1-11-2018

Klaas van der Linden 599644

Luca Hogeweide 601160

Klas: ITA-OOSE-A-s

Software Design Description Odinido

Case study OOAD OOSE

Docent: Marco Engelbart

1. Inhoud

[2 Introduction 2](#_Toc528825951)

[2.1 Opdracht omschrijving 2](#_Toc528825952)

[2.2 Doel document 2](#_Toc528825953)

[2.3 Definitions, acronyms, and abbreviations 2](#_Toc528825954)

[3 Detailed Design Description 3](#_Toc528825955)

[Odinido 3](#_Toc528825956)

[3.1 Design Class Diagram 3](#_Toc528825957)

[3.2 Sequence Diagrams 3](#_Toc528825958)

[3.2.1 Starten Kennistoets 3](#_Toc528825959)

[3.2.2 Genereren score 4](#_Toc528825960)

[3.2.3 Aanmaken kennistoets 5](#_Toc528825961)

[3.2.4 Registeren 6](#_Toc528825962)

[3.2.5 Uitvoeren Kennistoets 7](#_Toc528825963)

[3.2.6 Uitvoeren Teamgevecht 8](#_Toc528825964)

[3.3.1 Activity and State Diagrams 8](#_Toc528825965)

[3.3.2 Design decisions made for the system 8](#_Toc528825966)

[Bijlage 11](#_Toc528825967)

# Introduction

## Opdracht omschrijving

Voor een product omschrijving zie [SRS 2.1.](SRS%20Odinido%20Luca%20Hogeweide%20en%20Klaas%20van%20der%20Linden.docx)

## Doel document

In dit document staan de design en analyse onderdelen die relevant zijn voor de software. Hiermee kan de programmeur zien wat er gedaan moet worden om de applicatie te realiseren.

## Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| Term | Description |
| GRASP | Generalized Responsibility Assignment Software Patterns/Principles |
| GoF | Gang of Four |
|  |  |

