|  |
| --- |
| **Parola**  I-OSSE OOAD 2023/2024 S1  SDD document |
| Jochem de Jong  Sn: 1656142  Osama Halabi  Sn: 1628160  Docent: Michel Koolwaaij  26-10-2023  v. 1.0.7 |

Inhoudsopgave

[1 Introduction 3](#_Toc149207635)

[1.1 Overall Description 3](#_Toc149207636)

[1.2 Purpose of this document 3](#_Toc149207637)

[1.3 Definitions, acronyms, and abbreviations 3](#_Toc149207638)

[2 Architectural Overview 3](#_Toc149207639)

[3 Detailed Design Description 4](#_Toc149207640)

[3.1 Deployment Diagram 4](#_Toc149207641)

[3.1.1 Design Decisions related to deployment 4](#_Toc149207642)

[3.2 Design ParolaApp 5](#_Toc149207643)

[3.2.1 Design Class Diagram 5](#_Toc149207644)

[3.2.2 Sequence Diagrams 7](#_Toc149207645)

[3.2.3 Activity and State Diagrams 9](#_Toc149207646)

[3.2.4 Design decisions made for the subsystems 10](#_Toc149207647)

[3.3 Database Design 11](#_Toc149207648)

[3.3.1 Design decisions related to the database 11](#_Toc149207649)

# Introduction

## Overall Description

Voor het OOAD-vak wordt een quiz applicatie ontwikkeld. In deze applicatie kunnen spelers een quiz spelen. Deze quiz bestaat uit een aantal vragen die de speler moet beantwoorden. Voor elke goed antwoord krijgt de speler een karakter. Aan het eind van de quiz moet de speler een woord maken van alle verdiende karakters. Op basis van een bereken score strategie krijgt de speler een eindscore. Met deze score weet de speler hoe goed hij was tijdens het spelen van de quiz. Dit product wordt ontwikkeld voor opdrachtgever Michel Koolwaaij.

## Purpose of this document

In dit document wordt het ontwerp en de structuur van de Parola applicatie uitgebreid uiteengezet. We duiken diep in de modules en interfaces die gezamenlijk de complexiteit en functionaliteit van de app vormgeven. De keuzes dit we als programmeurs hebben gemaakt wat betreft het gebruiken van programmeerprincipes.

## Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| **Term** | **Beschrijving** |
| US | Use Case |

# Architectural Overview

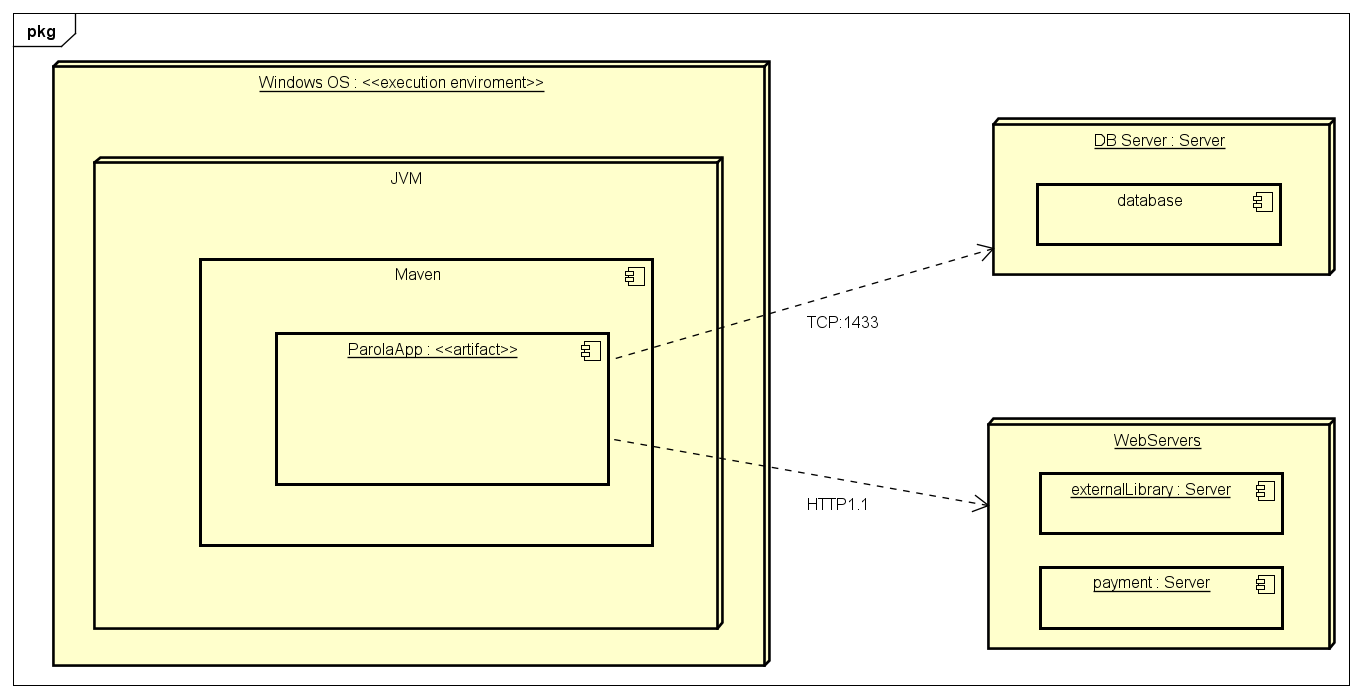
In dit diagram geeft een algemene schets van de architectuur van het systeem en zijn subsystemen.

