

---

# Dungeons and Gnoblins

*Semesterprojekt - Teknisk Bilag*

---

Aarhus Institut for Elektro- og Computerteknologi

Vejleder: Jung Min Kim (Jenny)

## Authors:

Magnus Blaabjerg Møller, 202006492

Sune Andreas Dyrbye, 201205948

Morten Høgsberg, 201704542

Rasmus Engelund, 202007668

Anders Hundahl, 202007859

Oscar Dennis, 202009975

Jacob Hoberg, 201807602

Luyen Vu, 202007393

Afleveringsdato: 03-06-2022

Eksamineringsdato: 22-06-2022

# Contents

<b>1</b>	<b>Forord</b>	<b>5</b>
<b>2</b>	<b>Ordforklaring</b>	<b>5</b>
<b>3</b>	<b>Spilbeskrivelse</b>	<b>6</b>
3.1	Minimal Viable Product . . . . .	6
3.2	Login-screen . . . . .	6
3.3	Core gameplay layout . . . . .	7
3.4	Settings . . . . .	9
3.5	MapBeskrivelse . . . . .	10
<b>4</b>	<b>Systembeskrivelse</b>	<b>12</b>
<b>5</b>	<b>Kravspecifikation</b>	<b>12</b>
5.1	Funktionelle krav - User Stories . . . . .	13
5.2	Ikke-funktionelle krav . . . . .	15
5.3	MOSCOW krav . . . . .	17
5.4	Accepttest . . . . .	17
5.4.1	Funktionelle . . . . .	17
5.4.2	Ikke funktionelle . . . . .	23
<b>6</b>	<b>Analyser</b>	<b>28</b>
6.1	Teknologiundersøgelse: Unity - .Net . . . . .	28
6.1.1	Unity: . . . . .	28
6.1.2	.NET framework: . . . . .	28
6.2	Teknologiundersøgelse: Database SQL/NOSQL . . . . .	29
6.2.1	SQL . . . . .	29
6.2.2	No-SQL . . . . .	30
6.2.3	Konklusion . . . . .	30
6.3	Teknisk Analyse Backend . . . . .	30
6.3.1	Mulighed 1: WPF ->ASP.NET Core ->Entity Framework Core ->Database . . . . .	30
6.3.2	Mulighed 2: WPF ->Entity Framework Core ->database . . . . .	30
6.3.3	Konklusion . . . . .	31
<b>7</b>	<b>Arkitektur</b>	<b>32</b>
7.1	Systemarkitektur . . . . .	32
7.2	Frontend Arkitektur . . . . .	32
7.2.1	Pseudo Frontend Arkitektur . . . . .	33
7.3	GameEngine Arkitektur . . . . .	38
7.3.1	States og Character interagering . . . . .	38
7.3.2	Room State og Character interagering . . . . .	39
7.3.3	Combat State og Combat simulering . . . . .	39
7.3.4	Forbindelser til andre moduler . . . . .	40
7.4	Backend Arkitektur . . . . .	41
7.4.1	MVC . . . . .	41
7.4.2	C3-Model for backend . . . . .	42
7.4.3	REST . . . . .	44
7.4.4	Konklusion . . . . .	44
7.5	Database Arkitektur . . . . .	45
7.6	DAL Arkitektur . . . . .	49

<b>8</b>	<b>Design</b>	<b>50</b>
8.1	Overordnet System Design . . . . .	50
8.2	Frontend Design . . . . .	52
8.2.1	Room . . . . .	52
8.2.2	Combat . . . . .	53
8.2.3	Login . . . . .	54
8.2.4	Settings . . . . .	55
8.3	Game Controller Design . . . . .	57
8.3.1	Map Creation . . . . .	57
8.3.2	Player class og Items . . . . .	57
8.3.3	Enemies . . . . .	57
8.3.4	Combat . . . . .	57
8.3.5	Saves . . . . .	58
8.4	Backend Design . . . . .	58
8.4.1	Analyse konklusion . . . . .	58
8.4.2	Routes . . . . .	58
8.4.3	Authentication/Authorization med JWT Token . . . . .	59
8.4.4	Hashing . . . . .	59
8.4.5	BackEndController på client siden . . . . .	59
8.5	Applikationsmodeller . . . . .	60
8.5.1	Applikationsmodel User Controller . . . . .	60
8.5.2	Applikationsmodel Save Controller . . . . .	62
8.5.3	Konklusion . . . . .	66
8.6	Software Design DAL Design . . . . .	67
8.6.1	User story funktioner . . . . .	68
8.7	Database Design . . . . .	71
<b>9</b>	<b>Implementering</b>	<b>73</b>
9.1	System Implementering . . . . .	73
9.2	Frontend Implementering . . . . .	73
9.2.1	Login view . . . . .	74
9.2.2	Room View . . . . .	75
9.2.3	Combat View . . . . .	76
9.2.4	Settings view . . . . .	77
9.2.5	Load view . . . . .	78
9.2.6	Note om Baggrundsfarver . . . . .	79
9.3	Game Engine . . . . .	80
9.3.1	Gamecontroller Implementering . . . . .	81
9.3.2	Generering af Map . . . . .	83
9.3.3	Log . . . . .	83
9.4	Kommentar til implementeringen . . . . .	84
9.5	Backend Implementering . . . . .	84
9.5.1	Data Transfer Object: . . . . .	84
9.5.2	SaveController . . . . .	85
9.5.3	UserController . . . . .	86
9.5.4	BackEndController Client . . . . .	86
9.5.5	Konklusion . . . . .	87
9.6	Database Implementering . . . . .	88

<b>10 Test</b>	<b>90</b>
10.1 Modultest Frontend . . . . .	90
10.1.1 Test metoder . . . . .	90
10.1.2 Eksempel på frontend test i forbindelse med Game Engine . . . . .	91
10.1.3 Eksempel på frontend test . . . . .	91
10.2 Modultest Game Engine . . . . .	91
10.2.1 GodkendelsesTabel Game Engine . . . . .	92
10.2.2 Mock testing . . . . .	95
10.2.3 Test Resultater for Game Engine . . . . .	96
10.3 Modultest database . . . . .	97
10.3.1 Diskussion af database modultest . . . . .	100
10.4 Backend Modultest . . . . .	100
<b>11 Integrationstest</b>	<b>104</b>
11.1 Tidlig Integrationstest . . . . .	104
11.2 Fulde Integrationstest . . . . .	104
<b>12 Accepttestspecifikation</b>	<b>105</b>
12.1 Funktionelle . . . . .	105
12.2 ikke-funktionelle . . . . .	111
12.3 Diskussion af testresultater . . . . .	116
12.4 Funktionsbeskrivelse . . . . .	116
12.4.1 Game Controller . . . . .	116
12.4.2 Log . . . . .	117
12.4.3 DiceRoller . . . . .	117
12.4.4 Room . . . . .	118
12.4.5 Combat Controller . . . . .	118
12.4.6 Backend Controller . . . . .	118
12.4.7 Base Map Creator . . . . .	119
12.4.8 Enemy . . . . .	119
12.4.9 Player . . . . .	120
<b>13 Fremtidigt Arbejde</b>	<b>120</b>

## 1 Forord

Følgende rapport for 4. semesterprojekt, Dungeons and Gnoblins, består og er udarbejdet af softwarestuderende fra Aarhus Universitets institut for Elektro- og computerteknologi. Gruppens medlemmer er følgende:

- Luyen Nhu Vu - Rasmus Munk Engelund - Oscar Høffding Koudahl Dennis - Morten Høgsberg Futtrup Kristensen - Magnus Blaagjerg Møller - Anders Hundahl - Jacob Hoberg - Sune Dyrbye  
Vejlederen for projektet er Jung-Min Kim som har deltaget i ugentlige møder for at vejlede og rådgive gruppen gennem hele projektet.

Afleveringsdatoen for projektet er d. 3 juni 2022 og vil blive bedømt ved en eksamination d. 22 juni 2022 for at konkludere projektets resultat.

## 2 Ordforklaring

### Ordforklaring

Table 1

Forkortelse	Betydning
DAL	Data Access Layer
RNG	Random Number Generator
PC	Player Character
EF-Core	Entity-Framework Core
LINQ	Language Integrated Query
REST	Representational State Transfer
SoC	Separation of Concerns
MVP	Minimum Viable Product

## 3 Spilbeskrivelse

Spillet designes som et tekst-baseret spil, hvilket er en spilgenre, hvor brugeren interagerer med spillet igennem tekstbeskrivelser. Spillet består overordnet af fire dele med hver deres ansvar; en grafisk brugergrænseflade, en database, en back-end til netværkskommunikation, samt selve spil logikken. Brugergrænsefladen gør det muligt for brugeren at integrere med spillet. Den vil bestå af en række forskellige vinduer med hver sit formål (login screen, gameplay screen og settings screen), der hovedsageligt vil indeholde knapper og beskrivende tekst. Databasens ansvar er at gemme de enkelte spil. Databasen er en relationel database som sammenholder de enkelte brugeres profiler (Username og password) med informationer omkring deres fremskridt i spillet, såsom placering, items og stats. Det skal være muligt for en bruger at hente sine gemte spil ned på flere forskellige enheder. Til dette skal der bruges en netværksforbindelse til databasen.

### 3.1 Minimal Viable Product

Minimal Viable Product er et tidligt stadie på et produkt, hvor det indeholder nok features til at blive præsenteret til kunden. Produktet er i dette stadie langt fra at være det endelige produkt, men kan bruges til at forbinde projektets moduler sammen. I gruppens projekt blev der lavet et Minimal Viable Product for at etablere en forbindelse mellem projektets moduler. Dette skabte dermed et fundament for projektet og der kunne derefter tilføjes flere features undervejs i forløbet. Projektets Minimal Viable Product er spil-mæssigt bygget så der kan startes et spil og man som bruger efterfølgende kan bevæge sig rundt på mappet. Herved kan der etableres en forbindelse mellem modulerne, Front-end og Game-logic. Derefter skal spillet kunne loade og gemme et save. Ved gem af save, så skal spillets aktuelle forløb gemmes, så det gemmes hvor spilleren er på mappet. Ved load af save, så skal spillets aktuelle forløb hentes og loades for bruger, så det loades hvor spilleren er på mappet og derved fortsætte spillet herfra. Dette etablere dermed forbindelsen overfor de resterende moduler, Back-end og database.

### 3.2 Login-screen

Det første brugeren bliver mødt af er en login-skærm. Her skal brugeren enten indtaste sine login-oplysninger eller oprette sig en profil i systemet. Systemet har en database hvor alle profiler er lagret. Herunder ses en illustration af spillets login-screen.



Figure 1: Udkast til loginscreen hvor man både kan logge ind på en eksisterende bruger og oprette en ny

### 3.3 Core gameplay layout

PC kommer ind i et rum (se Figure 2). Brugeren bliver præsenteret med en beskrivelse af rummet, en liste af elementer i rummet, og en række muligheder for at interagere med rummet (interaktionen foregår ved at trykke på knapperne markeret “Button” i Figure 2). Når brugeren er færdig med at interagere med rummet, forlader de det ved at vælge en retning (“go north/south/east/west”). Dette fører dem ind i et nyt rum, mappet opdateres og loopet starter forfra. Bevægelsen i spillet kan gøres i de forklarede fire retninger. Disse retninger indikerer for spilleren hvilke rum de kan bevæge sig ind i, relativt til det rum de er i, der kunne f.eks. være rum, som kun har en vej ind og en vej ud, så kan man ikke gå andre veje end vejen man kom ind. North/south/east og west retningerne er baseret på et kompas, så derfor ville North resultere i at bevæge spillerens karakter opad, South nedad osv. Rum på mappet loades efterhånden som man besøger dem.

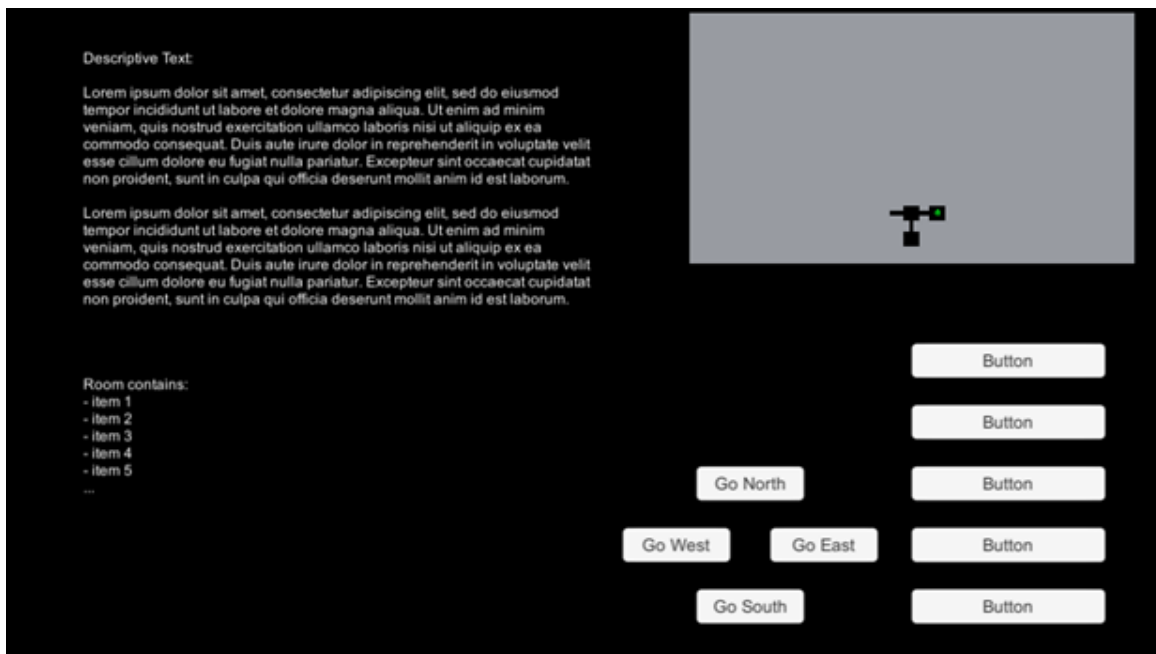


Figure 2: Udkast til core gameplay layout hvor der til venstre en en beskrivelse af historien efterfulgt af ting man kan interagere med. I højre side er der øverst en visuel præsentation af spillets map og hvor spilleren er derpå. Nederst til højre kan man bevæge sig i spillet og der er knappet til at lave forskellige interaktioner.

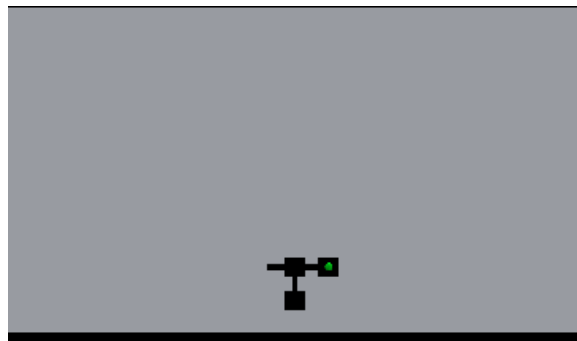


Figure 3: Closeup af et udkast til et start map, hvor man med prikken kan se spillerens lokation.

Et rum kan indeholde fjender, som skal bekæmpes før man kan tilgå resten af rummet. Kamp foregår ved at spillet “slår en terning” (RNG) for både PC og fjenden og lægger deres “combat stat” til slaget. Hvis det lykkes brugeren at slå højere end modstanderens “defense stats” gives skade til modstanderen ud fra et PCs “damage stat”, ligeledes sker det for modstanderen hvis brugeren ikke har besejret modstanderen på sin tur. Skaden trækkes fra PC/modstanderens liv. Når PCs liv bliver 0 dør karakteren. Hvis PC dør taber brugeren spillet. Hvis hverken PC eller fjenden er død efter en kamp får brugeren valget mellem at flygte (gå tilbage til det rum de kom fra) eller fortsætte kampen (start loopet forfra). Det er muligt at finde våben og udrustning i banen, som kan bruges til at forbedre PCs stats.





Figure 4: Udkast til combatscreen. Til venstre er der øverst en beskrivende tekst hvorunder der er en combatlog som beskriver fightens forløb. I højre side er der øverst en visuel præsentation af spillets map og hvor spilleren er derpå. Under mappet er der til venstre de to muligheder man har under en combat, fight og flee. Til højre for det, er der forskellige menu muligheder

### 3.4 Settings

Det skal være muligt for spilleren at tilgå en menu med spillets indstillinger. Her skal det være muligt for brugeren at tilpasse spillets layout indstillinger så som resolution og window size. Det skal ligeledes være muligt at justere lydstyrken for spillet. En illustration af hvordan denne menu kunne se ud er vist på Figure 5.