# Detailed Design Description

## Odinido

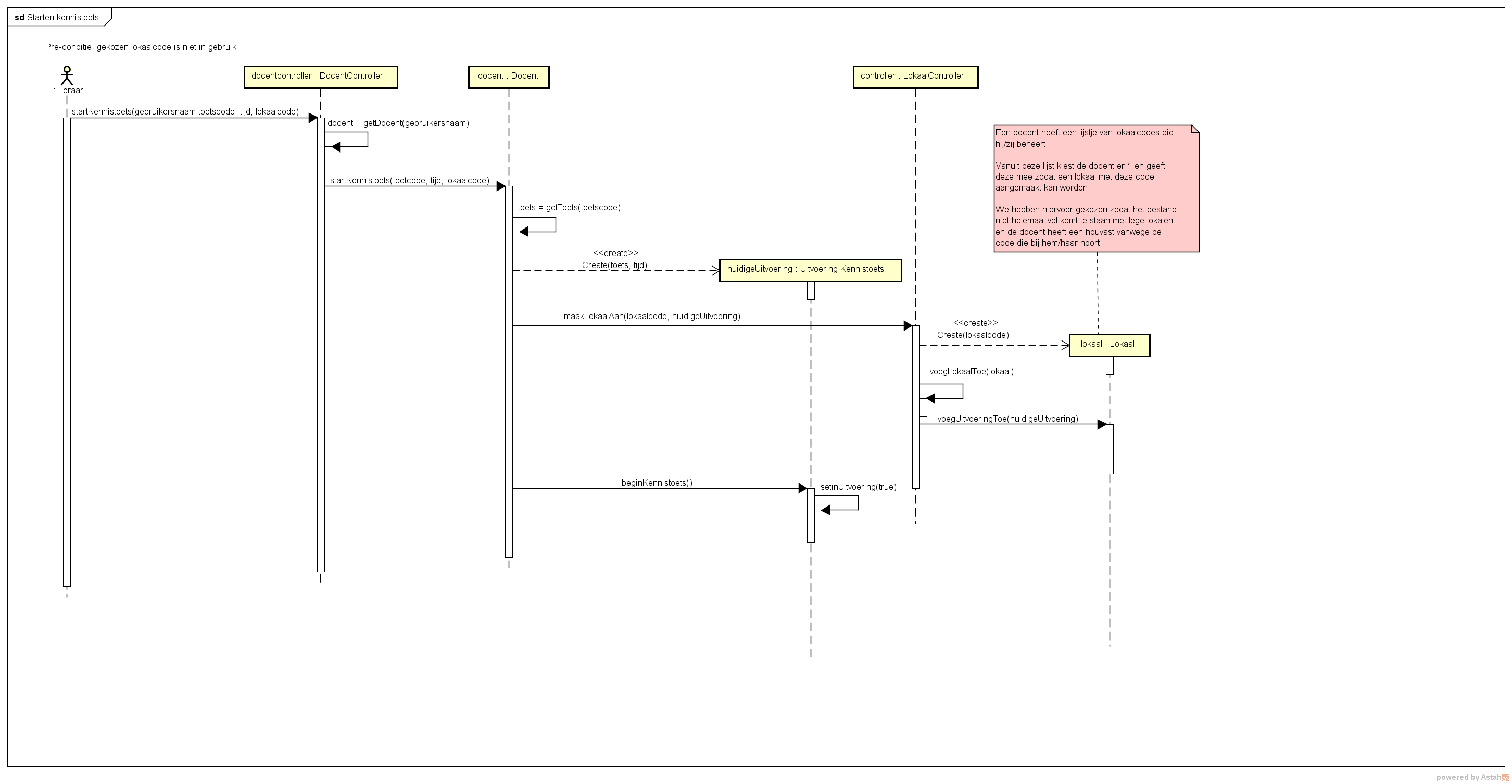
## Design Class Diagram

Zie bijlage voor de afbeelding van het Design Class Diagram.

## Sequence Diagrams

Hieronder bevinden zich alle sequence diagrammen inclusief eventuele toelichting en argumentatie voor de gemaakte keuzes. Op alle sequence diagrammen kan het control + click commando worden uitgevoerd om zo een JPEG variant van het diagram te openen waarop kan worden ingezoomd om het diagram beter te bekijken.

### 3.2.1 Starten Kennistoets

[](Sequence%20Diagrammen/Starten%20kennistoets.png)

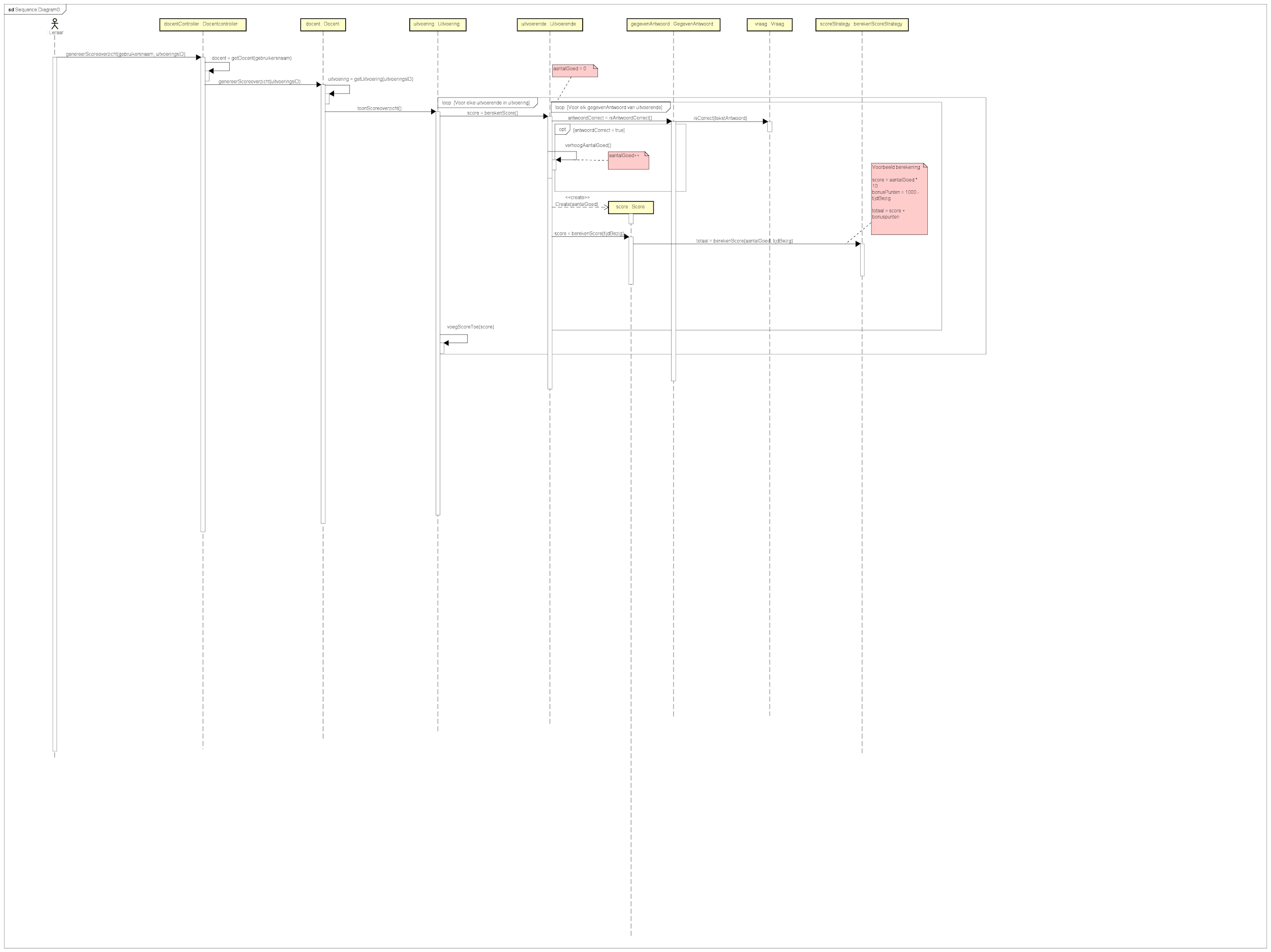
Figuur 1 Sequence diagram starten kennistoets