A screenshot of a computer

Description automatically generated

# Detailed Design Description

## Deployment Diagram



In dit diagram geven we aan dat de parolaApp contact maakt met een database om data op te slaan. Maar omdat we voor deze opdracht geen database hoefde te maken hebben we gekozen om data te mocken in de parolaApp in plaats van in de database op te slaan en te vragen.

Dit geldt ook voor de external webservices zoals payment server.

### Design Decisions related to deployment

In dit deployment diagram zijn er drie componenten die we voor deze casus niet hoeven te maken of implementeren. Dit geldt voor:

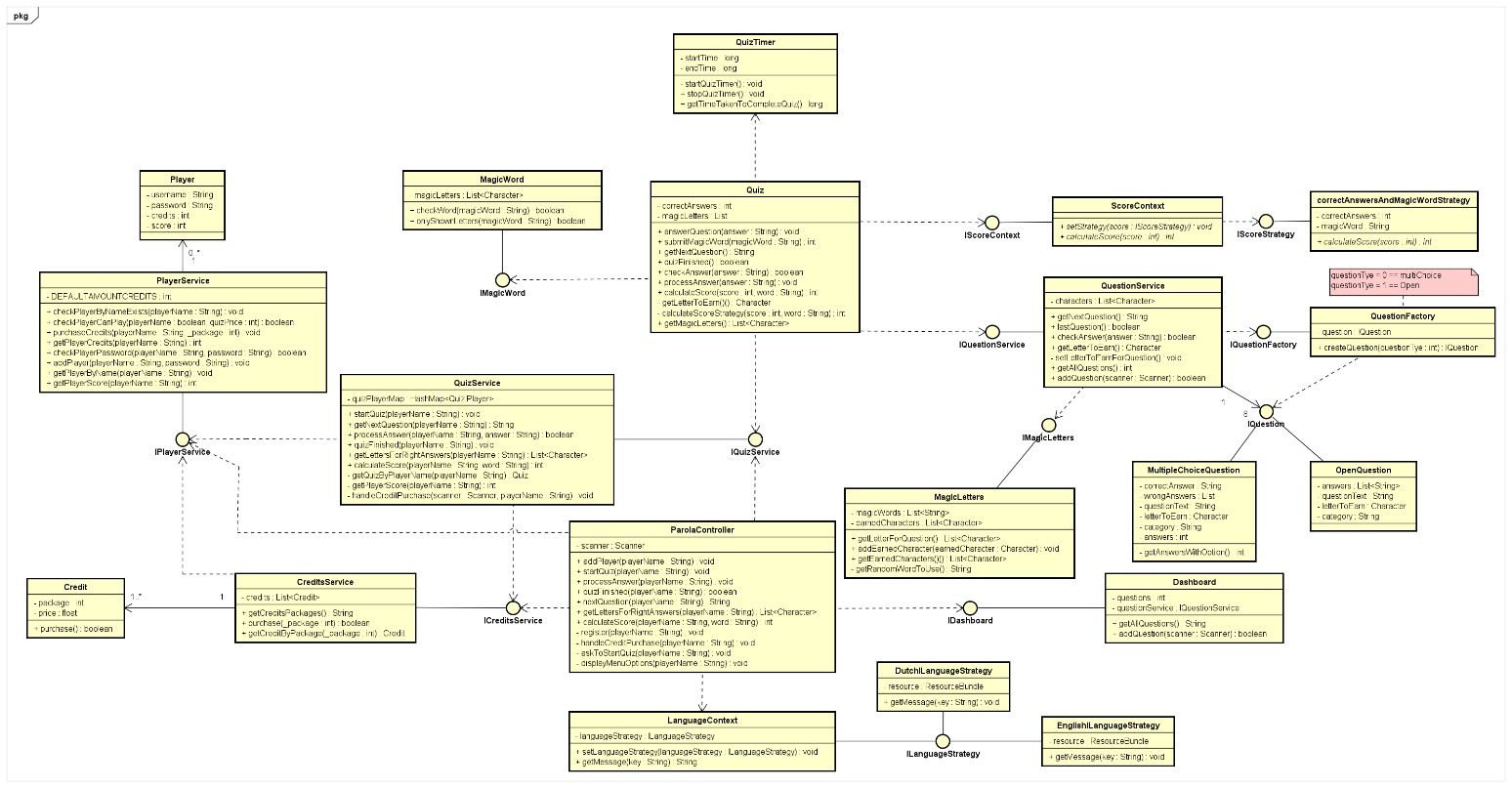
1. Database
2. externalLibrary
3. paymentLibrary

