Professionshøjskolen UCN
IT-Uddannelserne
Datamatiker
Gruppe 3

IronPlay

Teknologi- og Programmeringsrapport

Nikoline Halbak Mejer Christensen, Rasmus Skovborg Jensen, Thomas Mosbæk Jørgensen, Oliver Guldvang Koch, Carl Christian Brøkmann Rasmussen

29-05-2024

Professionshøjskolen UCN

IT-uddannelserne

Datamatiker

DMA-CSD-V23, gruppe 3

Vejledere: Karsten Jeppesen og Lars Landberg Toftegaard

Anslag: 47.364 Sider: 19,7

Bilag: 8

Database: Production: DMA-CSD-V23_10485521

Testing: DMA-CSD-V23_10478735

Repository path: https://github.com/RadenDK/ThirdSemesterProject.git

Antal commits: 571

Nikoline Halbak Mejer Christensen

Rasmus Skovborg Jensen

Thomas Mosbæk Jørgensen

Oliver Guldvang Koch

Carl Christian Brøkmann Rasmussen

Forord

Over følgende projekt, IronPlay, er der blevet udarbejdet to rapporter; Teknologi- og programmeringsrapport og Systemudviklingsrapport. Den følgende rapport er den ene rapport, Teknologi- og programmeringsrapport. I det samlede antal tegn indgår ikke Forside, Titelblad, Forord, Abstract, Indholdsfortegnelse, Litteraturliste og Bilag. Derudover står alle *Klasser* i kursiv.

Denne rapport beskriver det planlagte færdige system, men funktioner som Store, Chat, Friendlist osv. er endnu ikke implementeret.

I bilag 1 – IronPlay guide ses en guide til at bruge IronPlay.

Abstract

This project outlines the development process of a game client named IronPlay, highlighting key elements such as Concurrency Control, Security aspects, Frontend Development and RESTful Services. To achieve and ensure a well-structured project the Scrum methodology, an agile framework, has been used.

The functions and requirements of IronPlay were derived from User Stories, defined from the user's perspective, and a Domain Model, ensuring a well-structured system. These served as the foundation for the development process of IronPlay.

Implemented in C# with ASP.NET, IronPlay is structured as a Distributed System. Its API is a RESTful Service, seamlessly connecting a Web Client built with an MVC architecture and a Desktop Client featuring a Windows Forms interface.

Razor pages has been used to build dynamic web pages, simplifying the creation of user interfaces. Additionally, Dapper is used to allow efficient access to data in the database. The database used is MSSQL and the design is created from a Relational Model, to ensure correct and optimal structure for data management.

By employing Json Web Tokens IronPlay ensures robust security measures for both Admins and Players. Tokens are used as access keys, granting authorized interaction with the API.

The Web Client facilitates profile creation and authentication, allowing Players to log in and engage with the platform. Additionally, Players have the ability to create and join custom game lobbies. The Desktop Client is tailored for administrative tasks such as updating Player information, ensuring efficient management of the client.

IronPlay represents a well-structured solution, catering to both user experience and administrative functionality.

Indholdsfortegnelse

Indledning

Videospilsbranchen er i konstant udvikling med hurtige teknologiske fremskridt og en voksende brugerbase[1]. Inden for denne branche opstår der interessante problemstillinger, såsom hvordan man udvikler en spilklient af høj kvalitet, der undgår samtidighedsproblemer, når mange spillere forsøger at tilslutte sig den samme lobby, og hvordan man skaber en klient med høj skalerbarhed. For at undersøge nærmere, hvordan man kan løse disse udfordringer, har gruppen valgt at udvikle en spilleklient ved navn IronPlay. IronPlay er designet til at tackle disse problemstillinger og levere en optimal klient oplevelse for brugerne.

Afgrænsning

Gruppen har valgt at fokusere udelukkende på spilleklienten og har derfor afgrænset sig fra at udvikle selve spillet. I spilleklienten er det muligt for brugere at oprette en profil, logge ind og oprette enten en privat eller offentlig gamelobby. Dog har gruppen ikke valgt at implementere funktionalitet til at starte et spil. Derudover afgrænses der på at håndtere finansielle transaktioner i klienten.

Problemstilling

I dette afsnit uddybes det valgte emne; Spilleklient. Derudover specificeres problemformuleringen samt User Stories, som er essentielle til udviklingen.

Spilleklienten: IronPlay

IronPlay er en spilleklient, der er skabt til gamere og deres venner, der sætter pris på en personlig spiloplevelse og muligheden for social interaktion. Platformen giver spillere mulighed for at skabe private, brugerdefinerede gamelobbies, hvor de kan invitere et specifikt antal venner til at spille og chatte sammen. Hvis man ønsker at møde nye mennesker og deltage i større spilfællesskaber, kan man også oprette offentlige gamelobbies.

I tillæg til dette tilbyder IronPlay en venneliste-funktion, der giver spillerne mulighed for at tilføje og holde styr på deres venner. Her kan man se deres online-status og starte private samtaler. For dem, der ønsker at tilpasse deres spiloplevelse, tilbyder IronPlay en Store med et bredt udvalg af genstande som profilbilleder og emojis, af begrænset kvantitet. Disse genstande kan samles i en privat samling, som kan ses i brugerens collection, hvor det også er muligt at opdatere profilbilleder m.m. Derudover tilbyder IronPlay et rangsystem, der tilføjer en konkurrencepræget dimension til spillet, og rangen vises tydeligt på den pågældende profil.

Problemformulering

Formålet med projektet er at udvikle en funktionel spilleklient, der muliggør oprettelse og håndtering af gamelobbies.

> Hvordan kan udviklingen af en distribueret spilleklient til et fiktivt spil effektivt adressere samtidigheds problemer i forbindelse med oprettelse og tilslutning til brugerdefinerede gamelobbies?

User Stories

User Stories udgør grundlaget for udviklingsarbejdet og er generelle uformelle beskrivelser, fra brugerens synsvinkel, af de funktioner, som spilleklienten skal have. Ved at skabe disse User Stories er det sikret, at udviklingsprocessen forbliver fokuseret og målrettet mod at levere et produkt, som opfylder brugerens behov på bedst mulig vis. [2]

For at danne overblik over disse User Stories, er de blevet inddelt i Epics; Player, Login, Lobby, Friends, Store, Profile, Administrate Player, Administrate Lobby og Administrate Store. Under hver enkelt User Story er der beskrevet kortfattet hvilken funktion spilleklienten skal have samt de acceptkriterier, som følger med. Acceptkriterierne bruges til at validere, om funktionerne i den enkelte User Story er opfyldt samt et grundlag for at skrive automatiserede tests. [3]

For at vide om en User Story er færdig, så har gruppen beskrevet en Definition of Done:

- Alle acceptkriterier skal være opfyldt
- Koden compiler
- Alle tests skal bestå
- Der er blevet lavet et Code Review

Alle User Stories kan ses beskrevet i bilag 2 – User Stories.

Krav

De funktionelle og ikke-funktionelle krav beskrives i dette afsnit, hvilket tager udgangspunkt i de tidligere beskrevet User Stories. Derudover redegøres der for domænemodellen, som indeholder informationskravene til systemet.

Funktionelle og ikke-funktionelle Krav

For at sikre en brugervenlig og funktionel spilleklient, er de funktionelle krav baseret på de 39 User Stories, som afspejler de behov og forventninger, som brugeren har. Disse krav sikrer, at spilleklienten opfylder centrale funktioner som login, registrering, oprettelse af gamelobbies samt join en gamelobby. De ikke-funktionelle krav til spilleklienten er følgende:

- Brugervenlighed
- Sikkerhed
- Performance

Spilleklienten skal være intuitiv og nem at bruge, som gør det let for spillerne at finde og bruge de forskellige funktioner. Derudover skal den beskytte brugerdata og forhindre uautoriseret adgang gennem authentication og authorization. Klienten skal være optimeret og have høj ydeevne og minimal ventetid ved udførelse af funktioner.

Domænemodel

En domænemodel udgør en fundamental del af enhver softwareudviklingsproces ved at fungere som en visuel repræsentation af de informationsmæssige krav til et system. Denne model giver et klart overblik over de konceptuelle klasser i systemet og deres indbyrdes relationer. I forbindelse med udviklingen af domænemodellen for projektet, IronPlay, analyserede gruppen de initiale User Stories for at identificere substantiver, som kunne fungere som potentielle kandidater til konceptuelle klasser. Dette resulterede i følgende liste over kandidatklasser (se tabel 1):

	Har Attributter?	Flere Instanser?	Ikke Kun Information/Udskrift?	Godkendt Kandidat
Profile	Ja	Ja	Ja	**Nej**
Homepage	Nej	Nej	Nej	**Nej**
Account	Ja	Ja	Nej	**Ja**
Message	Ja	Ja	Nej	**Ja**
Chat	Ja	Ja	Nej	**Ja**
FriendList	Ja	Ja	Nej	**Ja**
Player	Ja	Ja	Nej	**Ja**
GameLobby	Ja	Ja	Nej	**Ja**
Order	Ja	Ja	Nej	**Ja**
Admin	Ja	Ja	Nej	**Ja**
Dashboard	Nej	Nej	Nej	**Nej**

Tabel 1: Kandidatliste over konceptuelle klasser

Med denne kandidatliste har gruppen udarbejdet følgende domænemodel (se illustration 1):

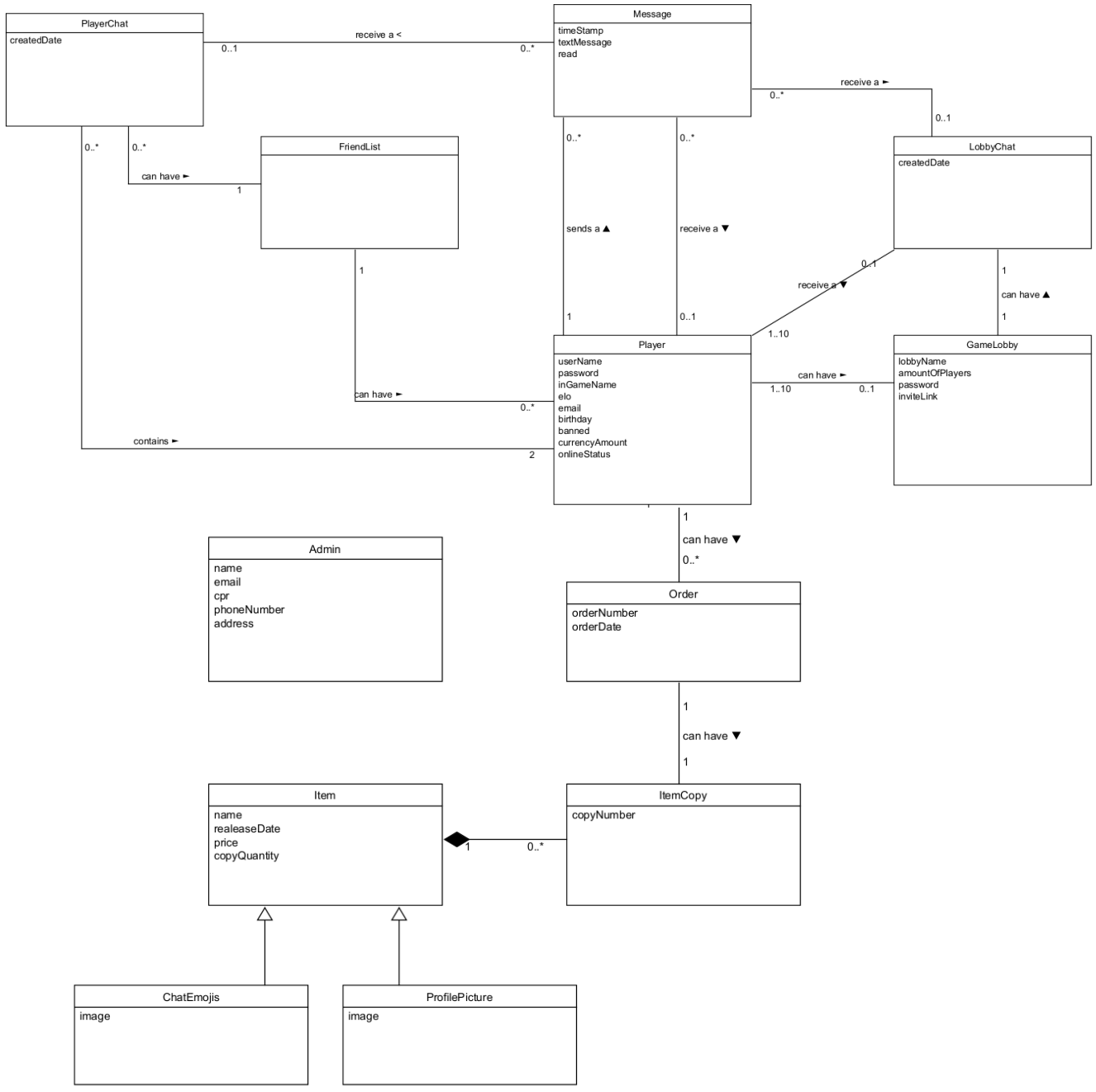


Illustration 1: Domænemodel

Gruppen har anvendt denne domænemodel til at opnå en dybere forståelse af domænet for spilleklienter, og den har været afgørende for at definere det samlede systems struktur. Domænemodellen fungerer også som en væsentlig dokumentationsressource, som kommende udviklere til IronPlay vil kunne benytte til at forbedre deres forståelse af systemets domæne.

Arkitektur

I dette afsnit vil der blive forklaret, hvorfor et distribueret system er blevet valgt til projektet, samt hvilke fordele det medfører.

Distribueret system

Systemet, som er blevet udviklet, er et distribueret system, da det er opdelt i separate projekter. I dette tilfælde består systemet af tre projekter: en Webklient, en Desktopklient og et API. Webklienten og desktopklienten håndterer alt brugerinput og kommunikerer med API'et, som står for databasehåndtering. Projekternes struktur er baseret på domænemodellens konceptuelle klasser. Disse projekter uddybes i højere detalje i de forudgående afsnit i rapporten.

Projektet er designet i fem tiers; Database, API, Webserver, Desktopklient og en Webklient. Disse tiers repræsenterer de fysiske maskiner, som disse er fordelt ud på. Illustration 2 nedenfor viser sammenhængen mellem disse tiers, og de protokoller der bruges til kommunikation mellem dem.

