**Software Architectuur Document (SAD)**

Project: Eternity Wars

Teamleden: Mario Hendriks (studentnummer)  
 Wiebe van Herwijnen (studentnummer)  
 Nicky Janse (421118)

Inhoud

[H1 Inleiding 3](#_Toc30344894)

[H2 Systeem Context (C1) 4](#_Toc30344895)

[H3 Containers en technologiekeuzes (C2) 5](#_Toc30344896)

[H4 Componenten (C3) 6](#_Toc30344897)

[H5 Klassendiagrammen en sequence diagrammen (C4) 7](#_Toc30344898)

[Websocket communicatie Sequence Diagram 14](#_Toc30344899)

[H6 Persistentie per component 15](#_Toc30344900)

[H7 Specificatie van interfaces 17](#_Toc30344901)

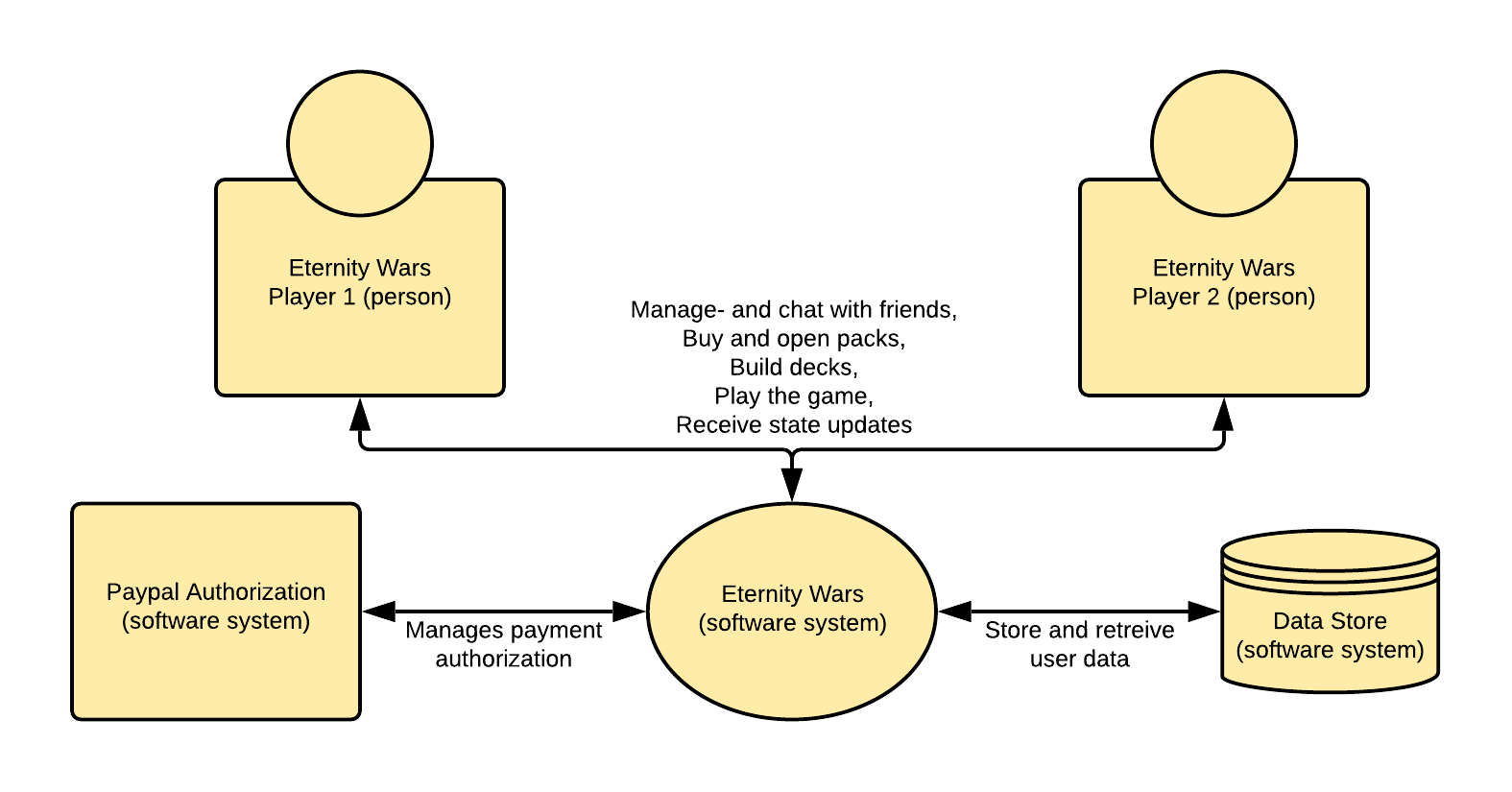
# H1 Inleiding

Eternity wars is een online card game voor twee spelers. Beide spelers hebben een hero met een aantal hitpoints en een stapel van kaarten. Deze kaarten kunnen met behulp van mana op het veld gespeeld worden, waarna ze de volgende beurt gebruikt kunnen worden om aan te vallen. Een speler heeft gewonnen als de hero van de tegenstanders 0 hitpoints heeft.

Het spelen van dit spel wordt mogelijk gemaakt door een lobby systeem. Via dit lobby systeem kunnen spelers hun vriendenlijst beheren, een spel starten, chatten, een deck maken, kaart pakketjes kopen en deze pakketjes open maken.

In dit document wordt duidelijk gemaakt hoe de architectuur van Eternity Wars eruit komt te zien. Dit wordt gedaan middels het C4-model, sequence diagrams en verdere uitleg over de communicatie van verschillende componenten onderling.