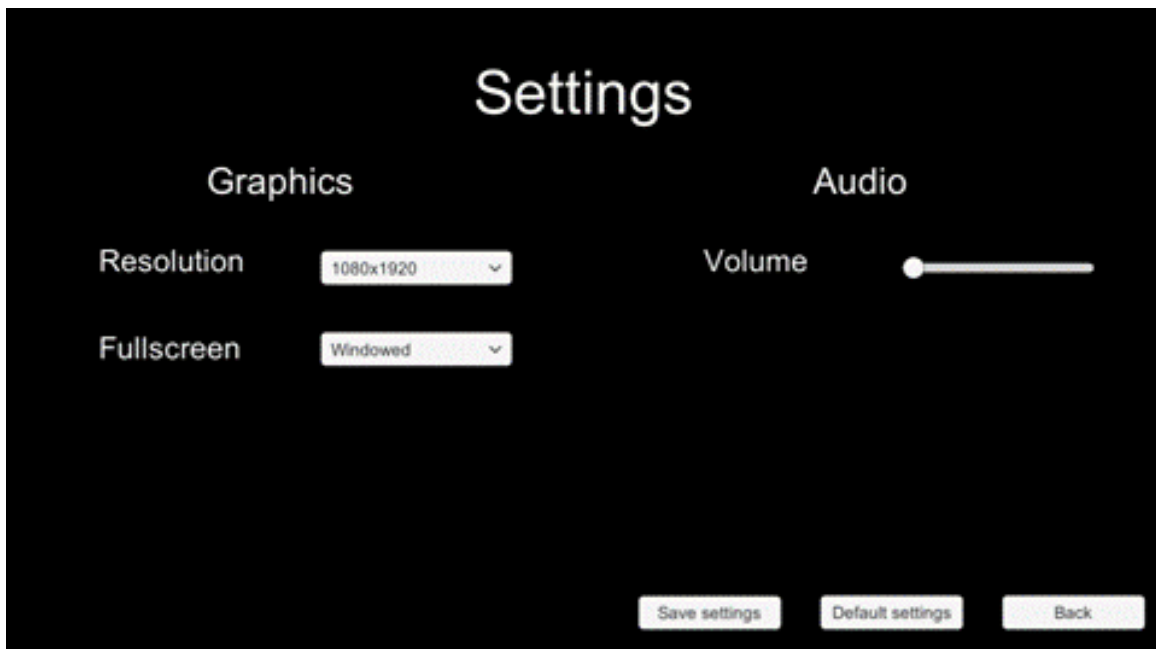


Figure 5: Udkast til settings menuen, hvor man til venstre kan sætte spillets resolution og til højre kan sætte spillets musik volume. Nederst er der 3 forskellige muligheder, gem settings, default settings og at gå tilbage

### 3.5 MapBeskrivelse

På Figure 6 herunder ses det endelige layout af spillets map som det er blevet implementeret. Spillets map består af 20 rum, som indeholder forskellige artefakter. Det første rum, rum 1, indeholder spillets spawnpoint, markeret med X, hvor spillet starter. De resterende 19 rum indeholder en række udfordringer, i form af fjender som skal overvindes, samt hjælp til dette i form af forskellige genstande. Der er i alt fem fjender på banen, fire svagere fjender markeret E, og en boss markeret B. Der findes to forskellige slags genstande som skal hjælpe spilleren på sin vej, våben(W) og skjold(S). Disse genstande henholdsvis forøger skaden man deler til fjender, og formindsker skade man tager. Bag bossen i rum 19, The Sword Of Destiny, markeret SOD, er placeret i rum 20 bag bossen. SOD er spillets endepunkt og når dette nåes, er spillet vundet.

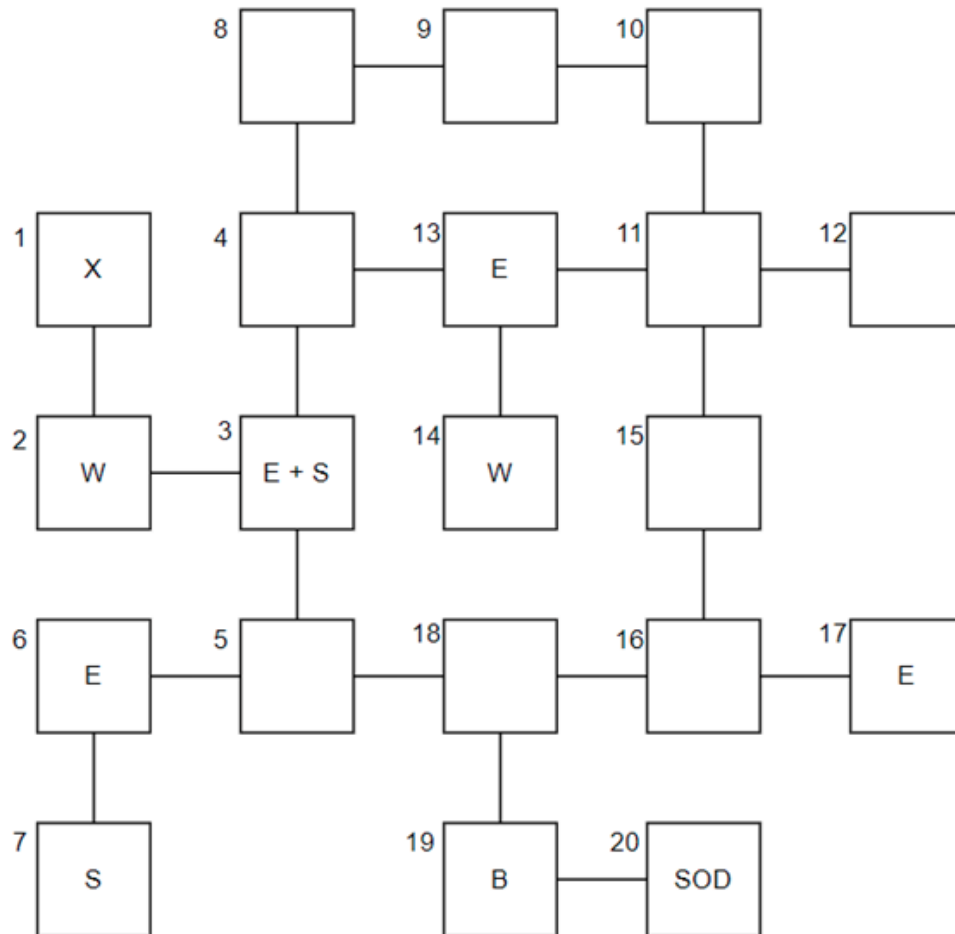


Figure 6: Final maplayout med 20 rum, som spawnpoint(x), Weapons(W), enemies(E), shields(S), 1 boss(B) og Sword of destiny(SOD) som man skal nå for at vinde

## 4 Systembeskrivelse

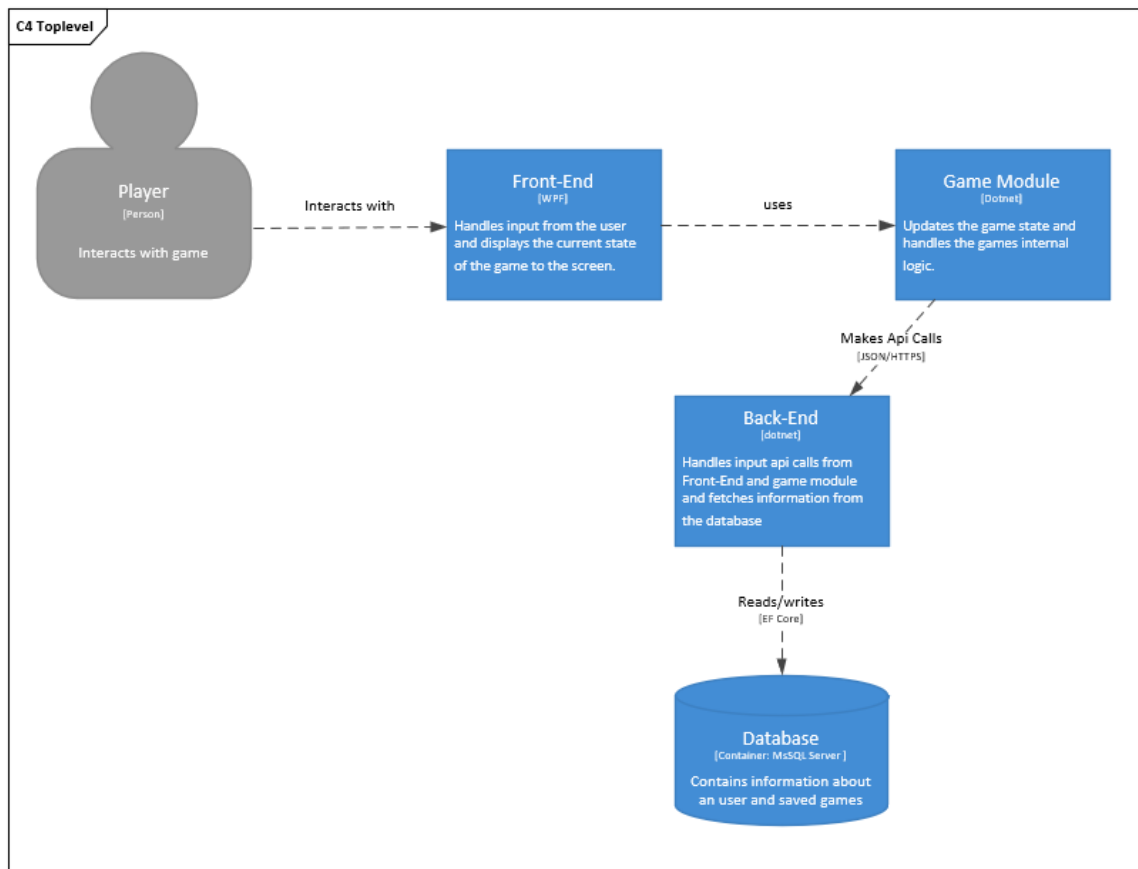


Figure 7: C4 top level diagram, som viser kommunikation mellem systemets segmenter

På figuren herover ses systemets toplevel arkitektur. Denne består af en bruger som interagerer med systemet gennem frontendapplikationen som skrives i WPF. States i denne applikation styres af Game modul som holder styr på hvor spilleren befinder sig, hvilke items der er samlet op og andre nyttige information som skal bruges gennem spillet. Spillets backend benyttes primært til bruger authentication og som bindeled til databasen. Databasen indeholder oplysninger om blandt andet brugere, de gemte spil og oplysninger om historien for de forskellige rum.

## 5 Kravspecifikation

I det følgende afsnit vil vi uddybe de forudbestemt krav til projektets specifikation. Dette indeholder en systembeskrivelse, samt en accepttestspecifikation til de opsatte krav. Kravene til systemet vil være opdelt i funktionelle og ikke funktionelle, hvor de opsatte ikke-funktionelle krav vil være prioriteret efter MoSCoW princippet. Der er ydermere opsat accepttest som tester alle scenarier i de opsatte user stories, samt systemets ikke-funktionelle krav.

## 5.1 Funktionelle krav - User Stories

I dette afsnit forklares systemets funktionelle krav i form af user stories som tager os igennem systemet og de ønskede funktioner.

User Story 1: Log in

Som Bruger

Kan jeg logge ind

For at kunne se en Main Menu så jeg kan spille spillet.

User Story 2: Opret profil

Som Bruger

Kan jeg oprette en ny profil

For at jeg kan logge ind på spillet.

User Story 3: Main Menu - Settings

Som bruger

Kan jeg tilgå Settings

For at jeg kan customize interfacet

User Story 4: Main Menu – Start Spil

Som bruger

Kan jeg starte et nyt spil

For at spille spillet

User Story 5: Main Menu – Exit game

Som Bruger

Kan jeg trykke "Exit Game"

For at lukke spillet.

User Story 6: Ingame Menu

Som Bruger

Kan jeg trykke Esc

For at se en menu med mulighederne "Return to game", "Save and Exit", "Exit without saving" and "Settings".

User Story 7: Ingame Menu - Resume Game

Som Bruger

Kan jeg trykke på "Resume game"

For at vende tilbage til spillet

User Story 8: Exit Menu - Save - No combat

Som Bruger

Kan jeg trykke på "Save Game"

For at bruger bliver vist en liste af saves

User Story 9: Exit Menu - Exit Without Saving - Combat

Som Bruger

Kan jeg under en combat trykke på "Escape"

For at se en ingame menu uden "Save Game"

User Story 10: Exit Menu - In game Menu - Main Menu

Som Bruger

Kan jeg trykke på Main Menu i in game menu

For at vende tilbage til Hovedmenuen uden at gemme

UserStory 11 : ingame Menu - Settings

Som Bruger

Kan jeg trykke på "Settings"

For at se Settings Menuen.

UserStory 12 : Settings Menu - Efter ændringer - Apply

Som Bruger

Kan jeg trykke "Apply"

For at ændre settings

UserStory 13 : Settings Menu - Efter ændringer - back

Som Bruger

Kan jeg trykke "back"

For at brugeren sendes tilbage til Esc Menu

UserStory 14 : Settings Menu - Efter ændringer - Default

Som bruger

Kan jeg trykke på "Default"

For at sætte settings til defaults

UserStory 15 : Save Menu - Save Game

Som bruger

Kan jeg trykke på et eksisterende spil, ændre navnet og trykke "Save Game"

For at gemme spillet.

UserStory 16 : Save Menu - Save Game - Back

Som bruger

Kan jeg trykke på "Back"

For at vende retur til in game menuen

UserStory 17 : Main Menu - Load Game

Som bruger

Kan jeg trykke på "load game"

For at få en liste af save games.

UserStory 18 : Main Menu - Load Game - Load

Som bruger

Kan jeg trykke på "load game"

For at komme ind i spillet og fortsætte med at spille det valgte spil

UserStory 19 : Main Menu - Load Game - Back

Som bruger

Kan jeg trykke "back"

For at vende tilbage til Main Menu

UserStory 20 : Spil Spillet - Bevægelse i spil

Som bruger

Kan jeg vælge mellem actioner ved at bruge tallene 0-9 eller piltasterne til at bevæge mig  
For at bevæge karakteren i spillet til et andet rum.

UserStory 21 : Spil spillet - Enter nyt room

Som bruger

Får jeg en beskrivelse af et rum, når jeg kommer ind i rummet  
Så jeg ved hvad jeg kan interagere med.

UserStory 22: Spil spillet - Combat

Som bruger

Kan jeg trykke på "attack" knappen  
For at skade fjenden

User Story 23: Spil spillet - Combat - flygt

Som bruger

Kan jeg trykke på "flee" knappen  
For at blive rykket tilbage til forrige rum

UserStory 24: Spil spillet - Inventory

Som bruger

Kan jeg trykke på "Inventory" knappen  
For at se genstande som jeg besidder

UserStory 25: Spil spillet - Interact

Som bruger

Kan jeg trykke på "interact" knappen  
For at interagere med objekter i det nuværende rum

UserStory 26 : Spil Spillet - Level klaret

Som bruger

Kan jeg færdiggøre det sidste rum  
For at få en besked om at spillet er fuldført og mulighed for at tilgå Main Menu

## 5.2 Ikke-funktionelle krav

I dette afsnit opsættes systemets ikke-funktionelle krav. Disse er opdelt efter kategori, hvorved der er krav til FURPS modellens forskellige dele.

Ikke-funktionelle:

GUI:

- GUI'en skal tilbyde valget mellem 3 resolutions
- GUI'en skal kunne være fullscreen
- GUI'en skal kunne være windowed

SOUND:

- Lyden skal kunne justeres mellem 0-100% relativt til PC'ens lyd niveau.

DATABASE:

- Skal kunne gemme maksimalt 5 save games
- Skal kunne loade et spil indenfor maksimalt 5s

- Skal gemme hvilke genstande man bruger lige nu
- Skal gemme hvor meget liv man har tilbage.
- Skal gemme hvilke fjender man har slået ihjel.
- Skal gemme hvilke puzzles man har løst
- Skal gemme hvilke rum man har været i.

#### GAMEPLAY:

- Spilleets kort skal holde styr på hvilke rum man kan komme til for et givet rum.
- Spilleets kort skal kun vise de rum som spilleren har været i.
- Spilleets kort skal, hvis spilleren har været i alle rum vise alle rum.
- Et rum kan have maksimalt 4 forbindelser til andre rum.
- Et rum skal have mindst 1 forbindelse til andre rum.
- Alle Rum skal kunne nås fra ethvert andet rum, måske ikke direkte, men man skal kunne komme dertil.
- Spillerens rygsæk skal kunne indeholde alle spillets genstande.
- Spilleren skal have mulighed for at bruge ét våben og én rustning af gangen.
- Spilleren skal have mulighed for at skifte hvilket våben og hvilken rustning der bruges.

#### COMBAT:

- Når spilleren/fjenden prøver at slå, rammer man kun hvis man på sit angreb slår højere end modstanderens rustningsværdi. Dette afgøres af et simuleret 20-sidet terninge kast, hvortil der lægges en værdi til, korresponderende til spilleren/fjendens våben bonusser.
- Hvis spilleren/fjenden rammer, bliver skaden bestemt af et/flere simulerede terningekast, afhængigt af hvilket våben der bruges
- Hvis spilleren, når nul liv inden fjenden, så dør spilleren og spillet er tabt.
- Hvis fjenden, når nul liv inden spilleren, så dør fjenden og spilleren kan nu frit udforske rummet, som fjenden var i.
- Hvis spilleren drikker en livseleksir bliver spillerens nuværende liv sat til fuldt.
- Hvis spilleren/fjenden rammer, bliver skaden bestemt af et/flere simulerede terningekast, afhængigt af hvilket våben der bruges.

#### PERFORMANCE:

- Spillet skal respondere indenfor maksimalt 5s
- Spillet må ikke have mere end én kommando i aktionskøen af gangen

#### STABILITY:

- MEANTIME BETWEEN FAILURE - 1Time +- 10Min

#### DOCUMENTATION:

- Spillet skal have en manual/help page til hvordan alting virker.

#### SECURITY:

- Username skal være mindst 6 characters
- Username skal være unikt
- Password skal være mindst 8 characters
- Password skal have store og små bogstaver
- Password må ikke indeholde username



### 5.3 MOSCOW krav

I dette afsnit er de ikke-funktionelle krav fra ovenstående afsnit prioriteret ved hjælp af MoSCow metoden.

#### MUST

- Skal have en GUI
- Skal have en Database
- Skal have Netværkskommunikation
- Skal have en serie af sammenhængende rum.
- Skal have et start og slut rum som ikke er det samme rum.
- Skal kunne gemmes på en Database.
- Skal kunne hente save game fra databasen
- Skal have en authentication system (Accounts)
- Hvert Rum skal bestå af et beskrivende element og en serie af actioner.
- Skal have instillinger.
- Skal have en Character.
- Skal kunne tage imod user input.
- Skal være udviklet til Windows.
- Skal være testbart i.e. skal være designet med testing in mind.

#### SHOULD

- Burde have Enemies
- Burde have Items
- Burde have Combat mechanics
- Burde have en End Screen
- Burde have Character Stats

#### COULD

- Kunne have Level development
- Kunne have Procedural world
- Kunne have Text Parser
- Kunne have Difficulty
- Kunne have Security

#### WON'T

- Vil ikke Have Graphics

### 5.4 Accepttest

I dette afsnit opsættes systemets Accepttest. Disse er opdelt efter kategori, først er der accepttest for User stories, og herefter de ikke funktionelle krav.

