toc61870211)

[*Could Have*: 3](#_Toc61870212)

[*Would Have*: 3](#_Toc61870213)

[Design verslag 4](#_Toc61870214)

[Opzetten externe werkzaamheden 4](#_Toc61870215)

[Trello 4](#_Toc61870216)

[Github 4](#_Toc61870217)

[Devops 4](#_Toc61870218)

[Inhoud van het project 5](#_Toc61870219)

[Microservices: 5](#_Toc61870220)

[UserApi 5](#_Toc61870221)

[AnimalApi 5](#_Toc61870222)

[ClockApi 5](#_Toc61870223)

[BankApi 5](#_Toc61870224)

[InventoryApi 5](#_Toc61870225)

[ShopApi 5](#_Toc61870226)

[Frontend 5](#_Toc61870227)

[Externe services 5](#_Toc61870228)

[Gateway 6](#_Toc61870229)

[MessageBroker 6](#_Toc61870230)

[Discovery service 6](#_Toc61870231)

[Testen 6](#_Toc61870232)

[FrontEnd 6](#_Toc61870233)

[Ui testen 6](#_Toc61870234)

[End-to-End testen 6](#_Toc61870235)

[BackEnd 6](#_Toc61870236)

[Unit testen 6](#_Toc61870237)

[Integration testen 6](#_Toc61870238)

# Project

Ons project was het maken van een Tamagotchi achtig spel. We wilden een spel maken waarbij je een huisdier had die je kon voeden en upgraden door middel van spellen spelen en geld verdienen.

Ons doel was om onze requirements op een goede manier uit te kunnen werken zodat we aan het eind van dit project een mooi resultaat hadden gehaalt. In dit verslag laten we zien wat we hebben gedaan en waar we eventueel tegen aanliepen.

## Requirements

Allereerst begonnen we met het opstellen van een analyse document met de requirements, vervolgens hebben we daar samen use en test cases voor geschreven zodat het duidelijk was voor de docent wat ons plan van aanpak was. Hieronder een kleine samenvatting van de requirements die we hebben gemaakt.