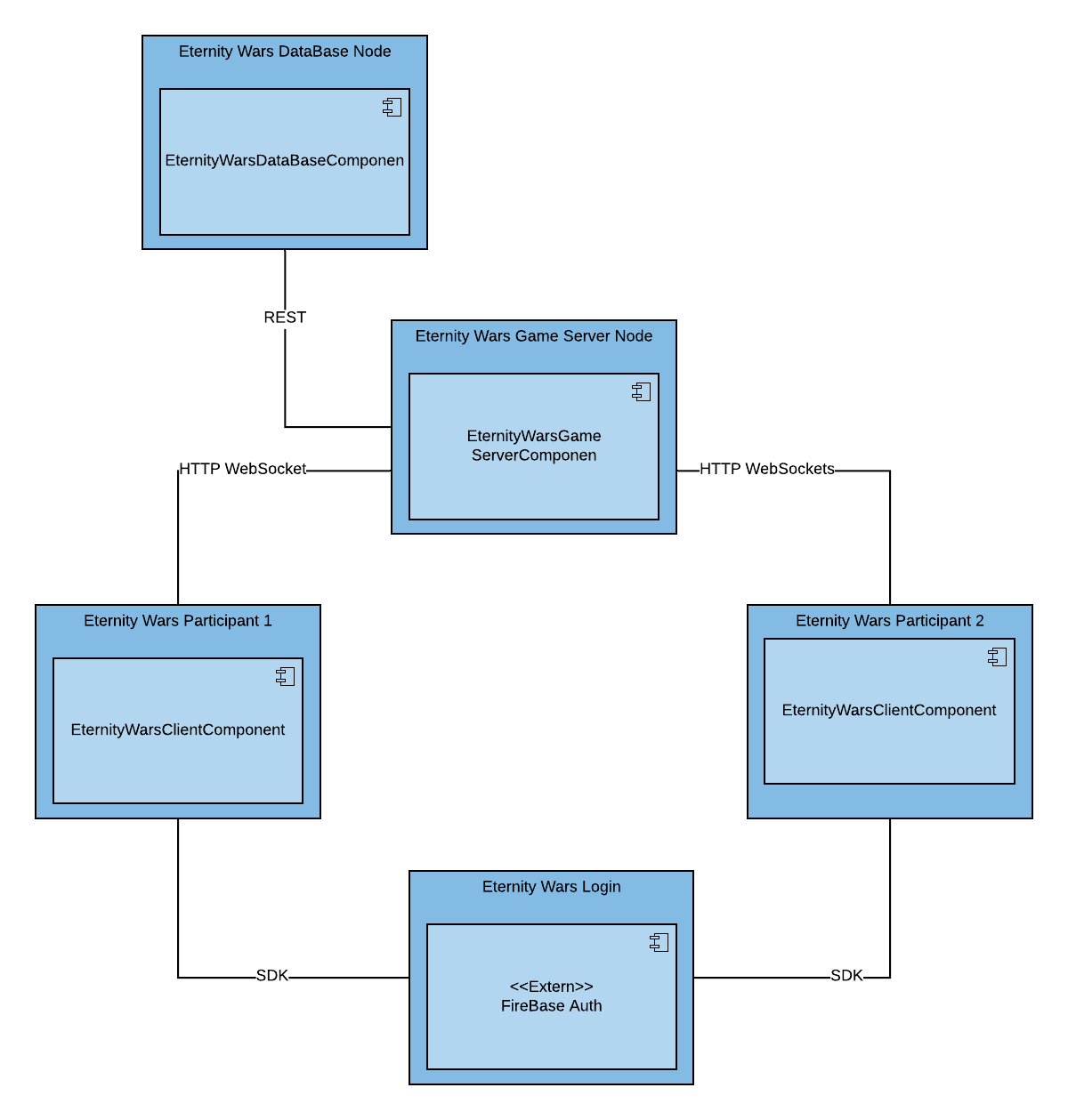
# H2 Systeem Context (C1)

In de onderstaande afbeelding is het C1-diagram van de Eternity Wars applicatie te zien. Deze applicatie maakt gebruik van Paypal Authorization service voor het autoriseren van betalingen en een datastore om speler gegevens op te slaan.

# H3 Containers en technologiekeuzes (C2)

Hieronder is ons C2 model te zien. We hebben gekozen voor de communicatiemiddelen REST en websocket. De reden dat we REST hebben gekozen is omdat het erg flexibel is. De data zit niet vast aan resources of methodes en dit zorgt ervoor dat er meerdere type calls gebruikt kunnen worden.