#### 5.4.1 Funktionelle

##### Test af User Story 1 - Login 1. Scenarie - Succesfuld login

*Givet at brugerens profil er oprettet på databasen og at serveren er oppe.*

- Når bruger indtaster sit login(Brugernavn og password)
- og trykker "Log in" på UI
- Så skifter skærmen til hovedmenu

Resultat:

Kommentar:

### 1. Scenarie - Fejlet login

*Givet at brugerens profil er oprettet på databasen og at serveren er oppe.*

- Når bruger indtaster et forkert login(Brugernavn og password)
- og trykker "Log in" på UI
- Så signaleres der om forkert login-oplysninger til bruger

Resultat:

Kommentar:

### Test af User Story 2 - Opret profil 2. Scenarie - Bruger opretter profil

*Givet at brugerens profil er oprettet på databasen og at serveren er oppe.*

- Når bruger vælger at oprette profil
- Så gemmes datanene for brugerens profil på databasen
- Og skærmen skiftes til log ind skærmen

Resultat:

Kommentar:

### 2. Scenarie - Bruger opretter samme profil

*Givet at brugerens profil er oprettet på databasen og at serveren er oppe.*

- Når bruger vælger at oprette en allerede-eksisterende profil
- Så signaleres der om at profil allerede eksisterer

Resultat:

Kommentar:

### Test af User Story 3-5 - Main Menu 3. Scenarie - Tilgå settings

*Givet at brugeren er logget ind og har adgang til spillet.*

- Når bruger trykker settings i UI
- Så går spillet til settings menuen

Resultat:

Kommentar:

### 4. Scenarie - Start spillet

*Givet at brugeren er logget ind og har adgang til spillet.*

- Når bruger trykker "New Game"
- Så vises game-interface for brugeren

Resultat:

Kommentar:

### 5. Scenarie - Exit game

*Givet at brugeren er logget ind og har adgang til spillet.*

- Når bruger trykker "Exit game"
- Så lukker spillet

Resultat:

Kommentar:

### Test af User Story 6-14 - Exit Menu 6. Scenarie - Ingame menu spil

*Givet at brugeren har trykket "New Game"*

- Når bruger trykker "Escape" på keyboardet
- Så popper et menuvindue op med mulighederne "Resume Game", "Save Game", "Main Menu" og "Settings"

Resultat:

Kommentar:

### 7. Scenarie - In game menu - Resume game

*Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard*

- Når bruger trykker "Resume Game" i In game menu
- Så forsvinder menuvinduet og spillet fortsætter

Resultat:

Kommentar:

### 8. Scenarie - In game menu - Save – No Combat

*Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard*

- Når bruger trykker på "Save Game" i In game menuen
- Så vises save menuen
- Og en liste af gemte spil

Resultat:

Kommentar:

### 9. Scenarie - In game menu - Save – Combat

*Givet at brugeren er i combat*

- Når bruger trykker "Escape" på keyboardet
- Så kan man ikke se en "Save Game" knap i game menuen

Resultat:

Kommentar:

#### **10. Scenarie - In game menu - Main Menu**

*Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard*

- Når bruger trykker "Main Menu" i In game menu
- Så vises hovedmenuen

Resultat:

Kommentar:

#### **11. Scenarie - In game menu - Settings**

*Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard*

- Når bruger trykker "Settings"
- Så åbnes et vindue med mulighed for konfiguration af spil for bruger

Resultat:

Kommentar:

#### **12. Scenarie - Exit menu - Settings - Apply**

*Givet at brugeren har trykket "Settings" og ændret på resolution indstilling*

- Når bruger trykker "Apply"
- Så ændres indstillinger som brugeren har ændret

Resultat:

Kommentar:

#### **13. Scenarie - Exit menu - Settings - Back**

*Givet at brugeren har trykket "Settings"*

- Når bruger trykker "Back"
- Så vises In game menuen

Resultat:

Kommentar:

#### **14. Scenarie - Exit menu - Settings - Default**

*Givet at brugeren har trykket "Settings" og har ændret mindst 1 indstilling*

- Når bruger trykker "Default"
- Så ændres alle indstillinger tilbage til default settings

Resultat:

Kommentar:

**Test af User Story 15 og 16 - Save Menu 15. Scenarie - Save Menu - Save Game**

*Givet at brugeren har trykket "Save game" fra Settings menu og ikke er i combat.*

- Når bruger vælger et gemt spil
- Og sætter navnet til det ønskede
- Og trykker på "Save Game" knappen
- Så genoptages spillet

Resultat:

Kommentar:

**16. Scenarie - Save Menu - Back**

*Givet at brugeren har trykket "Save game" fra Settings menu*

- Når bruger trykker "Back" i save game menuen
- Så vender spillet tilbage til In game menuen

Resultat:

Kommentar:

**Test af User Story 17-19 - Main Menu 17. Scenarie - Main menu - Load Game**

*Givet at brugeren er logget ind og har adgang til spillet.*

- Når bruger trykker "Load game"
- Så vises en liste af gemte spil på profilen

Resultat:

Kommentar:

**18. Scenarie - Main menu - Load Game - Load**

*Givet at brugeren er logget ind og har adgang til spillet og trykket "Load Game"*

- Når bruger vælger et gemt spil
- Og trykker "Load Game"
- Så loader spillet det gemte spil med det valgte game state

Resultat:

Kommentar:

**19. Scenarie - Main menu - Load Game - Back**

*Givet at brugeren er logget ind og har adgang til spillet og trykket "Load Game"*

- Når bruger trykker "Back" i load game-menuen
- Så vender spillet tilbage til hovedmenuen

Resultat:

Kommentar:

## **Test af User Story 20-26 - Spil spillet 20. Scenarie - Spil spillet - Start spillet**

*Givet at brugeren har trykket "New Game"*

- Når bruger anvender piltasterne på keyboard eller trykker på knappen på interfacet til at bevæge sig sydpå fra startrummet
- Så bevæger brugeren sig ind i rummet "Syd" for det rum de står i

Resultat:

Kommentar:

## **21. Scenarie - Spil spillet - Enter nyt room**

*Givet at brugeren har trykket "New Game"*

- Når bruger går ind i et nyt rum ved brug af keyboard
- Så giver UI en beskrivelse af rummet spilleren befinder sig i

Resultat:

Kommentar:

## **22. Scenarie - Spil spillet - Combat**

*Givet at brugeren møder en fjende i et rum*

- Når bruger trykker "Fight!"
- Så ruller brugeren et tal mod fjenden om at skade fjenden
- Og derefter ruller fjenden et tal om at skade brugeren

Resultat:

Kommentar:

## **23. Scenarie - Spil spillet - Combat - Flygt**

*Givet at brugeren møder en fjende i et rum*

- Når bruger trykker "Flee"
- Så flyttes brugeren tilbage til det rum han kom fra
- Og får ikke sit mistede liv tilbage igen

Resultat:

Kommentar:

## **24. Scenarie - Spil spillet - Inventory**

*Givet at brugeren har trykket "New Game"*

- Når bruger trykker "Inventory"
- Så vises genstande som bruger besidder

Resultat:

Kommentar:

## **25. Scenarie - Spil spillet - Interact**

*Givet at brugeren har trykket "Start spil" og at der ike er fjender i rummet*

- Når bruger trykker "Interact"
- Så kan bruger tage potentielle genstande i rummet til brugerens "Inventory"

Resultat:

Kommentar:

## 26. Scenarie - Spil spillet - Level klaret

*Givet at brugeren har fuldført det næstsidste rum*

- Når bruger bevæger sig ind i sidste rum
- Så får bruger en besked om at spillet er klaret og får mulighed for at gå til "Main Menu"

Resultat:

Kommentar:

### 5.4.2 Ikke funktionelle

#### GUI

Table 2: Ikke funktionelle tests for GUI

Beskrivelse	Verificering	Resultat	Kommentar
GUI'en skal tilbyde valget mellem 3 resolutions	Visuel		
GUI'en skal kunne være fullscreen	Visuel		
GUI'en skal kunne være windowed	Visuel		

#### SOUND

Table 3: Ikke funktionelle tests for SOUND

Beskrivelse	Verificering	Resultat	Kommentar
Lyden skal kunne justeres mellem 0-100% relativt til PC'ens lyd niveau.	Auditorisk/Visuel		

#### DATABASE

Table 4: Ikke funktionelle tests for DATABASE

Beskrivelse	Verificering	Resultat	Kommentar
Skal kunne gemme maksimalt 5 save games	Visuel		
Skal kunne loade et spil indenfor maksimalt 5s	Visuel		
Skal gemme hvilke genstande man bruger lige nu	Visuel		
Skal gemme hvor meget liv man har tilbage.	Visuel		
Skal gemme hvilke fjender man har slået ihjel.	Visuel		
Skal gemme hvilke puzzles man har løst	Visuel		
Skal gemme hvilke rum man har været i.	Visuel		

**GAMEPLAY**



Table 5: Ikke funktionelle tests for GAMEPLAY

Beskrivelse	Verificering	Resultat	Kommentar
Spillets kort skal holde styr på hvilke rum man kan komme til for et givet rum.	Visuel		
Spillets kort skal kun vise de rum som spilleren har været i.	Visuel		
Spillets kort skal, hvis spilleren har været i alle rum vise alle rum.	Visuel		
Et rum kan have maksimalt 4 forbindelser til andre rum.	Visuel		
Et rum skal have mindst 1 forbindelse til andre rum.	Visuel		
Alle Rum skal kunne nås fra ethvert andet rum, måske ikke direkte, men man skal kunne komme dertil.	Visuel		
Spillerens rygsæk skal kunne indeholde alle spillets genstande.	Visuel		
Spilleren skal have mulighed for at bruge ét våben og et skjold af gangen.	Visuel		
Spilleren skal have mulighed for at skifte hvilket våben og hvilket skjold der bruges.	Visuel		

## COMBAT

Table 6: Ikke funktionelle tests for COMBAT

Beskrivelse	Verificering	Resultat	Kommentar
Når spilleren/fjenden prøver at slå, rammer man kun hvis man på sit angreb slår højere end modstanderens rustningsværdi. Dette afgøres af et simuleret 20 sided terninge kast, hvortil der lægges en værdi til, korresponderende til spilleren/fjendens våben bonusser.	Visuel		
Hvis spilleren/fjenden rammer, bliver skaden bestemt af et/flere simulerede terningekast, afhængigt af hvilket våben der bruges.	Visuel		
Hvis spilleren/fjenden rammer, bliver skaden bestemt af et flere simulerede terningekast, afhængigt af hvilket våben der bruges.	Visuel		
Hvis spilleren, når nul liv inden fjenden, så dør spilleren og spillet er tabt.	Visuel		
Hvis fjenden, når nul liv inden spilleren, så dør fjenden og spilleren kan nu frit udforske rummet, som fjenden var i.	Visuel		
Hvis spilleren drikker en livseleksir bliver spillerens nuværende liv sat til fuldt	Visuel		

**PERFORMANCE**

Table 7: Ikke funktionelle tests for PERFORMANCE

Beskrivelse	Verificering	Resultat	Kommentar
Spillet skal respondere indenfor maksimalt 5s	Visuel		
Spillet må ikke have mere end én kommando i aktionskøen af gangen	Visuel		

**STABILITY**

Table 8: Ikke funktionelle tests for STABILITY

Beskrivelse	Verificering	Resultat	Kommentar
MEANTIME BETWEEN FAILURE 1Time + 10Min?	Visuel		

**DOCUMENTATION**

Table 9: Ikke funktionelle tests for DOKUMENTATION

Beskrivelse	Verificering	Resultat	Kommentar
Spillet skal have en manual/help page til hvordan alting virker.	Visuel		

**SECURITY**

Table 10: Ikke funktionelle tests for SECURITY

Beskrivelse	Verificering	Resultat	Kommentar
Username skal være mindst 6 characters	Visuel		
Username skal være unikt	Visuel		
Password skal være mindst 8 characters	Visuel		
Password skal have store og små bogstavers	Visuel		
Password må ikke indeholde username	Visuel		

## 6 Analyser

### 6.1 Teknologiundersøgelse: Unity - .Net

I projektgruppen har vi identificeret at der er to oplagte frameworks, til at lave projektet i. Det første framework er Microsofts "WPF" til frontend og "ASP.NET" til systemets backend. Det andet framework er Unity Technologies' framework "Unity" til kombineret frontend og backend. Ens for de to framework er at de begge understøtter blandt andet: - Understøtter C# - Versionskontrol i form af Git - Integrerede/let integrerbare Testframeworks

#### 6.1.1 Unity:

Unity lader dig bygge fulde spil fra bunden, dette giver rig mulighed for udvidelse og ændringer af spil designet, dette kombineret med at Unity er bygget til cross-platform kompilering, hvilket gør at ens spil automatisk bliver portabelt og ens målgruppe bliver udvidet og derved bliver markedsføring af produktet mere bred. Dette har naturligvis en pris, i form af at en Unity licens til firmaer med en årlig omsætning på over 100.000\$, koster minimum 1.800\$. Derudover er Unity umiddelbart intuitivt og nemt at bruge, med det menes der at Unity's frontend er informativ, dog er hele frameworket nyt og skal derfor læres fra bunden. Dette resulterer i at der skal undersøges mange basale ting for at der kan laves et spil, og tiden det tager at lave første iteration kan hurtigt eksplodere. Unity er også et program i aktiv udvikling og det betyder for det første at features hele tiden bliver udviklet og udvidet. Dette resulterer også i at dokumentationen omhandlende Unity hele tiden skifter og derfor kan det være svært at søge hjælp på nettet, når man sidder fast i udviklingen. Unity har udover sit udviklingsværktøj, en Asset store, som giver mulighed for at købe allerede programmerede Assets og trække dem direkte ind i ens program. Dette giver selvfølgelig mulighed for at spare en masse arbejde, men da projektarbejde netop handler om at lave tingene selv, giver dette ikke meget mening at bruge i dette tilfælde. Et negativ ved at bruge et så gennemført og feature rigt udviklingsværktøj, er at kompilering, loading og test tider bliver markant højere end ved brug af lettere udviklingsværktøjer. Til sidst skal der nævnes at udvikling i Unity ikke bliver opdelt i frontend/backend, sådan som det bliver lært at udvikling skal opdeles på 4. semester, Tværtimod sidder frontend/backend meget tæt sammen, når man udvikler med Unity.[Unity.com] [Choose-Unity] [Unity-Pros-Cons]

#### 6.1.2 .NET framework:

Til forskel fra Unity, er .NET Frameworket noget mere letvægt. Spillet kan stadig laves fra bunden og stadig med rig mulighed for ændringer og udvidelse af spil designet. Hvad man ikke får, er Unity's

indbyggede Cross-platform kompilering og derved kan denne del af ens udvidelses fase godt blive mere kompliceret, hvis man vil kompilere til flere styresystemer. At man misser den indbyggede Cross-platform kompilering, og Unity's Asset store er også med til, som sagt, at gøre .Net frameworket mere letvægt, det vil sige at man kan se frem til mindre overhead og derved potentielt hurtigere load og compile tider. Kigger man på de mere økonomiske forskelle, ser man hurtigt at de to frameworks er som to modsætninger, nemlig fordi at .NET frameworket kan fås gratis som extension til Visual Studio, som der også findes gratis versioner af. Ser man på .NET frameworket gennem mere praktiske øjne for os som studerende, dukker der umiddelbart tre argumenter frem. 1. Vi ved fra undervisningen, hvor vi har arbejdet med netop .NET frameworks, at der er massere af kendt viden at hente, altså dokumentation at hente på nettet og fra vores undervisere, om hvad og hvordan vi bedst kan gribe lige netop dette framework an. 2. Frontend og Backend er i .NET frameworket separeret, hvilket passer rigtig godt med vores projektarbejde i projektgruppen og med hvordan vi har lært at man skal udvikle software systemer i undervisningen, nemlig lige præcis med den separerede frontend og back end. 3. Vi har også tidligere arbejdet med hvordan CI (Continuous Integration) kan sættes op, netop sammen med .NET frameworket. En opsummering af de sidste 3 punkter giver altså at vi i .NET framework, på trods af at det er mere letvægt end unity, kan fokusere mere intenst på at lave noget godt projektarbejde og kode, og ikke fokusere så meget på de basale ting, som vi måske skulle hvis vi arbejdede med Unity.[[Microsoft.com](https://www.microsoft.com)]

## 6.2 Teknologiundersøgelse: Database SQL/NOSQL

Til at opbevare vores data til spillet og dennes loginsystem, skal vi benytte en database. Hertil skal der undersøges muligheder for at finde en database som gør det muligt for at bruge meget præcise tabeller med forhold til hinanden for at kunne genskabe et spil der er blevet gemt. Så derfor undersøges der fordele og ulemper ved SQL og NoSQL, og derefter ville der kunne vælges en af deres respektive applikationer, såsom MySQL (SQL) eller MongoDB (NoSQL).

### 6.2.1 SQL

En mulighed for vores database er at anvende en SQL baseret database. Denne form for database er relational, da den har en samling af data med forudbestemte forhold derimellem. Genstande organiseres i tabeller som indeholder information om objektet. Yderligere er SQL kendt og vel dokumenteret, som resulterer i at denne type database har et stort fællesskab bag den, med mulighed for støtte til arbejdet med den. Hernæst fungerer SQL også med ACID-principperne, som er;

- Atomicity: Hver transaktion ses som en samlet unit, derfor vil transaktion enten være fuldført komplet, eller fejle, hvis en eller flere operationer fejler.
- Consistency: Kun gyldige data kan skrives i databasen. Hvis inputtet data er ugyldigt så ville databasen returnere til, hvordan den var før transaktionen, og pga. dette kan fejl transaktioner ikke korruptere databasen.
- Isolation: Ufærdige transaktioner forbliver isoleret. Dette medfører at alle transaktioner bliver behandlet individuelt.
- Durability: Data bliver gemt af systemet selvom en transaktion fejler, og pga. dette vil data ikke blive mistet ved at f.eks. systemet crasher under transaktionen.

Derudover er SQL også portabel og kan anvendes i programmer på PC'er uanset servers, OS og embedded systemer. Ydermere har SQL også hurtige query processering, som betyder at store mængder af data, og operationer som indsættelse, at slette og datamanipulation, sker hurtigt og effektivt.

