|  |
| --- |
| Software Design Description Provo\_  Case Study OOSE-OOAD |

Namen: Marnix Wildeman 658705

Max Jansen 584700

Klas: ITA-OOSE-B

Datum: 21-10-2021

Docent: Joost Kraaijeveld

Inhoudsopgave

[1 Introductie 3](#_Toc85719965)

[1.1 Opdracht omschrijving 3](#_Toc85719966)

[1.2 Doel van het document 3](#_Toc85719967)

[1.3 Definities, acroniemen en afkortingen 3](#_Toc85719968)

[2 gedetailleerd designomschrijving 4](#_Toc85719969)

[2.1 Design Class Diagram 4](#_Toc85719970)

[2.2 Sequence Diagrams 4](#_Toc85719971)

[2.2.1 Registreren 4](#_Toc85719972)

[2.2.2 Starten Kennistoets 5](#_Toc85719973)

[2.2.3 Uitvoeren Kennistoets 5](#_Toc85719974)

[2.2.4 Genereren Totaalscore 6](#_Toc85719975)

[2.3 Gemaakte Design Keuzes 7](#_Toc85719976)

[2.3.1 GoF – Strategy Pattern 7](#_Toc85719977)

[2.3.2 GRASP – Creator 7](#_Toc85719978)

[2.3.3 GRASP – Controller 7](#_Toc85719979)

[2.3.4 GRASP – Information Expert 8](#_Toc85719980)

[2.3.5 Overerving 8](#_Toc85719981)

[2.3.6 SOLID – Single Responsibility Principle (SRP) 8](#_Toc85719982)

[2.3.7 SOLID – Open Closed Principle (OCP) 8](#_Toc85719983)

[2.3.8 SOLID – Interface Segregation Principle (ISP) 8](#_Toc85719984)

[3 Bronnen 9](#_Toc85719985)

# Introductie

## Opdracht omschrijving

[Zie 1.1 van SRS](../SRS/SRS.docx)

## Doel van het document

Het system design description document is de blauwdruk van het uiteindelijke stuk software die de programmeur(s) gebruiken om tot een goed werkend eindproduct te komen zonder al teveel struikelblokken. Die zijn tijdens het analyse proces al gladgestreken. Zoals een bouwvakker de blauwdrukken gebruikt om te zien welke maten en ruimtes het huis moeten hebben, kan de programmeur zien welke klassen en methodes nodig zijn. In beide gevallen hoeven ze niet te freestylen.

## Definities, acroniemen en afkortingen

|  |  |
| --- | --- |
| Term | Omschrijving |
| GRASP | Generalized Responsibility Assignment Software Patterns/Principles |
| GoF | Gang of Four |
| Coupling | Coupling is de onderlinge vorbondenheid van klassen. Hoge coupling kan voor veel problemen zorgen wanneer één onderdeel veranderd wordt, en andere onderdelen stoppen met werken omdat ze afhankelijk zijn van het eerste onderdeel. Coupling staat vaak tegenover Cohesion. Als de een hoog is, is de ander typisch gezien laag. |
| Cohesion | Cohesion is de samenhang van het systeem. Er is hoge cohesion wanneer elk stuk code zijn eigen taak heeft. Cohesion staat vaak tegenover Coupling. Als de een hoog is, is de ander typisch gezien laag. |
| Information Hiding | De implementatie van klassen moet verborgen zijn voor andere klassen. Zo kan een klasse B niet zomaar de klasse A veranderen of informatie uitlezen. |
| SOLID | Richtlijnen die helpen software makkelijker, begrijpelijker, flexibeler en onderhoudbaar te maken. |

# gedetailleerd designomschrijving

## Design Class Diagram

Diagram

Description automatically generated

1

## Sequence Diagrams

### Diagram Description automatically generatedRegistreren

Dit is het Sequence Diagram voor het registreren van een account. Een docent vult de gevraagde gegevens in, en als hij voor een premium account kiest zal er een betaling plaats moeten vinden.