Mellem databasen og API'et anvendes en TCP-protokol, hvilket sikrer, at data sendes korrekt og i den rigtige rækkefølge [4]. Frontendklienterne kommunikerer med API'et via en HTTPS-protokol, hvilket muliggør dataoverførsel gennem HTTP-requests. For at sende en request kræves en action og en Uniform Resource Identifier (URI) med et specifikt endpoint. Når en request er sendt, modtages en HTTP-response, som angiver, om requesten er godkendt med en kode 200 (OK), eller om der returneres en fejlmeddelese, ofte en kode 400 (Bad Request).[5]

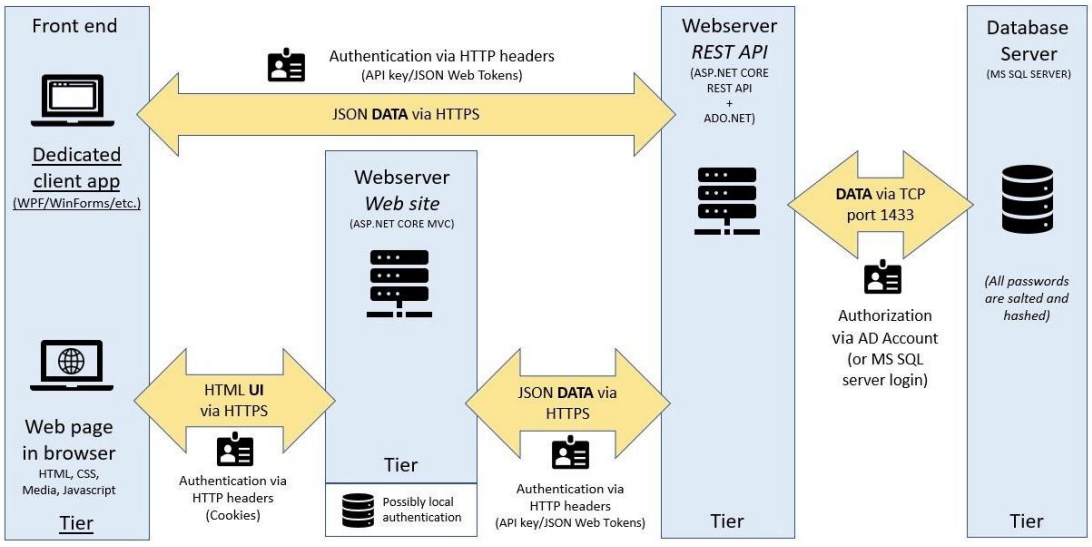


Illustration 2: Distribueret System – kan findes i Programmering/Klassemateriale på Microsoft Teams

Valget af et distribueret system blev truffet på baggrund af flere faktorer. Først og fremmest giver et distribueret system bedre skalérbarhed. Dette betyder, at hvis der pludselig skulle komme mange brugere, kan spilleklienten skaleres op uden at påvirke de andre applikationer såsom API'et og desktopklienten. Derudover tilbyder et distribueret system også større fleksibilitet i valget af teknologier, da forskellige teknologier kan anvendes til de forskellige tjenester. Denne fleksibilitet muliggør også enklere vedligeholdelse, da de enkelte tjenester kan deployes uafhængigt af hinanden, uden at hele systemet skal redeployes.

Distribuerede systemer gør det også nemmere at isolere fejl. Hvis spilleklienten oplever problemer, vil det højst sandsynligt ikke påvirke de andre tjenester. Desuden muliggør sikkerheden i distribuerede systemer implementeringen af forskellige sikkerhedsniveauer på forskellige tjenester, hvilket minimerer risikoen for angreb. Denne differentierede sikkerhedsstruktur bidrager til at beskytte systemet mod potentielle trusler.

Frontend klienter

Gruppen har udviklet to former for frontendklienter, der er designet til at løse hver deres form for brugerkrav og interaktionsformål. Disse vil blive beskrevet i dette afsnit, på baggrund af design og struktur.

Hvad udgør et website

Webklienten er blandt andet bygget ved hjælp af HTML, CSS og JavaScript, hver til deres formål. HTML er blevet anvendt til at strukturere og definere layoutet, samt individuelle elementer som f.eks. knapper, billeder, links og paragraffer. Efterfølgende er der anvendt CSS til at tilføje interaktivitet til disse HTML elementer samt for at style deres udseende og opførsel. Dette kan f.eks. være, at en knap skal blive større eller ændre baggrundsfarve, når man holder musen over den. Der er blevet anvendt JavaScript på klientsiden for at kunne håndtere ting som, at filtrere i gamelobbies baseret på brugerinput, EventListeners for at holde øje med hvilke tekstfelter brugeren interagerer med, f.eks. som at gøre Login-knappen grå og ikke klikbar når der ikke er skrevet noget i tekstfelterne.

Webklient

Brugerne af denne klient er spillere, hvorfor dette også har resulteret i at webklienten er blevet vægtet designmæssigt højere end desktopklienten. Spillere kommer til at være den primære indkomst via klientens Store, de interagere på daglig basis med klienten, og deres oplevelse, engagement og tilfredshed har direkte indflydelse på produktet.

I webklienten kan der oprettes en ny bruger og logges ind på login siden (se illustration 3). Når de indtastede oplysninger er blevet verificeret, bliver der omdirigeret til hjemmeskærmen. Gendannelse af password har været uden for scope, hvorfor denne funktionalitet endnu ikke er implementeret.

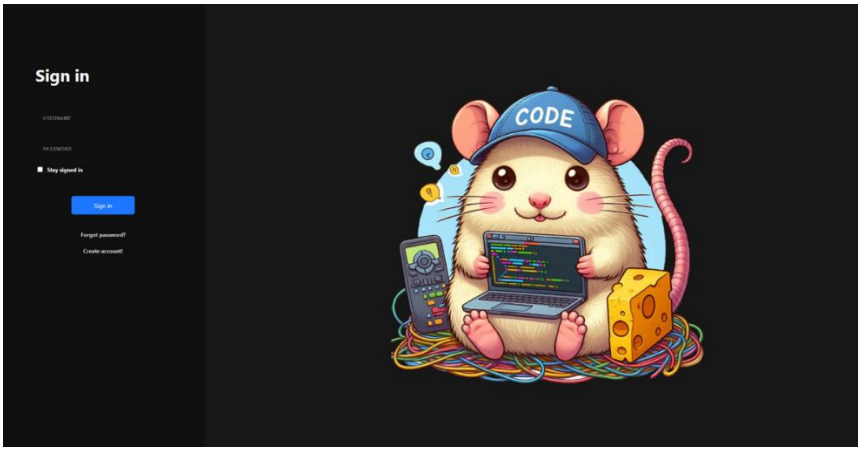


Illustration 3: Webklientens login side

På hjemmeskærmen er der en sektion med fremhævet indhold i midten af skærmen, mens der under dette er tre sektioner med nogle Patch notes og de seneste nyheder relateret til IronPlay. Navigationsbaren i venstre side indeholder et overblik over brugernavnet på den spiller der er logget ind, et spiller-ikon og log ud funktionalitet samt spillerens rang og mængde af IronPlay's virtuelle valuta. I navigationsbaren findes der også en Play knap, som omdirigerer til en liste over gamelobbies. De yderligere tre knapper; Profile, Collection og Store er endnu ikke funktionelle, men kan ses som User Stories i bilag 2 – User Stories. Derudover er der også implementeret en hardcoded venneliste (se illustration 4). Resten af hjemmesiden kan ses som skærmbilleder i bilag 3 - Webklient.

Illustration 4: Webklientens hjemmeskærm

Design

Før gruppen begyndte at lave webklienten, blev der udarbejdet en wireframe for at danne et overblik over hovedfunktionerne, og hvordan selve hjemmesiden skulle se ud. Dette var også med til at gøre det nemmere at opnå en samlet forståelse for designet og layoutet af webklienten. (se illustration 5).

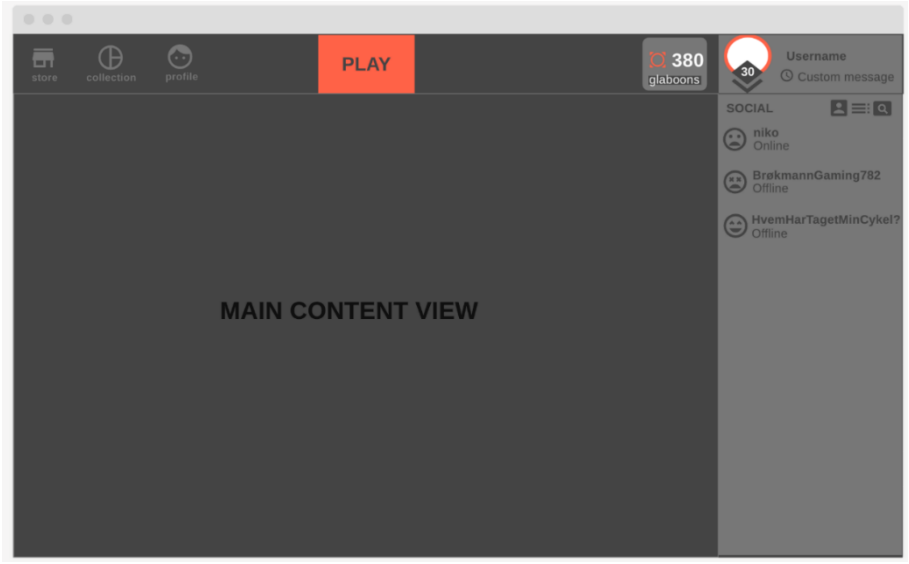


Illustration 5: Første udkast til Webklientens navigationsbar

Der opstod dog hurtigt en bred enighed om at det første udkast ikke var tilfredsstillende, derfor blev der udarbejdet nogle nye fortolkninger af navigationsbaren. (se illustration 6).

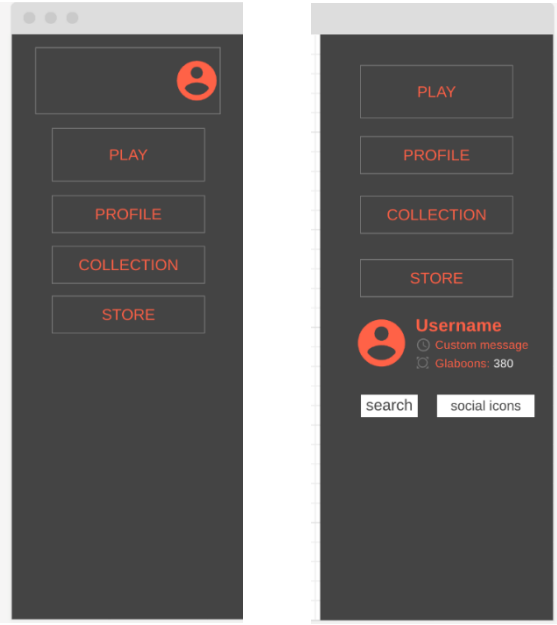


Illustration 6: Udkast af 2 mulige designs af navigationsbaren

Som det ses af de ovenstående Wireframes, har designet ændret sig fra at bestå af en navigationsbar i toppen og en venneliste i siden, til en mere kompakt version, hvor kerneelementerne blot er blevet vedligeholdt.

Struktur

Webklienten anvender ASP.NET Core MVC Framework. MVC står for Model-View-Controller og inddeler applikationen i disse 3 lag. Webklienten er dog struktureret efter i alt seks lag for at opnå en effektiv separation of concerns. Her er en kort beskrivelse af de første fem lag, hvor det sjette og sidste lag, Security-laget, bliver gennemgået mere dybdegående senere i rapporten:

BusinessLogic-laget: Dette lag håndterer al forretningslogik relateret til webklienten. For eksempel håndterer *GameLobbyLogic* alt, der relaterer sig til gamelobbies, og det samme gælder for *LoginLogic* og *RegistrationLogic*. Der er også oprettet interfaces til alle disse klasser for at gøre koden mere testbar og fleksibel.

Controller-laget: Dette lag står primært for at navigere brugeren videre til det korrekte View, som ved succesfuldt login eller når man trykker på knappen 'Play'.

Model-laget: Dette lag repræsenterer systemets data. Det sikrer, at alle lag har en fælles forståelse af, hvordan data skal struktureres, for eksempel hvordan en spillermodel skal se ud.

Service-laget: Dette lag er ansvarligt for at kommunikere med API'et. For eksempel vil *LoginService* håndtere kommunikationen med API'et for at verificere brugeroplysninger.

Views: Dette er de faktiske brugergrænseflader, som spillerne interagerer med. De er bygget med Razor-syntaks og kan indeholde både HTML og server-side C#-kode. Views bruger Razor Layout Pages til at inkludere genbrugelige sektioner i brugergrænsefladen for at sikre et ensartet og fleksibelt design. For eksempel er navigationsbaren, som indeholder en venneliste, Store og Play mm., en Razor Layout Page, der bruges sammen med andre Views for at sikre, at disse funktioner altid er tilgængelige, uanset hvor i systemet spilleren befinder sig.

Dette design sikrer, at hver del af systemet har et klart defineret ansvar, hvilket gør systemet lettere at vedligeholde og udvide. Det gør også systemet mere robust, da ændringer i ét lag er mindre tilbøjelige til at påvirke andre lag.

Desktopklient

Valg af applikationstype

Desktopklienten er udviklet i Windows Forms, netop som en desktopklient, da den er beregnet til administratorer af spilleklienten. Gruppen mente derfor, at det ville give bedst mening at have en administrerende klient som en applikation på pc'en, frem for en mobilapp. Dette gør også, at den administrerende klient er sværere at tilgå for uvedkommende, da man skal downloade den for at kunne køre den. Det er også en fordel, da en bruger, som ikke burde have adgang til den administrerende klient, ville kunne forårsage problemer for hele servicen.

Struktur

Strukturen for desktopklienten er opbygget med fem lag; Controller-laget, GUI-laget, Model-laget, Security-laget og Service-laget. I den udviklede desktopklient er det GUI-laget, der er ansvarlig for at håndtere alle brugerinputs og kalde videre ned i koden.

Controller-laget varetager både forretningslogikken og fungerer som mægler mellem GUI-laget og Service-laget. Service-laget har ansvaret for at kalde de korrekte endpoints til API'en for at hente data ud af databasen. Endelig er der Security-laget, som håndterer JWT-tokens. Dette lag er afgørende for, at desktopklienten kan kalde endpoints, da API-brugeren skal være autoriseret for at kunne lave en forespørgsel. Model-laget fungerer som i webklienten.

Denne struktur er blevet valgt da den giver en mere struktureret og vedligholdelsesvenlig tilgang til udviklingen af en desktopklient og resulterer i sidste ende til mere skalerbarhed.

Design

Intentionen med designet af desktopapplikationen var, at den skulle være intuitiv og nem at bruge, da dette også kommer fra de ikke-funktionelle krav. For at opnå dette blev der implementeret et overskueligt dashboard, der giver admin nem adgang til de forskellige administrative opgaver (se illustration 8).

I Player Management vinduet vises en liste over alle spillere, hvor det her også er muligt at søge efter specifikke spillere på deres InGameName eller Username (se illustration 7). Når der dobbeltklikkes på en spiller, så åbnes der et modalt vindue, hvor der kan foretages ændringer på den specifikke, spiller der er blevet valgt (se illustration 9).

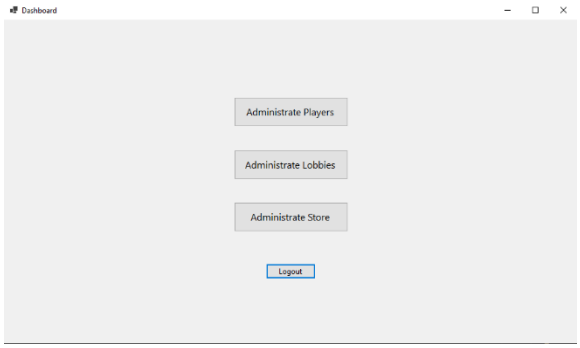


Illustration 8: Admin Dashboard

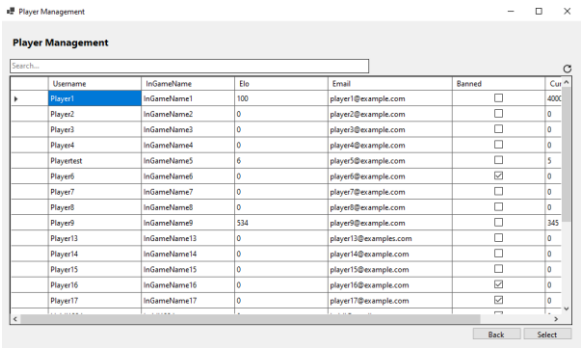


Illustration 7: Admin Player Management

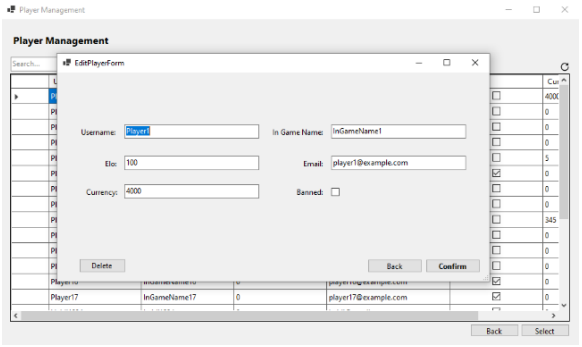


Illustration 9: Admin Update eller Delete Player

Det samlede design fremhæver enkelthed og effektivitet, hvilket sikrer at en admin nemt kan udføre administrative opgaver.