SQL har selvfølgelig også nogle ulemper såsom, at den har rigide og fastlåste skemaer. Disse resulterer

i at databasen ikke er fleksibel, og har ikke mulighed for ændringer eftersom at kravene ændres. For at akkommodere tilpasninger, så skal koden for databasen skrives om.

### 6.2.2 No-SQL

Et andet alternativ ville være en database baseret på No-SQL. Denne har, modsat SQL, fleksible datamodeller, som gør det nemmere at lave ændringer i databasen efterhånden som kravene til databasen ændres. Yderligere har No-SQL også horizontal scaling, som betyder at hvis der rammes den nuværende servers kapacitet, så har man muligheden for at tilføje mindre ekstra servers, hvor at man ville skulle migrere til en større server i SQL. Dette gør det muligt for hurtigt at justere kapaciteten. Mere er NoSQL data ofte opsat efter queries. Sådan at data der skal hentes sammen, er ofte opbevaret sammen. Dette resulterer i hurtigere queries.

### 6.2.3 Konklusion

I dette foranstående segment blev der diskuteret fordele og ulemper ved anvendelse af SQL kontra No-SQL. Hertil bliver der opstillet et valg om hvilket af disse to, der ville være bedst egnet til dette system. Valget vil blive taget senere i design fasen for databasen til projektet.

[Yalantis] [IBM] [MongoDB.com]

## 6.3 Teknisk Analyse Backend

Resultatet af den tekniske analyse af Front-end delen af projekt, blev at anvende .NET Core frameworket WPF til spillets GUI. Med dette på plads skal det besluttet, hvordan spillets back-end skal bygges op. Back-end'ens ansvar er at flytte og validere data mellem spillet GUI og spillets tilhørende Database. Beslutningen af Back-end er blevet taget på baggrund af beslutningerne for Front-end og Databasen, og da det er blevet besluttet at anvende en Microsoft SQL database, og at GUI'en udvikles i WPF, giver det mening, at back-end'en ligeledes udvikles i et .NET miljø. Derfor ser vi på følgende to muligheder.

### 6.3.1 Mulighed 1: WPF ->ASP.NET Core ->Entity Framework Core ->Database

Denne mulighed indebærer .NET frameworket ASP.NET Core. Her vil dataflowet altså blive følgende. Data'en skal flyttes først fra WPF til ASP.NET og derfra med EF Core[**Entity-Framework-Core**] til databasen, og så tilbage igen. Det skal altså være muligt for WPF App'en at kontakte et Web Api og sende/modtage dataobjekter, dette kan gøres med HTTP request/response, hvilket vil kræve at data objekterne seraliseres og deseraliseres til/fra JSON, afhængigt af om der modtages eller sendes. For at dataen kan blive behandlet på back-end siden vil det kræve at JSON dataen så konverteres tilbage til objekter. Når dataen er færdigbehandlet kan web api'et kontakte databasen og sendes/-modtage den nødvendige data. Med ASP.NET fåes muligheden for at lave et Web API server med en pipeline til at håndtere de forskellige forespørgelser, hvilket vil resultere i at systemets ansvarsområder bliver bedre fordelt. Hertil giver ASP.Net mulighed for authentication af brugere, samt Authorization af indkommende kald fra klienten, hvilket vil gøre systemet mere beskyttet.

### 6.3.2 Mulighed 2: WPF ->Entity Framework Core ->database

Dette er en mere simpel løsning, da det ikke kræver frameworket ASP.NET Core. Dataflowet vil her være direkte fra WPF til databasen ved hjælp af EF Core. På den måde opnås den samme funktionalitet, men med et framework mindre. Det vil også være hurtigere at implementere og give et mindre risiko fyldt design, da der fjernes interface mellem Front-end applikationen og databasen.

Med EF core fåes et mere Lightweight system med mindre overhead, som går fint i tråd med vores minimale valg af WPF fremfor Unity. Tilgængæld undværes et Web api med pipelines, hvilket resulterer i at back-end'ens opgaver, vil blive tættere koblet op ad WPF applikationen.

### **6.3.3 Konklusion**

De foranstående fordele og ulemper for de præsenterede muligheder giver mulighed for anvendelse af forskellige metoder. Valget vil blive taget i design fasen for dette system.

## 7 Arkitektur

### 7.1 Systemarkitektur

Dette afsnit forklarer hvordan systemets toplevel arkitektur er opbygget. Denne består af en bruger som interagerer med systemet gennem frontendapplikationen som skrives i WPF. States i denne applikation styres af Game modul som holder styr på hvor spilleren befinder sig, hvilke items der er samlet op og andre nyttige information som skal bruges gennem spillet. Spilleets backend benyttes primært til bruger authentication og som bindeled til databasen. Databasen indeholder oplysninger om blandt andet brugere, de gemte spil og oplysninger om historien for de forskellige rum. For at se en visuel repræsentation af hvordan de forskellige moduler snakker sammen se Figure 24

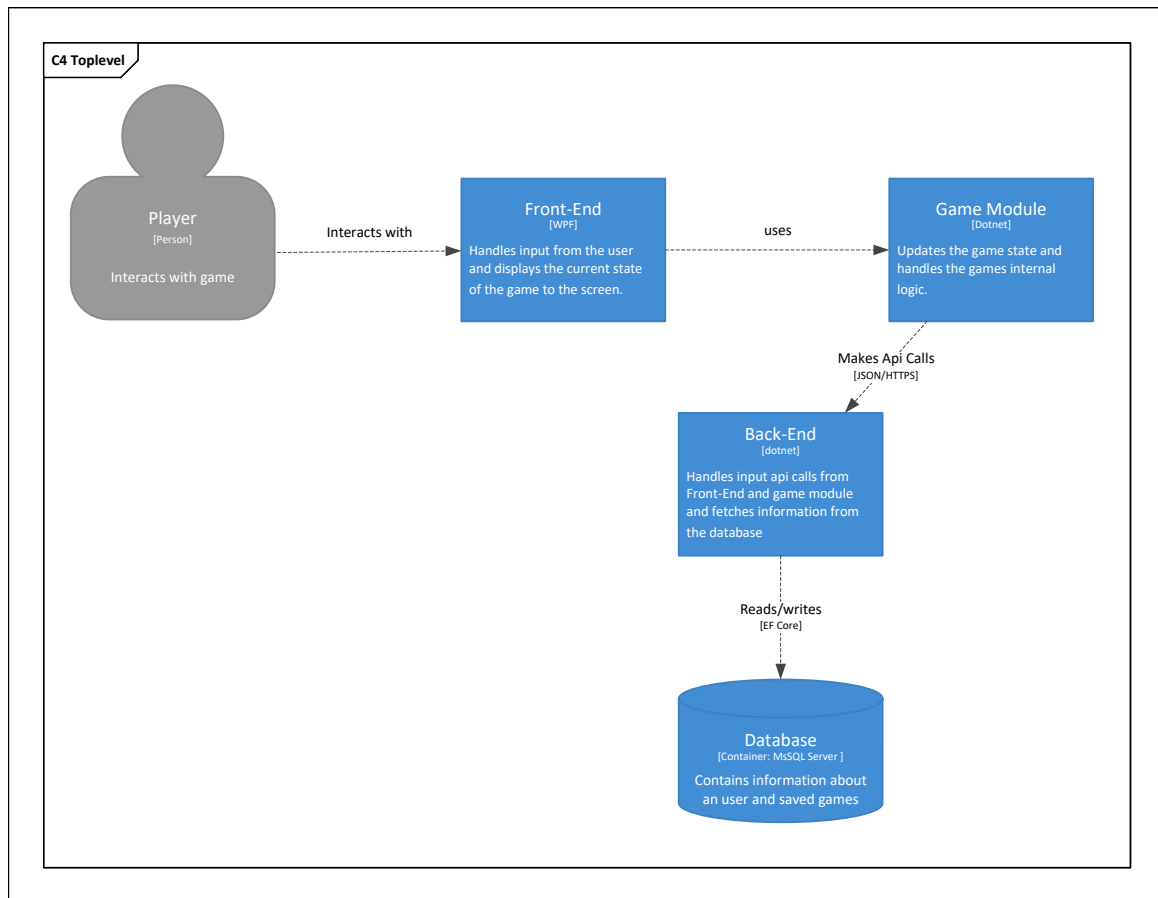


Figure 8: C4 Top-Level diagram for systemets arkitektur. Her ses et diagram for systemets Top-Level arkitektur, hvori der er skabt et overblik over hvilke moduler der er til stede i systemet og hvordan de kommunikerer. Heri er der også tilføjet kommunikationsmetode for de forskellige forbindelser.

### 7.2 Frontend Arkitektur

Frontend applikationen vil have til formål at håndtere brugerinput og output. Dvs. at der i Front-end'en vises det data fra gamelogic, som brugeren skal have, og at det præsenteres på en overskuelig og brugervenlig måde. Dette resulterer i at brugeren kan forstå og kan finde ud af at bruge spillet på den tiltænkte måde. Derudover skal Front-end'en tage hånd om bruger input, og sørge for at brugeren giver korrekt input og at der tages hånd om eventuelt forkert input.



Da der er mange forskellige menuer og skærme i spil i Front-end'en er der lavet følgende C3 model for Front-end'en (Figure 9), som giver en idé om hvilke skærme og menuer der kan gå til hvilke andre menuer/skærme. Udover dette fortæller modellen også om hvordan og hvilke skærme og menuer der snakker med noget uden for Front-end'en selv f.eks. skal der ved Login/Register kontaktes databasen for at få verificeret logind-oplysninger, og samtidig skal der ved save og load-game hentes en liste af gemte spil i databasen hvorefter der skal henholdsvis skrives og hentes fra databasen alt efter om man gemmer eller henter et spil. Der skal hertil nævnes at Login og Register står til at snakke med backenden direkte og ikke igennem gamelogic blokken, dette er valgt da funktionen ikke kaldes i gamelogic blokken, men den kaldes direkte i backend controlleren. Derudover er modellen mere overskuelig på denne måde og fungerer bedre til at give overblik over navigationen igennem menuerne.

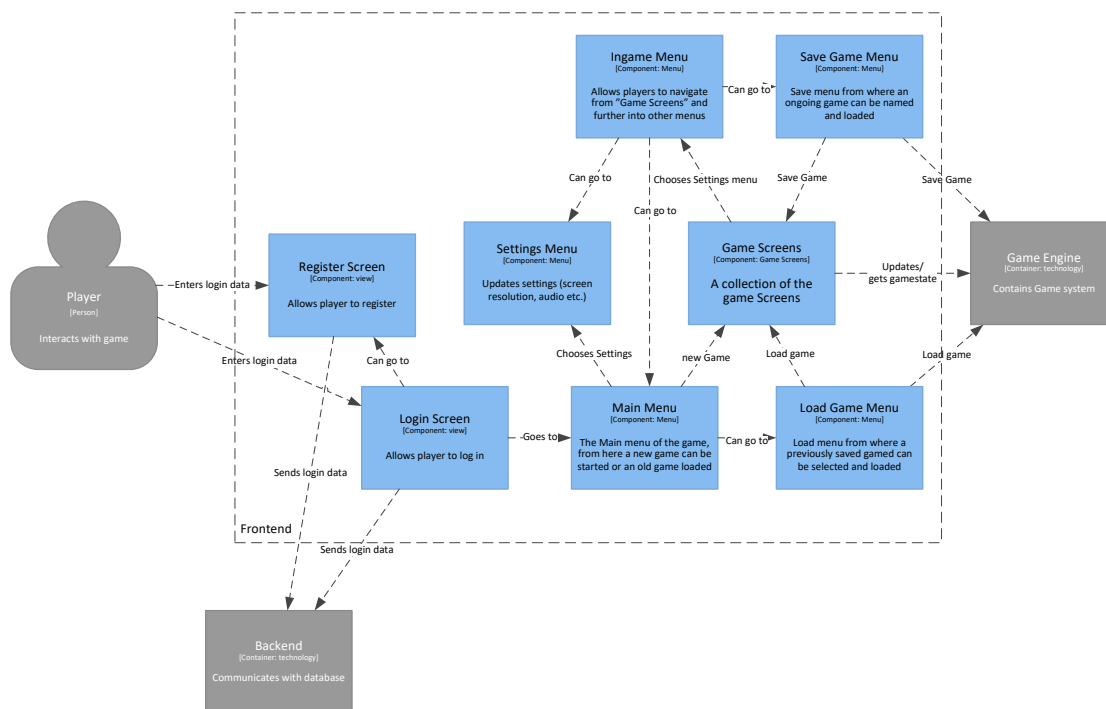


Figure 9: C3-Model for Frontend. Modellen fortæller hvordan man kan navigere igennem forskellige menuer og hvilke menuer der kan føre til hvad. Derudover kan man se hvilke blokke der snakker ud af frontend og sammen med resten af systemet.

### 7.2.1 Pseudo Frontend Arkitektur

(Hovedrapport stuff) For at give overblik over, hvordan kommunikationen mellem frontend, backend og gamecontroller kommer til at foregå, er der lavet et pseudo sekvensdiagram for følgende UserStories:

- Login
- Register
- Save Game
- Load Game

(/Hovedrapport stuff) Der er ikke lavet sekvensdiagrammer for alle af projektets userstories, da mange af disse fungerer på samme måde og derfor ikke bidrager med noget nyt ift. dokumentationen. Nedenfor ses pseudo sekvensdiagrammer for de fire userstories:

- Login
- Register
- Save Game
- Load Game

Først ses "Login" (Figure 10), som viser forløbet af userstory "Login", med en reference til userstory "Register", hvis brugeren ikke er registreret i forvejen. Udover dette er der ydermere vist håndtering fejlet login. Det skal her nævnes at brugeren kan ikke spille spillet uden at logge ind først, der tillades ikke at spillet kan spilles i Offline-tilstand.

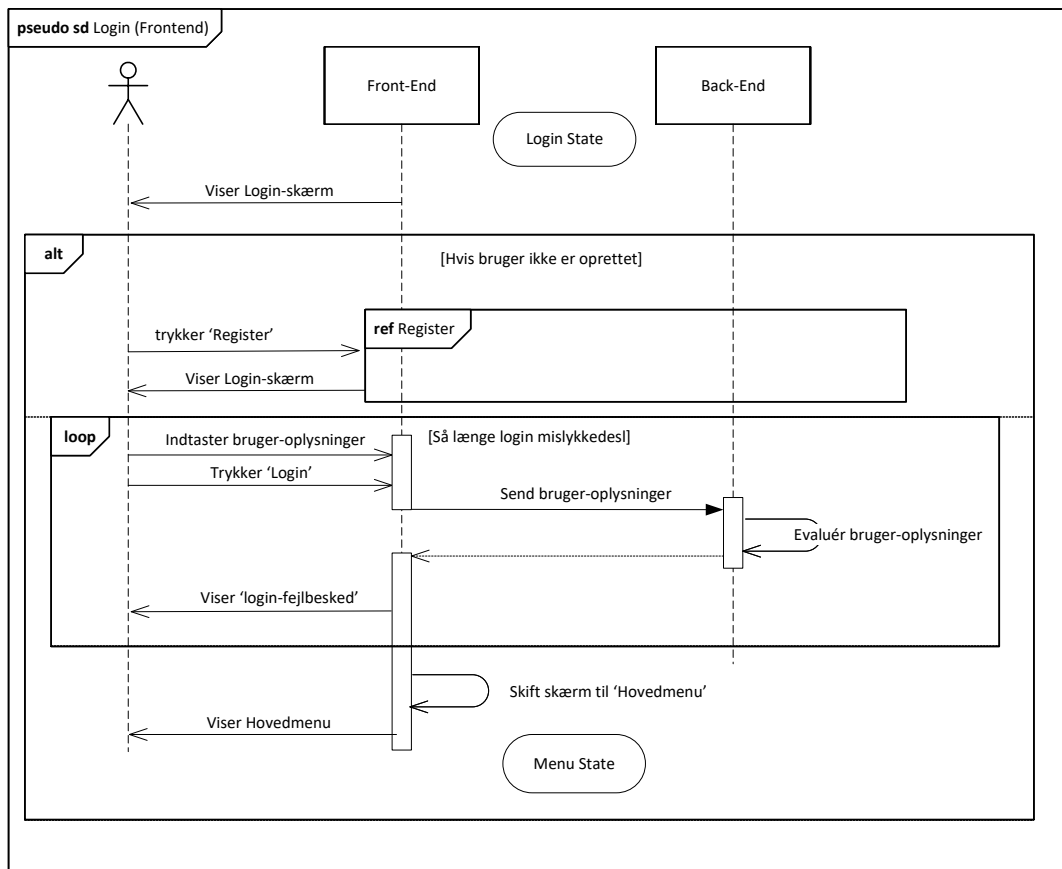


Figure 10: Pseudo sekvensdiagram af forløbet af userstory "Login", set fra Frontends perspektiv. Med reference til "Register" userstory og håndtering af forkerte login oplysninger.

Dernæst ses "Register" (Figure 11), som viser forløbet af hvordan en bruger kan registrere sig med en profil i systemet. Der kan ses på Figure 10, at hvis en bruger ikke er oprettet skal man gøre dette først. Derefter skal man vælge et brugernavn og kodeord, hvis brugernavnet er ledigt og alt ellers går godt, kommer man tilbage til "Login" skærmen. ellers får man en fejlbesked på skærmen og bliver bedt om at prøve med et andet brugernavn.