### 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

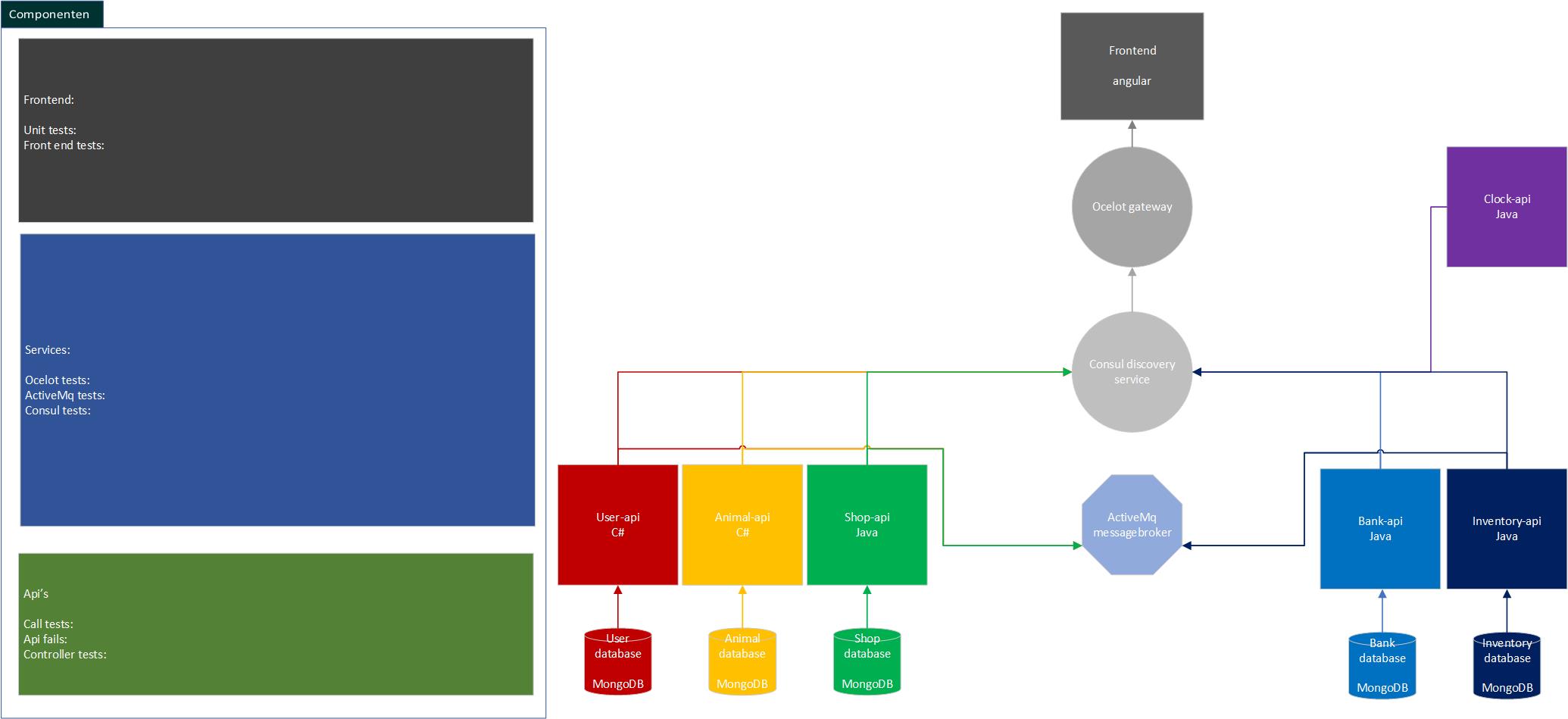
**FR-13** Een account moet zijn balans kunnen opwaarderen via paypal

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

## Design verslag

Nadat we de requirements hadden opgesteld hielden we ons steeds meer bezig met het maken van een aantal schetsen die ons zouden helpen als we dalijk bezig zouden zijn met de frontend. Hierin hebben we een aantal schetsen gemaakt waaronder: wireframes en C4 schetsen.

Hieronder een klein voorbeeld van een zo’n schets die we hebben gemaakt.



## Opzetten externe werkzaamheden

### Trello

Omdat we samen een project deden in plaats van individueel hebben we een trello bord opgezet zodat we beter de taken zouden kunnen verdelen en bij houden wat we nog moesten doen. Uiteindelijk hebben we deze niet zo vaak gebruikt als dat we dachten maar wel genoeg om er profijt uit te halen.

### Github

Ook vanwege de samenwerking maar voornamelijk voor het bijhouden van het project hebben we ervoor gekozen om een github repository aan te maken, dit hebben we ontzettend veel gebruikt om projecten op te houden voor veiligheid en voor de CICD pipeline.

### Devops

Voor de CICD pipeline hebben we gebruik gemaakt van Devops, hiermee konden we direct naar azure pushen en hadden we veel controle over de pipeline zelf.

Later kwamen we er ook achter dat devops een hele project omgeving had met zijn eigen project borden, deze hebben we helaas niet gebruikt.

# Inhoud van het project

Ons project bestaat uit een frontend, een aantal microservices, en een aantal extra’s die we gebruiken voor bijvoorbeeld communicatie.

We hebben een aantal microservices die elk zijn eigen taak kan voldoen.

## Microservices:

### UserApi

De userApi is gemaakt door Mickey en zorgt voor de opslag en de aanmaak van nieuwe gebruikers.

### AnimalApi

De animalApi is gemaakt door Mickey en houdt de data van je huisdier bij.

### ClockApi

De clockApi is gemaakt door Mickey en zorgt ervoor dat er live data van het dier afgehaalt wordt (Blijheid, honger, energie).