De system call van startenkennistoets komt binnen bij de docentController. Deze controller beheert alle docenten en vraagt op basis van de meegegeven gebruikersnaam de juiste docent op. De controller wordt alleen gebruikt voor binnen komende calls van de actor en voor het ophalen van het juiste object. De verdere taken worden door gedelegeerd naar het opgehaalde object.

De docent haalt bij zichzelf de kennistoets op, op basis van de meegegeven toetscode. Daarna maakt de docent een nieuwe uitvoering aan met de juiste paramaters. Hierbij wordt het creator pattern toegepast. De docent beschikt over de juiste initialisatie data om de uitvoering aan te maken en beheert de uitvoering, dus is deze klasse het meest geschikt om de uitvoering aan te maken.

Ook moet er een lokaal worden aangemaakt waarin de uitvoering zich afspeelt. Deze taak delegeert de docent door naar de lokaalcontroller. Deze beheert namelijk de lokalen. Ook hier word dust het creator pattern toegepast. De controller voegt het lokaal aan zichzelf toe en de meegegeven uitvoering aan het lokaal. Als dit gedaan is begint de docent de uitvoering.

### 3.2.2 Genereren score

[](Sequence%20Diagrammen/Genereren%20score.jpg)

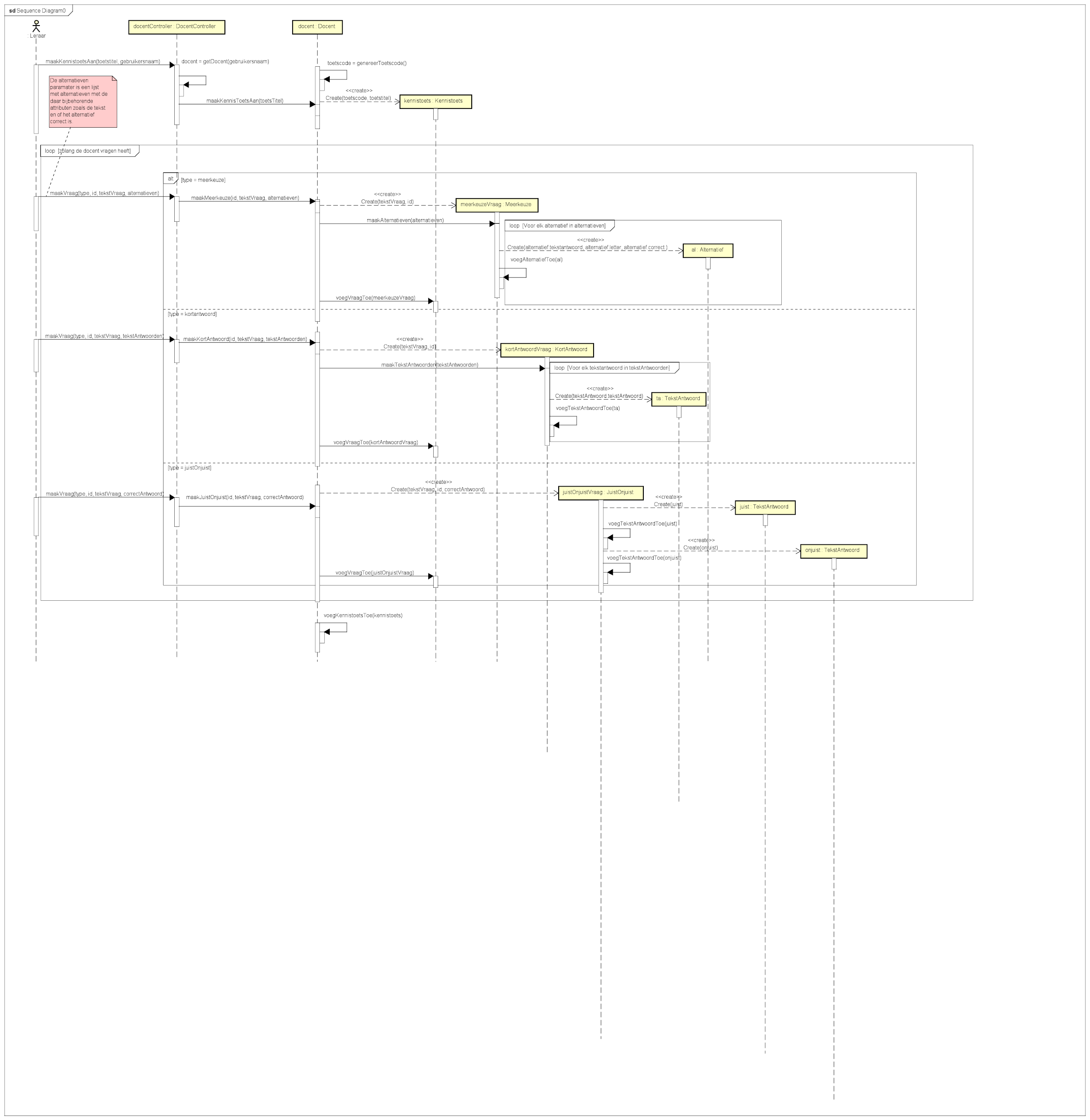
Figuur 2 Sequence diagram score genereren