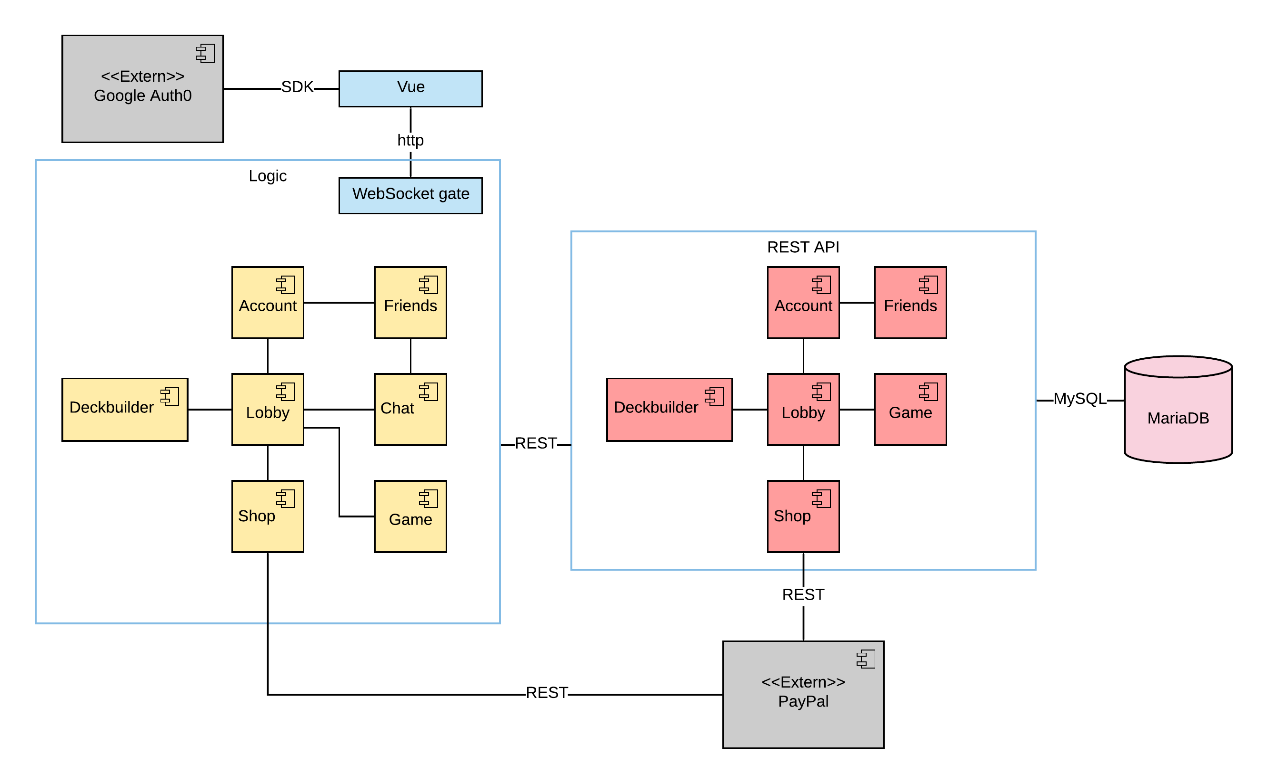
Ook kiezen wij HTTP WebSocket. De reden hiervoor is dat we een constante connectie willen krijgen vanuit de game server zodat beide spelers kunnen zien welke kaarten worden gespeeld.



# H4 Componenten (C3)

De onderstaande afbeelding bevat het C3-diagram van de Eternity Wars applicatie. Deze zal gebruik maken van een Vue front-end welke via http websockets communiceert met de logica binnen de applicatie. Vanuit de logica zullen er REST-calls gemaakt worden naar de REST API. Deze API zal met een MariaDB communiceren via MySQL.

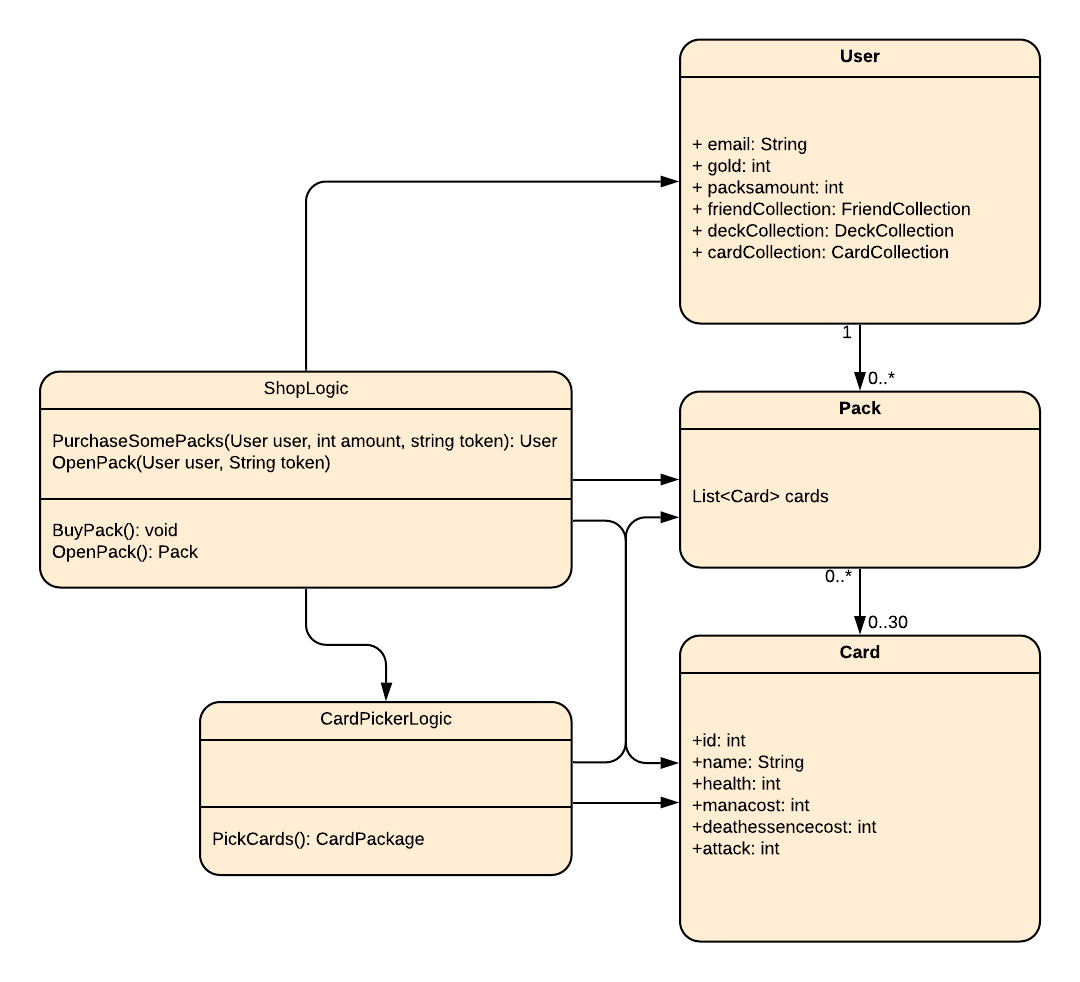
Ook wordt er gebruik gemaakt van twee externe softwaresystemen. Google Auth0 zal gebruikt worden voor de Google authenticatie om in te kunnen loggen in de applicatie. Om in de shop te kunnen betalen, zal er een PayPal checkout gebruikt worden.