### BankApi

De bankApi is gemaakt door Matthijs en zorgt ervoor dat al het geld op de juiste manier wordt afgeschreven.

### InventoryApi

De inventoryApi is door Matthijs gemaakt en zorgt voor het bijhouden van de inventaris.

### ShopApi

De shopApi is gemaakt door Matthijs en zorgt voor de koop en verkoop van producten.

## Frontend

De frontend hebben we voornamelijk samen gedaan Matthijs is begonnen met een structuur te maken zowel als een kleine opzet met css etc. Rond deze tijd was Mickey vooral bezig met het maken van zijn Api’s en een opzet creeren hiervoor.

Hierna zijn de rollen omgedraait. Mickey ging toen verder met het maken van de frontEnd terwijl Matthijs begon met het maken van zijn Api’s.

## Externe services

Buiten de front en backend hebben we veel gebruik gemaakt van externe applicaties om onze app beter te maken in het algemeen.

* We hebben een Gateway toegevoegd die ervoor zorgt dat de links naar de api’s niet direct gebruikt worden.
* We hebben ook gebruik gemaakt van een Messagebroker, die zorgt ervoor dat de berichten die api’s sturen naar elkaar niet verloren gaan.
* Ook hebben we een Consul discovery server gemaakt die ervoor zorgt dat de linkjes van de api’s goed naar de gateway worden gecommuniceert.

## Gateway

Voor de gateway hebben we gebruik gemaakt van Ocelot. Dit is een gateway die ons goed leek omdat deze zowel support had voor C# als voor Java waar Matthijs aan het begin voornamelijk mee bezig was. Mickey heeft deze gateway opgezet en ervoor gezorgd dat we dit altijd konden gebruiken.

## MessageBroker

Voor de messagebroker hebben we gebruik gemaakt van ActiveMq. Dit was een goede keus omdat dit zowel voor java als voor c# werkte, achteraf gezien hadden we toch beter rabbitmq kunnen gebruiken maar dat is voor een volgende keer. Matthijs heeft hier voornamelijk aan gewerkt met het opzetten van het project en met het uitzoeken hoe je dit linked. Mickey heeft hier uiteindelijk een C# library voor geschreven die ervoor zorgt dat het gemakkelijk te gebruiken is binnen onze api’s.

## Discovery service

De discovery service hebben we opgezet om ervoor te zorgen dat we niet constant de links van de gateway hoefde te updaten. Matthijs heeft deze opgezet op zijn lokale server terwijl Mickey druk bezig was met het linken van zijn projecten. Op deze manier kregen we het het snelste en gemakkelijkste aan de praat.

# Testen

Nadat we alle onderdelen hadden en een basis hadden gelegd konden we beginnen met het schrijven van tests. Voor meer informatie over de tests zelf kunt u naar ons testrapport, maar hier wordt de basis uitgelegd.

## FrontEnd

### Ui testen

Voor de User interface testen hebben we gebruik gemaakt van Karma, dit zat ingebouwd in angular zelf en Mickey heeft ontzettend veel gedaan om hiervoor de juiste testresultaten te krijgen.

### End-to-End testen

Om de connectie met de backend te kunnen testen hebben we op de frontend gebruik gemaakt van protractor. Hier is Mickey veel mee bezig geweest om dit werkend te krijgen en uiteindelijk hebben we alles kunnen testen wat we wilde testen

## BackEnd

### Unit testen

Om de api’s te kunnen testen hebben we gebruik gemaakt van microsoft’s eigen testplatform. Hiermee konden we gemakkelijk unit tests uitvoeren en de code coverage bekijken. We zijn hier beide erg hard mee bezig geweest.

### Integration testen

Om dingen zoals Activemq en consul te kunnen testen hebben we een aantal integration tests geschreven. Hier is Matthijs veel mee bezig geweest en samen hebben we uiteindelijk beide kunnen testen.