Systemet

I dette afsnit vil det specificeres hvilke teknologier der er blevet brugt til udvikling af projektet samt hvorfor. Derudover beskrives der hvilket middleware, der bliver gjort brug af, som muliggør f.eks. datahåndtering.

Brugte teknologier

C# blev valgt som programmeringssprog, da det var det sprog, gruppen havde mest erfaring med til udvikling af både API'er og webklienter. For at holde tech stacken simpel blev C# også brugt til desktopklienten.

ASP.NET Core blev brugt som framework til at bygge API'er og webklienter, da det gør det nemt at udvikle webapps i samme stack [6]. Desuden er det blandt de hurtigste frameworks ifølge TechEmpower's benchmarks [7].

Visual Studio blev anvendt som den primære IDE til udvikling. Visual Studio er et kraftfuldt og omfattende udviklingsmiljø, der understøtter en bred vifte af programmeringssprog og teknologier. Det tilbyder avancerede debugging-værktøjer, integration med versionsstyringssystemer og en række udvidelser, der kan tilpasses udviklerens behov, hvilket gør det ideelt til udvikling i .NET-økosystemet [8].

Windows Forms blev valgt som UI-framework til at skabe desktopapplikationen for IronPlay [9]. Med Visual Studio's integration, der gør det muligt for udviklere at designe brugergrænseflader ved hjælp af drag-and-drop-funktionalitet, var det et oplagt valg til dette projekt.

MSSQL blev brugt som databasesystem, da der var behov for en pålidelig og robust relationel database. MSSQL tilbyder høj sikkerhed og skalerbarhed, hvilket er essentielt for at håndtere større mængder data.

Middleware

Middleware er software, der fungerer som et lag mellem operativsystemet og de applikationer, der kører på det [10]. I udviklingsprocessen af IronPlay er forskellige middleware-komponenter blevet anvendt, typisk importeret via NuGet-pakker i Visual Studio.

Middleware spiller en afgørende rolle i et softwareprojekt, da det sikrer systemets overordnede funktionalitet og ydeevne. Ved at have styr på det anvendte middleware kan man bedre identificere potentielle fejlårsager og sikre, at integrationen mellem systemets forskellige dele fungerer problemfrit.

I udviklingen af IronPlay er der brugt forskellige libraries, skrevet af andre udviklere, såsom BCrypt til hashing og salting af brugerdata, Dapper som en micro ORM til databaseforespørgsler, og Newtonsoft.Json til mapping af JSON-data. En komplet liste over anvendt middleware for hvert projekt kan findes i bilag 4 – Middleware versions.

Backend

I dette afsnit skabes der overblik over hvordan API'en er opbygget i forskellige lag, som har forskelligt ansvar. Derudover beskrives hvilken database der er blevet brugt samt hvordan scripts er opbygget ud fra en Relationel Model.

API

I dette afsnit beskrives API'en som en RESTful Service samt hvordan denne er struktureret.

RESTful Service

API'en er opbygget som en Representational State Transfer Service (RESTful Service). Denne arkitektur er baseret på seks forskellige principper (se illustration 10), som promoverer simplicitet, scalability og statelessness i designet af API'en. En af de bærende principper, som skal opfyldes, for at opnå en RESTful API er det første princip Uniform Interface. Her ses det i IronPlay, at de forskellige HTTP requests bruges uniformt, altså GET metoden bruges til at hente data, POST bruges til indsættelse af data, PUT til opdatering af data og DELETE til at slette data fra databasen. Det andet princip Client-Server og femte princip Layered System beskrives yderligere i næste afsnit omkring strukturen af API'en. Derudover er systemet stateless i den forstand, at hver forespørgelse er ny og der ikke vil blive gemt information fra tidligere requests. [11]

Illustration 10: RESTful Service principper [11]

Struktur

For at sikre den ønskede separation of concerns er det blevet valgt at strukturere API'en i fire lag. Dette gør også at API'en opfylder de to tidligere nævnte principper til at være en RESTful Service.

Controller-laget håndterer alle indkommende anmodninger fra klienter. Dette lag er ansvarligt for at route anmodningerne til de relevante sider ved at returnere passende svar på de anmodninger, der er sendt fra klienten. De forskellige endpoints, som nuværende er implementeret for IronPlay, kan ses i tabel 2-5.

BusinessLogic-laget er dedikeret til al forretningslogik. Her udføres eventuelle komplekse operationer, og det er også her, at kaldene videreføres til DatabaseAccessor-laget. Denne struktur tillader en klar adskillelse af forretningslogikken fra databasetilgangen og muliggør en mere fleksibel og modulær struktur, hvor ændringer i DatabaseAccessor-laget ikke nødvendigvis kræver ændringer i BusinessLogic-laget.

DatabaseAccessor-laget håndterer alle databaserelaterede operationer. Dette lag er ansvarligt for at håndtere alt, hvad der vedrører databasen, og det fungerer som et bindeled mellem BusinessLogic-laget og den underliggende database.

Model-laget definerer hvordan data organiseres og relateres til hinanden. Modellerne i dette lag er tilgængelige fra både BusinessLogic-laget, Controller-laget og DatabaseAccessor-laget og sikrer en ensartet repræsentation af data på tværs af hele applikationen. Denne struktur sikrer ikke kun en klar og struktureret opdeling af ansvar, men muliggør også en mere effektiv vedligeholdelse og udvidelse af API'en over tid.

Endpoint (api/Player)	GET	POST	PUT	DELETE
/players	Get a list of all players	Creates a new player	Updates an existing player	N/A
/players/(playerId)	N/A	N/A	N/A	Delete Player
/players/login	N/A	N/A	Login player	N/A
/players/logout	N/A	N/A	Logout Player	N/A

Tabel 2: Tabel over endpoints for PlayerControlleren

Endpoint (api/Gamelobby)	GET	POST	PUT	DELETE
/gameLobbies	Get a list of all gamelobbies	Create a new gamelobby	N/A	N/A
/gameLobbies/join	N/A	N/A	Join an existing Gamelobby	N/A
/gameLobbies/leave	N/A	N/A	Leave an existing Gamelobby	N/A

Tabel 3: Tabel over endpoints for GamelobbyControlleren

Endpoint (api/Token)	GET	POST	PUT	DELETE
/tokens	N/A	Generates new access and refresh tokens	N/A	N/A
/tokens/refresh	N/A	Generates new access and refresh tokens	N/A	N/A

Tabel 4: Tabel over endpoints for TokensControlleren

Endpoint (api/Admin)	GET	POST	PUT	DELETE
Admins/login	N/A	N/A	Login admin	N/A

Tabel 5: Tabel over endpoints for AdminControlleren

Database

Gruppen har besluttet at anvende en Microsoft SQL Server (MSSQL) database, som er hostet på Hildur.dk. Dette valg blev baseret på en grundig risikoanalyse, som kan findes i bilag 5 – Risk Register, hvor prioriteringen var teknologier, som gruppen havde mest erfaring med. Formålet var at minimere risikoen for uventede komplikationer under udviklingen af IronPlay spilleklienten.

Derudover er MSSQL en relationel database, hvilket muliggør effektiv strukturering af data, dette er afgørende for en kompleks applikation som IronPlay. MSSQL's understøttelse af ACID (Atomicity, Consistency, Isolation, Durability) sikrer desuden en høj grad af dataintegritet, hvilket er af central betydning for projektet. Mens NoSQL-databaser også kan tilbyde visse fordele såsom skalerbarhed og fleksibilitet, er den relationelle struktur og stærke dataintegritet, som MSSQL giver, essentielt for projektets behov.

Relationel Model

Gruppen har udarbejdet en relationel model for at få et overblik over, hvilke tabeller som skal oprettes i databasen samt hvilke attributter disse skal have. Modellen giver også overblik over forholdet af foreign keys mellem forskellige tabeller. Under udviklingen af den relationelle model er der taget udgangspunkt i de tre normalformer samt Boyce-Codd normalformen, for at reducere redundans og sikre dataintegritet. Den relationelle model kan ses i bilag 6 – Relationel Model.

Database opbygning

Til opbygningen af databasen er der udarbejdet SQL-scripts, som kan findes som bilag 7 – Database Scripts. Disse scripts er oprettet ud fra den relationelle model og muliggør nemt nedrivning og genopbygning af alle tabeller, hvilket giver en høj grad af fleksibilitet, uanset om den brugte database i udviklingsmiljøet er på den lokale computer eller hosted på Hildur.

Inden for disse scripts er der defineret forskellige constraints med det formål at sikre, at databasen opretholder en konsistent tilstand og undgår inkonsistente data. Disse constraints inkluderer brugen af foreign keys samt definitionen af not null og default værdier i forskellige tabeller såsom Player, GameLobbies, Order osv. Disse constraints bidrager til at forhindre fejlindtastninger og sikre en stabil og pålidelig databasestruktur. Et eksempel på dette kan ses på illustration 11.

```
CREATE TABLE [dbo].[Player] (
    [PlayerID]       INT            IDENTITY (1, 1) NOT NULL,
    [Username]       NVARCHAR (50)  NOT NULL,
    [PasswordHash]   NVARCHAR (MAX) NOT NULL,
    [InGameName]     NVARCHAR (50)  NOT NULL,
    [Email]          VARCHAR (50)   DEFAULT (NULL) NULL,
    [Birthday]       DATETIME       NOT NULL,
    [Elo]            INT            DEFAULT ((0)) NULL,
    [Banned]         BIT            DEFAULT ((0)) NULL,
    [CurrencyAmount] INT            DEFAULT ((0)) NULL,
    [GameLobbyID]    INT            DEFAULT (NULL) NULL,
    [OnlineStatus]   BIT            DEFAULT ((0)) NULL,
    [IsOwner]        BIT            DEFAULT ((0)) NULL,
    PRIMARY KEY CLUSTERED ([PlayerID] ASC),
    FOREIGN KEY ([GameLobbyID]) REFERENCES [dbo].[GameLobby] ([GameLobbyID])
);
```

Illustration 11: Database script udsnit for oprettelse af Player tabel

Udfordringer med samtidighed

Da IronPlay er tiltænkt til at skulle håndtere mange brugere, der samtidigt interagerer med klienten, er det afgørende at sikre, at der ikke opstår samtidighedsproblemer, som kan forringe brugeroplevelsen eller i værste fald resultere i systemnedbrud.

Potentielle scenarier for samtidighedsproblemer

I forbindelse med spilleklienten IronPlay er der to scenarier, der kan føre til samtidighedsproblemer.

Det første scenarie opstår, når flere spillere forsøger at tilslutte sig den samme lobby, som kun har en eller få ledige pladser tilbage. Her opstår spørgsmålet om, hvilke spillere der får de sidste pladser.

Det andet scenarie omhandler IronPlay's in-game Store, hvor spillere har mulighed for at købe genstande til at personliggøre deres konto. Disse genstande er af begrænset kvantitet, og når alle eksemplarer af en genstand er solgt, kan andre spillere ikke købe flere af dem. Det er derfor vigtigt at undgå situationer, hvor flere spillere forsøger at købe de sidste eksemplarer af en genstand samtidig.

Konsekvenser af samtidighedsproblemer

I begge scenarier er målet at forhindre race conditions, som kan føre til inkonsistente data i databasen og dermed strider mod IronPlays forretningslogik. Begge scenarier kan medføre

non-repeatable reads, en databaseanomali, der opstår, når data ændres i løbet af en transaktion på grund af en anden samtidig transaktion [12].

Løsning af samtidighedsproblemer

For at løse samtidighedsproblemer kan der enten bruges Pessimistic Concurrency Control (PCC) eller Optimistic Concurrency Control (OCC). PCC håndterer samtidighed ved at sætte en lås på de data, som en transaktion forsøger at ændre [12]. Dette sikrer, at andre transaktioner ikke kan få adgang til disse data, før den nuværende transaktion er afsluttet.

OCC fungerer anderledes ved ikke at sætte låse på dataene i databasen. I stedet antager det, at der ikke er andre transaktioner, der forsøger at ændre de samme data som den aktuelle transaktion [12]. Typisk virker OCC ved, at der i starten af en transaktion laves en læsning af databasen, som gemmes i en variabel. Herefter udføres de nødvendige handlinger i transaktionen. Til sidst sammenlignes en ny læsning af databasen med den oprindelige for at se, om nogen har ændret dataene i mellemtiden. Hvis det viser sig, at dataene er ændret, rulles hele transaktionen tilbage.

Det skal dog bemærkes at begge løsninger kan føre til problemer som deadlocks, livelocks, starvation og dermed nedsat performance.

Implementering i IronPlay

Under projektarbejdet blev der implementeret funktionalitet til at understøtte det første scenarie. For at løse problemet med, at flere spillere ikke skal kunne tilslutte sig en lobby med et begrænset antal pladser, blev OCC brugt. API'et sikrer at race conditions undgås ved først at hente en liste over alle spillere i den pågældende lobby. Derefter tjekkes det, om der er plads i en lobby. Hvis der er plads, opdateres spillerens lobby-id, og endnu et kald foretages for at tjekke det aktuelle antal spillere i en lobby. Hvis der nu er for mange spillere, betyder det, at en anden transaktion har fundet sted samtidigt og afsluttet hurtigere. I så fald rulles hele transaktionen tilbage. Hvis der ikke er for mange spillere, bliver transaktionen committet.

Før denne implementering blev PCC overvejet. På grund af den måde, spillere tilknyttes en lobby, blev denne metode dog hurtigt afvist. PCC ville kræve enten en omstrukturering af databasens tabeller eller en låsning af hele Player-tabellen hver gang en spiller forsøgte at joine en lobby. Da Player-tabellen også anvendes til andre formål, såsom validering af loginoplysninger, ville dette have resulteret i længere responstider for spilleklienten som vil præge kvaliteten af produktet negativt. Derfor blev OCC valgt som den mest effektive løsning.

Sikkerhed

Sikkerhed for IronPlay er af høj betydning for at sikre en tryg og pålidelig brugeroplevelse. Programmet opbevarer følsomme brugerdata såsom email, brugernavne og adgangskoder, som brugerne forventer forbliver private og ikke lækkes tilgængeligt for alle på internettet. Desuden er det kritisk, at systemet forhindrer ondsindede aktører i at opbruge ressourcerne, hvilket kan resultere i langsommere responstider og dermed forringe kvaliteten af produktet. Effektive sikkerhedsforanstaltninger er derfor fundamentale for at opretholde integriteten og tilliden til spilleklienten.

