Produktrapport Kortklubben



[Produktrapport Kortklubben 1](#_Toc105506604)

[Kravspecifikation 3](#_Toc105506605)

[Brugervejledning 3](#_Toc105506606)

[Server 3](#_Toc105506607)

[Root server: 3](#_Toc105506608)

[Database (Virtuelle server 1): 3](#_Toc105506609)

[Anvendelse: 3](#_Toc105506610)

[Inbound rule: 3](#_Toc105506611)

[Outbound Rule: 4](#_Toc105506612)

[Game (Virtuelle server 2): 4](#_Toc105506613)

[Anvendelse: 4](#_Toc105506614)

[Overvejelser/beslutninger 4](#_Toc105506615)

# Kravspecifikation

Der er taget udgangspunkt efter FURPS og er den der vil blive refereret til igennem dokumentet.

Der er blevet lavet lidt om på use-case 4 ift. at se sin historik. I stedet så er der et leaderboard, hvor man ville kunne se sine egne stats 🡪 vundet, tabt og spil spillet, samt top x spillere.

# Brugervejledning

## Server

Der anvendes en server, som har 2 virtuelle servere til at administrere og køre vores system. Her er følgende informationer:

### Root server:

Type: x64-baseret computer

* Fabrikant: LENOVO
* Model: 10M9000CMT
* CPU: 4 x Intel® Core™ i7-7700 CPU @ 3.60GHz
* Hukommelse: 32 GB
* Opbevaring: 350 GB

### Database (Virtuelle server 1):

* Host navn: ZBC-RG-DBSERVER-2022
* Version: Microsoft Windows Server 2022 Standard
* Type: x64-baseret computer
* CPU: 2 x Intel® Core™ i7-7700 CPU @ 3.60GHz
* Hukommelse: 8 GB
* Opbevaring: 90 GB

#### Anvendelse:

På **Database serveren** er der lagt en SSMS (SQL Server Management Studio) på, samt SCCM (System Center Configuration Manager). Derudover er der åbnet for nogen port.

##### Inbound rules:

* Type: TCP
* Lokale port: 1433
* Remote port: Alle porte

Bruges til at åbne op for SQL connection.

###### Outbound Rule:

###### 1.

* Type: UDP
* Lokale port: 1434
* Remote port: 1434

### Game (Virtuelle server 2):

* Host navn: ZBC-RG-GAMESERVER-2022
* Version: Microsoft Windows Server 2022 Standard
* Type: x64-baseret computer
* CPU: 2 x Intel® Core™ i7-7700 CPU @ 3.60GHz
* Hukommelse: 8 GB
* Opbevaring: 90 GB

#### Anvendelse:

På **Game serveren** er der lagt Node.js og PM2 (Process Management).

##### Inbound rules:

###### 1.

* Type: TCP
* Lokale port 3000
* Remote port: Alle porte

Bruges til at åbne op for API kommunikationen.

* Type: TCP
* Lokale port: 5000
* Remote port: Alle porte

Bruges til at åbne op til WebSocket kommunikationen.

# Overvejelser/beslutninger

Som udgangspunkt da vi begyndte projektet var planen at vi henholdsvis koder vores spil-engine i Java og har den kørende på en Glassfish 5 server.

Under projektforløbet var vi blevet enig om at have daglige standup meetings lidt i retningen af SCRUM, vi tog brug af kanban boards. For at holde bedre styr på hvor langt vi var nået i game-engines udviklingsproces. Formålet ved dette var vi arbejdet agilt. Hvor vi endda også fik taget brug af pair-progamming.

Nogle af de overvejelser vi dvælet meget med, var omkring hvilke nogle af use-casene der var relevante for spil-engine og klienterne. Som for eksempel Use-case 7 (Check bilag). Handlingen der ligger bag ved at sortere kort, fandt vi ikke til at have en relevans for vores spil-engine. Da spil-engine bare skal vide hvilket kort den skal spilles på bordet og ikke i hvilken rækkefølge det håndteres.

Kort tid efter opstart støder vi derfra på et problem med Glassfish server og hvordan den håndterer Websocket endpoint. Derfra valgte vi at skifte fra Glassfish over til Tomcat.