# H5 Klassendiagrammen en sequence diagrammen (C4)

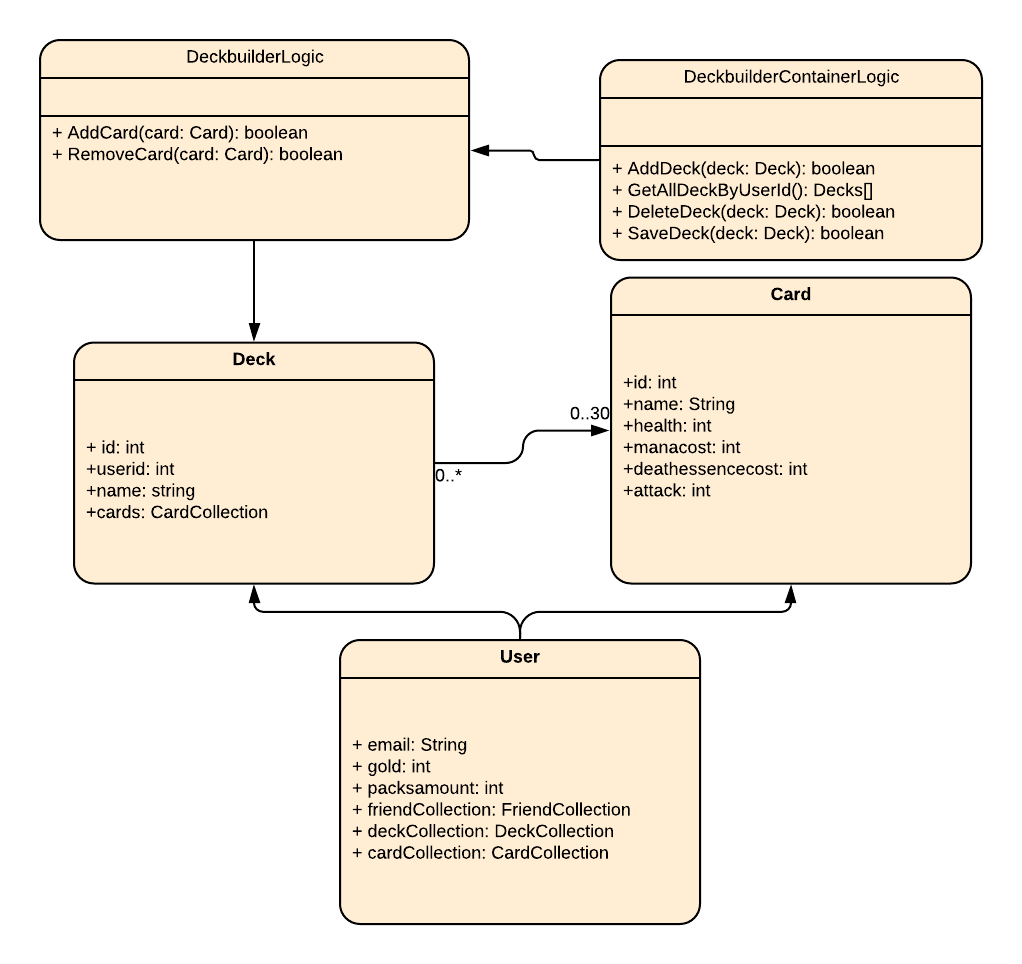
Voor de C4 hebben wij gekozen om het Logica gedeelte uit te werken van ons project. Deze bestaat uit meerdere stukken logica die op subjecten zijn gebaseerd. Hiermee voorkomen we dat methodes in de verkeerde classes worden geplaatst.