Forhindring af uautoriseret adgang til systemet

Authentication spiller en central rolle i sikkerheden, da det kræver, at brugerne logger ind for at få adgang til forskellige dele af systemet. Ved at bekræfte en brugers identitet gennem login-processen sikres det, at kun autoriserede brugere kan tilgå følsomme oplysninger og funktioner. Authorization kontrollerer derimod, hvilke ressourcer en bruger har adgang til efter at være blevet autentificeret. Denne adskillelse mellem authentication (bekræftelse af identitet) og authorization (tildeling af adgangsrettigheder) betyder, at ikke alle authenticated brugere har adgang til alle systemressourcer [13]. Dette muliggør en opdeling af adgangsrettigheder mellem almindelige spillere og administratorer. Ved at sikre, at kun de rette brugere har adgang til bestemte funktioner og data, forhindres uautoriserede handlinger og sikrer, at systemet fungerer effektivt og sikkert.

Authentication og authorization på spiller niveau

I IronPlay kræves det, at spillerne logger ind på systemet med det brugernavn og adgangskode, de registrerede deres bruger med. Hvis de ikke kan påvise, at de kender brugernavnet og adgangskoden til deres konto, nægtes de adgang til sider som oversigt over gamelobbies, Store eller Profile. Dette håndteres gennem brug af Microsofts AspNetCore Authentication package, som implementeres i *Program.cs* under projektet for webklienten [14](se illustration 12).

```
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
    {
        options.LoginPath = "/Login";
        options.AccessDeniedPath = "/Login/AccessDenied";
    });
```

Illustration 12: Webklient/Program.cs

Dette muliggør brugen af [Authorize] attributten på controllere i webklienten, som spillere ikke skal have adgang til uden at være blevet authenticated. Et eksempel på dette vises i illustration 13.

```
namespace WebClient.Controllers
{
        [Authorize]
        public class GameLobbyController : Controller
        {
```

Illustration 13: Authorize attribut på GameLobbyController

Hvis en spiller forsøger at tilgå sider, som de ikke er autoriseret til, vil de blive omdirigeret til login-skærmen via URL'en https://IronPlay/Login?ReturnUrl=%2FGameLobby%2F. De eneste sider, spillere har adgang til uden at være logget ind, er login-siden og konto-registreringssiden.

Authentication håndteres af kode vist i illustration 14. Webklienten sender et kald til API'et med brugernavn og adgangskode indtastet i login-formen. API'et verificerer disse oplysninger mod databasen. Hvis oplysningerne stemmer overens, får webklienten besked om, at informationerne er korrekte. Herefter oprettes en krypteret cookie, som gemmes af webbrowsere og sendes frem og tilbage ved hvert efterfølgende HTTP-kald til webserveren [14]. Denne cookie indeholder oplysninger om, at brugeren er authenticated og hvem de er.

```
[HttpPost]
public async Task<ActionResult> LoginToProfile(string username, string password)
{
    try
    {
        HttpResponseMessage response = await _loginLogic.VerifyCredentials(username, password);
        if (response.IsSuccessStatusCode)
        {
            PlayerModel player = await _loginLogic.GetPlayerFromResponse(response);
            ClaimsPrincipal principal = _loginLogic.CreatePrincipal(player);

            //Creates an encrypted authentication ticket(cookie) containing the user's principal
(identity) information and adds it to the current response.
            await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme,
principal);

            return RedirectToAction("HomePage", "Homepage");
        }
        else
        {
            if (response.StatusCode == HttpStatusCode.BadRequest)
            {
                ViewBag.ErrorMessage = "Username or password is incorrect.";
            }
            if (response.StatusCode == HttpStatusCode.Forbidden)
                        {
                                ViewBag.ErrorMessage = "Your account has been banned";
                        }
                        // If the API returned a 400 status code, return the same view
                        return View("Index");
                }
         }
         catch
         {
                return View("Error");
         }
}
```

Illustration 14: LoginToProfile metode i LoginControlleren

Andre måder at håndtere authentication på spiller niveau

En alternativ tilgang til håndtering af authentication for IronPlay kunne være at implementere social login-funktionalitet ved at lade brugerne logge ind med deres eksisterende Google, Facebook eller Discord-konto [15]. Dette kunne realiseres ved at integrere en social login-løsning, der giver tredjepartsapplikationer begrænset adgang til brugerens oplysninger uden at afsløre selve loginoplysningerne. Ved at benytte sig af denne tilgang ville brugerne kunne logge ind hurtigt og bekvemt uden at skulle dele deres adgangskoder med IronPlay. Fordelene ved denne metode omfatter forbedret sikkerhed, en mere streamlined brugeroplevelse og reducerede administrative byrder ved ikke at skulle vedligeholde en separat authentication infrastruktur. Dog er der også ulemper ved denne tilgang, herunder afhængighed af tredjepartsudbydere, potentielle bekymringer vedrørende databeskyttelse og privatliv samt en stigning i kompleksiteten af implementeringen.

Authentication og authorization på klient niveau

Alle brugere, der tilgår systemet fra webklienten, er spillere, mens alle brugere, der tilgår systemet fra desktopklienten, er administratorer. Derfor er der ikke behov for at autorisere på rolleniveau i de to klienter, men kun på API-siden. Dette sikrer, at en spiller, der er logget ind, ikke kan slette en anden brugers konto.

API'et anvender role-based authorization baseret på klienttypen, hvilket kontrolleres gennem brug af JSON Web Tokens (JWT) [16]. JWT er en kompakt, URL-sikker måde at repræsentere påstande mellem to parter. En JWT består af tre dele: header, payload og signature. Headeren specificerer algoritmen, der bruges til signaturen. Payloaden indeholder påstandene, som typisk inkluderer brugeroplysninger og roller. Signaturen skabes ved at tage headeren og payloaden, kombinere dem, og derefter signere dem med en hemmelig nøgle ved hjælp af en algoritme specificeret i headeren.

API'et genererer en JWT, når det modtager en POST-request til endpointet /tokens, hvor klienterne inkluderer information, der kun er kendt af dem. Denne hemmelige nøgle bruges til at generere signaturen, som sikrer, at JWT'en ikke kan ændres uden at blive opdaget. Når en JWT modtages, kan API'et verificere den ved at kontrollere signaturen med den samme hemmelige nøgle. Hvis signaturen stemmer overens, er JWT'en valid, og API'et kan stole på oplysningerne i payloaden.

Efter en klient har modtaget en JWT fra endpointet /tokens, vil de modtage både en access token og en refresh token. En access token vil blive sat som bearer token for hvert fremtidigt kald til API'et. Den har en udløbstid på en time, hvorimod en refresh token har en udløbstid på syv dage. Dette bidrager væsentligt til sikkerheden af systemet. Ved at have kortere udløbstid for en access token, reduceres risikoen for misbrug, da selvom den bliver kompromitteret, vil den kun være gyldig i en begrænset periode. Samtidig tillader en refresh token brugere at forny deres adgang uden at skulle logge ind igen hyppigt, hvilket forbedrer brugervenligheden. Dog skal det bemærkes, at en refresh token også er underlagt

sikkerhedsforanstaltninger og skal opbevares sikkert af klienten for at forhindre uautoriseret adgang.

Opbevaring af brugerdata

I IronPlay-systemet opbevares brugerdata i en database, hvilket gør det essentielt at beskytte følsomme oplysninger mod uautoriseret adgang. Beskyttelsen af brugernes adgangskoder er særlig vigtig. Adgangskoderne bliver hashed ved hjælp af Bcrypt-metoden HashPassword, som automatisk genererer et salt og inkluderer det i det hashed password [17]. Dette sikrer, at to identiske adgangskoder resulterer i forskellige hash-værdier, hvilket forhindrer ondsindede aktører i at matche hashed værdier for at gætte andres adgangskoder.

For at validere, om en brugers indtastede adgangskode stemmer overens med det hashed password i databasen, anvendes Bcrypts Verify metode. Dette sikrer, at kun brugere med de korrekte adgangskoder får adgang.

For yderligere at sikre brugerkonti mod brute force-angreb har IronPlay implementeret krav til adgangskoder: de skal indeholde mindst ét stort bogstav, et tal og være mindst 8 tegn lange. Dette forhindrer ondsindede aktører i at få adgang til konti med svage adgangskoder.

Desuden benyttes parameteriserede queries i koden for at beskytte mod SQL-injection angreb, hvilket yderligere sikrer integriteten og sikkerheden af brugernes data. Et kodeeksempel på dette kan ses i illustration 15.

```
public bool CreatePlayer(AccountRegistrationModel newPlayer)
{
    bool playerInserted = false;

    string insertQuery = "INSERT INTO Player (Username, PasswordHash, InGameName, Email,
Birthday) " +
                         "VALUES (@Username, @Password, @InGameName, @Email, @Birthday)";

    using (SqlConnection connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        var rowsAffected = connection.Execute(insertQuery, newPlayer);
        playerInserted = rowsAffected == 1;
    }

    return playerInserted;
}
```

Illustration 15: CreatePlayer metode i PlayerDatabaseAccessor

Testing

Dette afsnit går i dybden med udførelsen af tests, herunder Unit Tests og integrationstests, med tilhørende eksempler på disse. Derudover beskrives det hvordan systemtests og brugertests er blevet udført.

Introduktion Test

I dette projekt blev User Stories udviklet med acceptkriterier, som skulle testes for at sikre kvalitet og funktionalitet. xUnit blev anvendt som test framework til .NET-applikationer, hvor tests følger et AAA-pattern: Arrange (initialisering), Act (handling) og Assert (validering). [18] Automatiserede tests udføres hurtigt, effektivt og konsekvent, hvilket hjælper med at finde fejl tidligt og forbedrer softwarekvaliteten. I enkelte tilfælde blev Test Driven Development (TDD) brugt til at skrive tests før der blev kodet, hvilket sikrer, at koden opfylder kravene og reducerer fejl. For eksempel blev tests skrevet før udviklingen af User Story ”View Lobbies”, og koden blev derefter tilpasset, indtil alle tests bestod.

Unit Test

En Unit Test er en type test, der validerer, at individuelle komponenter af systemet fungerer korrekt isoleret fra hinanden. Formålet med disse tests er at sikre, at hver enkelt metode opfører sig som forventet under kontrollerede betingelser.

Moq frameworket er blevet brugt til at oprette mock objekter, der simulerer adfærden af eksterne afhængigheder, som f.eks. databasen. Disse mock objekter fungerer som stub-klasser og muliggør testning af BusinessLogic-laget uden interaktion med den faktiske database. Dette har gjort det muligt at fokusere på selve logikken i metoderne. Et eksempel på hvordan Moq framework er brugt i forbindelse med at teste *PlayerLogic* mod en mock *PlayerDatabaseAccessor* kan ses i illustration 16.

```
[Fact]
public void CreatePlayer_TC1_ReturnsTrueIfPlayerIsValid()
{
    // Arrange
    AccountRegistrationModel mockPlayer = new AccountRegistrationModel
    {
        Username = "username1",
        Password = "password1",
        Email = "email1@example.com",
        InGameName = "InGameName1",
        BirthDay = DateTime.Now
    };

          _mockAccessor.Setup(a => a.UsernameExists(mockPlayer.Username))
                    .Returns(false);
          _mockAccessor.Setup(a => a.InGameNameExists(mockPlayer.InGameName))
                    .Returns(false);
          _mockAccessor.Setup(a => a.CreatePlayer(It.IsAny<AccountRegistrationModel>()))
                    .Returns(true);

          PlayerLogic SUT = new PlayerLogic(_mockConfiguration, _mockAccessor.Object);

          // Act
          bool testResult = SUT.CreatePlayer(mockPlayer);

          // Assert
          Assert.True(testResult);
}
```

Illustration 16: CreateGameLobby_TC1

Formålet med Unit Testen CreatePlayer_TC1 er at sikre, at metoden CreatePlayer i *PlayerLogic* klassen korrekt opretter en ny spiller, når de angivne data er gyldige. Testen anvender et mock-objekt af *PlayerDatabaseAccessor*, der er konfigureret til at returnere forventede værdier.

Mock-objekterne simulerer et kontrolleret miljø, hvor CreatePlayer metoden skal operere. Denne opsætning er afgørende for at verificere, at metoden korrekt håndterer scenariet, hvor alle betingelser for oprettelse af en spiller er opfyldt. Testen fokuserer på, at metoden returnerer true, hvilket indikerer en vellykket oprettelse af spilleren. Dette bidrager til at sikre, at logikken for oprettelse af nye spillere fungerer som forventet, hvilket er essentielt for systemets overordnede kvalitet og stabilitet. Denne test er en del af en større samling test cases, som kan findes i bilag 8 – Test Case Tabeller.

Integrationstest

Integrationstestning er en testmetode, der validerer samspillet mellem tiers af systemet for at sikre, at de fungerer sammen som forventet uden problemer.

En testdatabase, identisk med produktionsdatabasen, anvendes for at isolere testmiljøet og undgå påvirkning af produktionsdata. Testmiljøet konfigureres ved hjælp af klassen *TestDatabaseHelper* og Dispose-metoden i selve testklassen. *TestDatabaseHelper* administrerer testdatabasen ved at nulstille den til en kendt tilstand før og efter hver test ved

hjælp af metoden TearDownAndBuildTestDatabase. Denne proces sikrer, at testene udføres i et isoleret og forudsigeligt miljø, hvor de ikke påvirker hinanden, og de afhænger ikke af databasens tilstand før testkørsel. For at undgå lækage af data mellem tests køres de sekventielt, hvilket er afgørende for at forhindre data fra én test i at påvirke en anden.

Integrationstests er primært fokuseret på metoder, der interagerer med databasen og andre systemkomponenter. Ved at holde koncentration om disse områder sikres det, at systemet fungerer korrekt i et realistisk miljø. Et eksempel på dette ses på illustration 17.

```
[Fact]
 public void GetAllGameLobbies_TC1_ReturnsExpectedGameLobbies()
 {
     // Arrange
     InsertMockGameLobbies();
     InsertMockPlayers();
     AssosiatePlayersWithGameLobbies();
     SetValidOwnershipOfGameLobbies();

     GameLobbyDatabaseAccessor gameLobbyDatabaseAccessor = new
GameLobbyDatabaseAccessor(_configuration);
     PlayerDatabaseAccessor playerDatabaseAccessor = new PlayerDatabaseAccessor(_configuration);
     PlayerLogic playerLogic = new PlayerLogic(_configuration, playerDatabaseAccessor);
     GameLobbyLogic SUT = new GameLobbyLogic(_configuration, gameLobbyDatabaseAccessor,
playerLogic);

     // Act
     IEnumerable<GameLobbyModel> testResult = SUT.GetAllGameLobbies();

     // Assert
     Assert.True(testResult != null, "Test result is null.");
     Assert.True(testResult.Count() == 3, "Test result count is not 3.");

     foreach (var gameLobby in testResult)
     {
         Assert.True(gameLobby.GameLobbyId != 0, "Game lobby ID is 0.");
         Assert.True(gameLobby.LobbyName != null, "Game lobby name is null.");
         Assert.True(gameLobby.InviteLink != null, "Game lobby invite link is null.");
         Assert.True(gameLobby.PlayersInLobby != null, "Players in game lobby is null.");
         Assert.True(gameLobby.PlayersInLobby.Count(player => player.IsOwner) == 1, "There is
not exactly one owner in the game lobby.");
         Assert.True(gameLobby.PlayersInLobby.All(player =>
!string.IsNullOrEmpty(player.InGameName)), "One or more players in the game lobby do not have an
InGameName.");
         Assert.True(gameLobby.PlayersInLobby.Count <= gameLobby.AmountOfPlayers, "The number of
players in the game lobby exceeds the amount of players.");
     }
 }
```

Illustration 17: GetAllGameLobbies_TC1_ReturnsExpectedGameLobbies

Formålet med integrationstesten GetAllGameLobbies_TC1 er at sikre, at metoden GetAllGameLobbies i *GameLobbyLogic* korrekt henter og returnerer gamelobbies sammen med de tilknyttede spillere fra databasen. Testen bruger mock data til at simulere realistiske scenarier og verificere, at systemet fungerer som forventet, når forskellige komponenter interagerer.