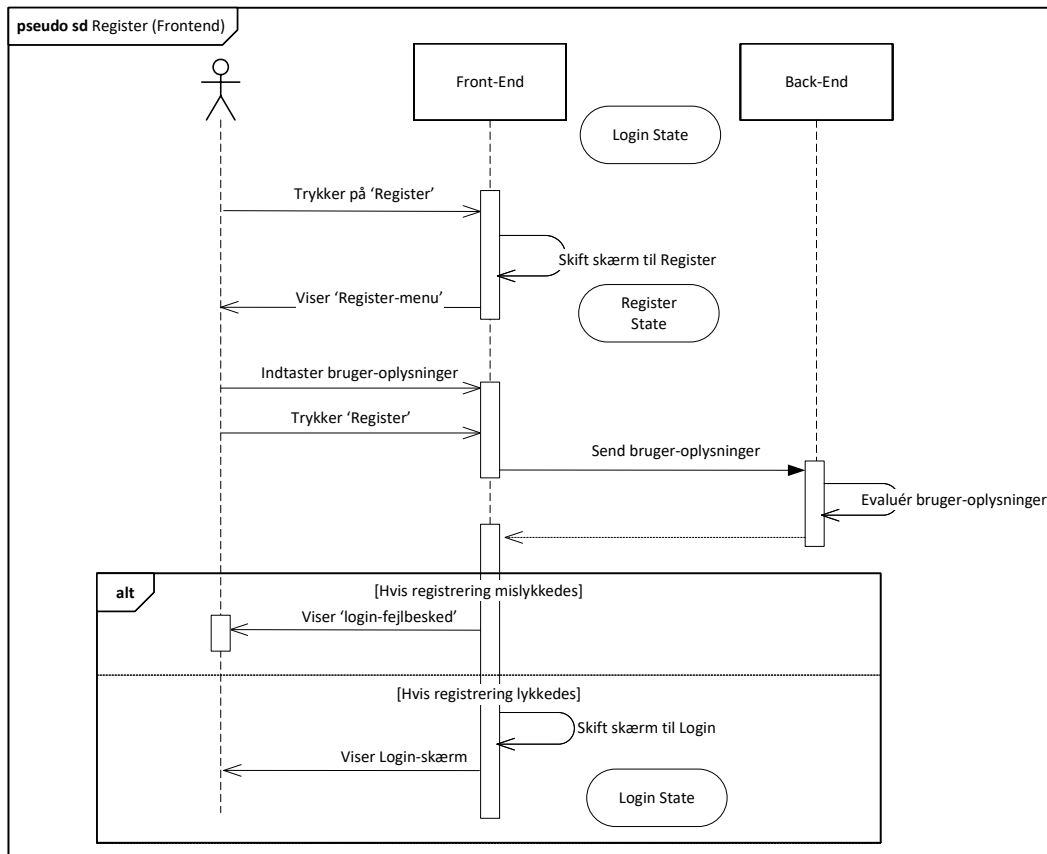


Figure 11: Pseudo sekvensdiagram af forløbet af userstory "Register", set fra Frontends perspektiv. Med håndtering af forkerte login oplysninger.

På Figure 12 ses "Save Game", som viser forløbet når en bruger gerne vil gemme sit igangværende spil, set fra Frontends perspektiv. Her kan man bide mærke i, at når der skiftes skærm, vil den nye skærm få initialiseret sine variabler i sin constructor og derfor er der kun et selv kald hver gang skærmen skiftes. Hertil skal der nævnes at hvis brugeren er i "Combat State" er det ikke muligt at gemme spillet og knappen "Save Game" på "In Game Menu" vil ikke kunne ses eller bruges.

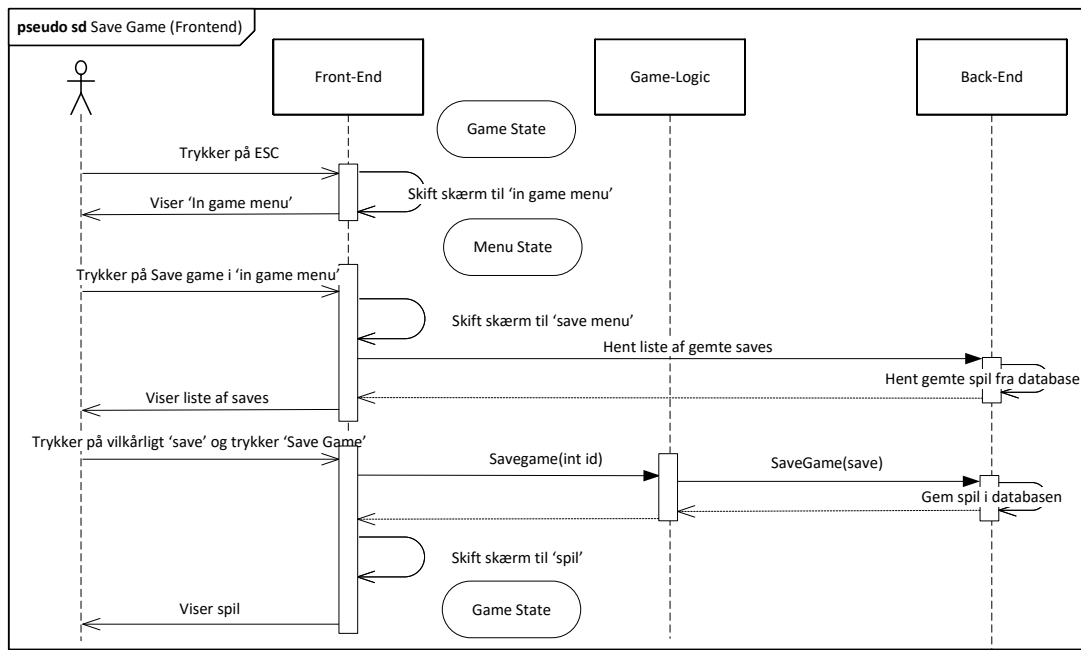


Figure 12: Pseudo sekvensdiagram af forløbet af userstory "Save Game", set fra Frontends perspektiv. Der laves 2 kald til databasen igennem Backenden, hvori der i det første kald, "Hent liste af gemte saves" hentes en liste af brugerens gemte spil og i andet kald gemmes brugerens nuværende spil henover det valgte spil.

Tilslidst kan der på Figure 13 ses "Load Game", som viser forløbet når en bruger gerne vil hente et tidligere gemt spil fra databasen, set fra Frontends perspektiv. Denne userstory minder på mange måder om "Save Game", men i stedet for at sende et element til databasen, skal der hentes et gemt spil fra databasen, med alle de data der skal bruges for at kunne loade sit spil op præcis som man forlod det.

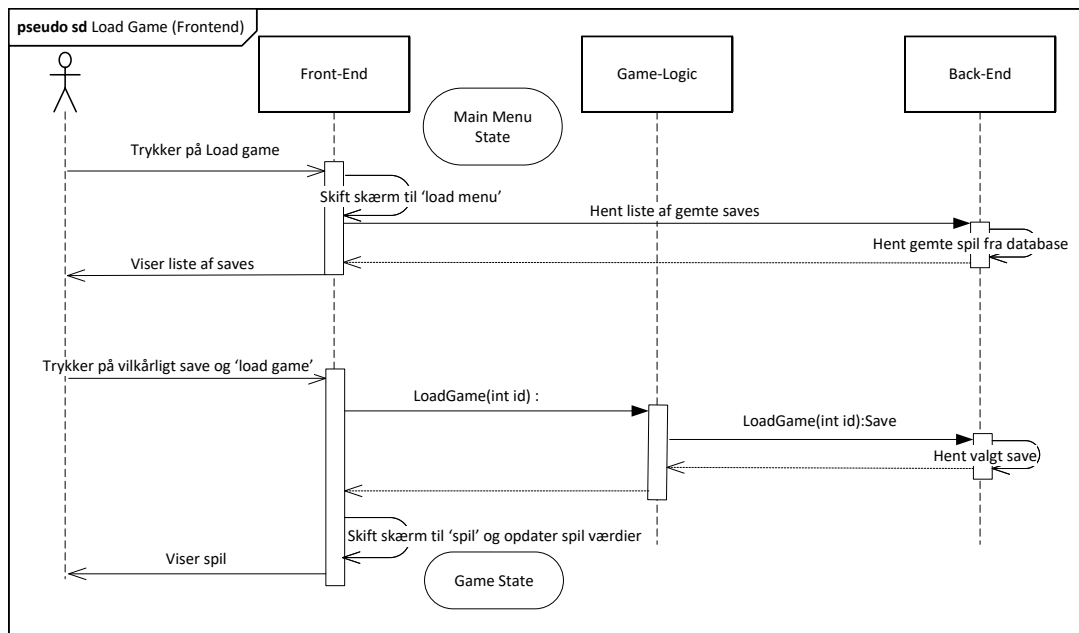


Figure 13: Pseudo sekvensdiagram af forløbet af userstory ”Load Game”, set fra Frontends perspektiv. Der laves 2 kald til databasen igennem Backenden, hvori der i det første kald, ”Hent liste af gemte saves” hentes en liste af brugerens gemte spil og i andet kald hentes brugerens valgte spil og spillet startes op.

### 7.3 GameEngine Arkitektur

GameEnginens rolle er at skabe logikken for brugeren i spillet. GameEnginens rolle for systemet er afgørende, når spillet er startet for brugeren. GameEnginens funktionalitet indebærer blandt andet at skabe et map for brugeren, så spilleren kan flytte fra rum til rum ved start af spil. Her forekommer det at brugeren anvender Front End til at interagere med spillet og derefter kan funktionerne fra GameEngine kaldes.

#### 7.3.1 States og Character interagering

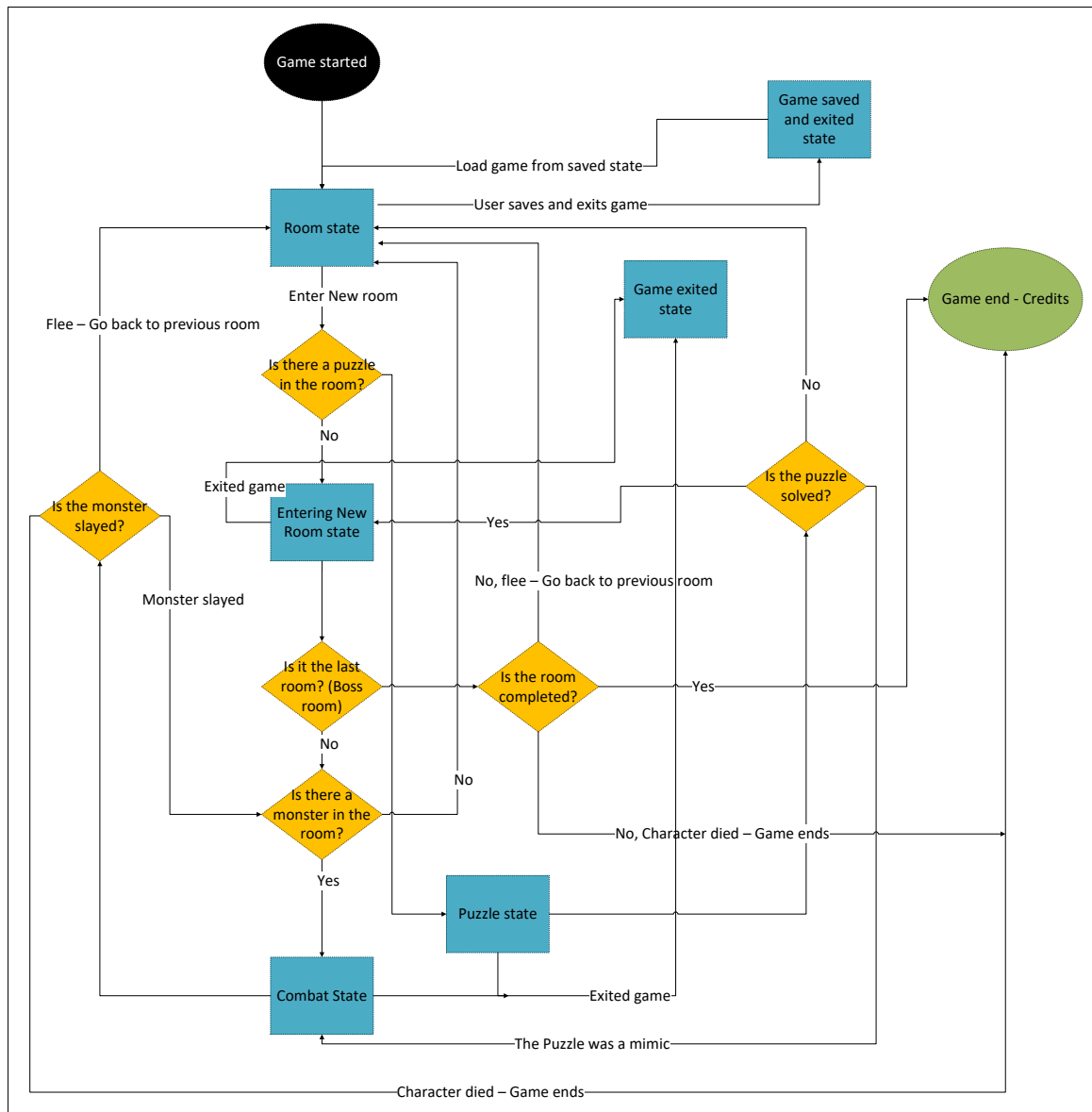


Figure 14: Flow Chart over systemets states når spillet er startet. Dette bilag viser arkitekturen over det ønskede states i systemet når bruger har startet spillet. Dette viser også hvordan spillets fremgang er ønsket og hvordan bruger kommer videre i spillet igennem de forskellige states.

### 7.3.2 Room State og Character interagering

GameEngineen er delt op i spillets game states som kan ses i bilaget ovenover. Der er hovedsageligt 2 states når spillet er startet. Room State og Combat state. Når spillet er startet for brugeren, starter brugeren i et room state. I dette state kan der interageres med rummet, hvis der er genstande til stede. Her kan brugeren også brugerdefinere sin spiller ved brug af knappen inventory og skifte våben eller skjold. Her kan der også se spillerens evner ved brug af knappen Character. Derudover er der også implementeret beskrivelser fra de forskellige rum som hentes fra modulet Database via modulet Back end.

### 7.3.3 Combat State og Combat simulering

Combat state indebærer når spilleren møder en fjende. Dette sker når man går ind i forskellige rum. Der er oprettet klasser for spillerens evner i form af Attack(Spillerens evne til at slå) og Armor(Spillerens evne til at modstå angreb). Når spilleren angriber en fjende foregår det ved at brugeren trykker på knappen 'Attack'. Når dette forekommer er der implementeret en terning i GameEnginenen. Dette skal implementeres ved to implementeringer. En simuleret terning i form af en pseudo random-number generator som genererer et tal mellem 1 og parameteren numOfSides og en En Rekursiv Psudo-Random number generator som tager en tuple (numOfSides, numOfDice). Den gentager rekursivt implementering 1. Et antal gange svarende til NumOfDice parameteren og summere alle resultaterne. Hvis brugeren taber kampen, skal brugeren starte et nyt spil eller load et save. I figuren under ses et Flow chart over forløbet når spiller går ind i combat state.

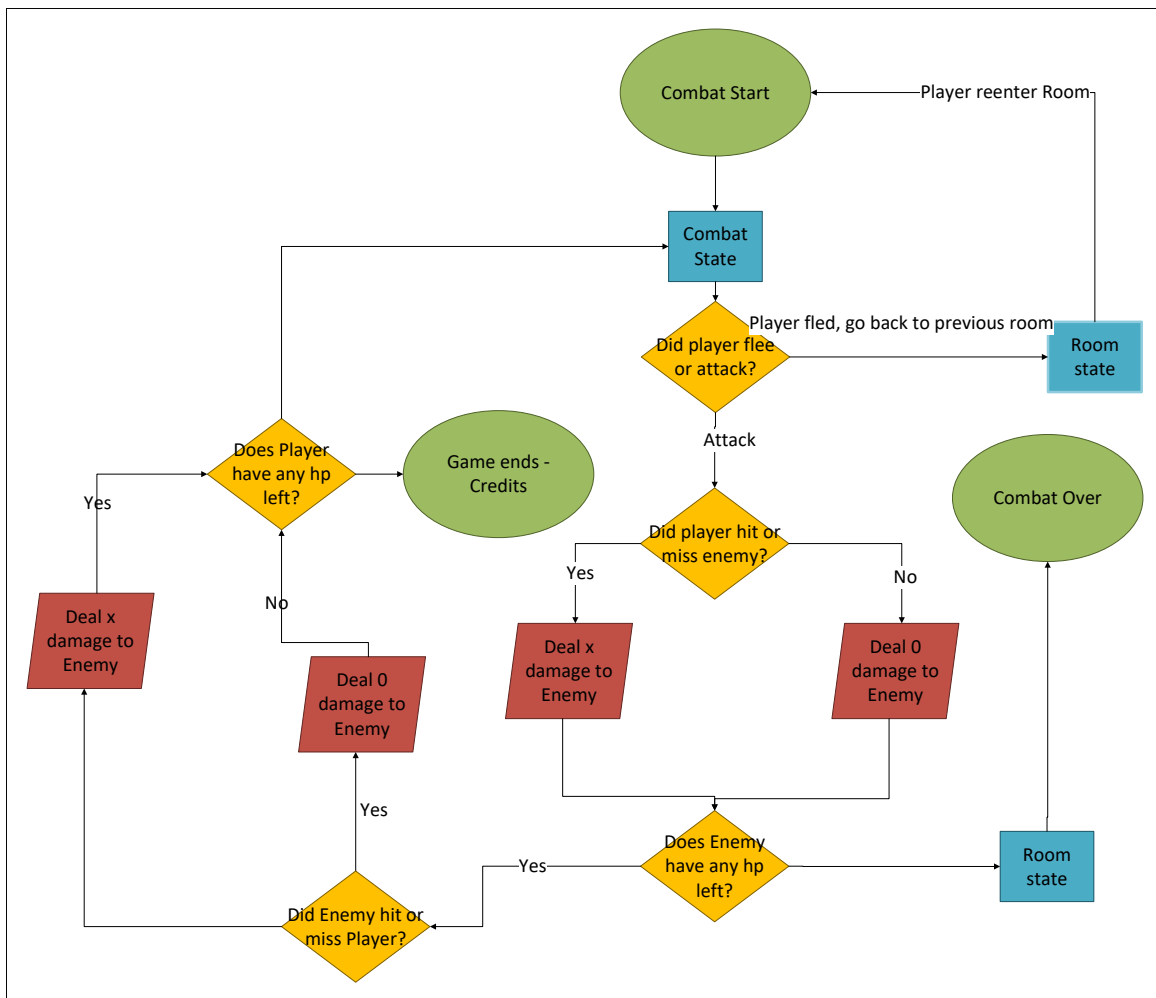


Figure 15: Flow Chart over spillets fremgang når spiller går ind i combat State. Når spiller går ind i et rum med en fjende i rummet, går spillet ind i combat state. Herfra skal spillets terning simulere et tilfældigt nummer ud fra spillerens evner. Fjendens evner er bestemt i forvejen. Angreb fastlægges om spiller ruller højere end fjendens Armor Class og vice versa. De tre outcomes for spilleren er følgende: Spiller flygter (flee-knap), Spilleren mister alt liv (Spil sluttet) og fjende mister alt liv (Spiller går til room state).