Bij het genereren van de score wordt het pattern information expert toegepast. Elke klasse krijgt voor het berekenen van de score de juiste verantwoordelijkheid toegewezen op basis van welke data de klasse bevat. Dit zorgt ervoor dat information hiding in stand wordt gehouden. Klassen weten namelijk niet van elkaar hoe ze de score berekenen. Bovendien houdt dit de koppeling tussen de klasse laag en de cohesie hoog.

De uitvoerende maakt de score. Deze bevat namelijk de gegevenAntwoorden. Aan de gegevenAntwoorden kan de taak worden door gedelegeerd om te kijken of het antwoord van de student overeenstemt met het beantwoorden van de vraag. Deze delegeert de taak door naar de bijbehorende vraag. De vraag bevat een method isCorrect waarbij de implementatie afhankelijk is van het type. Om dit op te lossen is er gebruik gemaakt van polymorphism waarbij vraag een abstracte klasse is en ieder type klasse van vraag een eigen implementatie heeft voor het kijken of het gegeven antwoord correct is.

Op basis hiervan kan de uitvoerende het aantal goede antwoorden ophogen en hiermee een nieuwe score object creëren. Het score object berekent vervolgens zelf de score met behulp van het strategy pattern. Het strategy pattern is hier toegepast zodat de strategie voor het berekenen van de score gemakkelijk kan worden aangepast en zodat er in de toekomst eenvoudig nieuwe score strategieën kunnen worden toegevoegd, aan gezien de punten systematiek bij Odinido op het moment nog niet definitief is.

### 3.2.3 Aanmaken kennistoets

[](Sequence%20Diagrammen/aanmakenKennistoets.png)

Figuur 3 Sequence diagram aanmaken kennistoets

Bij het maken van een kennistoets is ervoor gekozen om gebruik te maken van verschillende system calls van de actor om zo het maken van de kennistoets en de vragen te onderscheiden.

