




SOFTWARE DESIGN DOCUMENT (SDD) – CASE STUDY: FINCH – OOAD 2022/2023



Martijn Gerritsen (1644170) en Dani Botland (617442)
DOCENT: MARCO ENGELBART Datum: 30-03-2023

Inhoud

1. Introduction.....	2
1.1 Algemene beschrijving.....	2
1.2 Doel van het document.....	2
1.3 Definities, acroniemen en afkortingen.....	2
2. Gedetailleerde ontwerpbeschrijving.....	3
2.1 Klassendiagram.....	3
2.2 Uitleg per ontwerpprincipe.....	3
2.3 Sequence diagrammen.....	4
2.4 Gemaakte ontwerpbeslissingen voor subsysteem.....	6

1. Introduction

1.1 Algemene beschrijving

In dit document wordt het SDD beschreven dat bij de case study 'Finch' hoort. Dit is een beroepsproduct van OOSE-OOAD. Het bedrijf PeeWee Games wil een kennisquiz-applicatie Finch ontwikkelen die beschikbaar komt op de meest gangbare devices. Dit is een spel waarbij vragenlijsten beantwoord worden door gebruikers.

1.2 Doel van het document

Het doel van dit document is om de technische vertaling te weergeven/beschrijven van de uitgewerkte usecases uit het Software Requirement Specification (SRS). Dit document zal het ontwerp van Finch beschrijven aan de hand van de casusopdracht.

1.3 Definities, acroniemen en afkortingen

Term	Beschrijving
SDD	Software Design Document

2. Gedetailleerde ontwerpbeschrijving

2.1 Klassendiagram

Hieronder is een klassendiagram te zien van Finch. Dit diagram is gebaseerd uit het Software Requirements Specification (SRS). Verderop zullen toelichtingen te vinden zijn voor bepaalde keuzes.

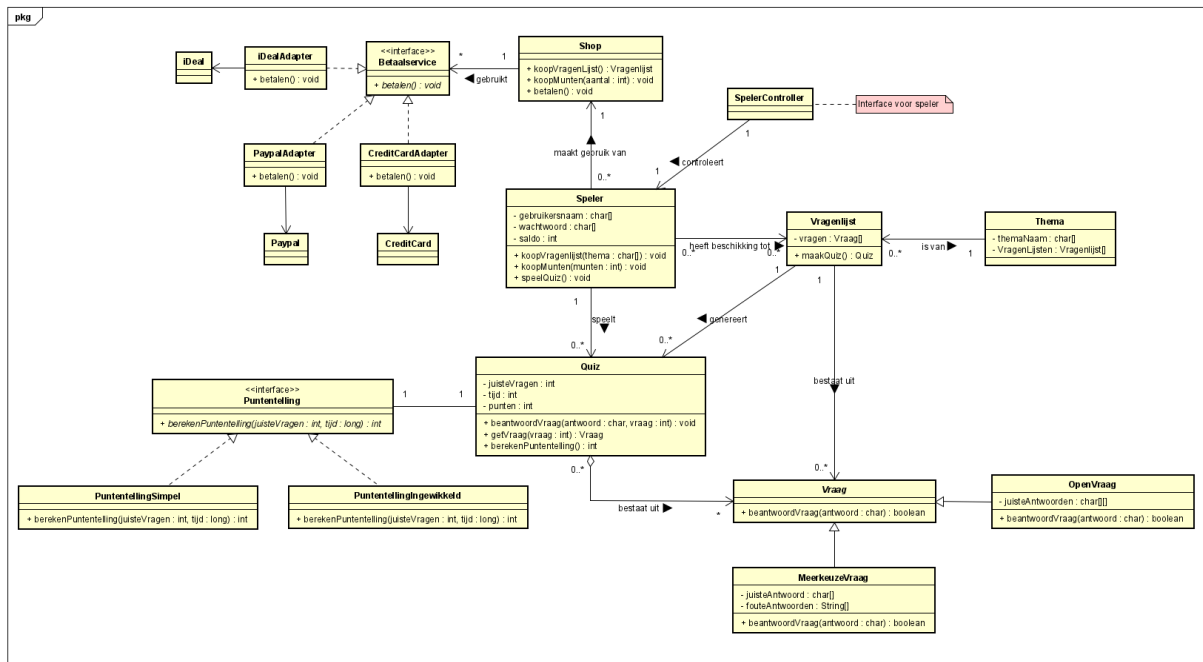


Figure 1: Design Class Diagram

2.2 Uitleg per ontwerpprincipie

Hieronder is een beschrijving met entiteiten/principes die afwijken van het domeinmodel in het Software Requirements Specification (SRS).