### 7.3.4 Forbindelser til andre moduler

GameEngineen er forbundet med systemets Back-end og Front-End. Den håndterer data fra Back-end som derefter kan håndteres, så Front-end kan kalde funktionerne fra GameEnginen. Det data som skal håndteres er primært gemmets forløb som kan lagres i databasen under brugeren. Herefter kan brugerne load det samme gem igen med samme fremgang som da brugeren gemte spillet. Det vil være nuværende rum, rum der er blevet besøgt, fjender der er blevet bekæmpet og genstande der er samlet op og taget på.



## 7.4 Backend Arkitektur

I dette afsnit redegøres for arkitekturen af backend containeren. Dokumentet vil først komme med en kort redegørelse for MVC mønstrets opbygning og dets bidrag til arkitekturen. Dernæst gennemgås de relevante krav, hvad angår backenden, disse krav bruges som input til arkitekturen. Med kravene på plads præsenteres et C4 level 3 diagram over backend'en, som viser hvordan Web api'et opbygges. Til slut præsenteres REST principper samt hvordan de anvendes. Systemets backend vil bestå af et Web API udviklet i frameworket ASP.NET Core. Web API'et skal give mulighed for containeren GameEngine at tilgå databasen igennem HTTP request/responses, for at hente og sende den nødvendige data til databasen. Hertil skal backend'en sørge for Authentication og Authorization af de indkommende kald.

### 7.4.1 MVC

Selve Web API'ets arkitektur bygges op omkring MVC (Model-View-Controller) mønstret. En illustration af MVC mønstrets struktur kan ses på Figure 16 [MVCpattern].

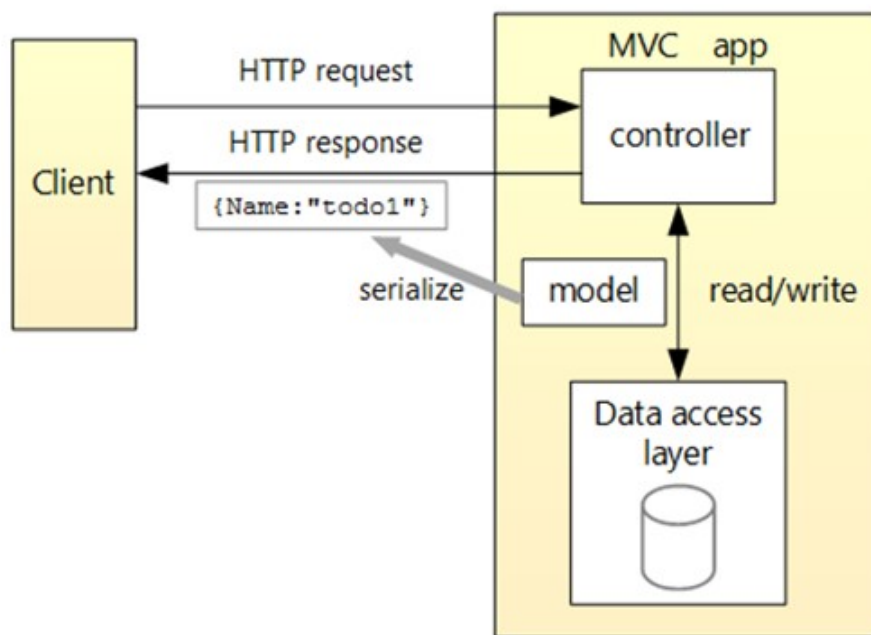


Figure 16: Illustration af MVC pattern, bestående af clienter, controller, modeller, og et DAL, som viser hvordan de kommunikerer

Som det kan ses på figuren består mønstret overordnet af 4 dele: client, controller, model og et DAL (Data Access Layer). Da der er tale om et Web API gøres der ikke brug af view moduler. De enkelte moduler har følgende ansvar.

**Client:**

Clientens rolle er at kontakte Web API'et med henblik på at få udført en service, eksempelvis at logge ind.

**Controller:**

Controllernes ansvar er at håndtere indkommende kald fra klienten, og route til de rigtige Http endpoints, hvorefter databasen kontaktes igennem DAL.

**DAL:**

DAL's ansvar er at kontakte og administrere kommunikation med databasen i form af queries. Det er også her relationerne imellem data objekterne defineres.

**Model:**

Modelernes ansvar er at håndtere og definere den data som kan sendes frem og tilbage mellem klienten og databasen, de sørger altså for at binde alle modulerne sammen, således at der er enighed omkring hvordan data objekterne ser ud.

### 7.4.2 C3-Model for backend

De relevante user stories for backend containeren udvælges. Disse omhandler håndtering af bruger konti, samt loading og saving af et game state.

- User Story 1 : Log in
- User Story 2 : Opret Bruger
- User Story 8: Exit Menu - Save - No combat
- UserStory 15 : Save Menu - Save Game
- UserStory 18 : Main Menu - Load Game - Load

Af de ikke funktionelle krav er følgende krav relevante:

- Skal kunne gemme maksimalt 5 save games
- Skal kunne loade et spil indenfor maksimalt 5 s.
- Skal kunne respondere indenfor maksimalt 5 s.

De ikke funktionelle krav tages hånd om under designet og implementeringen af backenden. For yderligere detaljer omkring funktionelle og ikke funktionelle krav henvises til afsnit omkring kravspecifikation section 5

Der bygges videre på C4 modellen for systemet, et level 3 component diagram over backend containeren udarbejdes, som viser hvilke componenter containeren indeholder, deres ansvarsområder, hvordan de kommunikere indbyrdes og med de omkring liggende containere, samt hvilke teknologier der anvendes i implementeringen. Diagrammet kan ses på Figure 17.

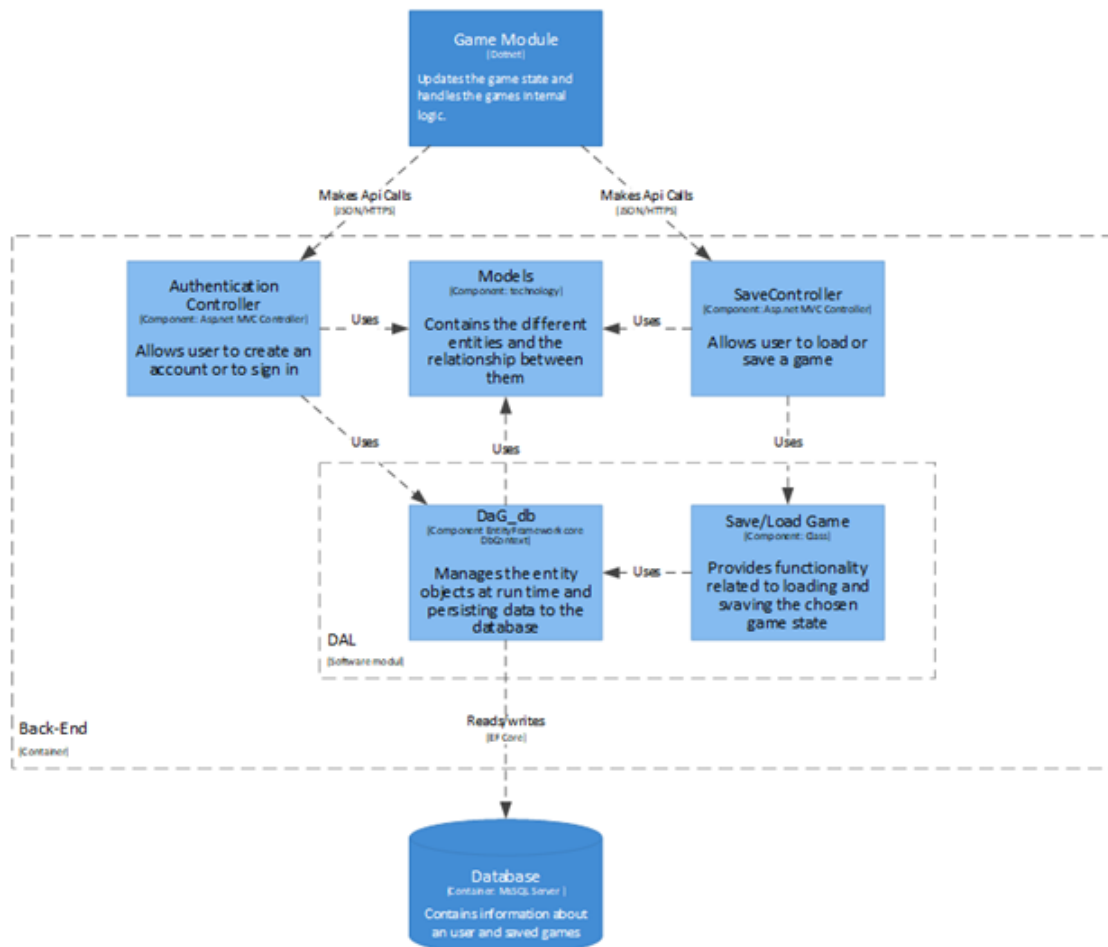


Figure 17: C3-Model over Backend container, figuren giver et overblik over de componenter backenden består, samt hvordan disse kommunikerer indbyrdes og med omkring liggende containere.

Figuren illustrerer en lagdelt struktur, hvor der ses en client som kalder ned i controllerne Authentication og Save Controller, som kalder videre til DAL, som så tilgår databasen denne kommunikationsvej går begge veje.

#### Client:

Der eksisterer en client, som kommunikerer med backenden Game Module. Game Module's kald til backenden kan overordnet set indeles i to undergrupper, bruger håndtering og save håndtering. Bruger håndtering indeholder kald som omhandler login og registrering (User story 1 og 2), hvor save håndtering omhandler loading og saving af Game state (User story 8, 15, 17 og 18). Kommunikationen her vil foregå ved netværkskald med protokollen HTTPS. Clienten udfører request/response kald til backenden igennem specifikke URL's. For at data objekterne kan sendes frem og tilbage med HTTPS kæves disse seraliseres og deseraliseres mellem JSON-objekter og .Net objekter.

#### Controller:

Inde i backenden anvendes ASP. Net MVC controllere [MVCcontroller] til at modtage og svare på kald fra clienten. Der laves en controller for Bruger håndtering og en controller for Save håndtering. Hver request mappes til en specifik controller. Controllerne vil så indeholde action metoder, som udføres når den modtager et kald. Når controlleren har udført sin action returneres et Action Result, hvor i et data objekt kan wrappes eller den kan indeholde en fejlmeddelelse hvis noget går galt.

**DAL:**

De to controllere kalder som sagt videre ned i DAL laget, dette lag består af en database kontekst DaG\_db og en hjælper klasse til at udfører Queries. Her gøres brug af EF (Entity Framework) Core [EF Core], som gør det muligt at arbejde med DbContext klassen, denne bruges til skabe en kontekst af databasen, hvor i de enkelte modeller kan mappes til entities.

**Model:**

Modellerne fungerer som bindeledet mellem alle komponenterne, de definerer den data som der arbejdes på, og vil blot bestå af en række C# klasser.

Som det kan ses passer MVC mønstret perfekt til den funktionalitet der ønskes. Mønstret gør det muligt at lave en logisk gruppering af relaterede opgaver, og er med til at sikre høj samhørighed i applikationen, ved at de enkelte ansvarsområder fordeles ud på hver sine moduler. På figur 1 Figure 17 ses det tydeligt hvordan koblingen mellem de enkelte moduler holdes på et lavt niveau, hvilket gør det lettere at bygge videre på og tilføje ny funktionalitet.

**7.4.3 REST**

Web Api'et arkitektur vil gøre brug af REST principper, for ikke at gøre data overførelserne for komplekse og for at overholde SoC, ved at adskille repræsentationen og dataen fra hinanden. På den måde sikres det at dataen, kan blive repræsenteret på den ønskede måde, og ikke er tvunget til at blive gemt på en specifikt måde. Dette kommer til udtryk ved følgende 5 REST principper[REST] som søges opnået:

1. Alle de data objekter der arbejdes med tilhører en bestemt unique URL. Dataen hentes, sendes og manipuleres igennem standard HTTP metoder som (GET, POST, PUT, DELETE).
2. Der arbejdes ud fra en client/server arkitektur med data som resource.
3. Web Api'et er stateless, hvilket vil sige der gemmes ikke nogen tilstand omkring clienten på server siden.
4. Der arbejdes ud fra en lagdelt struktur, som betyder at hver component kun kan se de componenter som grænser op til den selv.
5. Der anvendes Caching på server siden (Dette vil ikke blive implementeret).

**7.4.4 Konklusion**

Systemets backend består af Web Api, som bygges op omkring MVC mønstret, som gør det muligt at lave en logisk gruppering af den enkelte funktionalitet. Web Api'et vil ligeledes gøre brug af REST principper for at adskille data resourcerne fra deres repræsentation på clientens side.

## 7.5 Database Arkitektur

I oprettelsen af systemets database skal der tages hånd om, hvordan kommunikationen skal foregå imellem systemets segmenter, samt databasens funktionalitet. Her er der blevet besluttet at der anvendes en DAL, der fungerer ved at hver gang databasen skal kontaktes, så foregår det igennem denne. Yderligere vil denne DAL også simplificere kommunikationen mellem databasen og Back-End.

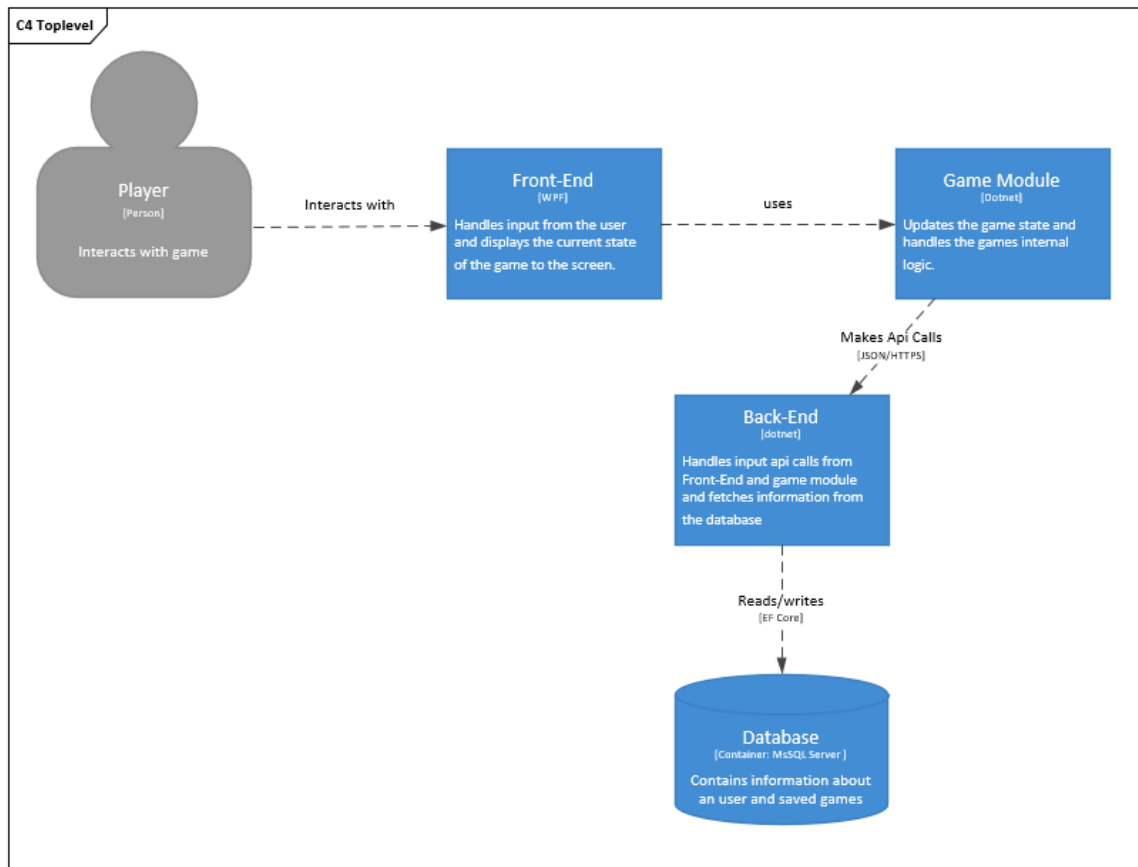


Figure 18: C4 top level diagram, som viser kommunikation mellem systemets segmenter

Måden kommunikationen vil foregå igennem systemet vises på Figure 18. Her ses der at en bruger interagerer med Front-End, data går videre til Game Module, som kommunikerer med Back-End, hvor der til sidst enten bliver skrevet til databasen eller hentet data fra databasen. For at illustrere databasens funktionalitet er der blevet dannet sekvensdiagrammer, som viser hvordan databasen vil kunne gemme et spil og hente et gemt spil.

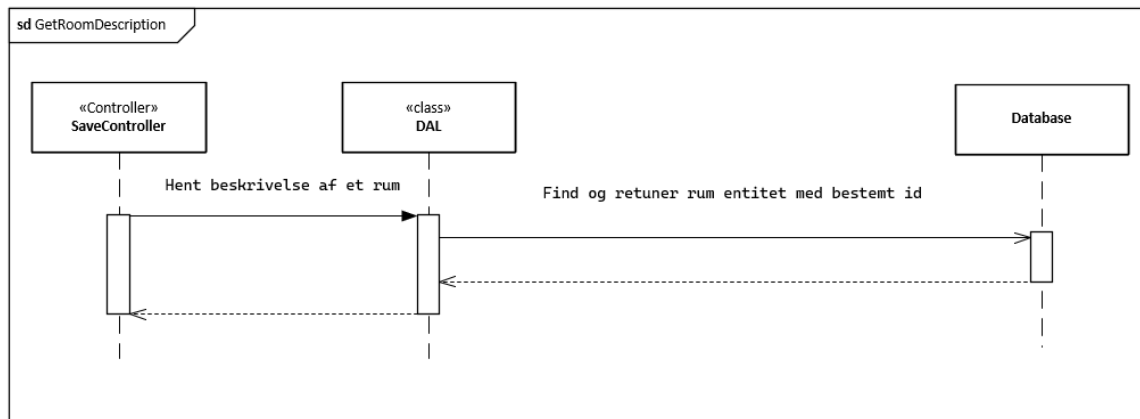


Figure 19: Sekvensdiagram for hvordan rum beskrivelser vil blive hente fra databasen af SaveController

I Figure 19 ses diagrammet GetRoomDescription ses der, hvordan kommunikationen ville foregå for at hente rum beskrivelsen. Her ses der at SaveController, fra Back-End, går til DAL, som så henter rum beskrivelsen fra databasen, og herefter returneres dataene fra databasen til DAL og så fra DAL til SaveController.