In de opdracht beschrijving is duidelijk aangegeven dat we voor de data mogen mokken in een class. Bovendien staat ook dat de external libraries hoefde te implementeren.

Bron: Case Study Parola OOSE-OOAD 2023-2024 P1.

## Design ParolaApp

### Design Class Diagram



|  |  |
| --- | --- |
| **Class** | **Explanation** |
| ParolaController | De hoofdcontroller van de applicatie die de interacties van de gebruiker coördineert en doorgeeft aan de relevante services. |
| Quiz | Vertegenwoordigt een quiz-sessie met details zoals het aantal juiste antwoorden en de magische letters. Biedt ook methoden voor het beheren van de quiz en het berekenen van scores. |
| QuizTimer | Houd de start- en eindtijd van de quiz bij. |
| QuizService | Biedt diensten voor het beheren van de quiz, zoals het starten van een nieuwe quiz, het ophalen van de volgende vraag en het verwerken van antwoorden. |
| MagicWord | Een entiteit die een verzameling magische letters bevat en functies biedt om te controleren of een woord enkel uit deze letters bestaat. |
| Player | Representeert een speler met basisinformatie zoals gebruikersnaam, wachtwoord, credits en score. |
| PlayerService | Biedt diensten voor het beheer van spelers, zoals het controleren van bestaande namen, het ophalen en instellen van credits, en het toevoegen van nieuwe spelers. |
| CreditsService | Beheert de verschillende kredietpakketten en biedt functionaliteit voor het ophalen van kredietinformatie en het aankopen van kredieten. |
| Credit | Een entiteit die de details van een specifiek kredietpakket vertegenwoordigt, met informatie over het aantal credits en de prijs. |
| LanguageContext | Context voor de taalstrategie. Het stelt gebruikers in staat om een bepaalde taalstrategie in te stellen en berichten in die taal op te halen. |
| Dutch- & EnglishILanguageStrategy | Implementaties van de **ILanguageStrategy** interface. Ze bepalen hoe berichten worden opgehaald voor respectievelijk de Nederlandse en Engelse talen. |
| Dashboard | Representeert het dashboard waar gebruikers alle vragen kunnen zien en nieuwe vragen kunnen toevoegen |
| MagicLetters | Bevat een verzameling magische woorden en biedt methoden voor het beheren en ophalen van deze woorden. |
| QuestionService | Biedt diensten met betrekking tot vragen, zoals het ophalen van vragen, het toevoegen van nieuwe vragen en het instellen van de letter voor elke vraag. |
| ScoreContext | Context voor de strategie voor scoreberekening. Het gebruikt de geïmplementeerde scorestrategie om de score te berekenen. |
| ScoreStrategy | Een interface die de strategie definieert voor het berekenen van scores. Er zijn verschillende implementaties van deze strategie, zoals **correctAnswersAndMagicWordStrategy**. |
| QuestionFactory | Een fabriek die verantwoordelijk is voor het aanmaken van verschillende soorten vragen, zoals meerkeuzevragen en open vragen. Door gebruik te maken van een factory pattern kunnen er gemakkelijk verschillende soorten vragen aangemaakt worden |
| MultipleChoiceQuestion & OpenQuestion | Subklassen van **IQuestion** die specifieke soorten vragen vertegenwoordigen met hun respectievelijke eigenschappen |