Shop / Betaalservice / Adapters

Om de Finch-shop te kunnen realiseren hebben wij gekozen om een Adapter-pattern toe te passen. Dit is handig, omdat twee klassen met incompatibele interfaces met elkaar kunnen communiceren met behulp van een adapter-klasse. Hierdoor wordt de herbruikbaarheid bevorderd.

Een alternatief voor de betaalservice was het strategy-pattern geweest. Via een interface 'Betaalservice' hadden dan meerdere betaalservice-strategieën toegevoegd kunnen worden. Wij hebben gekozen voor het adapter-pattern, omdat verschillende betaalservices buiten onze macht liggen en dat interface niet compatibel is met ons systeem. Ook is het strategy pattern al te vinden bij de puntentelling, dit zorgt voor variatie.

Puntentelling

In de casus is er vereist dat er een strategy pattern gebruikt moest worden voor het flexibiliseren van de puntentelling. Dit hebben wij dan ook toegepast in Finch. Via dit patroon kan er een gemakkelijk een extra puntentellingsysteem toegevoegd worden, zonder dat 'Puntentelling' aangepast moet worden, dit

wordt ook wel het open/closed principe genoemd (open voor uitbreiding, gesloten voor verandering).

SpelerController

SpelerController is de 'hoofdklasse'. Hierin wordt een speler aangemaakt en wordt de quiz gespeeld (speler.speelQuiz). Dit wordt gekoppeld aan het Single Responsibility pattern. SpelerController doet alleen wat hij hoort te doen.

2.3 Sequence diagrammen

Hieronder zijn sequence diagrammen te vinden. Het laat de belangrijkste systeemoperaties zien van de uitgewerkte use cases in het Software Requirements Specification (SRS).