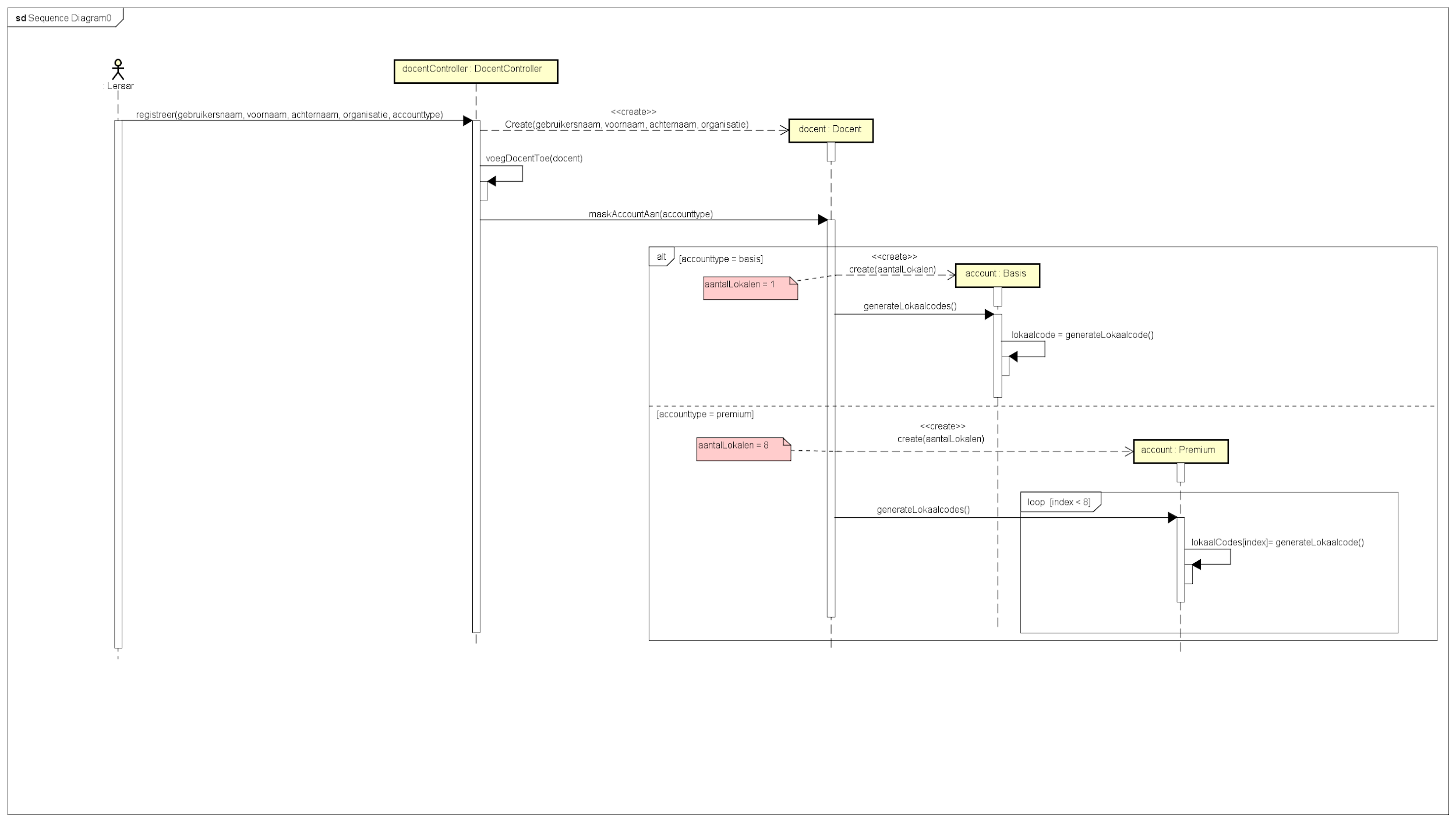
Als eerst komt het maken van de kennistoets aan bij de docentcontroller. Deze haalt de juiste docent op, op basis van de gebruikersnaam en delegeert het maken van de kennistoets door naar de docent zelf. Deze beheert namelijk de kennistoetsen.

Vervolgens wordt er gekeken welk type vraag de docent heeft gekozen en op basis hiervan de juiste vraag aangemaakt. De docent maakt de vraag aan en delegeert het creëren van de antwoorden door naar de vraag zelf. Als de complete vraag is aangemaakt, wordt deze toegevoegd aan de toets. De kennistoets wordt aan het docent object toegevoegd wanneer deze klaar is met het maken van de vragen.

Door de taken en verantwoordelijkheden zo goed mogelijk te verdelen wordt de koppeling zo laag mogelijk gehouden door elk klasse de juiste objecten te laten aanmaken en waar mogelijk taken door te laten delegeren.