Testen kontrollerer, at den returnerede liste af gamelobbies ikke er null og indeholder det forventede antal lobbies. Derudover sikrer den, at hver lobby har et gyldigt ID, navn, og invite link, samt at listen af spillere ikke er null. Det bekræftes også, at der er præcis én ejer i hver

lobby, at alle spillere har et InGameName, og at antallet af spillere i hver lobby ikke overstiger det specificerede antal.

Ved at gennemføre denne integrationstest valideres det, at systemet korrekt håndterer dataudvekslingen mellem databasen og forretningslogikken. Dette bidrager til at sikre systemets stabilitet og pålidelighed og forhindrer potentielle fejl, der kan påvirke brugeroplevelsen. Flere test cases kan tilgås i bilag 8 – Test Case tabeller.

Systemtest

En systemtest er en omfattende testmetode, der evaluerer hele systemets funktionalitet ved at simulere reelle brugsscenarier. Dets formål er at verificere, om systemet opfylder de specificerede krav og forventninger.

For at udføre systemtesten manuelt gennemgår testen typisk forskellige funktioner og scenarier i systemet som en bruger ville gøre det. Dette har omfattet at udføre handlinger, navigere gennem desktop- og webklienterne, og verificere om systemet reagerer korrekt på input og giver de forventede resultater, så dette er blevet gjort løbende igennem udviklingsfasen.

For at teste samtidighedsproblematikken i systemet, blev der foretaget en manuel systemtest. Programmet blev kørt fra to forskellige computere - en med et breakpoint placeret midt i joinLobby-funktionen i API'et, og en uden. Først udførtes kaldet fra computeren med breakpointet, hvor koden ville stoppe midt i kaldet på grund af breakpointet. Derefter udførtes kaldet fra computeren uden breakpointet for at observere, hvordan systemet håndterede samtidighed og interaktionen mellem de to kald.

Denne tilgang til systemtesten gav mulighed for at validere, at systemet fungerer som forventet i en reel brugssituation og identificere eventuelle problemer eller samtidighedsfejl, der kan opstå under brug.

Konklusion

Gennem projektarbejdet med udviklingen af spilleklienten IronPlay, der har til formål at adressere samtidighedsproblemer ved oprettelse og tilslutning til brugerdefinerede gamelobbies, har gruppen skabt et effektivt system. For at besvare problemformuleringen har gruppen udarbejdet User Stories for at fastsætte de funktionelle krav, beskrevet de ikke-funktionelle krav, og defineret de informationsmæssige krav gennem en domænemodel. Arkitekturen for IronPlay er designet med fem tiers bestående af en Webklient, Desktopklient, Webserver, API og en Database. Denne distribuerede arkitektur muliggør en mere effektiv skalering sammenlignet med en monolitisk struktur.

Under udviklingen af en frontend til IronPlay har brugeroplevelsen været central. For spillere var det vigtigt at mødes med et responsivt og brugervenligt interface, mens administratorer havde behov for et let navigerbart interface, der altid viser de nødvendige informationer.

For at understøtte den distribuerede arkitektur i backend er der udviklet et RESTful API, som leverer den nødvendige data til de to klienter gennem HTTP requests. API'et håndterer samtidighedsproblematikken, når spillere forsøger at tilslutte sig den samme lobby, ved at anvende Optimistic Concurrency Control (OCC). Ydermere, for at beskytte IronPlay mod ondsindede aktører, anvendes access tokens som bearer tokens til API'et. Disse access tokens kan kun opnås af de to klienter, da de er de eneste enheder, der har de nødvendige informationer for at generere en access token.

For at sikre kvaliteten af IronPlay's kode er der løbende skrevet automatiserede tests, der sikrer, at funktionaliteten til diverse User Stories fungerer korrekt. Samlet set udgør disse elementer et system, der effektivt håndterer samtidighedsproblemer og tilbyder en robust spilleklient, der imødekommer brugernes behov.

Dette projekt har ikke blot demonstreret, hvordan en distribueret spilleklient kan udvikles til at adressere samtidighedsproblemer ved oprettelse og tilslutning til gamelobbies, men har også givet gruppen værdifuld indsigt i sammensætningen af en distribueret systemarkitektur.

Refleksioner

Gennem udviklingsprocessen for IronPlay har gruppen gjort diverse overvejelser i forhold til systemets udvikling. Emner såsom løsningens organisering, dokumentation og mere, er blevet diskuteret løbende under udviklingen. Dette afsnit vil uddybe de refleksioner, som har fyldt meget eller er kommet op gentagne gange i løbet af udviklingen.

Solution struktur

Før projektets egentlige igangsætning diskuterede gruppen, hvilken form for solutionsstruktur IronPlay skulle følge. Her blev der overvejet, om det skulle være en løsning med mange underprojekter eller flere løsninger med få underprojekter.

Baseret på to forskellige spikes i projektet - en som fulgte en struktur med én solution og mange underprojekter, og en anden spike som fulgte en struktur med flere solutions og få underprojekter - valgte gruppen at gå med én solution med mange underprojekter. Dette skyldtes, at det ifølge spiken var mest overskueligt, at al koden til projektet kunne findes under den samme solution. Gruppen har dog i løbet af udviklingen fortrudt denne beslutning, da der i starten af projektet ikke blev taget højde for den mængde kode, som projektet ville komme til at indeholde efter flere ugers udvikling med flere udviklere.

Det har ikke haft store konsekvenser, men det har gjort det sværere at finde rundt i de forskellige projekter. Dette kunne løses ved refactoring, men gruppen ønskede ikke at gøre dette så tæt på projektets afslutning.

Håndtering af model klasser

Gruppen har ikke haft en tilstrækkelig samtale om, hvordan håndteringen af modelklasser skal foregå, hvilket har ført til, at modelmapperne til tider kan virke uoverskuelige. For eksempel har der ikke været en gennemgang af, hvordan data fra API'er og klienter skal håndteres. Som resultat bliver alt data sendt frem og tilbage via JSON, og i de fleste tilfælde er der blevet oprettet modelklasser, hvis eneste formål er at mappe denne data til og fra JSON. Her burde gruppen nok have fastsat klare retningslinjer for, hvornår det er nødvendigt at oprette en hel klasse for at mappe noget data til JSON, og hvornår det ikke er nødvendigt.

Derudover burde gruppen også have diskuteret håndteringen af Data Transfer Objects (DTO'er). API'et kunne have haft en model, som repræsenterede data i databasen, som så ville blive konverteret til en DTO. I stedet blev der blot oprettet model klasser, som indeholdt den data, klienten har brug for, men disse klasser hedder ikke noget, der indikerer, at de er DTO'er. Dette har skabt en vis forvirring omkring konceptet og formålet med disse klasser. En klarere definition og navngivning af disse klasser kunne have afhjulpet denne forvirring og gjort koden mere overskuelig.

Litteraturliste

[1] "Video Games - Worldwide." Accessed: May 21, 2024. [Online]. Available: https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide#revenue

[2] Max Rehkopf, "User stories with examples and a template." Accessed: May 16, 2024. [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories

[3] Piyush Rahate, "Exploring the Difference: Definition of Done vs Acceptance Criteria." Accessed: May 16, 2024. [Online]. Available: https://agilemania.com/definition-of-done-vs-acceptance-criteria

[4] H. M. Deitel, P. J. Deitel, and D. R. Choffnes, "16.6.1 Transmission Control Protocol (TCP)," in *Operating Systems*, 3rd ed., Pearson Prentice Hall, 2004, pp. 760–761.

[5] H. M. Deitel, P. J. Deitel, and D. R. Choffnes, "16.5.1 Hypertext Transfer Protocol (HTTP)," in *Operating Systems*, 3rd ed., Pearson Prentice Hall, 2004, pp. 757–758.

[6] "ASP:NET Core." [Online]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet

[7] "Web Framework Benchmarks." May 2023. [Online]. Available: https://www.techempower.com/benchmarks/#hw=ph&test=composite&section=data-r22

[8] "GitHub Copilot and Visual Studio 2022." [Online]. Available: https://visualstudio.microsoft.com/vs/

[9] "Desktop Guide (Windows Forms .NET)." May 2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-8.0

[10] "What is middleware?" May 2022. [Online]. Available: https://www.redhat.com/en/topics/middleware/what-is-middleware

[11] Lokesh Gupta, "What is REST?" Accessed: May 22, 2024. [Online]. Available: https://restfulapi.net/

[12] L. K. Jeppesen, "Lecture Note Anomalies in Databases."

[13] "Authentication vs. Authorization", [Online]. Available: https://auth0.com/docs/get-started/identity-fundamentals/authentication-and-authorization

[14] "Overview of ASP.NET Core authentication." May 2023. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-8.0

[15] F. James, "What Is Social Login and Is It Worth Implementing?" May 2020. [Online]. Available: https://www.okta.com/blog/2020/08/social-login/

[16] “Introduction to JSON Web Tokens.” [Online]. Available: https://jwt.io/introduction/

[17] “Bcrypt.net.” May 2022. [Online]. Available: https://github.com/BcryptNet/bcrypt.net

[18] “Unit testing C# in .NET using dotnet test and xUnit.” 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test

Bilag

Bilag 1 – IronPlay guide

Setup af IronPlay på lokal maskine

Trin 1: Hent kode

Klon koden fra GitHub-repositoriet ved at køre følgende kommando i din terminal:

```
git clone https://github.com/RadenDK/ThirdSemesterProject.git
```

Trin 2: Indsæt environment variabler

Du skal oprette konfigurationsfiler til de forskellige projekter. Nedenfor er eksempler på, hvordan disse filer kan se ud.

Desktop applikation (appsettings.json)

Opret en fil ved navn `appsettings.json` i Desktop-projektet med følgende indhold:

```
{
    "AllowDesktopApp": {
        "Username": "DesktopClient",
        "Password": "erpoeiuifdp-3fb3-1234678-9634-lskjfdslkjdsfh",
        "Role": "DesktopClient"
    }
}
```

Webklient (appsettings.json)

Opret en fil ved navn `appsettings.json` i Webklient-projektet med følgende indhold:

```
{
  "AllowWebApp": {
    "Username": "WebClient",
    "Password": "bb175ae2-3fb3-418c-9634-b124e7286473",
    "Role": "WebClient"
  }
}
```

API testprojekt (appsettingsForTesting.json)