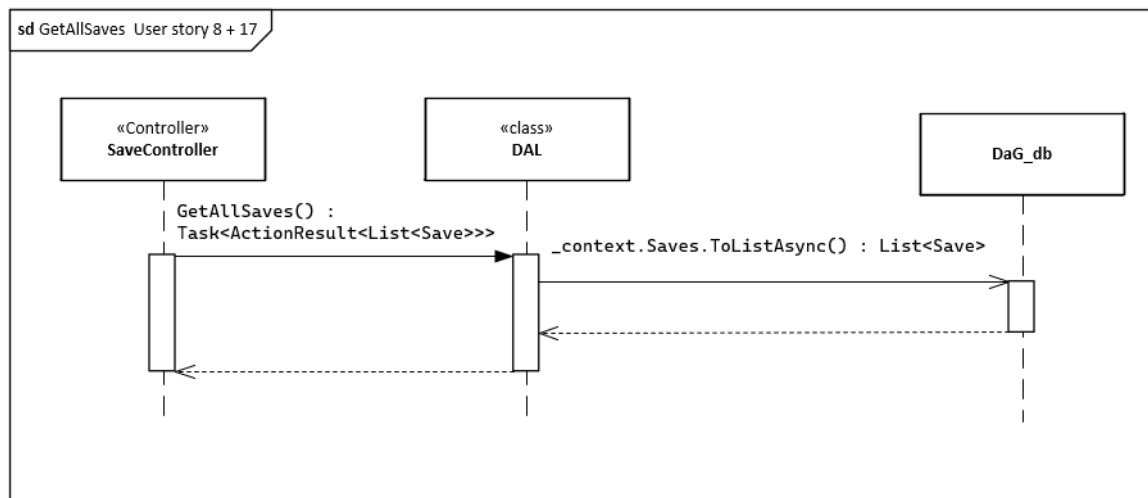


Figure 20: Sekvensdiagram for User story 8 + 17: GetAllSaves, i relation til hentet data fra database

I figuren Figure 20 ses diagrammet GetAllSaves foregår kommunikation på samme måde som ved at hente rum beskrivelser. Forskellen her er dog at der i stedet for hentes en list af alle gemte spil.

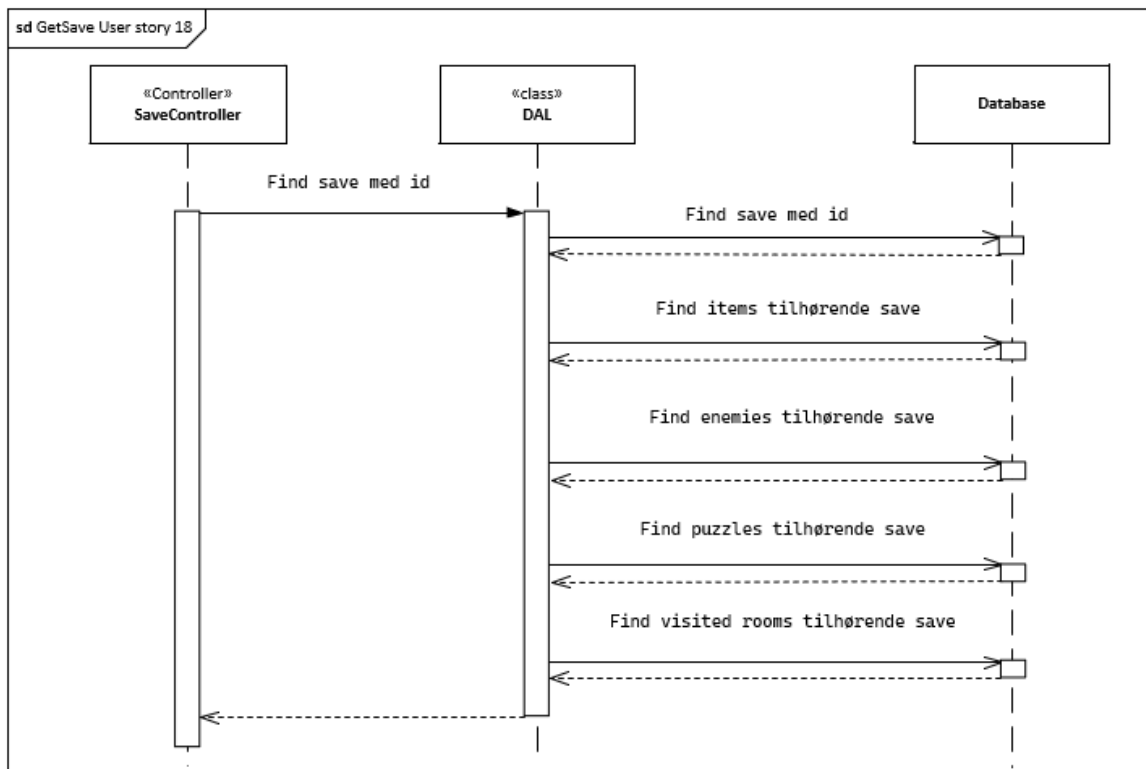


Figure 21: Sekvensdiagram for User story 18: GetAllSaves, i relation til hentet data fra specifikt gemt spil fra database

I Figure 21 ses diagrammet GetSave ses der hvordan et specifikt gemt spil hentes. Her ses der at dataene igen går først igennem DAL. Et gemt spil vil have et ID, som der kan anvendes til at finde de korresponderende værdier som dette gemte spil har. Her ses der at der vil findes items, enemies, puzzles og hvilke rum en spiller har besøgt. Dataene returneres herefter til DAL og så til SaveController.

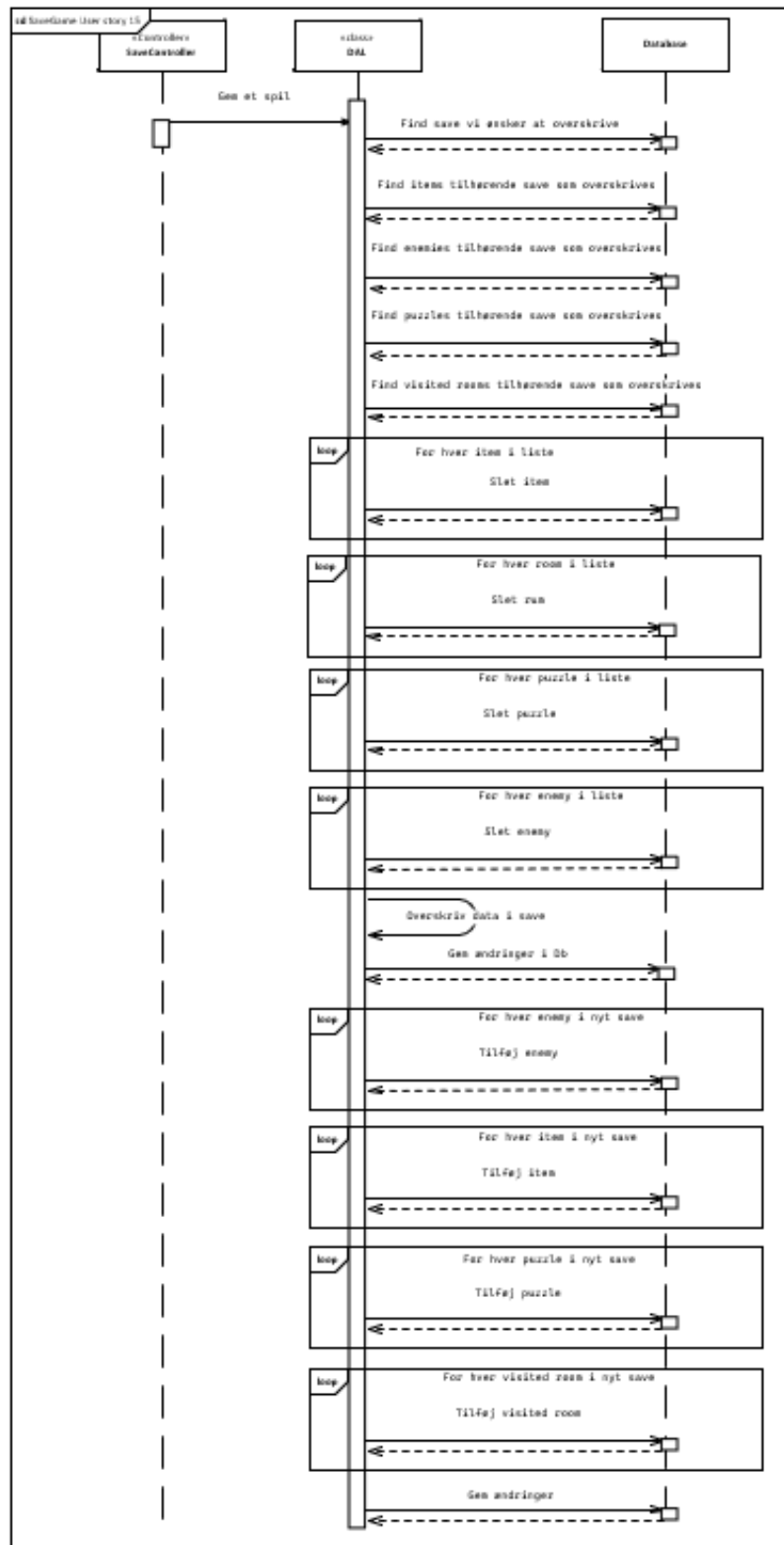


Figure 22: Sekvensdiagram for User story 15: SaveGame. Dette diagram viser hvordan et spil gemmes



På Figure 22, SaveGame, ses der hvordan et spil vil gemmes. Måden dette vil fungere på er at spilleren vil starte med fem tomme gemte spil, som bliver seeded til databasen. Dette bliver gjort for at begrænse en bruger til fem gemte spil. Så for at gemme et spil findes der først hvilket gemte spil der skal overskrives. Derefter findes de relevante værdier i dette gemte spil. Hernæst slettes disse værdier og der tilføjes de nye korrekte værdier.

Brugeren vil derfor ikke have mulighed for at oprette et nyt spil, men kun mulighed for at overskrive en af fem tomme spil, som bliver oprettet for hver bruger.

## 7.6 DAL Arkitektur

I dette segment vil der blive forklaret tankerne bag arkitekturen vedr. oprettelsen af en funktionel DAL, som kan anvendes til at styre kommunikationen til og fra databasen. Til oprettelsen af en funktionsdygtig database, i dette projekt, kræves der et relativt tæt sammenhold mellem databasen og backend api'en. API'en er ansvarlig for kommunikation mellem Front-end, Game Module og databasen.

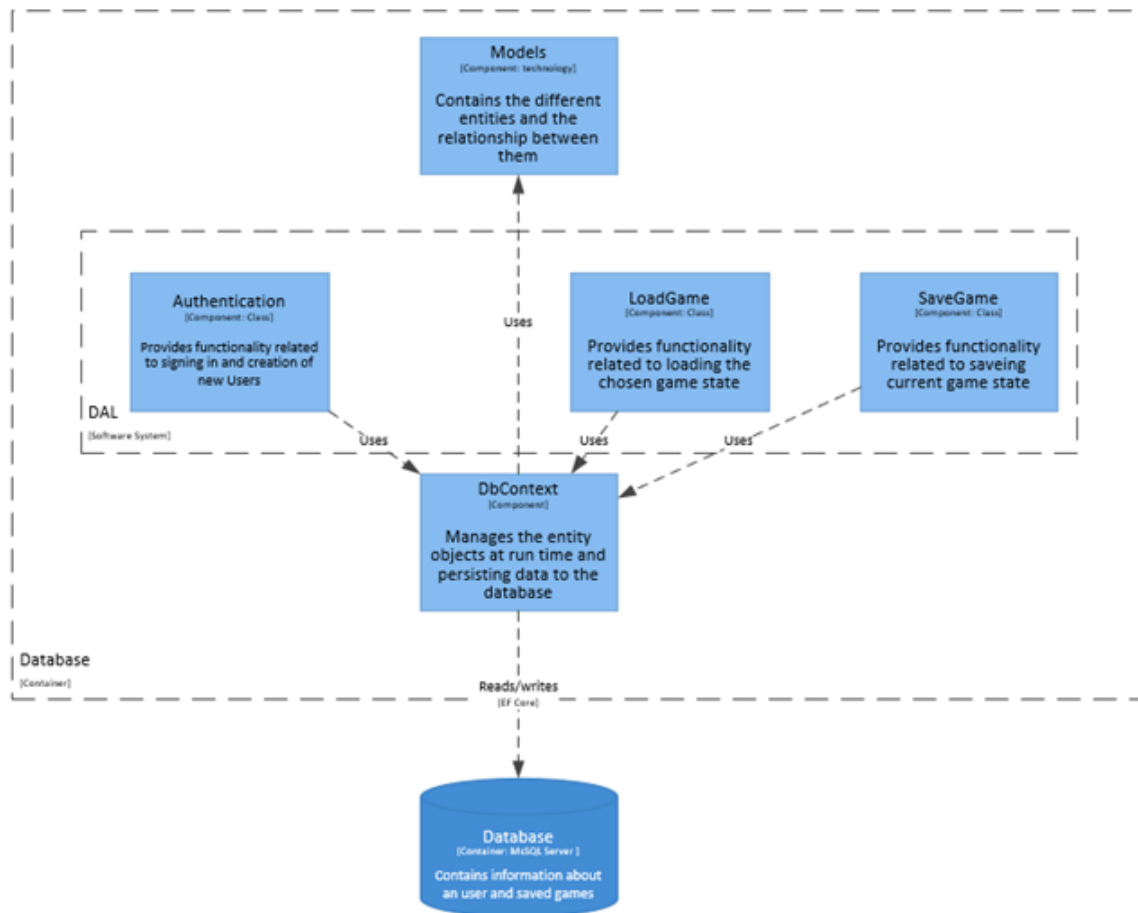


Figure 23: C3 diagram for blokkene involveret i DAL's funktionalitet

Når data enten skal sendes til eller hentes fra databasen så er det igennem et DAL. Systemets DAL giver os en simplificeret adgang til dataene gemt i databasen, og fungerer som en mellemmand for systemet, da alt kommunikation til databasen går igennem den. I denne DAL har vi en Authen-

tication, som bliver anvendt når en bruger logger ind, og når der oprettes en ny bruger. Denne blok kontrollerer brugernavn og kodeord, som sendes og hentes i databasen. Ydermere vil der ikke sættes fokus på sikkerhed, som ses i kravene stillet for systemet. Yderligere vil det også være muligt at både gemme et spil og indlæse et spil. Begge af disse vil operere på det samme data, dog ville den ene, LoadGame, hente data fra systemets database, og den anden, SaveGame, vil sende data til systemets database. LoadGame i DAL er ansvarlig for at hente spillerens data fra databasen, såsom hvilket rum de var i og mængden af liv de har tilbage. Hernæst er der SaveGame. Denne indeholder funktionaliteten for at sende et gemt spil, altså dataene for spillerens nuværende spil. Heri vil der også gemmes information og spillerens nuværende tilstand i spillet. Det ville f.eks. være rummet som spilleren er i når spillet bliver gemt. Begge af disse vil være ansvarlige for at håndtere data som er individuelle for hver spiller og hvert gemt spil. Udenfor systemets DAL ville der også være inkluderet Models i konstruktionen af systemets database. Models vil indeholde de entities som Authentication og Load-/SaveGame vil bestå af. Yderligere vil Models også være ansvarlig for forholdene imellem de forskellige entities.

## 8 Design

### 8.1 Overordnet System Design

Dette afsnit repræsenterer hvordan modulerne ønskes at kommunikere med hinanden samt give et overblik over hvor meget der kommunikeres mellem modulerne i de forskellige stadier. Der vil i dette afsnit indgå 4 User stories til at repræsentere systemets design. Disse User Stories er relevante for design, da de involverer alle moduler samtidigt og giver et indblik i kommunikationen på tværs af systemet.

Herunder ses et sekvensdiagram for User Story 7-10 - Save. Her ses hvordan det ønskes at de forskellige moduler kommunikerer på tværs af systemet når en bruger loader et save fra databasen.

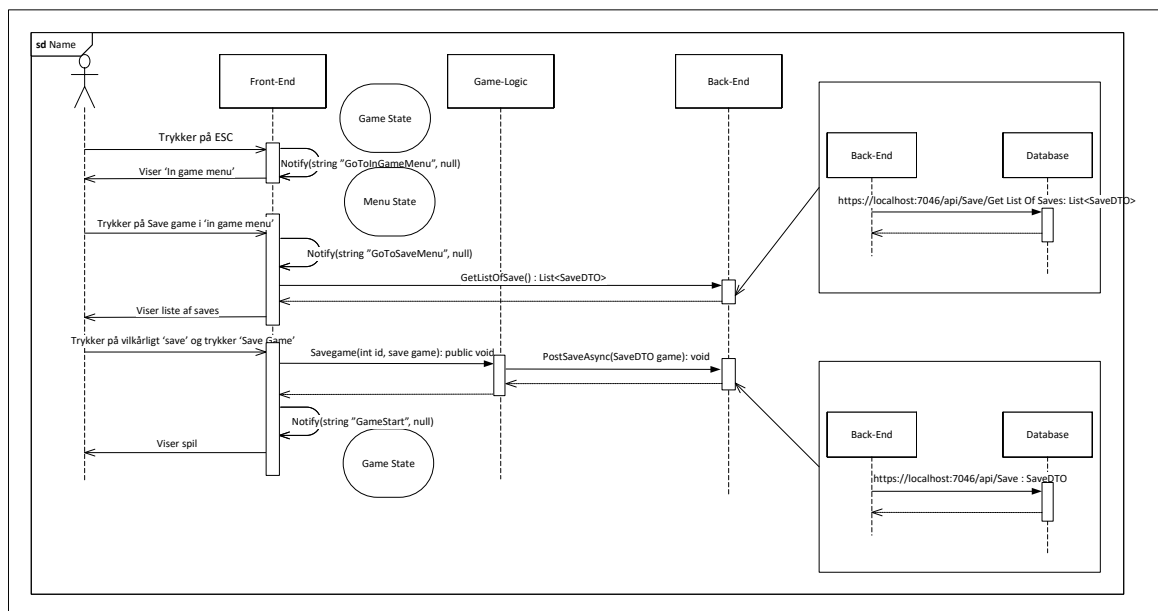


Figure 24: SD diagram for User Story 7-10. Diagrammet viser hvordan det ønskes at systemet overordnet skal kommunikere på tværs af hinanden når en bruger skal gemme sit aktuelle save

Herunder ses et sekvensdiagram for User Story 17-18 - Load. Her ses hvordan det ønskes at de

forskellige moduler kommunikerer på tværs af systemet når bruger gemmer et save i databasen.