## shop logic



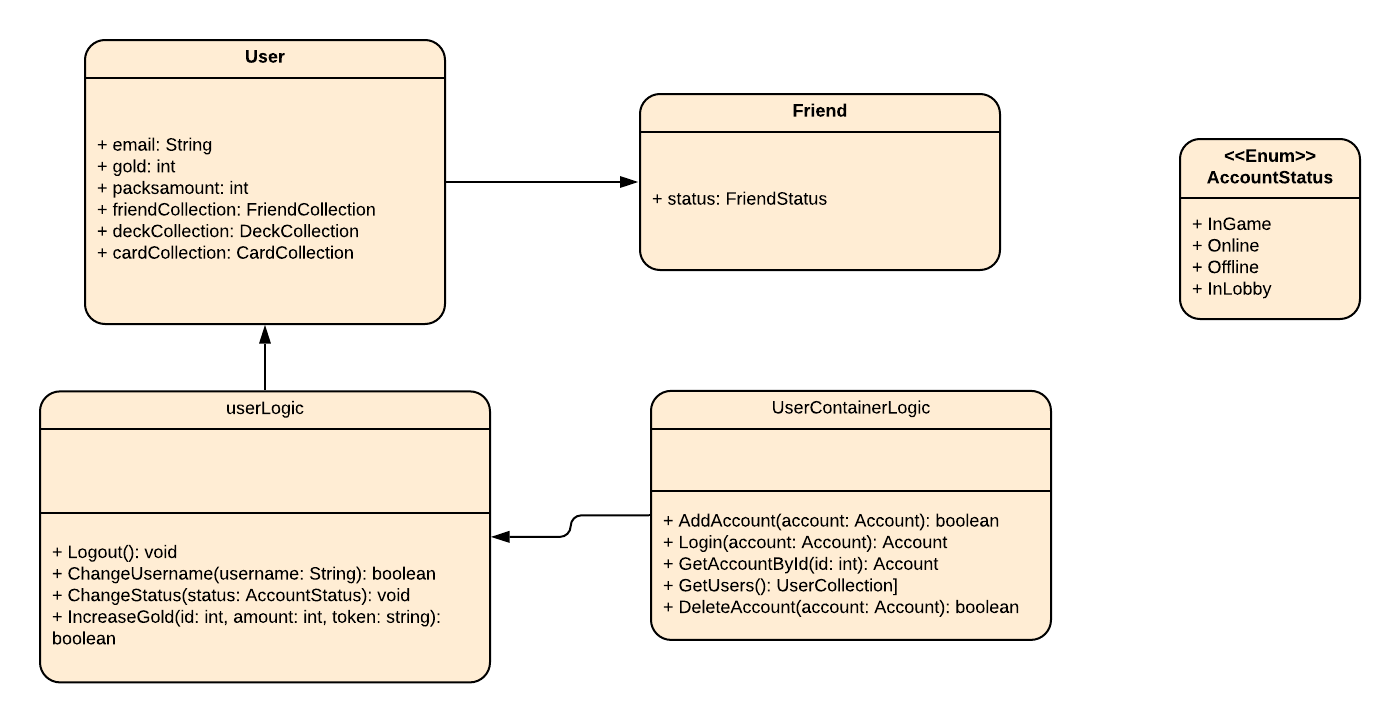
De shop logic is het stukje logica wat alles gerelateerd aan shop afhandelt. We zien hier de twee logic classes die methodes uitvoeren om het gewenste resultaat te krijgen, daarnaast zijn er ook deze modellen bij nodig.

## DeckBuilder logic



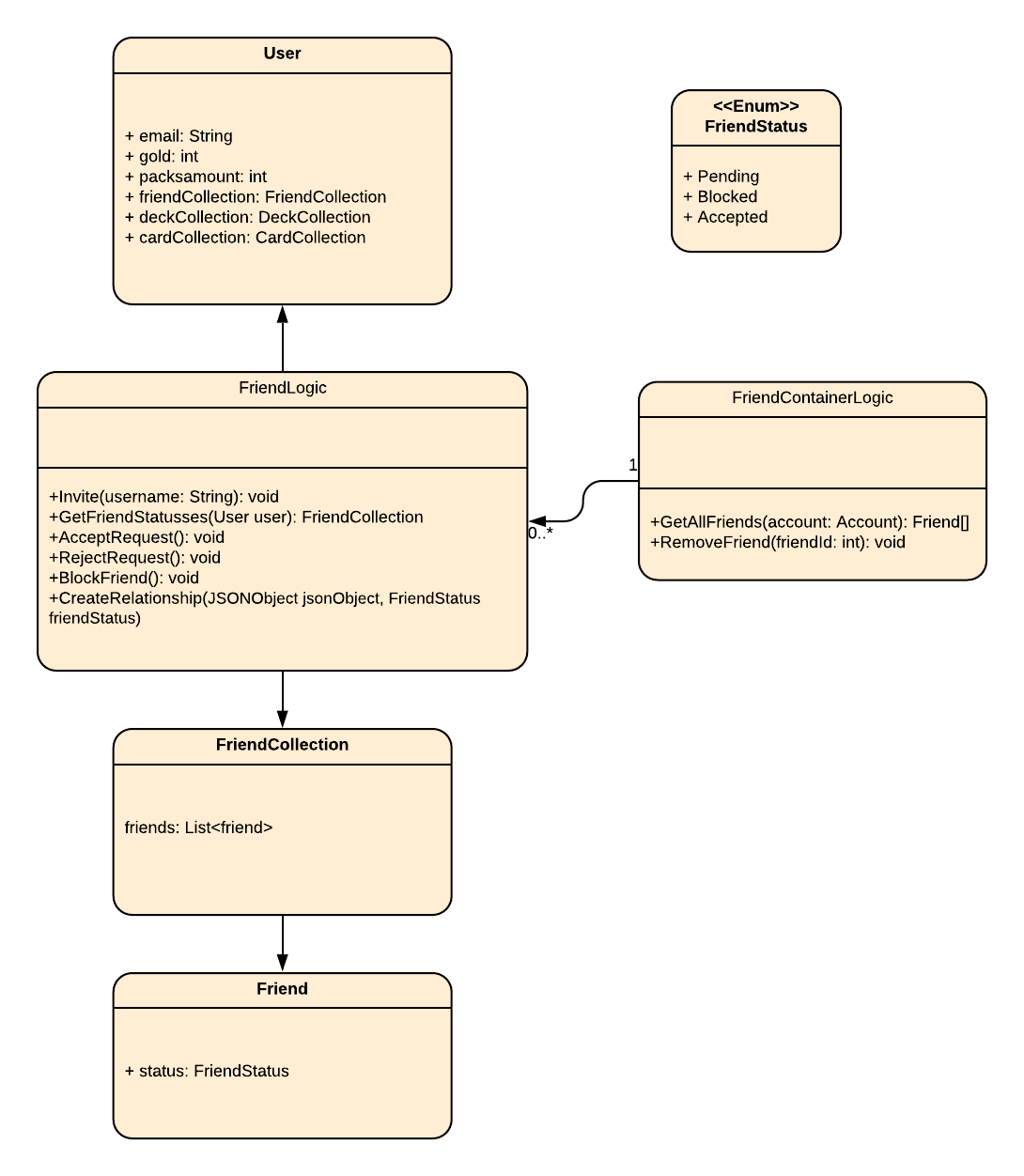
De deckbuilder logic zorgt ervoor dat deck gemaakt en aangepast kunnen worden. we hebben hier een logic voor een specifiek deck aan te passen(deckbuilderlogic), en een logic die deze decks kan maken en oproepen(deckbuildercontainer logic).

