Gedistribueerde Systemen

UNO

Axel Vulsteke

Benoît Masschelein

Technologiecampus gent

Inhoudsopgave

[1 Inleiding 3](#_Toc501311434)

[2 Architectuur 3](#_Toc501311435)

[3 Databank 4](#_Toc501311436)

[4 Ontwerpbeslissingen 4](#_Toc501311437)

[4.1 Consistentie en Replicatie 4](#_Toc501311438)

[4.2 Caching 4](#_Toc501311439)

[4.3 Beveiliging 4](#_Toc501311440)

[4.4 Recovery 4](#_Toc501311441)

[5 API 4](#_Toc501311442)

[5.1 Dispatcher 4](#_Toc501311443)

[5.2 Applicationserver 4](#_Toc501311444)

[5.3 Databaseserver 4](#_Toc501311445)

[6 Reflectie 4](#_Toc501311446)

[6.1 Sterktes 4](#_Toc501311447)

[6.2 Zwaktes 4](#_Toc501311448)

[6.3 Uitbreidingen 4](#_Toc501311449)

# Inleiding

Eenvoudig om op te pikken, maar moeilijk om weg te leggen. Uno is 's werelds meest gespeelde gezinskaartspel. Voor het vak gedistribueerd systemen wordt er een een digitale versie van dit eenvoudig maar prachtig spel geïmplementeerd. Het spel wordt online gespeeld, er worden spelers op andere clients uitgedaagd. Aangezien dit gebeurt voor het vak gedistribueerde systemen wordt er rekening mee gehouden dat de applicatie een killer-applicatie kan worden. Servers worden gedupliceerd en er wordt nagedacht over het cachen van gegevens zodat het aantal requests op servers ingeperkt wordt. Een goede architectuur wordt essentieel. De databank krijgt een doordachte structuur, ook wordt er nagedacht over hoever de gegevens op verschillende databanken consistent moeten zijn en hoe die consistent zullen gehouden worden. Hoe groter het systeem, hoe groter de kans op fouten. Kan het systeem terug in een stabiele toestand geraken na de crash van een van de servers? Welke gegevens zijn er verloren gegaan? Verderop in dit verslag vindt u een antwoord op deze vragen en lichten we andere beslissingen toe. Er wordt kritische gekeken naar dit systeem in een sterkte-zwakteanalyse, evenals de uitbreiding die mogelijk zijn om er een state-of-the-art-applicatie van te maken.

# Architectuur

Onderaan de architectuur hebben we een databank. De inhoudt van de databank wordt later besproken. De databank wordt beheerd door een databankserver, dit is nodig omdat we meerdere databanken hebben, zo kunnen we requests verspreiden over meerdere databanken. Deze databankservers staan in verbinding met één applicatie server die gegevens zal opvragen via RMI aan de databankservers. Deze zal op zijn beurt gegevens ophalen uit zijn persoonlijke databank. De gegevens op deze databanken moeten echter op een bepaalde manier gesynchroniseerd worden, dit is een tweede taak van de databankservers. Alle databankservers staan met elkaar in verbinding via RMI, databankenservers zullen ervoor zorgen dat de nodige gegevens in iedere databank terechtkomt.

Boven de databankservers hebben we applicationservers, zoals de naam doet vermoeden zal deze de core functie van het UNO-spel bevatten. De applicationserver zal eerst en vooral de login verwerken. Daarna kom je terecht in de lobby, iedere applicationserver bevat een lobby. In deze lobby bevinden zich alle spelen die op een specifieke applicationserver te vinden zijn. Ook worden spellen getoond die op andere servers te vinden zijn, maar dan moet de speler verbonden worden met een andere applicationserver als hij een spel uit een andere lobby wil spelen. Wanneer een spel opgestart wordt, maakt de speler verbinding met de juiste applicationserver. Om deze spellen te kunnen delen staan alle applicationservers ook in verbinding met elkaar.

De applicationserver houdt de status van het spel bij. Hij zal kijken of een kaart die gelegd wordt op de stapel past, hij kan controleren of de juiste speler wel probeert te leggen, hij kijkt of die speler wel effectief die kaart heeft, … Het idee achter de architectuur is om enkel de nodige informatie te sturen naar de speler zodat hij het spel kan weergeven, alles van ‘backend’ wordt in de applicationserver geschreven. Bepaalde eenvoudige checks worden ook uitgevoerd op de cliënt zodat het aantal requests niet overdreven groot wordt. Zo zal de client ook bijvoorbeeld al kijken of de juiste speler wel probeert te leggen, anders wordt de requests niet doorgegeven. Er is dus een dubbele veiligheid ingebouwd. Het is dus een afweging tussen het beperken van calls en het behouden van een mooie architecturale splitsing.

Nu alle code aanwezig is om een spel te spelen, moeten we dit natuurlijk nog visualiseren. Iedere speler kan een client applicatie opstarten. Eerst en vooral moet ingelogd worden, iedere speler kan slechts voor een bepaalde tijd ingelogd blijven. Daarna komt de speler in de lobby terecht, hier kan hij alle spellen zien van alle applicationservers. Hij kan hieruit een spel kiezen of zelf een spel aanmaken. Wanneer hij een spel kiest, wordt verbinding gemaakt met de juiste applicationserver. Eenmaal het juiste aantal spelers bereikt is, kan het spel gestart worden. Het spel bevat 2 speciale kaarten: omwisselen van richting, en een speler overslaan. Wanneer een spel ten einde is wordt teruggekeerd naar de lobby, waar de totale scores van alle spelers te zien zijn. Een speler kan na een bepaalde tijd geen nieuwe spellen meer starten en zal bij het proberen starten van een nieuw spel automatisch uitgelogd worden. Er kan ook handmatig uitgelogd worden.