### Sequence Diagrams

#### Register een nieuwe speler.

Deze sequence diagram geeft de flow hoe een speler kan registeren om een quiz te kunnen spelen.   
Zoals in de opdracht beschrijving aan gegeven. Elke nieuwe speler krijgt 1000 credits bij het maken van de nieuw account. Dat wordt meegegeven in de methode AddSpeler met de parameter DEFAULTAMOUNTCREDITS. Wanneer een gebruiker probeert te registreren, wordt er eerst gecontroleerd of de speler al bestaat. Zo niet, dan wordt deze aangemaakt en krijgt de nieuwe speler het eerder beschreven aantal credits.

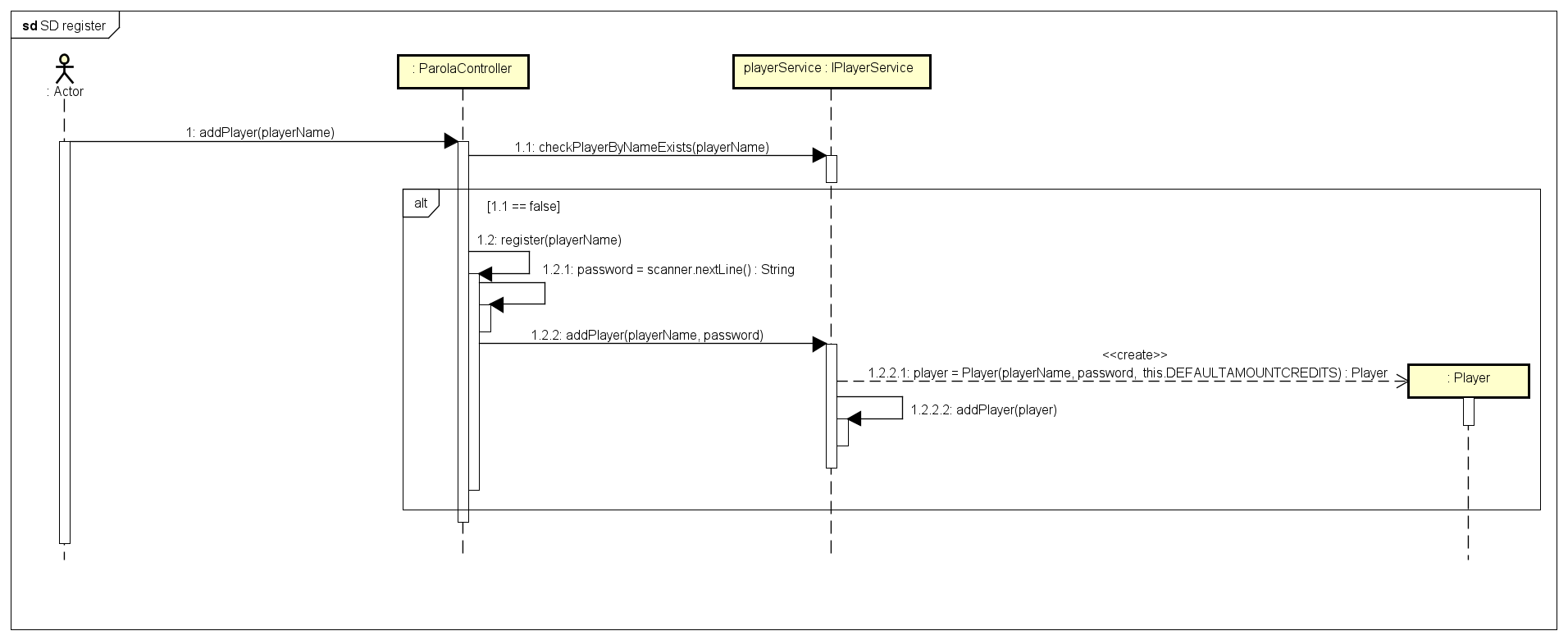


Figure 1 SD register