## User Logic



De User logic is vooral rond het account van de spelers gebaseerd. Ook hier hebben we een logic die voor specifieke users logica afhandelt en voor meerdere.

## Friend logic



De friend logic is gemaakt om relaties tussen users af te handelen, ook hier kijken we naar logica voor specifieke relaties, en om er meerdere tegelijk te verkrijgen.

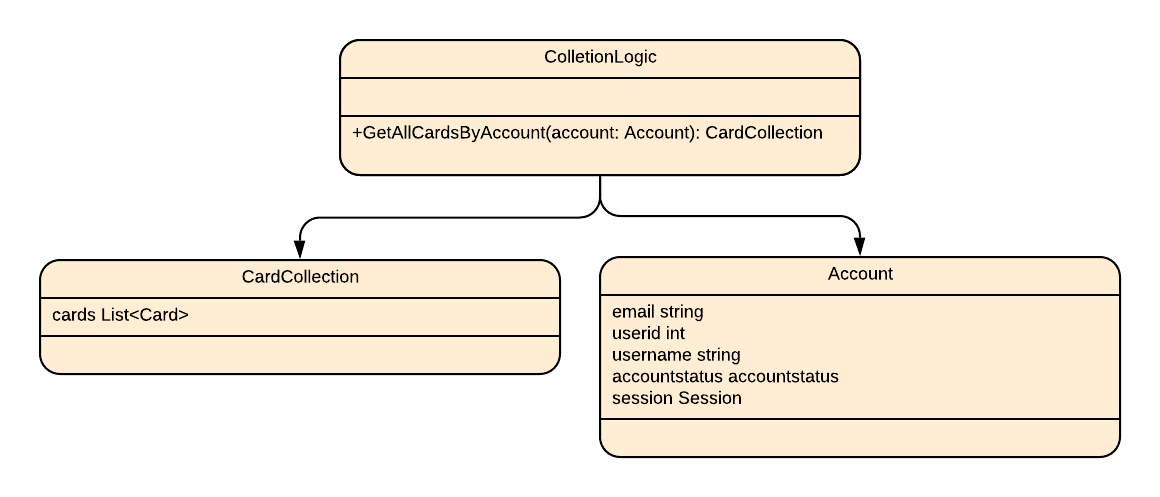
## Lobby logic

De lobby logic is gemaakt om de lobby’s te handelen die gedurende de game worden aangemaakt. Hierin wordt en aanmaken van een lobby afgehandeld en specifieke aanpassen in een lobby zoals het aanmaken van een naam.

## Game logic

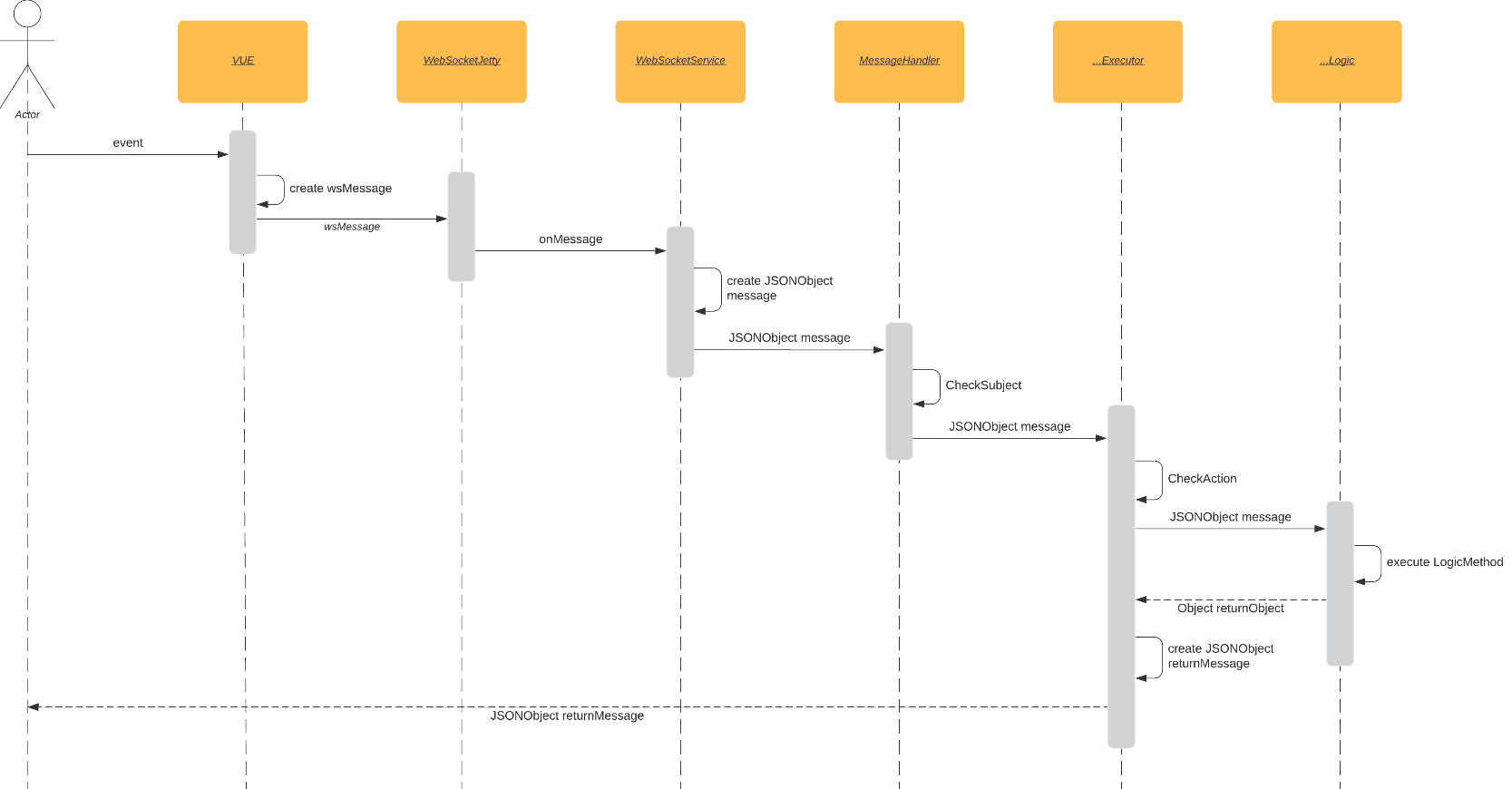
De Game logic zorgt ervoor dat de kaarten worden geplaats en dat ze doen wat ze moeten doen. Hier wordt met behulp van models data terug gestuurd naar de front-end om vervolgens daar de game levend te maken.