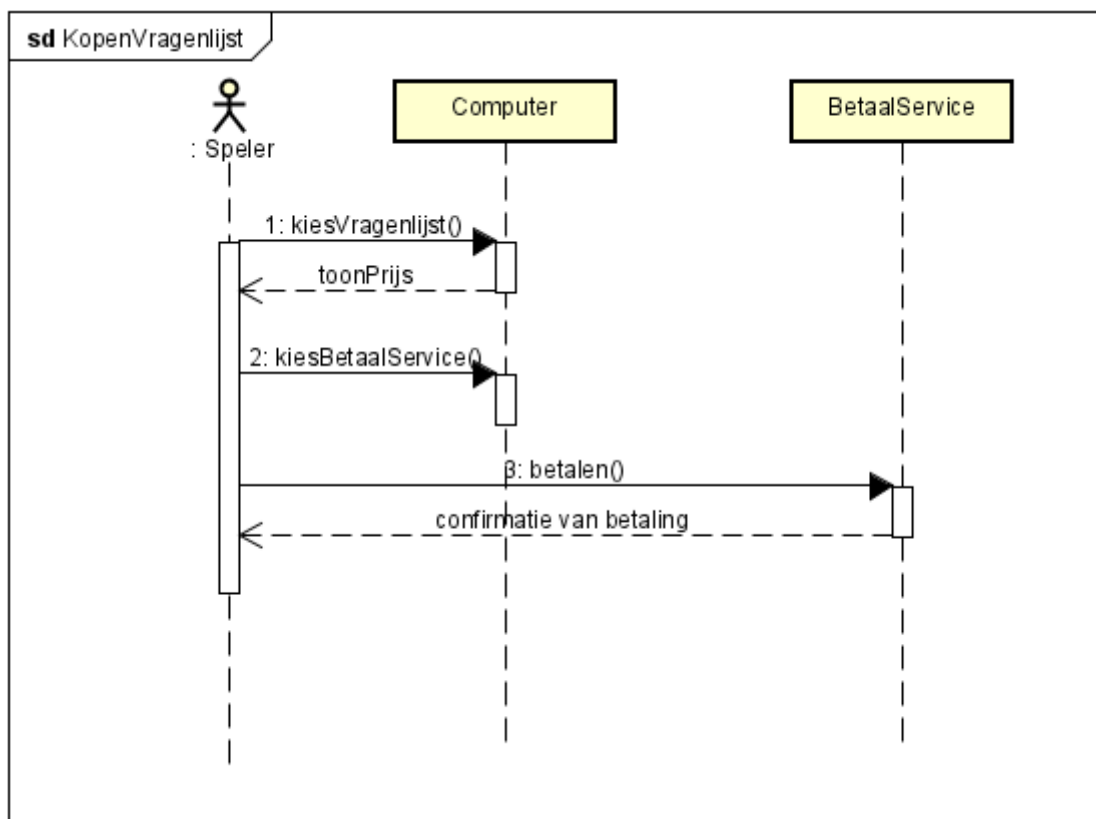


Figure 2: System Sequence Diagram kopen Vragenlijst(kopen game assets)