Opret en fil ved navn `appsettingsForTesting.json` i API testprojektet med følgende indhold:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=hildur.ucn.dk;Initial Catalog=DMA-CSD-V23_10478735;User ID=DMA-CSD-V23_10478735;Password=Password1!;Trust Server Certificate=True"
  }
}
```

Når du har oprettet disse filer og indsat de nødvendige environment variabler, burde din lokale opsætning af IronPlay være klar til brug.

Webklient

Eksisterende Login:

Username: Player1

Password: Player1

Create account:

1. Tryk på “Create Account”
2. Indtast oplysninger I alle felter
3. Tryk på ”Create Account”
4. Denne konto kan nu bruges til at logge ind med

Create gamelobby:

1. Log ind med eksisterende login eller eget login
2. Tryk på ”Play”
3. Tryk på ”Create Custom Lobby”
4. Indtast et gamelobby navn
5. Indtast team size
6. Indtast et password (dette er optional)
7. Tryk ”Create Lobby”

Join gamelobby:

1. Log ind med eksisterende login eller eget login
2. Tryk på ”Play”
3. Tryk på en gamelobby uden password for at joine
4. Tryk på ”leave” for at forlade en gamelobby
5. Tryk på ”gamelobby2” og indtast dette password: ”password”
6. Tryk på ”leave” for at forlade en gamelobby

Desktopklient

Eksisterende login:

Username: 1

Password: Player1

Update Player:

1. Log ind med eksisterende login
2. Tryk på ”Administrate Players”
3. Dobbeltklik på den spiller, som skal opdateres
4. Indtast nye oplysninger i alle eller nogle af felterne
5. Tryk på ”Confirm”

Delete Player:

1. Log ind med eksisterende login
2. Tryk på ”Administrate Players”
3. Dobbeltklik på den spiller, som skal opdateres
4. Tryk på ”Delete”
5. Tryk på ”Ja”

Bilag 2 – User Stories

Epic: Player

Userstory: Update profile

- As a: Max Thomsen
- I want to: Update my profile
- So that: I can update old information

  Acceptkriterier

  - Can update Password
  - Can update Email
  - Can Update In game name

Epic: Login

Userstory: Create profile

- As a: Max Thomsen
- I want to: Create a profile
- So that: I can be able to play

  Acceptkriterier

  - Unique username which must not be explicit
  - Valid password which includes at least 6 characters, an Uppercase letter, a number, and a special character
  - Entered email and birthdate
  - A create button, to create the profile

Userstory: See Homepage

- As a: Max Thomsen
- I want to: See the homepage.
- So that: I can get an overview of the gameclient

  Acceptkriterier:

  - You should load into the Homepage when logged in

Userstory: User login

- As a: Max Thomsen
- I want to: Login to my account.
- So that: I can be able to play.

  Acceptkriterier:

- You should have entered a valid username and password
- You load into the game client on your own profile
- Player status is changed to online

Userstory: Admin login

- As a: Tom Hansen
- I want to: Login to my account.
- So that: I can be able to administer game client.

Acceptkriterier:

- You should have entered a valid username and password
- You load into the administrator desktop app
- Admin status changed to online.

Userstory: See Admin Dashboard

- As a: Tom Hansen
- I want to: See the dashboard.
- So that: I can get an overview of the gameclient management

Acceptkriterier:

- You should load into the Dashbboard when logged in

Userstory: Logout

- As a: Max Thomsen
- I want to: Logout of my account.
- So that: I can be able to quit the client

Acceptkriterier:

- A logout button can be clicked
- Clicking the browser exit button exits the client
- A popup window is presented with the options exit or sign out
- Player status is changed to offline

Epic: Lobby

Userstory: Make Lobby

- As a: Max Thomsen
- I want to: make a lobby

- So that: I can setup a game for my friends

  Acceptkriterier

  - Set maximum players
  - Set an optional password
  - Set game lobby name

Userstory: Update Lobby

- As a: Max Thomsen
- I want to: update my lobby
- So that: adjust the settings if I need to change them

  Acceptkriterier

  - Change maximum players
  - Set or change optional password
  - Change lobby name

Userstory: Start game

- As a: Max Thomsen
- I want to: start game
- So that: I can play the game

  Acceptkriterier

  - Button to start the game
  - Minimum one player in the lobby

Userstory: View lobbies

- As a: Max Thomsen
- I want to: view lobbies
- So that: I can play

  Acceptkriterier

  - See a list of all lobbies
  - See the information of each lobby (private/public, amount of players, lobby name)
  - Be able to search for lobbies by lobby name in a searchbar
  - Be able to sort lobbies by category
  - Be able to filter the lobbies

Userstory: Join a lobby by lobbylist

- As a: Max Thomsen

- I want to: join a lobby
- So that: I can play with my friends

  Acceptkriterier

  - Should be able to enter lobby by doubleclicking on the lobby in the lobbylist
  - Enter password, if there is one
  - Lobby isn't full

Userstory: Join a lobby by invite

- As a: Max Thomsen
- I want to: join a lobby
- So that: I can play with my friends

  Acceptkriterier

  - Should be able to enter lobby by accepting invite from friend
  - Enter password, if there is one
  - Lobby isn't full

Userstory: Join a lobby by link

- As a: Max Thomsen
- I want to: join a lobby
- So that: I can play with my friends

  Acceptkriterier

  - Should be able to enter lobby by following the sent link
  - Enter password, if there is one
  - Lobby isn't full

Userstory: Leave a lobby

- As a: Max Thomsen
- I want to: leave a lobby
- So that: I can quit playing or go to another lobby

  Acceptkriterier

  - Should be enabled to leave a lobby by clicking a quit button

Userstory: Chat in lobby

- As a: Max Thomsen
- I want to: chat with people in lobby
- So that: I can chat with people

  Acceptkriterier

  - A window where there is a chat bar
  - All players in the lobby can send messages in the chat window
  - All players in the lobby can read each other's messages
  - No explicit words allowed

Userstory: Kick from lobby

- As a: Max Thomsen
- I want to: kick people from lobby
- So that: I don't have to play with them

  Acceptkriterier

  - The lobby owner should be able to kick any player
  - The player who is kicked is removed from the lobby
  - The team can votekick a player
  - The lobby owner can't be kicked

Userstory: Invite people to lobby

- As a: Max Thomsen
- I want to: Invite people to lobby
- So that: I can invite my friends

  Acceptkriterier

  - The players can invite people from their friendlist
  - The players can copy an invite link and send to friends

Epic: Friends

Userstory: Add a friend

- As a: Max Thomsen
- I want to: add a friend
- So that: I can socialize in game

  Acceptkriterier

  - Search a friends username

- Add the friend to your friendlist
- The pending friend request list is updated

Userstory: Accept a friend request

- As a: Max Thomsen
- I want to: accept a friend request
- So that: I can socialize in game

Acceptkriterier

- Accept a pending friend request
- The friendlist is updated with the new friend

Userstory: See all friends

- As a: Max Thomsen
- I want to: see all my friends
- So that: I know who is online and able to play

Acceptkriterier

- See a tab with awaiting friend requests
- See a tab with a list of your friends
- See friends information (username, online status)
- Sort friendlist in online/offline

Userstory: Chat with friend

- As a: Max Thomsen
- I want to: Chat with my friend.
- So that: I can communicate with them.

Acceptkriterier

- Click on a friend from the friendlist to chat
- Can send chat messages to eachother
- No explicit words allowed
- The thread of messages sent is saved for 2 weeks

Userstory: Remove friend

- As a: Max Thomsen
- I want to: Remove a friend

- So that: I don’t have to see them in my friends list

  Acceptkriterier

  - Rightclick shows a drop down with the option to unfriend
  - The friendlist is updated and the friend is removed

Epic: Store

Userstory: View store

- As a: Max Thomsen
- I want to: buy a product
- So that: I can personalize my account

  Acceptkriterier

  - Click store tab
  - See a list of every skin
  - Filter by name and price
  - Sort by releasedate

Userstory: Buy a product

- As a: Max Thomsen
- I want to: Buy a product
- So that: I can personalize my account

  Acceptkriterier

  - Click on a product, which reveals a buy screen
  - Click on buy with currency
  - The player have enough in game currency
  - The players balance is subtracted by the skin cost
  - The skin is now shown in collection

Userstory: Buy in game currency

- As a: Max Thomsen
- I want to: buy something in the store
- So that: I can personalize my account

  Acceptkriterier

  - To buy currency the currency icon is clicked
  - A pop up window is shown with different amounts of currency

- Choosing a amount and clicking buy
- You are redirected to a NETS popup, which handles the payment transaction
- The players currency is updated with the bought amount

Epic: Profile

Userstory: See collection

- As a: Max Thomsen
- I want to: see what I have bought in the store
- So that: I can use the products

Acceptkriterier

- Click Collection tab
- See a list of your skins
- Filter your skins by name
- Sort by release date
- Click on a skin in collection to set as profile image

Userstory: Look at own stats

- As a: Max Thomsen
- I want to: look at my own stats
- So that: I can improve

Acceptkriterier

- Click profile tab
- See players stats in a dashboard

Userstory: Look at friends stats

- As a: Max Thomsen
- I want to: look at my own stats
- So that: I can improve

Acceptkriterier

- Right click on friend to access a dropdown and click on profile
- On friends profile it shows a dashboard of friends stats

Epic: Administrate Player

Userstory: Find a user

- As a: Tom Hansen
- I want to: Find a player
- So that: I can see information about the player

  Acceptkriterier

  - *The admin should be able to search by such things as username and ingame name*

Userstory: Ban a user

- As a: Tom Hansen
- I want to: Ban a player
- So that: I can improve the community

  Acceptkriterier

  - *By clicking on a player the admin should be able to click ban which would turn the banned attribute on a player to true*

Userstory: Unban a user

- As a: Tom Hansen
- I want to: unban a user
- So that: I can give a player a second chance

  Acceptkriterier

  - *By clicking on a player the admin should be able to click ban which would turn the banned attribute on a player to false*

Userstory: Delete user

- As a: Tom Hansen
- I want to: Delete a user
- So that: I can improve the community

  Acceptkriterier

  - *From a list of found players there should be a button that would delete the user. A confirm window should popup after the button is clicked*
  - *The player is then deleted from the database*

Userstory: Update user

- As a: Tom Hansen
- I want to: Update a user
- So that: I can help a user who is having problems with their account

  Acceptkriterier

- *The user should be able to change information such as the account password, ingamename, username, email address and ban status*

Epic: Administrate Lobby

Userstory: Find a lobby

- As a: Tom Hansen
- I want to: Find a lobby
- So that: view details about a certain lobby

Acceptkriterier

- *The user should be able to find a lobby from a list of lobbies based on lobbyname or lobby owner*

Userstory: Delete lobby

- As a: Tom Hansen
- I want to: Delete a lobby
- So that: I can improve the lobby selection

Acceptkriterier

- *From a list of found lobbies there should be a button that would delete the lobby. A confirm window should popup after the button is clicked*
- *The lobby is then deleted from the database*
- *If there were players in that lobby the player lobby status is updated to not be in a lobby*

Epic: Administrate Store

Userstory: Find a skin

- As a: Tom Hansen
- I want to: Find a skin
- So that: view details about a certain skin

Acceptkriterier

- *The user should be able to find a skin from a list of skins based on the name of the skin*

Userstory: Add skin

- As a: Tom Hansen
- I want to: Add a skin to the store
- So that: I can improve the store selection

Acceptkriterier

- *The user should be able to add a skin by uploading the skin files, setting a price and an amount of skins.*
- *The skin is then saved in the database*
- *The skin is then able to be purchased by players*

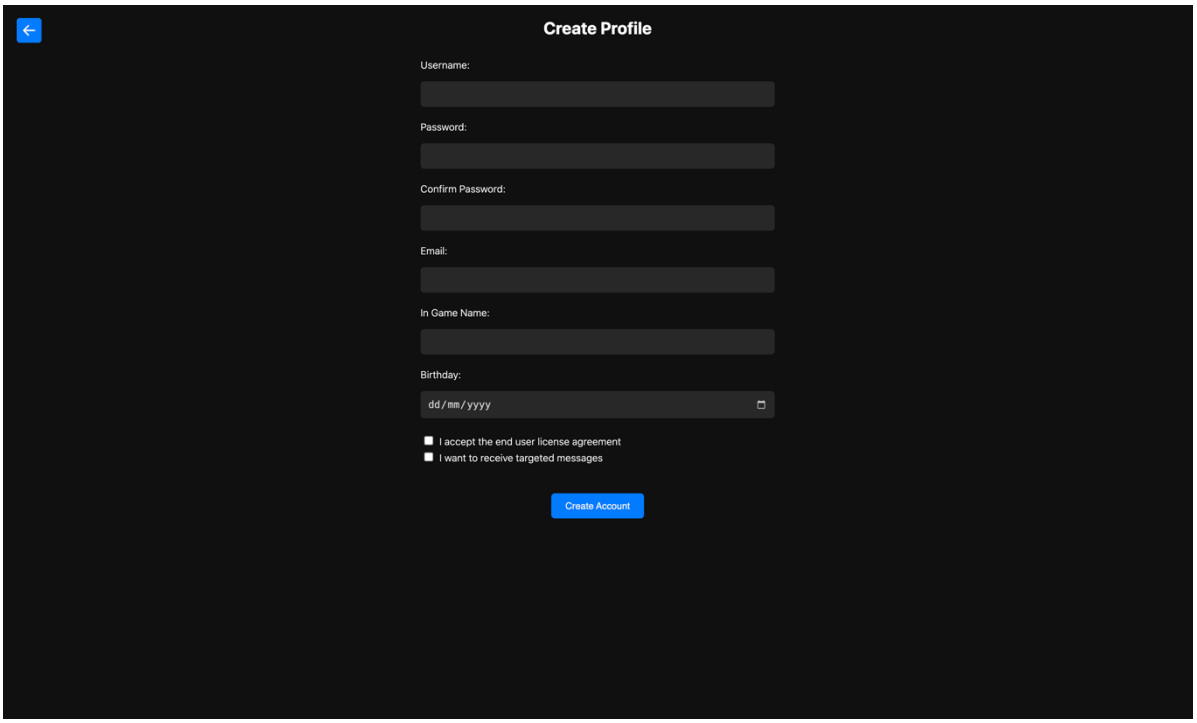
Userstory: Remove skin

- As a: Tom Hansen
- I want to: Remove a skin from the store
- So that: I can improve the store selection

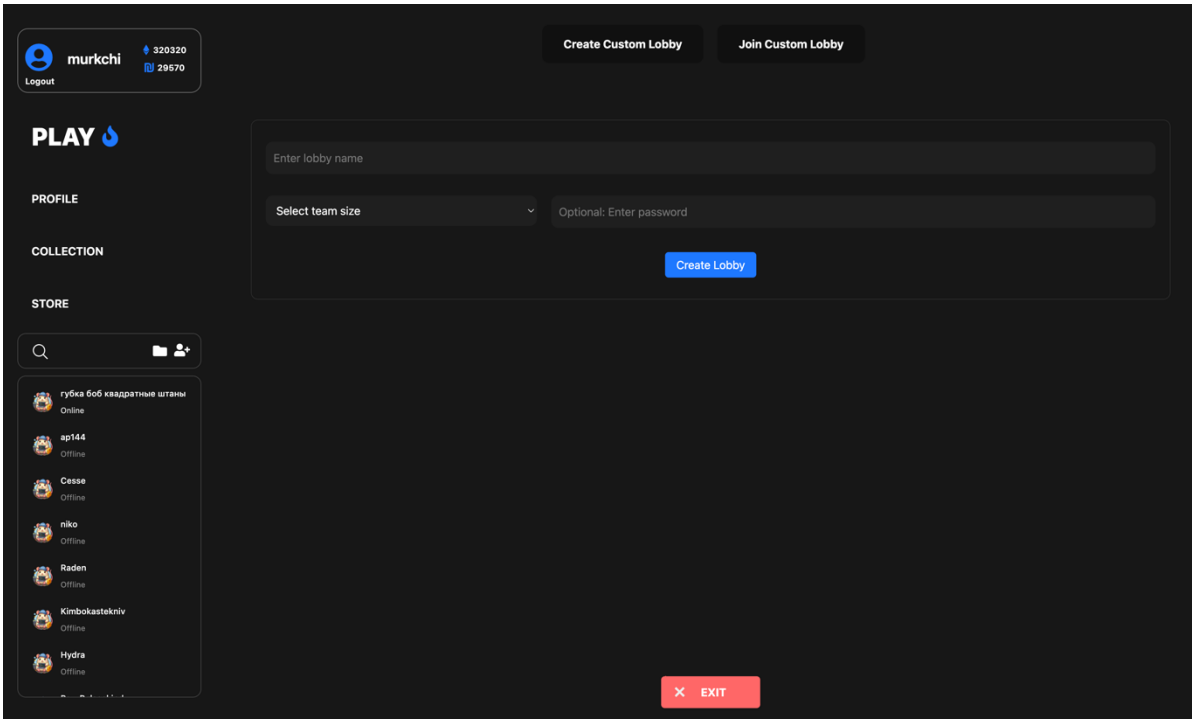
Acceptkriterier

- *From a list of found skins there should be a button that would remove the skin from the store. A confirm window should popup after the button is clicked*
- *The skin is then no longer able to be purchased*

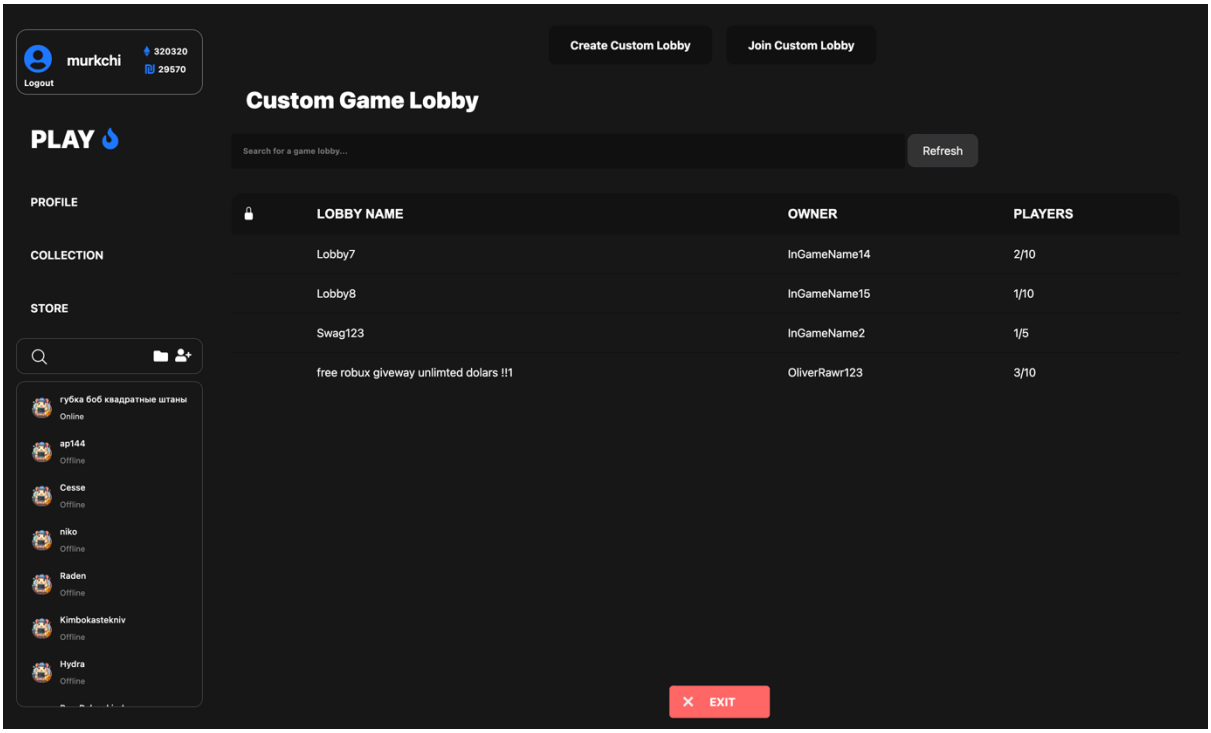
Bilag 3 – Webklient



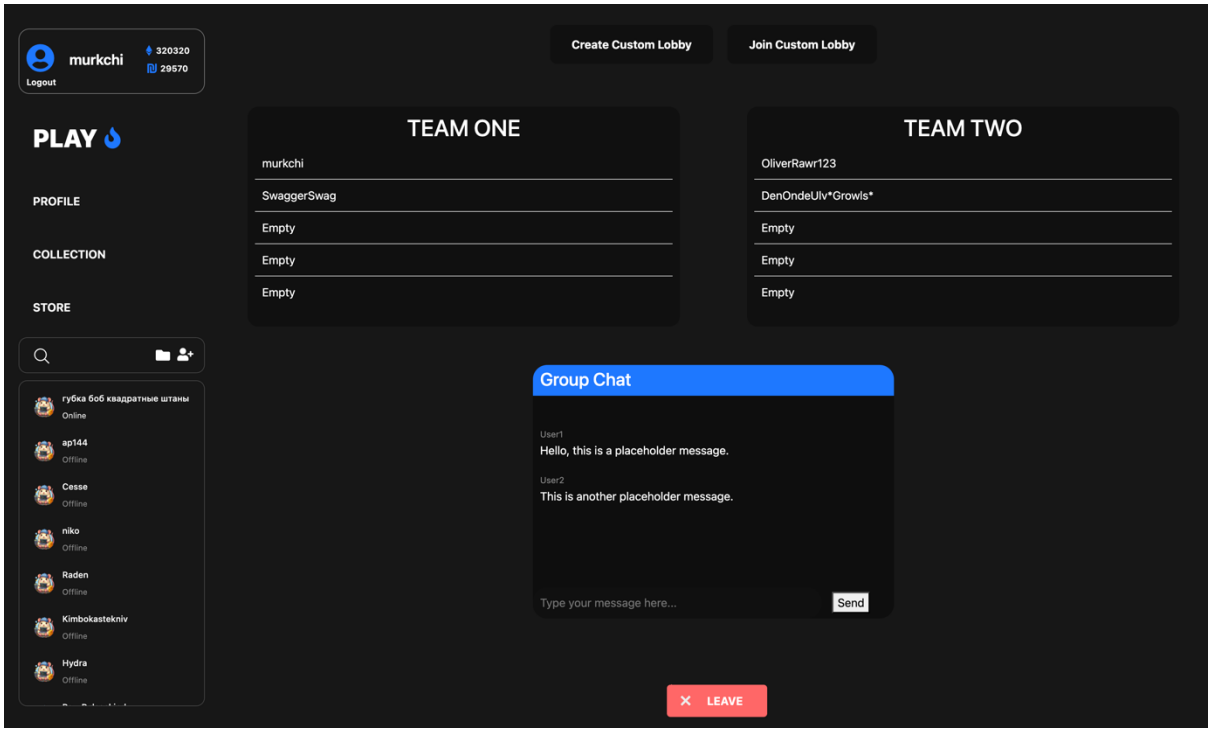
'Create account' view fra Loginskærmen.



'Create lobby' view



'View lobby' view



'Join lobby' view

Bilag 4 – Middleware versions

API

Middleware	Version
.net version	Net8.0
BCrypt.Net-Next	4.0.3
Dapper	2.1.35
Microsoft.AspNetCore.Authentication.JwtBearer	8.0.4
Microsoft.Data.SqlClient	5.2.0
Newtonsoft.Json	13.0.3
Swashbuckle.AspNetCore	6.4.0

Desktopklient

Middleware	Version
.net version	Net8.0
Microsoft.Extensions.Configuration	8.0.0
Microsoft.Extensions.Configuration.Abstractions	8.0.0
Microsoft.Extensions.Configuration.FileExtensions	8.0.0
Microsoft.Extensions.Configuration.Json	8.0.0
Newtonsoft.Json	13.0.3
System.IdentityModel.Tokens.Jwt	7.1.2

Webklient

Middleware	Version
.net version	Net8.0
Microsoft.VisualStudio.Web.CodeGeneration.Design	8.0.2
System.IdentityModel.Tokens.Jwt	6.35.0

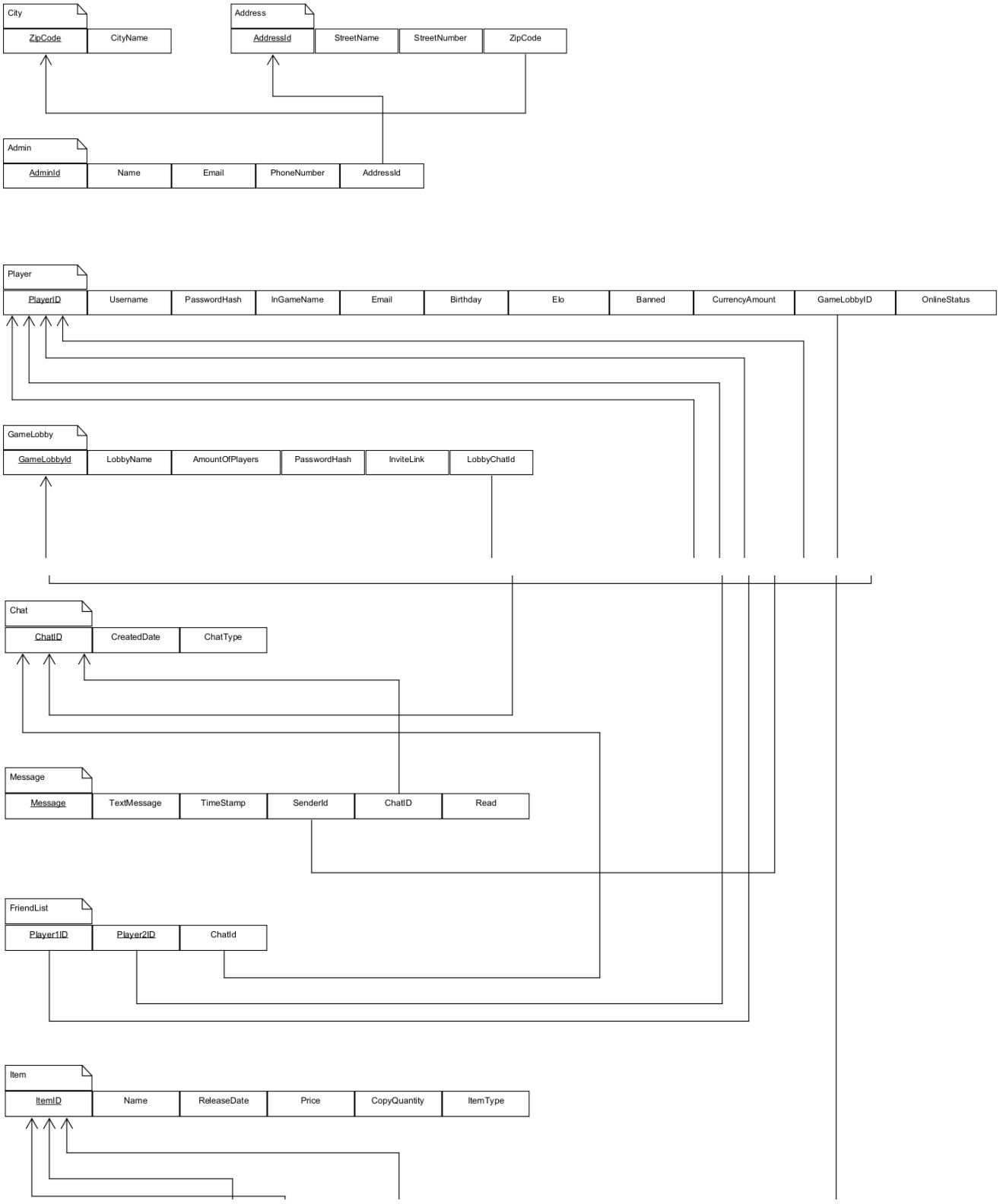
API tests

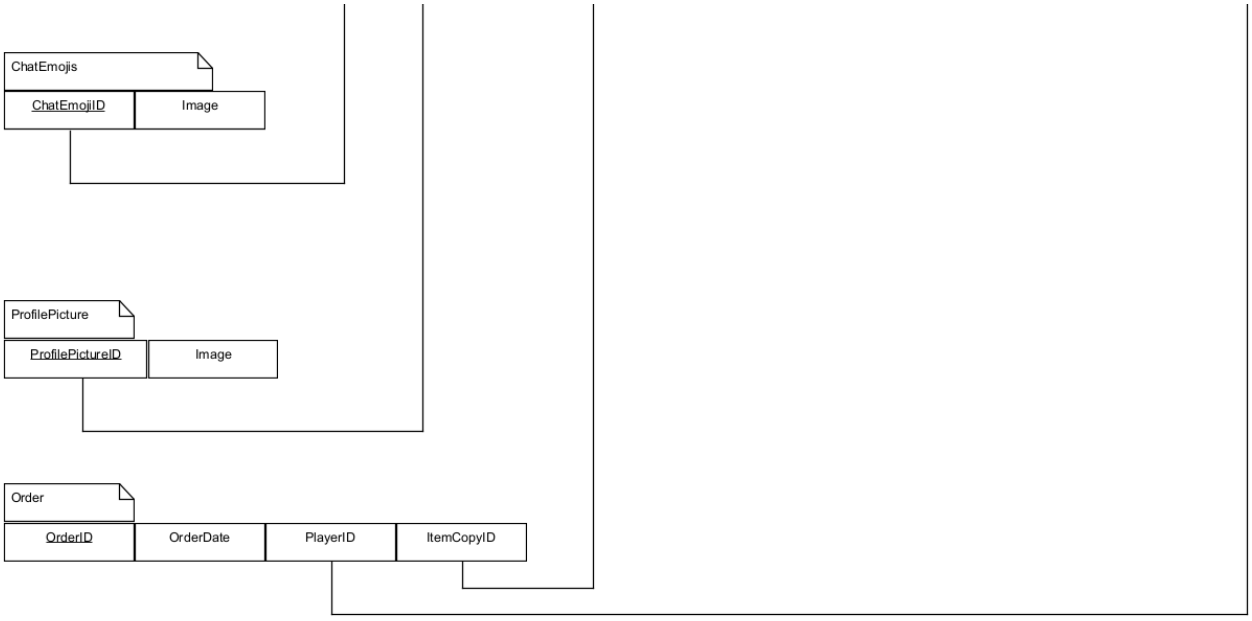
Middleware	Version
.net version	Net8.0
coverlet.collector	6.0.0
Moq	4.20.70
xunit	2.5.3
xunit.runner.visualstudio	2.5.3
Microsoft.NET.Test.Sdk	17.8.0

Bilag 5 – Risk Register

Risk	Probability	Effect	Action
Sygdom/Frafald (1)	**Høj**	**Medium**	Hold løbende gruppemedlemmer informeret om overordnet helbreds status.
Forsinkelser ift. Projektet (2)	Medium	Medium	Fælles identificeres og løses eventuelle flaskehalse eller forsinkende faktorer i projektplanen
Fysiske forsinkelser over 30 min (3)	Lav	Lav	Løbende hold gruppen opdateret om forsinkelse
Holde fokus (4)	**Høj**	**Medium**	Udnævn et gruppemedlem hvert sprint med ansvar om at få fokus tilbage på sporet
Ukendte teknologier (5)	**Høj**	**Medium**	Lav spikes inden teknologien skal anvendes i projektet. Nem adgang til vidensdeling blandt gruppemedlemmer
Tekniske problemer (6)	Medium	Medium	Pair program resten af dagen sammen med en anden
Udbrændthed (7)	Medium	Medium	Hold regelmæssige gruppeaktiviteter i fritiden
Ny arbejdsmetode SCRUM (8)	Medium	Medium	Planlæg løbende evalueringer af SCRUM-processen for at identificere og løse eventuelle udfordringer.
Dårlig kommunikation internt (9)	Lav	Medium	Hold løbende diskussioner om hvordan kommunikation skal behandles
Dårligt implementering af arkitektur (10)	**Medium**	**Høj**	Holde løbende review fortaget af minimum 3 gruppemedlemmer inkluderet dem som skrev koden
Manglende opfyldelse af krav (11)	Lav	Høj	Implementer regelmæssige krav gennemgange og valideringer for at sikre, at projektet lever op til kravene
Kontakt til vejleder (12)	Medium	Medium	Opret en backup-plan for at sikre kontinuitet, hvis der opstår kommunikationsproblemer med vejlederen
Håndtering af personale, som er planlægningsdrevet (13)	**Høj**	**Medium**	Løbende kontakt med vejledere omkring diverse problemstillinger som skyldes manglende erfaring

Bilag 6 – Relationel Model




ChatEmojis
ChatEmojiID
Image
ProfilePicture
ProfilePictureID
Image
Order
OrderID
OrderDate
PlayerID
ItemCopyID

Bilag 7 – Database Scripts

Database tables