#### Start quiz

Deze sequence diagram geeft de flow van hoe de speler een quiz kan starten. Er wordt eerst gecheckt of de speler genoeg credits heeft om te kunnen spelen. Vervolgens wordt een object quiz gemaakt met een aantal vragen, daarna wordt een object magicLetters gemaakt die een list van karakters bevat voor elke vraag van de quiz.

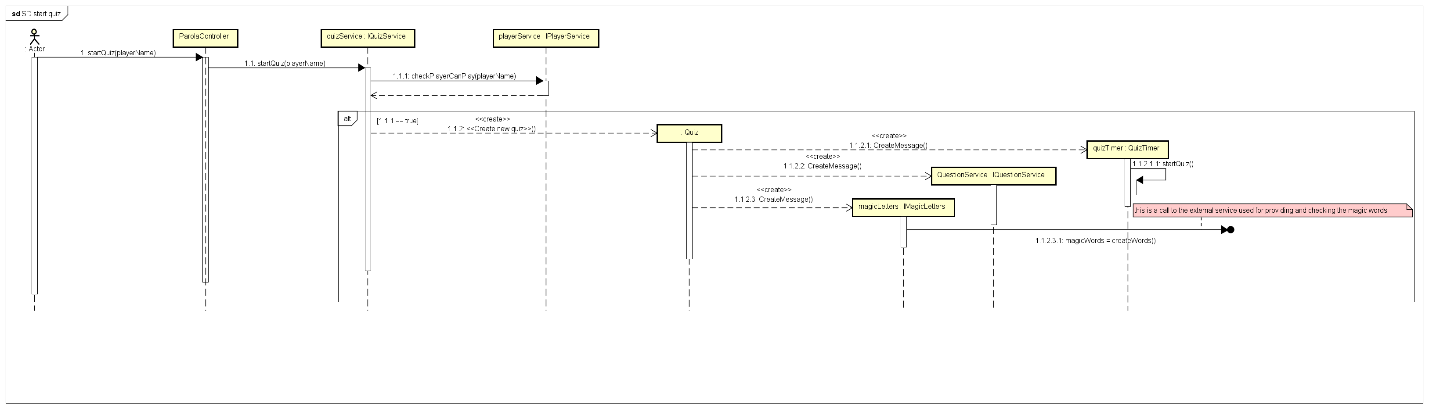


Figure 2: SD start quiz

#### Answer question

Deze sequence diagram geeft de flow van hoe het systeem het antwoord van de speler wordt behandeld. Het systeem controleert eerst het ingevoerde antwoord. Mits deze correct is, geeft het systeem een van de magic letters terug.