### 

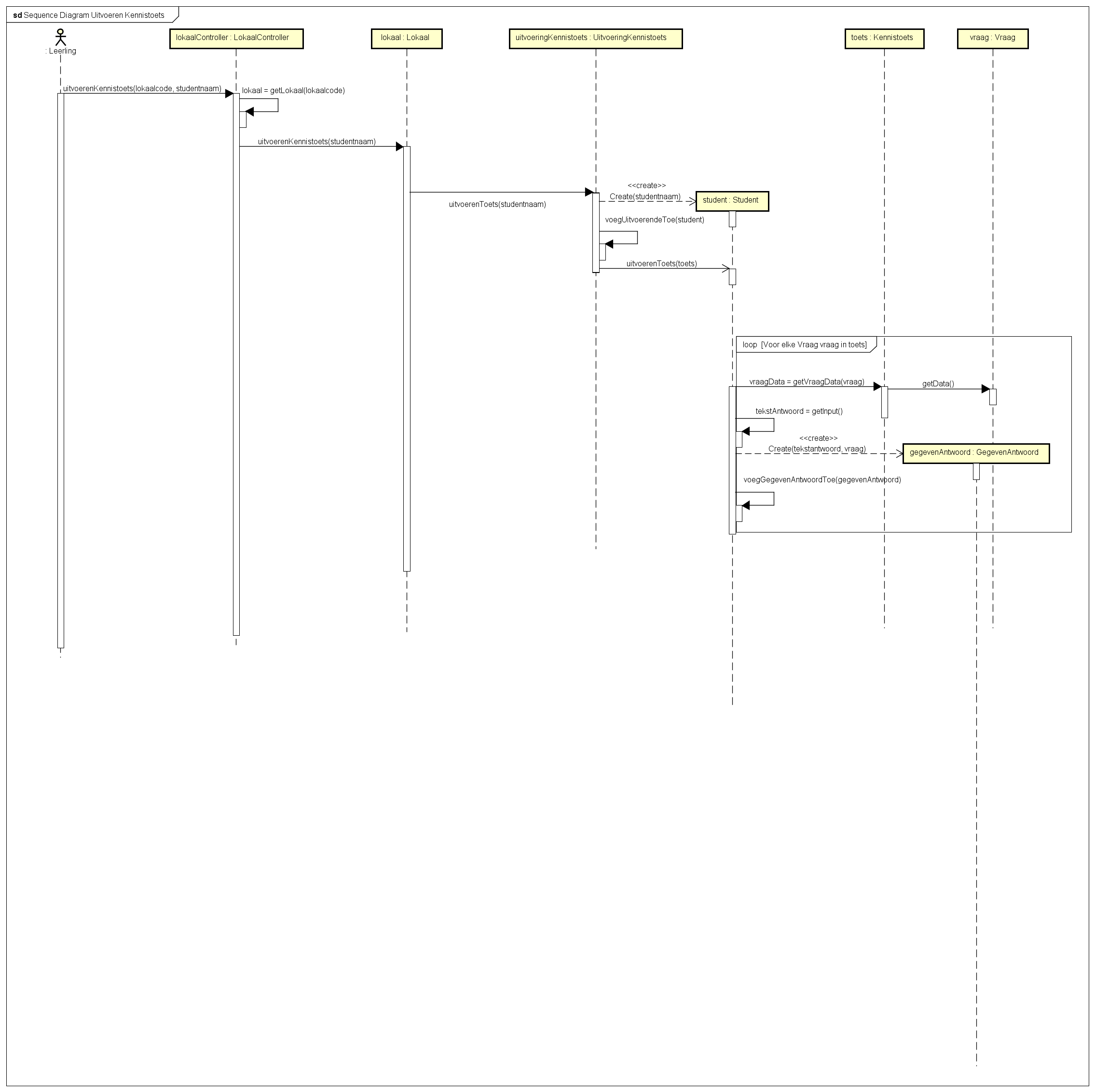
### 3.2.4 Registeren

[](Sequence%20Diagrammen/Registreren.png)

Figuur 4 Sequence diagram registreren

Bij het registreren wordt wederom het creator pattern toegepast. Als eerst maakt de docentcontroller een docent aan. Deze beheert namelijk de docent en krijgt via de system call van de actor alle benodigde informatie om een docent aan te maken. Als de docent is aangemaakt wordt het aanmaken van een account door gedelegeerd naar de docent zelf. De docent beheert namelijk zijn eigen account en maakt gebruik van zijn account, dus maakt hij zelf zijn account aan in plaats van de controller. Bovendien houdt dit de koppeling tussen de klassen laag, aangezien het aanmaken van het account door de docent controller een extra afhankelijkheid betekent.

### 3.2.5 Uitvoeren Kennistoets

[](Sequence%20Diagrammen/Uitvoeren%20kennistoets.jpg)

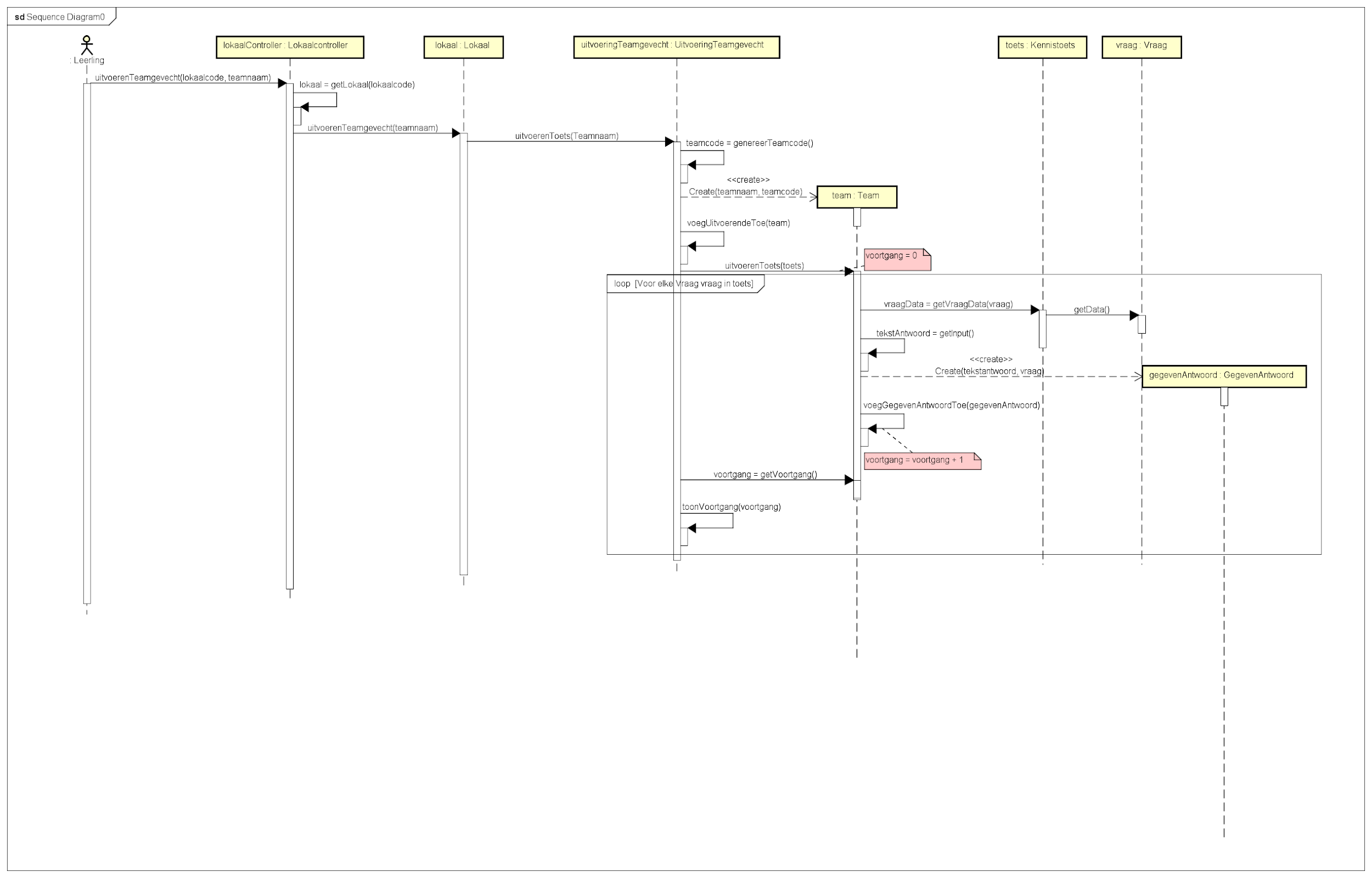
Figuur 5 Sequence diagram uitvoeren kennistoets