```
use [GameClient];

-- Drop tables in reverse order of creation
DROP TABLE IF EXISTS [Order];
DROP TABLE IF EXISTS ItemCopy;
DROP TABLE IF EXISTS ProfilePicture;
DROP TABLE IF EXISTS ChatEmojis;
DROP TABLE IF EXISTS Item;
DROP TABLE IF EXISTS FriendList;
DROP TABLE IF EXISTS [Message];
DROP TABLE IF EXISTS Player;
DROP TABLE IF EXISTS GameLobby;
DROP TABLE IF EXISTS Chat;
DROP TABLE IF EXISTS [Admin];
DROP TABLE IF EXISTS [Address];
 DROP TABLE IF EXISTS [City];
GO
-- Creating the tables

CREATE TABLE Chat(
    ChatID INT IDENTITY(1,1) PRIMARY KEY,
    CreatedDate DATETIME DEFAULT GETDATE(),
    ChatType NVARCHAR (50) NOT NULL
);


CREATE TABLE GameLobby (
    GameLobbyID INT IDENTITY(1,1) PRIMARY KEY,
    LobbyName NVARCHAR (50) NOT NULL,
    AmountOfPlayers INT DEFAULT 10,
    PasswordHash NVARCHAR (MAX) DEFAULT NULL,
    InviteLink VARCHAR (50) NOT NULL,
    LobbyChatId INT NOT NULL,
    FOREIGN KEY (LobbyChatId) REFERENCES Chat(ChatID)
);


CREATE TABLE Player(
    PlayerID INT IDENTITY(1,1) PRIMARY KEY,
    Username NVARCHAR (50) NOT NULL,
    PasswordHash NVARCHAR (MAX) NOT NULL,
    InGameName NVARCHAR (50) NOT NULL,
    Email VARCHAR (50) DEFAULT NULL,
    Birthday DATETIME NOT NULL,
    Elo INT DEFAULT 0,
    Banned bit DEFAULT 0,
    CurrencyAmount INT DEFAULT 0,
    GameLobbyID INT DEFAULT NULL,
    OnlineStatus bit DEFAULT 0,
    IsOwner bit DEFAULT 0,

    FOREIGN KEY (GameLobbyID) REFERENCES GameLobby(GameLobbyID)
);