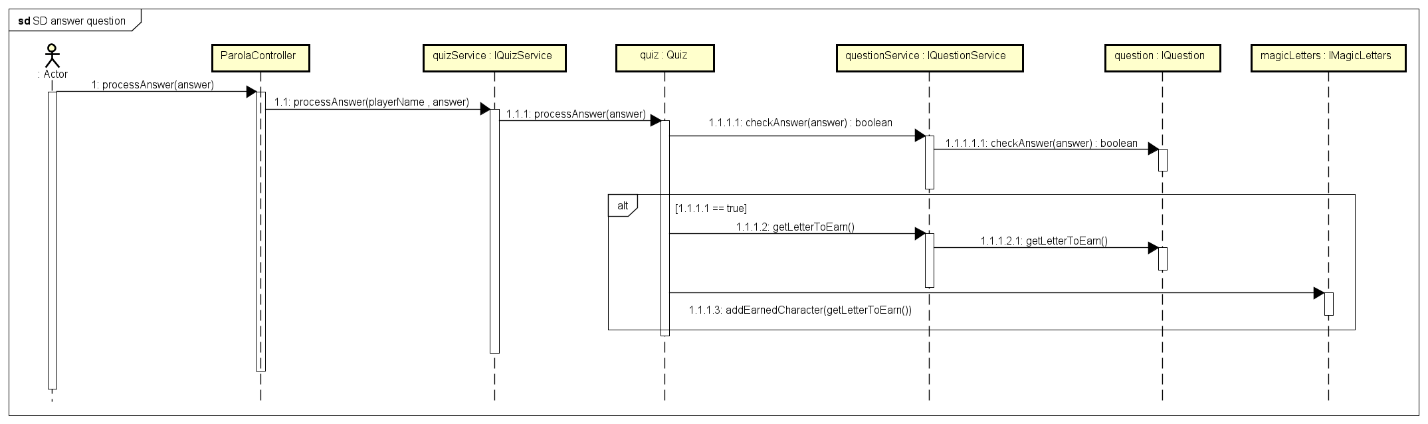


Figure 3: SD answer question

#### Calculate score.

Deze sequence diagram geeft de flow van hoe het systeem de score gaat bereken van de speler. Het systeem neemt controleert het ingevoerde magic word. Mits dit woord geldig is, wordt het gebruikt samen met de score strategy om de score te berekenen. Als het woord niet geldig is wordt deze niet meegegeven.

