Piotr Tadrala

AP2O-S201

Personal Development Report

27-05-2022

2022

Inhoud

[**Functioneel ontwerp** 2](#_Toc105529729)

[**Technisch ontwerp** 2](#_Toc105529730)

[**Implementation** 2](#_Toc105529731)

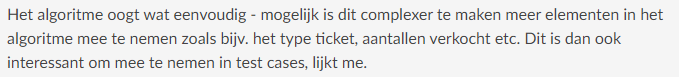
[**Algorithmics** 3](#_Toc105529732)

[**Quality** 3](#_Toc105529733)

[**Professionalism** 3](#_Toc105529734)

# **Functioneel ontwerp**

Tijdens het maken van mijn functioneel ontwerp heb ik geleerd om de applicatie goed te omschrijven, zo zijn de mogelijkheiden en de flow duidelijk voor de klant. Dit heb ik gedaan door een conceptueel model te maken met alle concepten van het systeem en relaties tussen die concepten. Vervolgens heb ik een use case diagram met de belangrijkste methodes toegevoegd aan het functioneel ontwerp. Methodes uit het use case diagram heb ik vervolgens in de user stories omschreven met de correcte en exceptional flow. Vervolgens heb ik een algoritme bedacht. In eerste instantie had ik moeite om het principe van een uitbreidbare algoritme te begrijpen.



Na feedback bleek mijn algoritme niet complex genoeg te zijn. Maar na een overleg met John is het mij gelukt om er een te bedenken die goed in het systeem paste. Als laatste ging ik aan de slag met een acceptatietest. Hierin heb ik mijn teststrategie toegelicht over het testen van de use cases en het algoritme.

**Self Assessment: Proficient**

# **Technisch ontwerp**

Tijdens het opstellen van mijn technisch ontwerp heb ik geleerd hoe je een use case diagram kan omzetten naar een klassendiagram. Hierin staan alle classes en interfaces met methodes. Deze kunnen verwerkt worden door een developer om OOP structuur van een applicatie te programmeren. Ook heb ik geleerd hoe je bepaalde use cases kunt testen dankzij unit-tests en mocking.

Toen ik mijn eerste concept van het klassendiagram had ingeleverd kreeg ik de volgende feedback:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Structuur leek in eerste instantie goed te zijn maar ik miste methodes die mijn use cases ondersteunde. Na dat ik die heb toegevoegd ben ik aan de slag gegaan met de architectuur. Daar kreeg ik de volgende feedback op:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

DataModel interface die geïmplementeerd wordt door alle data models had alleen get() als functie, die heb ik na de feedback aangepast.

**Self Assessment: Proficient**

# **Implementation**

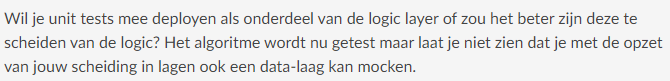
Na dat ik klaar was met mijn technisch ontwerp ging ik aan de slag met de implementatie. Wat ik geleerd heb is het maken van een applicatie die heel goed maintable is dankzij een goede lagen architectuur. Wat ook nieuw was voor mij was om verschillende lagen in het applicatie ‘loosely coupled’ te houden voor nog betere maintability.

Feedback op mijn implementatie was het volgende:



Wat inderdaad niet klopte was een class die console read & writes uitvoerde. Dit heb ik verplaatst naar de interface layer.

Unit tests bleken ook niet helemaal correct te zijn aangezien ze in mijn implementatie een onderdeel van de logica layer waren.



Ook heb ik tests toegevoegd die gebruik maken van layer mocking.

**Self Assessment: Beginning**

Om mijn rating naar proficient te verhogen moet ik me meer oriënteren in de scheiding van lagen binnen de applicatie.

# **Algorithmics**

Een relatief moeilijke onderdeel voor mijn was het algoritme. Het was namelijk moeilijk voor mij om het concept van een uitbreidbare algoritme te begrijpen. Gelukkig naar overleg feedback, en bekijken van verschillende voorbeelden is het mij gelukt om een algoritme te bedenken. Het algrotime past goed binnen de applicatie en is ook uitbreidbaar dankzij het gebruik maken van interfaces. Wat ik alleen nog miste was uitleg van mijn algoritme om die duidelijk te maken voor de klant

Afbeelding met tekst

Automatisch gegenereerde beschrijving

**Self Assessment: Proficient**

# **Quality**

Tijdens het programmeren heb ik voor versie beheer GitHub gebruikt. Hierdoor kan ik altijd terugkijken naar verschillende versies van mijn code en heb ik een back-up. Nadat ik met bepaalde use cases klaar was, heb ik gebruik gemaakt van unit tests om stukjes code individueel te testen. Wanneer er een unit test fout ging heb ik gebruik gemaakt van mocking om de range binnen welke class het fout gaat te verkleinen.

**Self Assessment: Begining**

Om mijn rating naar proficient te verhogen moet ik meer kritisch zijn op wat ik inlever. Er waren namelijk momenten dat ik feedback kreeg over iets wat niet goed wat omdat ik er niet goed genoeg naar had gekeken. Bijvoorbeeld mijn technisch ontwerp die methodes miste.

# **Professionalism**

Tijdens het hele proces heb ik mijn TO, FO en code bij verschillende docenten uitgelegd en overlegd wat de aannames en overweging waren achter mijn keuzes. Wanneer een keuze niet helemaal logisch was heb ik geprobeerd om die te verbeteren totdat het uiteindelijk wel correct was.

**Self Assessment: Beginning**

Om mijn rating naar proficient te verhogen moet ik vooral mijn tijd beter inplannen. Ik kwam namelijk laat achter dan ik naar mijn gevoel achterliep.