CREATE TABLE Message (
    MessageID INT IDENTITY(1,1) PRIMARY KEY,
    TextMessage NVARCHAR (MAX) NOT NULL,
    [TimeStamp] DATETIME DEFAULT GETDATE(),
    SenderID INT NOT NULL,
    ChatID INT NOT NULL,
    [Read] bit DEFAULT 0,
    FOREIGN KEY (SenderID) REFERENCES Player(PlayerID),
    FOREIGN KEY (ChatID) REFERENCES Chat(ChatID)
);


CREATE TABLE FriendList(
    Player1ID INT NOT NULL,
    Player2ID INT NOT NULL,
    ChatId INT DEFAULT NULL,
    FOREIGN KEY (Player1ID) REFERENCES Player(PlayerID),
    FOREIGN KEY (Player2ID) REFERENCES Player(PlayerID),
    FOREIGN KEY (ChatId) REFERENCES Chat(ChatID),
    CONSTRAINT PK_FriendList PRIMARY KEY (Player1ID, Player2ID)
);
```

```
CREATE TABLE [Order] (
    OrderID INT IDENTITY(1,1) PRIMARY KEY,
    OrderDate DATETIME DEFAULT GETDATE(),
    PlayerID INT NOT NULL,
    ItemCopyID INT NOT NULL,
    FOREIGN KEY (ItemCopyID) REFERENCES ItemCopy(ItemCopyID),
    FOREIGN KEY (PlayerID) REFERENCES Player(PlayerID)
);

 CREATE TABLE City (
    ZipCode INT PRIMARY KEY,
    CityName NVARCHAR (50) NOT NULL
);

CREATE TABLE [Address] (
        AddressId INT IDENTITY(1,1) PRIMARY KEY,
        StreetName NVARCHAR (50) NOT NULL,
        StreetNumber INT NOT NULL,
        ZipCode INT NOT NULL,
        FOREIGN KEY (ZipCode) REFERENCES City(ZipCode)
    );

CREATE TABLE Admin (
        AdminID INT IDENTITY(1,1) PRIMARY KEY,
        [Name] NVARCHAR (50) NOT NULL,
        Email VARCHAR (50) NOT NULL,
        CprNumber VARCHAR (10) NOT NULL UNIQUE,
        PhoneNumber VARCHAR (10) NOT NULL,
        AddressId INT NOT NULL,
        PasswordHash NVARCHAR (MAX) NOT NULL,
        FOREIGN KEY (AddressId) REFERENCES Address(AddressId)
    );

GO
```

```
CREATE TABLE Item (
    ItemID INT IDENTITY(1,1) PRIMARY KEY,
    [Name] NVARCHAR (50) NOT NULL,
    ReleaseDate DATETIME DEFAULT GETDATE(),
    Price INT NOT NULL,
    CopyQuantity INT NOT NULL,
    ItemType NVARCHAR (50) NOT NULL
);


CREATE TABLE ChatEmojis (
    ChatEmojiID INT PRIMARY KEY,
    [Image] VARBINARY(MAX) NOT NULL,
    FOREIGN KEY (ChatEmojiID) REFERENCES Item(ItemID)
);


CREATE TABLE ProfilePicture (
    ProfilePictureID INT IDENTITY(1,1) PRIMARY KEY,
    [Image] VARBINARY(MAX) NOT NULL,
    FOREIGN KEY (ProfilePictureID) REFERENCES Item(ItemID)
);


CREATE TABLE ItemCopy (
    ItemCopyID INT IDENTITY(1,1) PRIMARY KEY,
    ItemID INT NOT NULL,
    FOREIGN KEY (ItemID) REFERENCES Item(ItemID)
);
```

Database mockdata

```
    -- inserting mock data

    -- Insert data into Chat
    INSERT INTO Chat (chatType) VALUES ('GameLobby');
    INSERT INTO Chat (chatType) VALUES ('GameLobby');
    INSERT INTO Chat (chatType) VALUES ('Friend');
    INSERT INTO Chat (chatType) VALUES ('Friend');
    INSERT INTO Chat (chatType) VALUES ('Friend');
    INSERT INTO Chat (chatType) VALUES ('GameLobby');
    INSERT INTO Chat (chatType) VALUES ('GameLobby');
    INSERT INTO Chat (chatType) VALUES ('GameLobby');
    INSERT INTO Chat (chatType) VALUES ('GameLobby');
    INSERT INTO Chat (chatType) VALUES ('GameLobby');
    INSERT INTO Chat (chatType) VALUES ('GameLobby');
    INSERT INTO Chat (chatType) VALUES ('GameLobby');
    INSERT INTO Chat (chatType) VALUES ('GameLobby');


-- Insert data into GameLobby
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('Lobby1', '$2a$11$GKvgzKXgwSzYug.moJPTw.DN.vw5ZgM.nSaZkw212GTuDUu8lyXL6', 'link1', 1); -- Password is 'password'
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('Lobby2', NULL, 'link2', 2);
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('MyLobbu', '$2a$11$GKvgzKXgwSzYug.moJPTw.DN.vw5ZgM.nSaZkw212GTuDUu8lyXL6', 'link3', 3); -- Password is 'password'
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('Lobby4', NULL, 'link4', 4);
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('Lobby5', NULL, 'link5', 5);
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('Lobby6', '$2a$11$GKvgzKXgwSzYug.moJPTw.DN.vw5ZgM.nSaZkw212GTuDUu8lyXL6', 'link6', 6); -- Password is 'password'
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('Lobby7', NULL, 'link7', 7);
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('Lobby8', NULL, 'link8', 8);
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('Lobby9', '$2a$11$GKvgzKXgwSzYug.moJPTw.DN.vw5ZgM.nSaZkw212GTuDUu8lyXL6', 'link9', 9); -- Password is 'password'
INSERT INTO GameLobby (LobbyName, PasswordHash, inviteLink, lobbyChatId) VALUES ('Lobby10', NULL, 'link10', 10);


-- Insert data into Player
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player1', '$2a$11$GsmfIz3OPipR6f5avJUDTuFMItDbPZtiCmYScex0uZxo1z4Q6iP/i', 'InGameName1', GETDATE(), 'player1@example.com', 1);
-- Password is a hashed version of "Player1"
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player2', '$2a$11$j3D2bz5NUgeivWNZOc0dyOmSlfsHkMgmmcXK5cJTxUg6xZAZTZbKe', 'InGameName2', GETDATE(), 'player2@example.com', 1);
-- Password is a hashed version of "Player2"
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email) VALUES ('Player3', '$2a$11$I9Z4yRD6qBWux3xeoKZmDuKB4VQRBg.yDTCdRE4toQPVHA6b54Oze', 'InGameName3', GETDATE(), 'player3@example.com');
-- Password is a hashed version of "Player3"
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email) VALUES ('Player4', '$2a$11$V8M5WAQWb9cl68sm0LFHCe0mIaxXexs4yj0QnAy352LDtRsBF5TsK', 'InGameName4', GETDATE(), 'player4@example.com');
-- Password is a hashed version of "Player4"
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player5', '$2a$11$6ule5Buo7y1LSAKCA5VLreoKA5gFLIa.RnVH1Cg66HHucedfuSu3C', 'InGameName5', GETDATE(), 'player5@example.com', 1);
-- Password is a hashed version of "Player5"
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player6', '$2a$11$8nKGqpk2d4SRpXo2Nn13Z.E1nvCzoPBph79pD/pYgc.62mB3vP/a.', 'InGameName6', GETDATE(), 'player6@example.com', 1);
-- Password is a hashed version of "Player6"
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email) VALUES ('Player7', '$2a$11$o0ZNk3a0JBhXjqSzUP0ZQOlMFBVjwqsLkaxFhJhihWOhgyQHErwyK', 'InGameName7', GETDATE(), 'player7@example.com');
-- Password is a hashed version of "Player7"
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email) VALUES ('Player8', '$2a$11$3Y.Pqf9.U4ydTMEYvr4QFuAiWx6P9pQHAI1M/31pcXTsHa3pbHIHC', 'InGameName8', GETDATE(), 'player8@example.com');
-- Password is a hashed version of "Player8"
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player9', '$2a$11$GsmfIz3OPipR6f5avJUDTuFMItDbPZtiCmYScex0uZxo1z4Q6iP/i', 'InGameName9', GETDATE(), 'player9@example.com', 2);
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player10', 'CannotBeLoggedInTo', 'InGameName10', GETDATE(), 'player10@example.com', 3);
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player11', 'CannotBeLoggedInTo', 'InGameName11', GETDATE(), 'player11@example.com', 4);
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player12', 'CannotBeLoggedInTo', 'InGameName12', GETDATE(), 'player12@example.com', 5);
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player13', 'CannotBeLoggedInTo', 'InGameName13', GETDATE(), 'player13@example.com', 6);
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player14', 'CannotBeLoggedInTo', 'InGameName14', GETDATE(), 'player14@example.com', 7);
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player15', 'CannotBeLoggedInTo', 'InGameName15', GETDATE(), 'player15@example.com', 8);
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player16', 'CannotBeLoggedInTo', 'InGameName16', GETDATE(), 'player16@example.com', 9);
INSERT INTO Player (Username, PasswordHash, InGameName, Birthday, Email, GameLobbyID) VALUES ('Player17', 'CannotBeLoggedInTo', 'InGameName17', GETDATE(), 'player17@example.com', 10);
```

```
    -- Insert data into Message

    -- Messages in gamelobby 1
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('Hello', 1, 1);
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('How are everyone doing?', 1, 1);
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('I am fine', 2, 1);
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('I am also fine', 5, 1);
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('How are you doing player 1?', 2, 1);


    -- Messages between different players
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('Wanna game?', 3, 3);
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('HELLOO?', 3, 3);
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('Sorry I did not see you message', 4, 3);
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('I am ready later if you are still down', 4, 3);


    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('Tip top tissemand', 1, 4);
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('Hvad fanden snakker du om?', 5, 4);
    INSERT INTO Message (textMessage, senderID, ChatID) VALUES ('Flink flot fissekone', 1, 4);


    -- Insert data into FriendList
    INSERT INTO FriendList (Player1ID, Player2ID, chatId) VALUES (3, 4, 3);
    INSERT INTO FriendList (Player1ID, Player2ID) VALUES (2, 1);
    INSERT INTO FriendList (Player1ID, Player2ID) VALUES (2, 7);
    INSERT INTO FriendList (Player1ID, Player2ID) VALUES (7, 8);
    INSERT INTO FriendList (Player1ID, Player2ID) VALUES (1, 3);
    INSERT INTO FriendList (Player1ID, Player2ID, chatId) VALUES (1, 5, 4);

    -- Insert data into City
INSERT INTO City (ZipCode, CityName) VALUES (1000, 'Copenhagen');
INSERT INTO City (ZipCode, CityName) VALUES (2000, 'Frederiksberg');
```

```
-- Insert data into Address
INSERT INTO Address (StreetName, StreetNumber, ZipCode) VALUES ('Main Street', 1, 1000);
INSERT INTO Address (StreetName, StreetNumber, ZipCode) VALUES ('Second Street', 2, 2000);

-- Insert data into Admin

    INSERT INTO Admin ([Name], Email, CprNumber, PhoneNumber, AddressId, PasswordHash)
VALUES ('admin1', 'admin1@example.com', '1234567890', '1234567890', 1, '$2a$11$GsmfIz3OPipR6f5avJUDTuFMItDbPZtiCmYScex0uZxo1z4Q6iP/i');
```

Bilag 8 – Test Case tabeller

Test Case	Description	Expected Output	Epic / User Story
GetAllGameLobbies_TC1	Returns the expected game lobbies	True	Lobby / View Lobbies
GetAllGameLobbies_TC2	Returns an empty list if no lobbies could be found	True	Lobby / View Lobbies
GetAllGameLobbies_TC3	Assigns an owner if no owner could be found	True	Lobby / Join Lobby
GetAllGameLobbies_TC3	Owner status of random players gets updated in database if no owner was found	True	Lobby / Join Lobbies
GetAllGameLobbies_TC4	Removes ownership if multiple owners were found	True	Lobby / Join Lobbies
GetAllGameLobbies_TC4	Owner status of random players gets updated in database if too many owners were found	True	Lobby / Join Lobbies
GetAllGameLobbies_TC5	Does not include game lobbies with no players	True	Lobby / View Lobbies
GetAllGameLobbies_TC5	Deletes game lobbies with no players	True	Lobby / View Lobbies
GetAllGameLobbies_TC6	Removes players if too many were found	True	Lobby / Join Lobbies
GetAllGameLobbies_TC6	Updates players' GameLobbyId if removed from lobby with too many players	True	Lobby / Join Lobbies
CreateGameLobby_TC1	Rolls back transaction when error occurs	This test case doesn't have a boolean assertion, it verifies that a method was called	Lobby / Make lobby

Test cases : Game Lobby Integrations Test.

Test cases : Game Lobby Logic Unit Test

Test Case	Description	Expected Output	Epic / User story
GetAllGameLobbies_TC1	Returns the expected game lobbies	True	Lobby / View Lobbies
GetAllGameLobbies_TC2	Returns an empty list if no lobbies could be found	True	Lobby / View Lobbies
GetAllGameLobbies_TC3	Assigns an owner if no owner could be found	True	Lobby / Join Lobbies
GetAllGameLobbies_TC4	Removes ownership if multiple owners were found	True	Lobby / Join Lobbies
GetAllGameLobbies_TC5	Does include game lobbies with no players	True	Lobby / View Lobbies
GetAllGameLobbies_TC6	Removes players if too many were found	True	Lobby / Join Lobbies
CreateGameLobby_TC1	Creates game lobby and returns it	True	Lobby / Make Lobby
CreateGameLobby_TC2	Returns null if game lobby could not be created	True	Lobby / Make Lobby
CreateGameLobby_TC3	Hashes password before creating game lobby	True	Lobbyy / Make Lobby

Test cases : Player Logic Integration Test

Test Case	Description	Expected Output	Epic / User story
CreatePlayer_TC5	Verifies that the method hashes and stores password in the database	True	Login / Create Profile

Test cases : Player Logic Unit Test

Test Case	Description	Expected Output	Epic / User story
VerifyLogin_ReturnsTrue	Verifies that the login is successful when the username and password are correct	True	Login / Login
VerifyLogin_ThrowsException	Verifies that an exception is thrown when the username is invalid	Exception	Login / Login
VerifyLogin_ThrowsException	Verifies that an exception is thrown when the password for the username is not found	Exception	Login / Login
VerifyLogin_ThrowsException	Verifies that an exception is thrown when the username and password are empty	Exception	Login / Login
VerifyLogin_ThrowsException	Verifies that an exception is thrown when the username and password are null	Exception	Login / Login
CreatePlayer_TC1	Verifies that a player is created when the player information is valid	True	Login / Create Profile
CreatePlayer_TC2	Verifies that a player is not created when the username already exists	False	Login / Create Profile
CreatePlayer_TC3	Verifies that a player is not created when the in-game name already exists	False	Login / Create Profile

CreatePlayer_TC4	Verifies that a player is not created when the player creation process fails	False	Login / Create Profile
CreatePlayer_TC5	Verifies that a player is not created when the player information is missing	False	Login / Create Profile
UpdatePlayer_TC1	Verifies that a player is updated successfully	True	Administrate User / Update User
UpdatePlayer_TC2	Verifies that an exception is thrown when the updated username already exists	Exception	Administrate User / Update User
UpdatePlayer_TC3	Verifies that an exception is thrown when the updated in-game name already exists	Exception	Administrate User / Update User