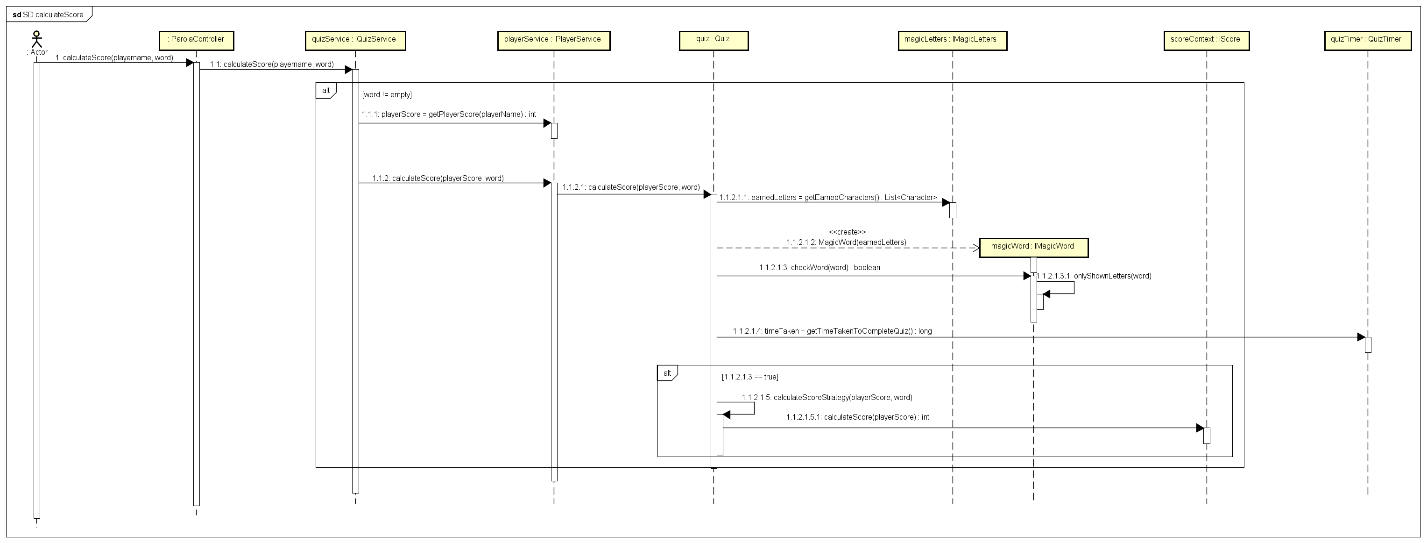


Figure 4: SD calculate score

#### purchase credits

Deze sequence diagram geeft de flow van hoe de speler credits kan kopen. Wanneer een speler een creditspackage selecteerd, wordt de speler verbonden met de externe betaalservice. Wanneer de betaalservice aangeeft dat de transactie succesvol is uitgevoerd, wordt het betreffende aantal credits toegevoegd aan de balans van de speler.

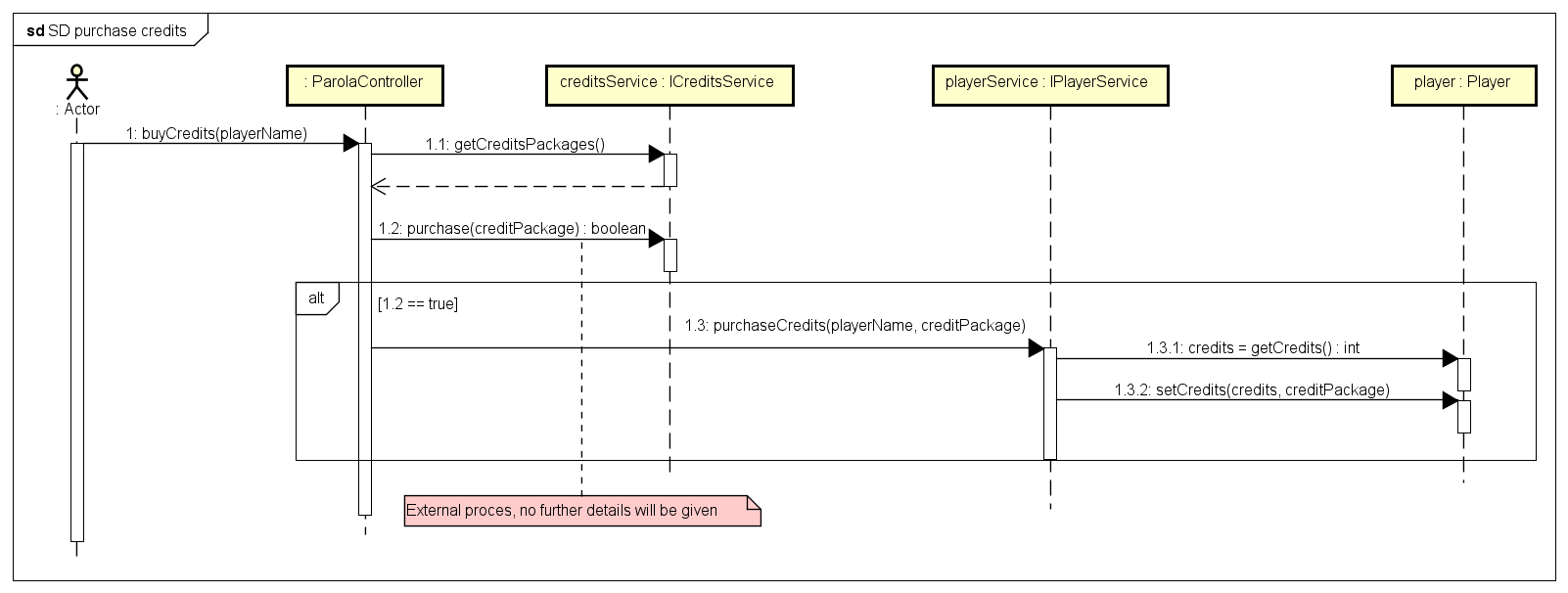


Figure 5: SD purchase credits

### Activity and State Diagrams

We hebben geen activity and state diagrams gemaakt voor deze applicatie.