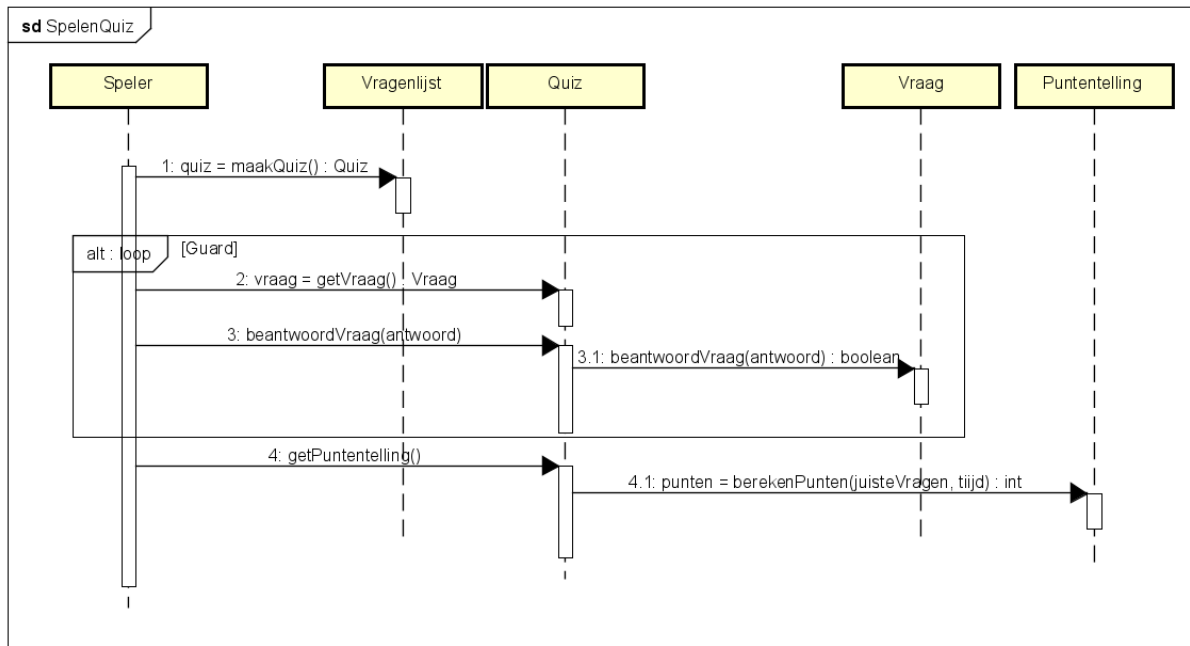


Figure 3: Sequence Diagram spelen quiz

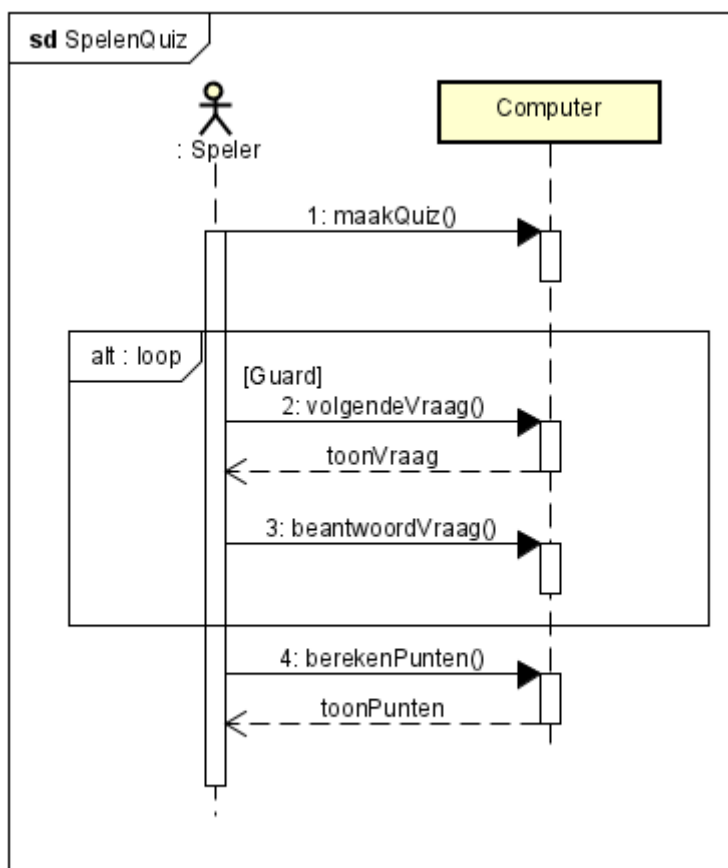


Figure 4: System Sequence Diagram spelen quiz

In bovenstaande (system) sequence diagrammen is de functie berekenPunten() te zien, hieronder valt ook het controleren van het gegeven antwoord.

2.4 Gemaakte ontwerpbeslissingen voor subsysteem

In hoofdstuk 2.1 is het klassendiagram te vinden, hieronder zullen de gemaakte ontwerpbeslissingen toegelicht worden. Voornamelijk zullen de volgende principes en patterns toegelicht worden:

- SOLID principes.
- Gang of Four design patterns.
- GRASP principes.

SOLID principes:

Principe	Plek	Toelichting
Open/closed principle	Vraag	De entiteit 'Vraag' is een abstracte klasse, hierdoor is het mogelijk om nieuwe soorten hiervan toe te voegen met dezelfde interface. Dit gebeurt nu voor 'Meerkeuze' en 'Open'. Het is gesloten voor modificatie.
Open/closed principle	Betaalservice	Bij de betaalservice is het mogelijk om meerdere betaalservices toe te voegen. Hiervoor hoeft Betaalservice zelf niet aangepast te worden (closed for modification), maar kan het wel uitgebreid worden (open for extension).
Single Responsibility	Puntentelling	Puntentelling is verantwoordelijk voor één functionaliteit, het regelen van de puntentelling. Niks meer en niks minder.
Single Responsibility	Betaalservice	Betaalservice is verantwoordelijk voor één functionaliteit, het regelen van de betaalservice.
Interface Segregation	Puntentelling	De klasse Puntentelling is een interface en kan dus zelf niet gebruikt worden, PuntentellingSimpel en PuntentellingIngewikkeld wél. Interface Segregation stelt dat geen enkele code gedwongen afhankelijk mag zijn van methoden die het niet gebruikt.
Interface Segregation	Betaalservice	De klasse Betaalservice is een interface en kan dus zelf niet gebruikt worden, IDDealAdapter, PaypalAdapter en CreditCardAdapter wél. Interface Segregation stelt dat geen enkele code gedwongen afhankelijk mag zijn van methoden die het niet gebruikt.
Dependency Inversion	Vraag	'Quiz' roept 'Vraag' aan, hierbij weet 'Quiz' niet of het een 'Openvraag' is of een 'MeerkeuzeVraag'.

Gang of Four design patterns:

Principe	Plek	Toelichting
Strategy pattern	Puntentelling	In de casus is er vereist dat er een strategy pattern gebruikt moest worden voor het flexibiliseren van de puntentelling. Dit hebben wij dan ook toegepast in Finch. Via dit patroon kan er een gemakkelijk een extra

		puntentellingsysteem toegevoegd worden, zonder dat 'Puntentelling' aangepast moet worden.
Adapter	Betaalservice	Voor de betaalservice is gekozen voor het adapter patroon. Dit is gekozen, omdat verschillende betaalservices buiten onze macht liggen en dat interface niet compatibel is met ons systeem.

GRASP principes:

Principe	Plek	Toelichting
Low coupling	Vraag	Vaak gaan dependency inversion en low coupling samen. Dit betekent dat de onderlinge afhankelijkheid tussen klassen laag ligt (lage koppeling). Dit is het geval tussen 'Quiz' en 'Vraag'.