Bij het uitvoeren van een kennistoets kom de student met zijn system call binnen bij de lokaal controller. Deze beheert alle lokalen en zoekt op basis van de meegegeven lokaalcode het juiste lokaal. Deze delegeert vervolgens het uitvoeren van een kennistoets door naar het lokaal. Deze geeft het weer door aan het uitvoering, aangezien deze zowel de kennistoets als de student bevat.

De uitvoering maakt de aangemelde student aan en voegt deze toe aan zijn uitvoering. Dit zorgt ervoor dat de docent kan zien welke studenten deelnemen aan de uitvoering en zodat hij later als de uitvoering voltooid is kan zien welke studenten hebben deelgenomen. De uitvoering bevat de kennistoets en deze geeft hij mee aan de student om uit te voeren.

De student gaat door alle vragen van de kennistoets heen. Hierbij maakt hij gegeven antwoorden aan die hij aan zichzelf toevoegt Deze gegeven antwoorden voegt hij aan zichzelf toe. Dit zorgt ervoor dat de score later kan worden berekend wanneer de student hier toe opdracht krijgt. Zie 3.2.2 Genereren score

### 3.2.6 Uitvoeren Teamgevecht

[](Sequence%20Diagrammen/UitvoerenTeamgevecht.png)

Figuur 6 Sequence diagram uitvoeren teamgevecht

De ontwerpkeuze bij het sequence diagram uitvoeren teamgevecht zijn grotendeels hetzelfde als bij het uitvoeren van de kennistoets. Het verschil bij het teamgevecht is dat er live de voorgang wordt getoond van het team. Daarom haalt de uitvoering per vraag de voorgang bij het team op en toont hij deze.

#### 

## Activity and State Diagrams

Voor het opgestelde model van Odinido is geen State Diagram gerealiseerd. Het enige object wat in het huidige model van state verandert is de uitvoering van de kennistoets. Deze kan verkerend in states als actief, non-actief en score berekenend. Deze states en de verandering er tussen zijn echter voor de hand liggend, vrij gemakkelijk te begrijpen en hebben nauwelijks impact op de werking van het systeem. Daarom is het niet relevant en essentieel om voor de verandering bij de uitvoering van de kennistoets een geheel state diagram op te stellen.

## Design decisions made for the system

Gedurende het analyseren en ontwerpen van het systeem voor Odinido zijn verschillende ontwerp beslissing gemaakt. Hieronder volgt de argumentatie voor de gemaakte ontwerp beslissingen aan onder andere de hand van GRASP en de GoF patterns.

**Overerving en Polymorphism**

In het ontwerp van het systeem vindt meerdere malen overerving plaats, te zien in zowel het [Domein model](Domein%20Model/Domain%20Model.png) als het [Design Class Diagram](Design%20Class%20Diagram/Design-class%20diagram.jpg), en polymorphism tevens te zien in het Design Class Diagram.