# Databank

Er wordt gebruik gemaakt van de SQLite technlogie voor het opzetten van de databank. Er bevinden zich twee tables in iedere databank. De ene houdt alle gegevens van de spelers bij, terwijl de andere de afbeeldingen opslaat van de kaarten om weer te geven bij de client.

Iedere entry in de spelers table stelt een speler voor. Deze speler heeft een unieke username, een hash van zijn paswoord, een salt, sessiontoken, timestamp en een score. De username moet uniek zijn aangezien we tijdens het inloggen zoeken naar één user. Dit wordt bij het aanmaken van een nieuwe user ook gecontroleerd. Iedere user heeft een paswoord, dit wordt niet zomaar plain-text opgeslagen. In de applicationserver wordt voor iedere nieuwe speler, met zijn paswoord een hash gemaakt. Voor de veiligheid wordt hier een salt aan toe gevoegd die verschillend is bij iedere user. Deze twee byte array’s worden opgeslagen als BLOB. Als iemand inlogt met een username en passwoord wordt voor die gebruiker zijn hash en salt opgehaald. Aan de hand van die salt kan het ingegeven paswoord gehashed worden. Komen de twee hashes overeen is de speler succesvol ingelogd. Telkens de speler een nieuw spel wil spelen wordt er gecontroleerd of deze speler niet langer dan 24 uur actief is. Daarom wordt voor iedere speler een timestamp bij gehouden. Deze bevat de tijd wanneer de speler laatst succesvol heeft ingelogd. Bovendien wordt er een sessiontoken gegeneerd bij iedere login. Deze sessiontoken is uniek voor iedere speler. Hiermee kan de speler zich doorheen het spel identificeren bij de applicationservers. Als een speler zicht uitlogt wordt deze sessiontoken verwijderd. Tenslotte wordt voor iedere speler zijn score bij gehouden. Telkens we een lijst van de beste spelers willen weten wordt een query op de database uit gevoerd waarbij de spelers geordend worden op score en waarbij de beste 50 terug gegeven worden. Deze lijst kan weergegeven worden in de lobby.

De andere table houdt afbeeldingen bij van kaarten. De mogelijkheid is geïmplementeerd waarbij speciale gelegenheden speciale versies van kaarten te voorzien. Deze table bestaat uit drie kolommen: BLOB van een afbeelding, string die aangeeft voor welk event de kaart is en een naam van de kaart. Momenteel zijn er twee events: *standard* of *christmass*, maar dit kan eenvoudig uitgebreid worden. Telkens een user inlogt worden alle kaarten naar de client over gebracht via RMI, deze afbeeldingen worden bij de client op zijn apparaat geplaatst. De huidige datum wordt bekeken, en afhankelijk hiervan worden de juiste kaarten terug gegeven. Om het aantal requests te beperken naar de database worden de kaarten van de huidige periode gecached bij de applicationserver. Tenslotte heeft iedere entry een naam. Dit is de naam van de kaart die gebruikt wordt om de afbeeldingen op te kunnen slaan aan de client zijde. Die kaarten kunnen vervolgens tijdens het spel opgehaald worden aan de hand van de correcte naam.

//hoe blijven databanken synchroon..?

# Ontwerpbeslissingen

## Consistentie en Replicatie

## Caching

## Beveiliging

Er wordt verondersteld dat de communicatiekanalen veilig zijn. De client is zeer lightweight, doordat we confidentiële communicate gebruiken mogen we de paswoorden van de client naar de applicationserver plain door sturen. We mogen niet veronderstellen dat de database veilig is, daarom worden de paswoorden gehashed met een salt. Bij iedere user hoort ook een salt. Telkens een user wil inloggen controleren we aan de hand van deze salt de hash. Iedere speler krijgt ook een sessiontoken terug, deze wordt opgeslagen aan de client zijde. Eenmaal de gebruiker ingelogd is, kan hij deze sessiontoken meesturen, zodat de applicationserver dit kan gebruiken als message authentication. We hebben dit echter niet overal gedaan.

## Recovery

# API

## Dispatcher

## Applicationserver

## Databaseserver

Hier worden de API’s overlopen die de databaseserver aanbiedt.

* User getUser(username): geeft alle gegevens van een user terug uit de database met die username
* Boolean addNewUserToAllDatabases(username, salt, hashedpswd): stopt de nieuwe gebruiker in de database en geeft acknowlegment terug als alles goed ging.
* Void createSessionToken(username): maakt een sessiontoken en stopt die in de database.
* Void setScoreOnAllDatabases(score, username): zorgt ervoor dat in alle databases de scores geüpdatet worden voor die user.
* List<User> getBestPlayers(): geeft een lijst terug met de 50 beste spelers op dit moment.
* Void logout(username): zorgt er voor dat de sessiontoken verwijdert wordt uit de database als de speler uitgelogd wordt.
* Integer getSessionToken(username): geeft de sessiontoken terug voor deze user.
* List<Picture> getCards(event): geeft de juiste kaarten terug afhankelijk van de opgegeven gelegenheid.

# Reflectie

## Sterktes

* Stabiel:
* Overzichtelijke UI
* Veilig opslaan van user gegevens
* Goed gebruik gemaakt van caching
* Goede architectuur
* Mogelijkheid van speciale kaarten..
* Overzichtelijke logging en terminals…

## Zwaktes

* Dispatcher is spof en bottleneck
* Geen spectator mode
* Niet genoeg rekening gehouden met failures 🡪 wel zeer complex
* Meerdere keren inloggen met hetzelfde account

## Uitbreidingen

* Spectator mode
* Stoppen tijdens het spelen van een spel
* Betere recovery?
* Uitvallen van databases simuleren