En af de næste større beslutning var omkring håndtering af kommunikation mellem klienter og serversiden ude fra spillet, hvor første forslag der lå på bordet var at ALT kommunikation foregik over websocket server. Men efter nærmere overvejelser begrænsede vi Websocket ned til kun at kommunikationen kun foregår under spil. Imens alt andet kommunikation med serveren vil foregå over en RestApi ved GET/POST statements.

Den næste større beslutning der skulle tages, var at for hvilken form format af tekst vi ville sende og kommunikere med Api og Websocket på. Vores eget foretrukket valg, blev vi førte en JSON format som følgende {”game”: ”whist”, ”command”:”start”,”info”:{}}

Ideen herover ligger simpelthen i at få en nem og simpel tekst der kan konverteres til JSON-objekt der kan håndtere beskeder mellem klienter og spil-engine. Og kort fortalt om de forskellige nøgler i JSON-Objektet.

* **GAME:** Er en måde at sikre det faktisk er det spil som spilles, kommandoen er beregnet til.
* **COMMAND:** Er hvilken kommando der skal udstedes på enten spil-engine eller klient. Som for eksempel: *start*: er til når de spiller der skal deltage i spillet er inde, så kan brugeren starte spillet.
* **INFO:** Er den del der vil håndtere noget af de ekstra data der skal sendes med som for eksempel: Klienten spiller et kort, så sendes der *”info”:{”symbol”: ”heart”, ”number”: 1}.* info er fleksibel i den hvis der bare skal fyldes ind efter hvad kommando man har valgt, der fra kan der fyldes ind med de data der skal sendes. En af de to slut punkter, hvis de har kommandoen i deres håndtering vil de udføre kommandoen. Hvis de ikke har vil de returnere en fejl meddelelse.

Den største beslutning vi fik taget var da vi tog det valg at ændre fra at kører vores spil-engine. Fra Java over til typescript og node. Samt fra Tomcat til node.js PM2. Det var en beslutning vi valgte at tag da vi desværre var udsat for store vanskeligheder med Tomcat og måden den håndterede sin API, samt hvordan Java EE webservlet, fungere med andre klassere og objekter. For at reflektere over denne beslutning var det nok det bedste valg hvis vi var startede ud med typescript og node, eller muligvis et helt tredje sprog. Men da vi tilbage i starten af projektet valgte at udvikle i Java. Tænkte vi at de var simpelt nok, og da det minder meget omkring andre objekt-orienteret sprog vi ellers har arbejdet i. Denne beslutning har derfor været den mest omfattende, og kode kvaliteten er op for diskussion som en selvfølge i forhold til hvis vi havde startet direkte ud med typescript og node, i stedet for at ligge x antal timer i først at skrive det i Java for efter at oversætte det til Typescript. Og opdatere dokumentation i en meget større omfang i forhold til hvis det var udviklet efter den beregnet hensigt.

# Test rapport

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case**  **Description** | **Test Data** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| TU01 | Check if a card is the right value | Card1: Diamond, 2  Input: 2 | Is the same number | Same number | Pass |
| TU02 | Check if 2 cards have the same symbol | Card1: Spades, 5  Input: Clubs | Is not the same symbol = false | Not same | Fail |
| TU03 | Check if 2 cards have the same symbol | Card1: Diamond, 5  Input: Diamond | Is the same symbol | Same symbol | Pass |
| TU04 | Check if a card has a higher number | Card1: Spades, 8  Input: 5 | Is higher than input | Is higher | Pass |
| TU05 | Check if a card has a higher number | Card1: Spades, 2  Input: 7 | Is not higher than input | Is not higher | Fail |
| TU06 | Check if a dealer can shuffle a card deck | Input: Standard Card Deck | Deck should be shuffled | Is shuffled | Pass |
| TU07 | Check if players have received x cards to their hand | Input: Standard Card Deck, 4 players | Players should get 13 cards each | Each player got 13 cards | Pass |

# Bilag

|  |  |
| --- | --- |
| **Use-case** | Sorter kort |
| **Id** | 7 |
| **Version** | 1.0 |
| **Aktør(er)** | Spiller |
| **Trigger** | En spiller ønsker at sortere sine kort |
| **Pre-condition** | En spiller er i gang med et spil |
| **Post-condition** | Spilleren har sorteret sin hånd efter eget ønske |
| **Beskrivelse** | En spiller ønsker at sortere sine kort efter en bestemt rækkefølge ud fra hans eller hendes præferencer |