Bij de klasse account vind overerving plaats. Hierbij is gekozen om de klasse abstract te maken, aangezien de docent alleen basis of premium als account kan hebben en niet een account zonder type. Beide klasse bevat het attribuut aantal lokalen. Bij de account typen bevatten namelijk een lokaal aantal, een bij basis en acht bij premium. Dit zorgt ervoor dat de docent een vast aantal lokaal codes heeft die hij kan gebruiken zodat de studenten elke keer de zelfde lokaal code kunnen gebruiken om zich bij een lokaal aan te melden.

Beide account typen erven de abstracte methode generateLokaalCodes over waar bij elk typ een eigen implementatie heeft. Bij basis wordt één lokaalcode gegenereerd, bij premium acht codes. Hierbij wordt de methode generateLokaalCode aangeroepen die een lokaalcode genereert en retourneert.

Bij de abstracte klasse vraag vindt tevens overerving plaats. Er zijn namelijk verschillende typen vragen die een docent kan toevoegen aan een kennistoets. Bovendien kan er nu gemakkelijk een type vraag worden toegevoegd door deze te laten overerven van de superklasse vraag. Vraag bevat een abstracte methode isCorrect die kijkt of een gegeven antwoord van een student klopt. De implementatie per type vraag verschilt hier, aangezien elke vraag verschillende antwoordtypes heeft.

Ook bij uitvoering en uitvoerende is gebruik gemaakt van overerving om zo de attributen met betrekking tot het team en de methodes met betrekking tot de uitvoering te scheiden van de student en het uitvoeren van de kennistoets.

**Strategy pattern**

Bij het berekenen van de score is het strategy pattern toegepast. De punten systematiek voor het berekenen van de score is op het moment van ontwerpen nog niet bekend. Het gebruik van het strategy pattern zorgt hierbij voor flexibiliteit. Zodra er een of meerdere punten methodes bekend zijn, dan kunnen deze gemakkelijk worden toegevoegd een de interface van berekenScoreStrategy en snel worden geïmplementeerd in de score klasse. De bijbehorende klassen en interfaces zijn te zien in het [Design Class Diagram](Design%20Class%20Diagram/Design-class%20diagram.jpg). De uitwerking is te zien in het sequence diagram [score genereren](Sequence%20Diagrammen/Genereren%20score.jpg).

**GRASP**

***Creator***

Tijdens het ontwerpen van het systeem zijn verschillende richtlijnen van GRASP toegepast. Zoals eerder benoemd bij de sequence diagrammen, is er gebruik gemaakt van de creator. De creator zorgt ervoor dat de juiste klasse de instantie van een andere klasse aanmaakt. Als één klasse alle instanties van de andere klassen zou aanmaken, dan zou deze klasse veel afhankelijkheden hebben wat zorgt voor een hoge koppeling. Een hoge koppeling zorgt ervoor dat het systeem lastig aan te passen en uitbreidbaar is. Bovendien is hoge koppeling in strijd met de GRASP richtlijnen. De creator is bij alle sequence diagrammen toegepast.

***Controller***

Voor de afhandeling van de systeemoperaties hebben wij gebruik gemaakt van verschillende use controllers. Deze zijn terug te vinden in het [Design Class Diagram](Design%20Class%20Diagram/Design-class%20diagram.jpg) alsmede de [sequence diagrammen](Sequence%20Diagrammen). Voor de acties die de docentcontroller uitvoert is gebruik gemaakt van een docentcontroller die alle docenten in het systeem beheert. Deze haalt op basis van een meegegeven gebruikersnaam de docent op en delegeert vervolgens de operatie door naar de docent klasse.

Ook voor de lokalen is een controller toegepast. Deze bevat alle lokalen die binnen het systeem actief zijn. Zodra een uitvoerende een uitvoering wil gaan uitvoeren, komt hij eerst in de lokaalcontroller terecht. Daar wordt op basis van de lokaalcode het juiste lokaal opgehaald. De rest van handeling worden door gedelegeerd naar het lokaal waar de flow verdergaat.

***Information Expert, Low Coupling and High Cohesion***

In de sequence diagrammen is er rekening gehouden met de information expert richtlijn. Taak worden door gedelegeerd naar een andere klasse wanneer de klasse waar de taak binnenkomt niet over de informatie beschikt om de taak uit te voeren. Dit zorgt ervoor dat koppeling laag wordt gehouden, aangezien er niet enkele klassen zijn die taken uitvoeren, maar het uitvoeren van de taken over de klassen wordt verdeeld. Het door delegeren van de taken en het laag houden van de koppeling zorgt automatisch voor high cohesion, aangezien het juist door delegeren zorgt voor samenhang tussen de klassen.

# Bijlage

[Link naar de afbeelding van het Design Class Diagram](Design%20Class%20Diagram/Design-class%20diagram.jpg)