### Design decisions made for the subsystems

| **Decision** | **Description** |
| --- | --- |
| Problem | Het uitgangspunt is dat de puntentelling flexibel moet zijn, d.w.z. er moet makkelijk overgestapt kunnen worden op een andere systematiek |
| Decision | Strategy Pattern toepassen |
| Arguments | * Het maakt het makkelijk om een nieuw puntentelling toe te voegen zonder de bestaande code aan te passen |
| Alternatives | * **Factory Patroon**: Dit zou kunnen worden gebruikt om verschillende scoring-algoritmes te creëren, maar het biedt niet dezelfde flexibiliteit voor het dynamisch veranderen van gedrag als het Strategy Patroon. * **Hardcoded if-else condities**: Dit zou minder flexibel zijn en zou leiden tot een grotere kans op fouten en complexiteit bij het toevoegen van nieuwe scoring-systematieken. * **Adapter Patroon**: Dit kan nuttig zijn als er verschillende interfaces zijn die ineen moeten passen, maar het is niet specifiek ontworpen voor het dynamisch wisselen van algoritmes of gedragingen. |

| **Decision** | **Description** |
| --- | --- |
| Problem | In de toekomst wil MEC Games ook versies in andere talen |
| Decision | Resource Bundles met Strategy Pattern toepassen |
| Arguments | * Het is gemakkelijk om nieuwe talen toe te voegen zonder de hoofdapplicatie te wijzigen. * Aanpassingen aan een specifieke taal hebben geen invloed op andere talen. * Het Strategy-patroon biedt flexibiliteit in de implementatie, waardoor het mogelijk is om verschillende vertalingsmechanismen te hebben indien nodig |
| Alternatives | * + Alleen **Resource Bundles**: Rely puur op Java's ingebouwde ResourceBundle mechanisme zonder het Strategy-patroon.   + Voordeel: Eenvoudiger te implementeren en beheren.   + Nadeel: Minder flexibel dan een combinatie met het Strategy-patroon. |

| **Decision** | **Description** |
| --- | --- |
| Problem | Het uitgangspunt is dat de puntentelling flexibel moet zijn, d.w.z. er moet makkelijk overgestapt kunnen worden op een andere systematiek |
| Decision | Factory Pattern toepassen |
| Arguments | * Dit maakt het gebruiken van een nieuwe soort vraag makkelijker omdat dan alleen de factory pattern zelf aangepast hoeft te worden. |
| Alternatives | * **Hardcoded if-else condities**: Dit zou minder flexibel zijn en zou leiden tot een grotere kans op fouten en complexiteit bij het toevoegen van nieuwe vraagsoorten * **Abstract Factory:** Vergeleken met een Factory Pattern, is een Abstract Factory verdeeld over meer classes, dit betekent dat het aanpassen van de abstract factory meer werk is met grotere complexiteit en grotere kans op fouten. |

### Design principles

#### SOLID

De volgende SOLID principles zijn toegepast:

* Single Responsibility
  + Iedere klasse heeft enkele verantwoordelijkheid: quiz klasse is bijvoorbeeld niet verantwoordelijk voor de speler, alleen voor de informatie die direct relevant is voor de quiz zelf.
* Open Closed Principle
  + Door gebruik van de strategy en factory patterns, is het relatief makkelijk om nieuwe score systeem of vraag types toe te voegen zonder dat de hele code-base herschreven hoeft te worden.

#### GRASP

De volgende GRASP principles zijn toegepast:

* Information Expert, Low coupling + High Cohesion
  + Door de implementatie van Single Responsibility (zie 3.2.5.1), heeft iedere klasse alleen de informatie die nodig is om de betreffende verantwoordelijkheid te vervullen. Dit leidt vanuit zichzelf tot low coupling en high cohesion.

## Database Design

Er is geen database gemaakt voor deze applicatie.

### Design decisions related to the database

Er zijn geen beslissingen gemaakt voor de database.