2

### Starten Kennistoets

Diagram

Description automatically generated

3

Bij het starten van een kennistoets zal de systemcall naar de DocentController gestuurd worden en heeft het e-mailadres/gebruikersnaam van de docent nodig. De DocentController heeft als enige functie het ophalen van de juiste Docent uit de database, wanneer dit gelukt is zal de controller de call delegeren.

De docent kan d.m.v. een toetscode of toetsnaam de gewilde kennistoets opvragen. Als dit gelukt is zal de docent het lokaalnummer invullen waarna het Creator Pattern wordt toegepast om de lokaalcode aan te maken. Als dit gelukt is zal de docent het lokaal open zetten zodat leerlingen kunnen joinen waarbij het systeem de lokaalcode aan de docent geeft.

### Uitvoeren Kennistoets

Diagram

Description automatically generated

4

Dit Sequence Diagram begint bij de student die aan Provo vraagt om een vraag te laten zien d.m.v. een vraagnummer. Provo laat die vraag aan de student zien waarbij het systeem bij de klasse vraag de vraagtekst opvraagt. Als de vraag een meerkeuze vraag is, zal Provo aan de vraag de mogelijke antwoorden opvraagt.

De student zal hierna zijn antwoord aan invullen bij de vraag waardoor toetsdeelname via het Creator Pattern een gegevenAntwoord aanmaakt met het antwoord als string meegegeven.

### Genereren Totaalscore

Diagram

Description automatically generated

5

De systemcall voor het genereren van de totaalscores wordt naar de use-case controller DocentController gestuurd waarbij via het emailadres de juiste docent wordt gebruikt om naar toe te delegeren. Via de docent wordt naar de kennistoets gedelegeerd waar de loop met het berekenen van de scores plaats vind.

De loop zal plaats vinden voor elke toetsdeelname en begint met de call berekenPunten. De punten beginnen op nul en worden bij elke correcte antwoord met één verhoogd bij elk correcte antwoord. Daarom is de eerste actie de check of het antwoord correct is of niet. Dit wordt voor elke vraag gedaan. Als alle vragen gecontroleerd zijn zullen met de gebruikte tijd de bonuspunten berekent worden. Wanneer deze twee variabelen bekend zijn zal de score berekend worden en tot slot in een overzicht geplaatst worden.

## Gemaakte Design Keuzes

Tijdens de analyse- en ontwerp fases hebben wij een aantal keuzes gemaakt met betrekking tot het systeemontwerp. Hieronder zal de argumentatie voor onze gemaakte ontwerp keuzes staan. Deze keuzes zullen beargumenteert worden met GRASP en GoF Patterns.

### GoF – Strategy Pattern

Het Strategy Pattern hebben wij toegepast bij de Puntentelling. Dit is omdat er op dit moment nog weinig bekend is over welke systematiek gebruikt gaat worden voor het berekenen van de totaalscores. Het kan ook voor komen dat er meerdere systematieken komen waarbij de docent er één kan kiezen. Door de puntentelling een interface te maken kan er later gemakkelijk één of zelfs meerdere manieren van puntentelling toegevoegd worden. Dit os ook te zien in het Domeinmodel en het Design Class Diagram. Het gebruik van het Strategy Pattern zorgt hier voor een grote hoeveelheid flexibiliteit waarbij later in het ontwerp geen code aangepast hoeft te worden wanneer er een nieuwe systematiek wordt toegevpoegd.

### GRASP – Creator

Om coupling/koppeling te verminderen is er gebruik gemaakt van de Creator Pattern. Wanneer er hoge coupling is zijn er veel klassen met afhankelijkheden van elkaar. Hierdoor wordt het heel moeilijk gemaakt om aanpassingen en toevoegingen binnen het systeem te maken zonder dat andere klassen problemen krijgen of zelfs helemaal stoppen met werken. Volgens het Creator Pattern moet de juiste klasse een instantie van een andere klasse aanmaken. Als dit allemaal wordt gedaan door één klasse, dan zou deze veel afhankelijkheden hebben en zou de coupling verlaagd worden.