## Collection logic



De collection logic is gemaakt om de collectie van kaarten van een user af te handelen. Deze werkt vooral samen in combinatie met de deckbuilder.

# Websocket communicatie Sequence Diagram

Het onderstaande sequence diagram beschrijft de communicatie tussen de VUE front-end client en de Java back-end logic.

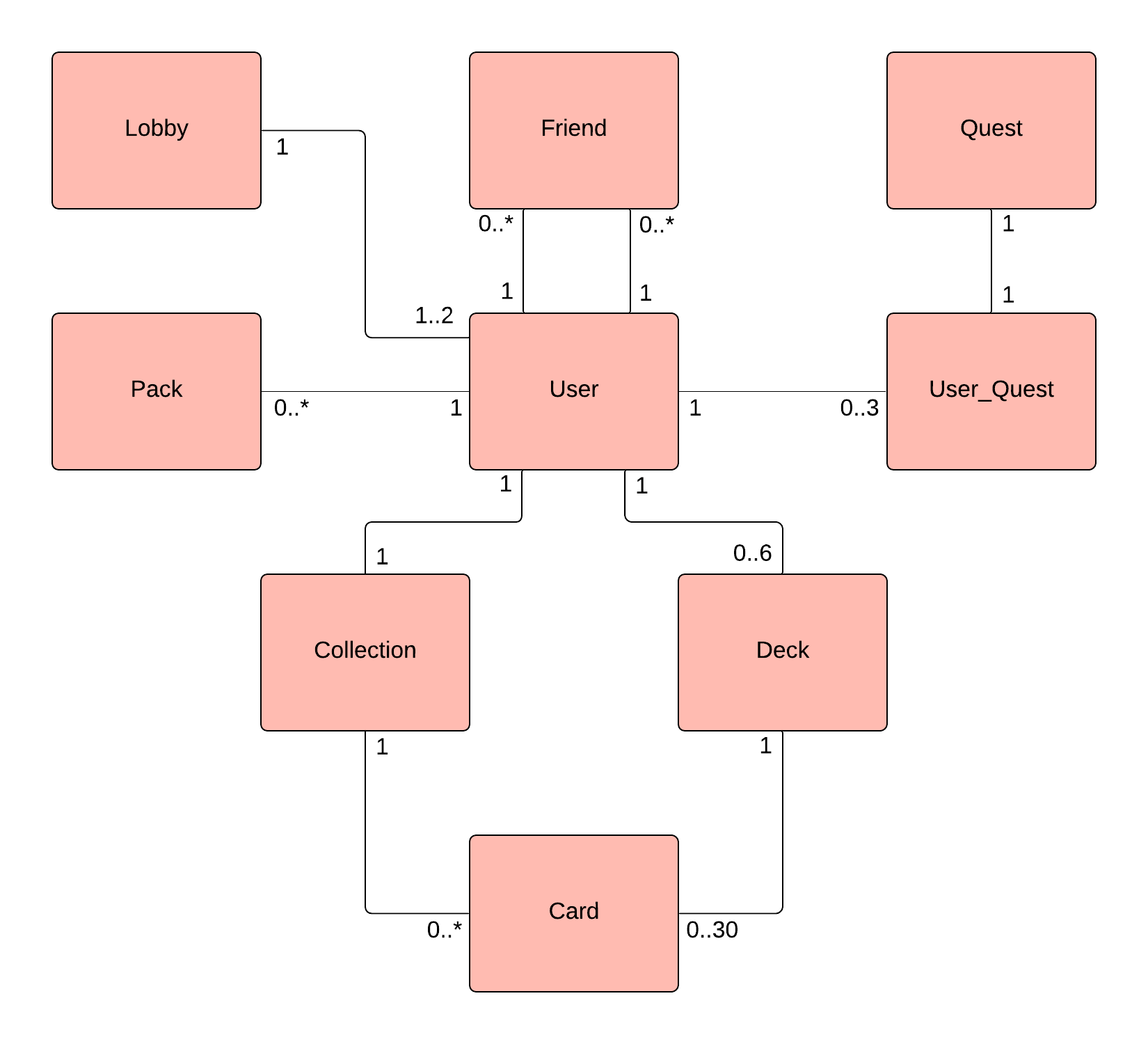
Event beschrijft het ophalen of aanpassen van data. Hier wordt in de VUE client een wsMessage object van gemaakt met de properties: Subject, Action, Content en Token. Dit wsMessage model wordt in JSON formaat doorgestuurd naar het logica systeem.