### GRASP – Controller

Wij maken gebruik van een use-case controller die system events afhandelt. In de Design Class Diagram en de Sequence Diagram van kennistoets starten en -uitvoeren is een DocentController toegepast. Deze beheert alle docenten in het systeem van Provo. Bij deze wordt het e-mailadres/gebruikersnaam (is hetzelfde in dit systeem) opgevraagd en haalt dan de juiste docent naarboven. Daarna delegeert de controller de operatie door naar de docent klasse.

### GRASP – Information Expert

Bij het maken van alle Sequence Diagrams hebben wij de richtlijnen van de Information Expert gevolgd. Dit houdt in dat wij zo veel mogelijk gebruik maken van taak delegatie wanneer de huidige klasse niet genoeg informatie heeft om de huidige taak uit te voeren zonder afhankelijkheden toe te voegen en daarmee coupling te verhogen. Dit houdt bovendien het Information Hiding Principe in stand en verhoogt de cohesion/samenhang van het systeem.

### Overerving

Op meerdere plekken binnen het systeem hebben wij gebruik gemaakt van overerving. Dit is zichtbaar in zowel het Domeinmodel als het Design Class Diagram. De klasse vraag is een abstracte klasse waarbij er drie implementaties zijn. Namelijk: de Kort-antwoord-vraag, de Meerkeuzevraag en de   
Juist-/Onjuistvraag. Er kunnen tevens in de toekomst andere soort vragen toegevoegd worden die erven van de superklasse Vraag. De klasse vraag heeft een methode juisteAntwoordCheck, deze zal door elke klasse geërfd worden maar op een andere manier geïmplementeerd worden omdat de drie klassen een ander soort antwoord verwachten.

De docent kan uit twee typen accounts kiezen, deze zijn Basis en Premium. Hierom hebben wij een klasse aangemaakt genaamd AccountType waarbij het verschil is dat Basis gratis is, en Premium met een prijs per jaar komt.

### SOLID – Single Responsibility Principle (SRP)

SRP houdt in dat elke module, klasse of methode maar één responsibility(verantwoordelijkheid) mag hebben. Dit zodat je geen godclasses of -functies in je systeem introduceert. Deze verlagen cohesion en verhogen coupling. Dit principe hebben wij bij al onze klassen en methodes gebruikt.

### SOLID – Open Closed Principle (OCP)

Het OCP betekent: “Open for extensions, closed for modification.” Je mag de code wel uitbreiden, maar niet aanpassen. Dit hebben wij toegepast door op plekken waar wij toevoegingen verwachten zoals de vragen en de puntentellingen gebruik te maken van het Strategy Pattern dat bij 3.4.1 is uitgelegd.

### SOLID – Interface Segregation Principle (ISP)

Wij hebben onze interfaces zo klein mogelijk gehouden om te voldoen aan ISP. ISP houdt in dat en gebruiker geen afhankelijkheid mag hebben van iets dat hij/zij niet gebruikt. Dit is zodat wanneer de code veranderd, er geen foutmeldingen mogen komen voor libraries/imports die niet eens gebruikt worden.

# Bronnen

*Case Study OOAD: Provo.* (sd). Opgehaald van: https://onderwijsonline.han.nl/elearning/lessonfile/BylEz7MD/https://onderwijsonline.han.nl/elearning/lessonfile/BylEz7MD/eyJpdiI6IktuajJDTkhFMStWcXA2bFAydUlSOGc9PSIsInZhbHVlIjoiNlRCMENCeXAySHQvTHdCTlBCYWhxNkd1UTcvSTV4eFA0Yk1PZDRNQVYzRWhXL0NyTFE4NE9TcVVYT