Subject beschrijft welk deel van de logica aangesproken moet worden. Hier wordt door middel van een switch case op gecontroleerd in de MessageHandler. Deze maakt vervolgens een instantie van Executor aan afhankelijk van het Subject.

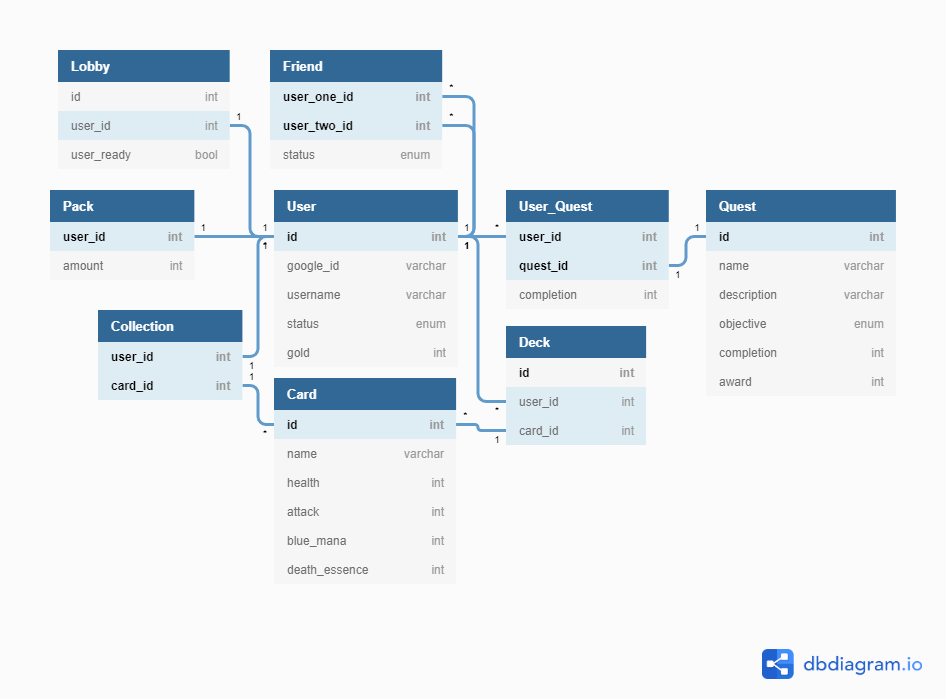
Deze Executor controleert op de Action uit de wsMessage en spreekt de bijbehorende methodes uit de logica aan. Afhankelijk van deze methode wordt er een REST-call gemaakt naar de API of wordt er een check uitgevoerd in de logica laag.

De logica laag returned vervolgens een object of boolean welke, omgezet in een wsMessage, als JSONObject weer teruggestuurd wordt naar de VUE client.

# H6 Persistentie per component

In de onderstaande afbeelding is het ERD van Eternity Wars te zien met alle onderlinge relaties.

Vervolgens is er een database diagram gemaakt bij het ERD. Het DBD bevat per tabel alle primary keys, fields, welk type deze fields zijn.



Het opslaan van objecten word op verschillende plekken gedaan in het systeem. In de meeste gevallen zal er een request met data van de front-end komen. In dit bericht staan een aantal properties. Op basis van deze properties weet de logica welke functie er aangeroepen moet worden. De logica stuurt op zijn beurt een request naar de bijbehorende API functie. Hierdoor zal de data of opgeslagen worden in de database of er word data opgehaald uit de database op basis van de data die mee werd gestuurd.

# H7 Specificatie van interfaces

Binnenkomende en uitgaande communicatie van de REST API is gedocumenteerd met gebruik van Swagger. Deze is via de onderstaande link te bereiken. Om de link te bezoeken is het wel nodig om de API lokaal te draaien op poort 8083, omdat EternityWars (nog) niet gehost is.

<http://localhost:8083/swagger-ui.html>

Swagger beschrijft alle endpoints met bijbehordende http mehtods, parameters en parametertypes en responses. Ook zijn alle models aanwezig in de API te zien via Swagger.

De websocket maakt op basis van het JSON object een keuze over welke subject het gaat. Dit kan een van de volgende subjects zijn: ‘Game, User, Lobby, Deck, Friend, Shop, Card, Collection, Register, Chat’. Vervolgens word er op basis van de Action bepaald welke functie er word uitgevoerd, dit verschilt per subject. Als laatst word de content gebruikt om de parameters te vullen van de functie die word aan geroepen. De token die mee word gestuurd word gebruikt voor een request te maken naar een van de private API-endpoints die anders niet te bereiken zijn.

Hieronder ziet u hoe het JSON object die de front-end stuurt is opgebouwd.

**WsMessage = {**

**"Subject": “String",**

**"Action" : "String",**

**"Content": "Object",**

**“Token" : "JWT",**

**};**