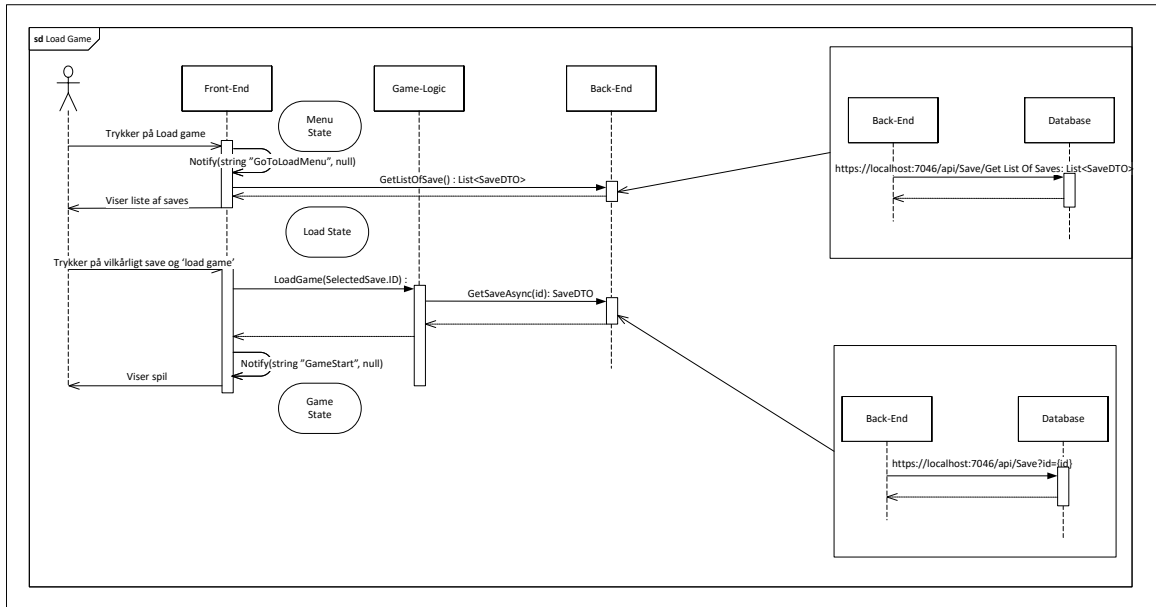


Figure 25: SD diagram for User Story 17-18. Diagrammet viser hvordan systemet overordnet skal kommunikere på tværs når bruger skal laode sit aktuelle save.

Herunder ses et sekvensdiagram for User Story 1 - Log in. Her ses hvordan det ønskes at de forskellige moduler kommunikerer på tværs af systemet når bruger skal logge ind.

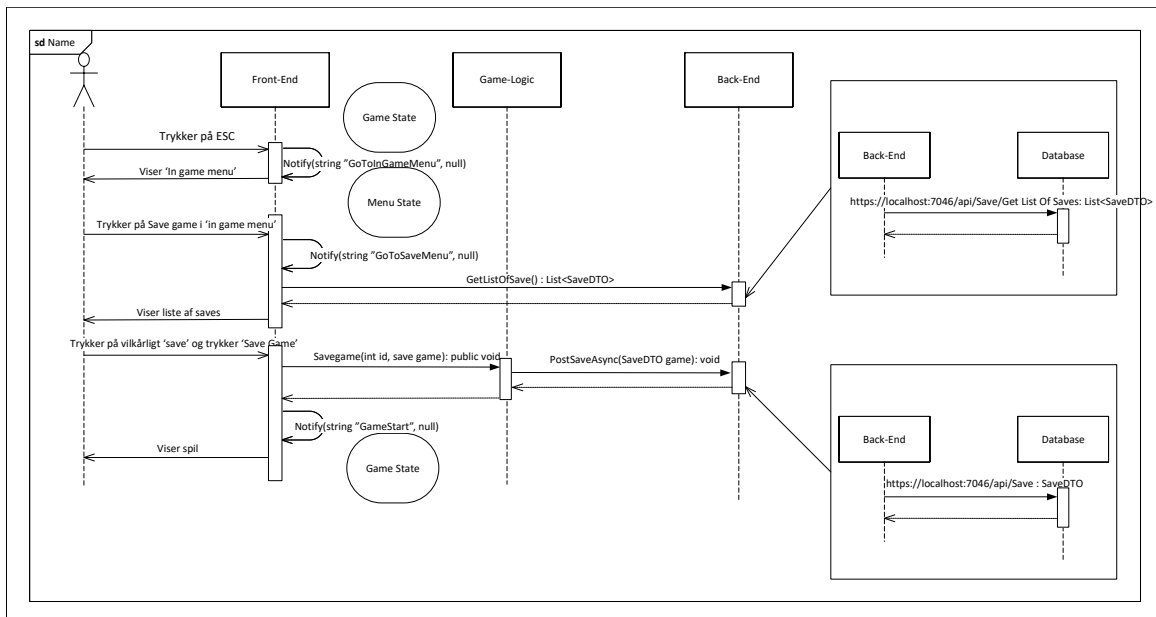


Figure 26: SD diagram for User Story 1. Diagrammet viser hvordan systemet overordnet skal kommunikere på tværs når bruger skal logge ind

Herunder ses et sekvensdiagram for User Story 2 - Opret Bruger. Her ses hvordan det ønskes at de forskellige moduler kommunikerer på tværs af systemet når bruger skal oprette en bruger i systemet.

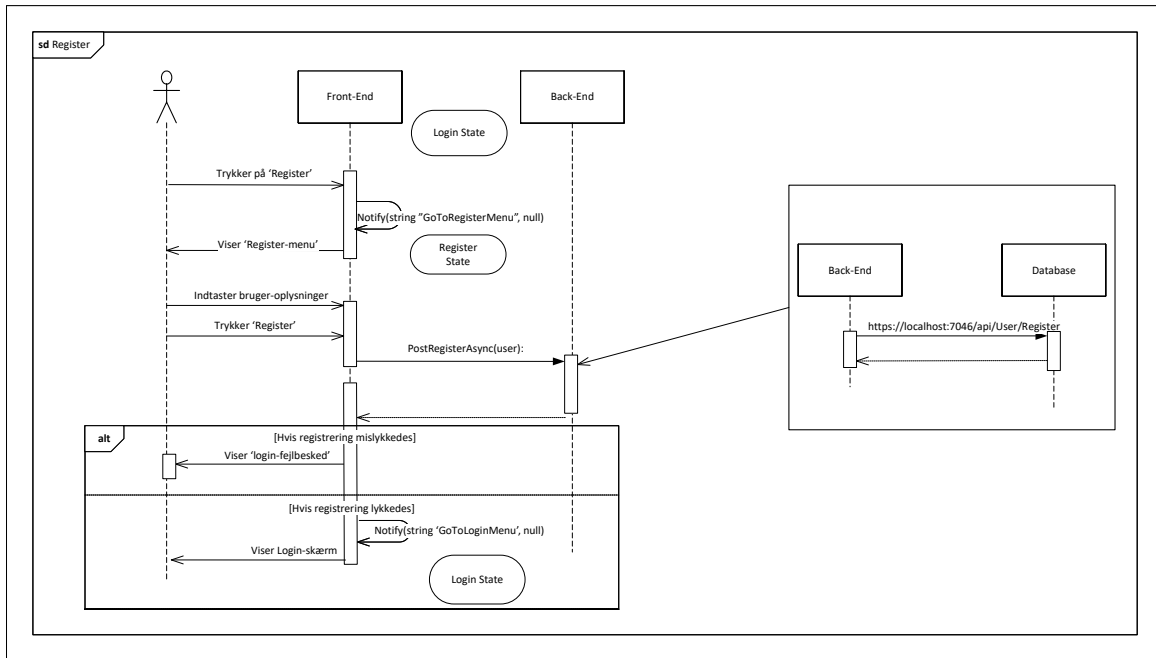


Figure 27: SD diagram for User Story 2. Diagrammet viser hvordan systemet overordnet skal kommunikere på tværs når bruger skal oprette en bruger i systemet

## 8.2 Frontend Design

I frontenden ønskes det at hele spillet foregår i et vindue. Det er derfor nødvendigt at programmet kan skifte mellem forskellige views uden at skulle åbne et nyt vindue, hver gang der skiftes. Løsningen til denne udfordring beskrives nærmere i implementerings afsnittet, i subsection 9.2. Spillet vil bestå af en række af vinduer, som giver spilleren den nødvendige information for at de kan spille spillet (så som at logge ind, gemme og hente spil, samt spille spillet).

Inden arbejdet på Frontend arkitekturen begyndte, er der blevet lavet en teknologiundersøgelse subsection 6.1, om hvilket udviklingsværktøj Frontenden og derigennem spillet skulle udvikles i. Baseret på denne teknologiundersøgelse er der blevet valgt, at spillet vil blive udviklet i et .NET framework. Dette valg er blandt andet truffet, da dette framework passer bedre med den opdeling af arbejde der er lavet i projektgruppen, altså opdelingen af Frontend og Backend. For andre grunde, se subsection 6.1, hvor flere fordele og ulemper for både unity og .NET frameworket er sat op. Her følger en række af illustrationer<sup>1</sup> af nogle af spillets views.

### 8.2.1 Room

Room view (Figure 28) er det primære spil-vindue. Her præsenteres spilleren for en beskrivelse af det rum de er i, samt hvilke elementer i rummet de kan interagere med. Der vises også et kort over banen. Kortet viser kun de rum spilleren allerede har været i, mens resten holdes skjult. Når brugeren så besøger et nyt rum, kan dette ses selvom spilleren forlader rummet. Dette lader spilleren udforske og oplåse hele kortet.

<sup>1</sup>Lavet i Unity

En række knapper nederst i højre hjørne på skærmen giver spilleren mulighed for at interagere med spillet. Fire knapper ("Go North/West/South/East") lader spilleren gå fra et rum til et andet. Ikke alle rum har forbindelse til alle sider, så det er f.eks. ikke altid muligt at trykke på "Go North". Kortet og rum beskrivelsen fortæller hvilken vej det er muligt at bevæge sig i.

Udover de fire retningsknapper er der et antal andre knapper. Disse bruges til at gemme spillet, gå til menuer, samt interagere med elementerne i rummet. Det specifikke antal og deres funktion er afhængig af den præcise implementering.

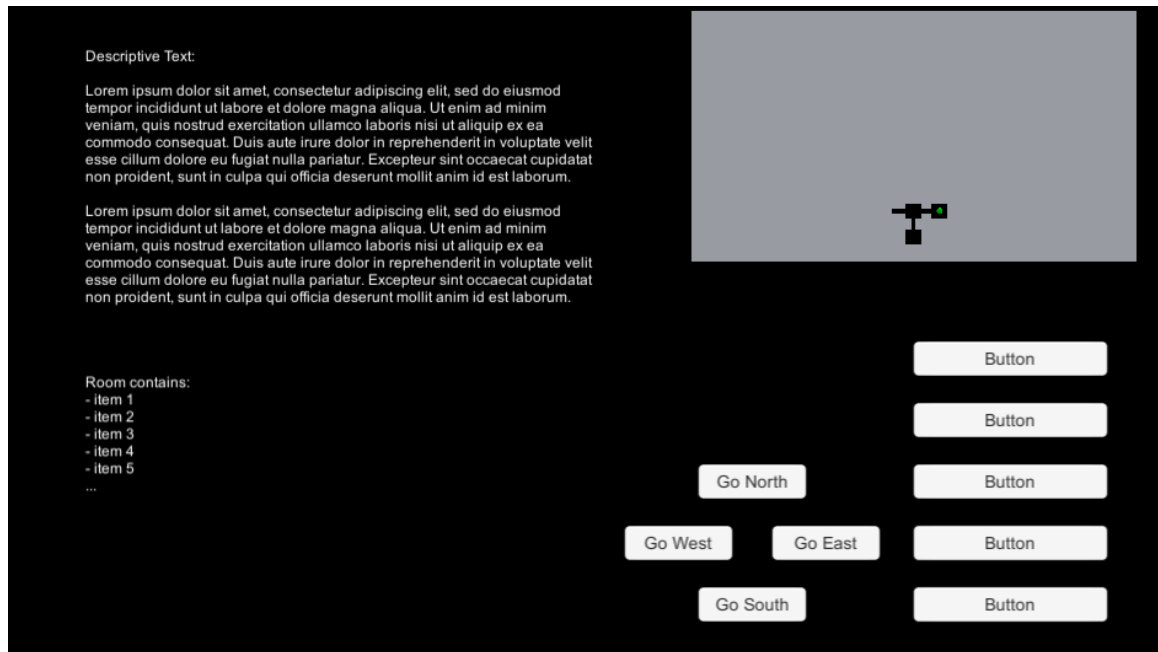


Figure 28: Et mockup af det primære spil vindue. Tekst øverst i venstre side af skærmen giver en beskrivelse af det rum spilleren er i, samt en liste af elementer i rummet som spilleren kan interagere med. Øverst til højre vises et billede af banen. Spilleren interagerer med spillet via knapper nederst i højre hjørne. Knapperne "Go North/West/South/East" fører spilleren ind i et andet rum, mens de resterende knapper (markeret "Button") bruges til andre funktionaliteter i spillet.

### 8.2.2 Combat

Combat vinduet er baseret på room vinduet. Strukturen er den samme: der er et kort øverst til højre, en beskrivelse øverst til venstre og knapper nederst til højre. Nederst til venstre er der information om hvordan kampen går, i stedet for en liste af elementer i rummet.

Knapperne består af nogle "menu" knapper, som lader dig gå til spil menuer, samt en 'Fight' knap og en 'Flee' knap. 'Fight' knappen lader spilleren kæmpe mod fjenden, mens 'Flee' knappen lader spilleren flygte fra kampen og tilbage til rummet som spilleren kom fra.

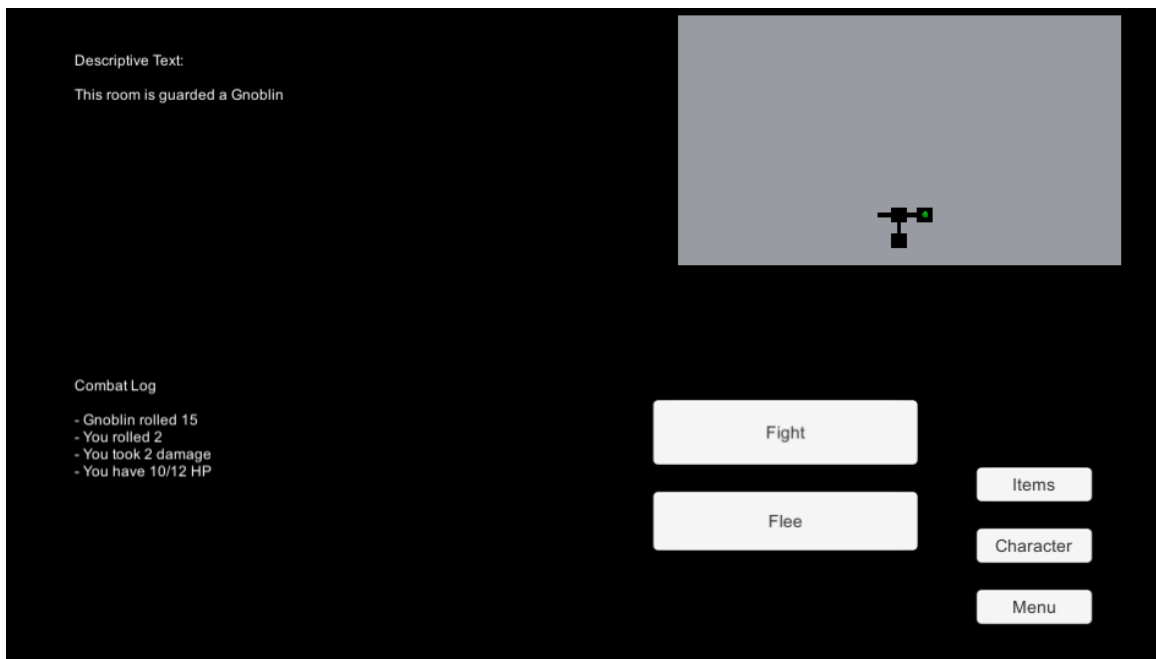


Figure 29: Combat view. Spilleren præsenteres for en fjende, og får information om hvordan kampen mod fjenden går. Der er knapper til at kæmpe og flygte, samt gå til menuer. Øverst til højre er kortet over banen, ligesom i Room vinduet Figure 28.

### 8.2.3 Login

Når spillet startes bedes spilleren prompte at logge ind på deres profil. Dette sker i login vinduet. Spilleren kan indtaste sit brugernavn og kodeord i de to tekstfelter 'Username' og 'Password'. Knappen login fører dem videre til spillet, hvis det indtastede login er korrekt.

Knappen create user fører spilleren til et vindue som ligner login vinduet, og som lader spilleren oprette en ny bruger.

Exit lukker spillet.

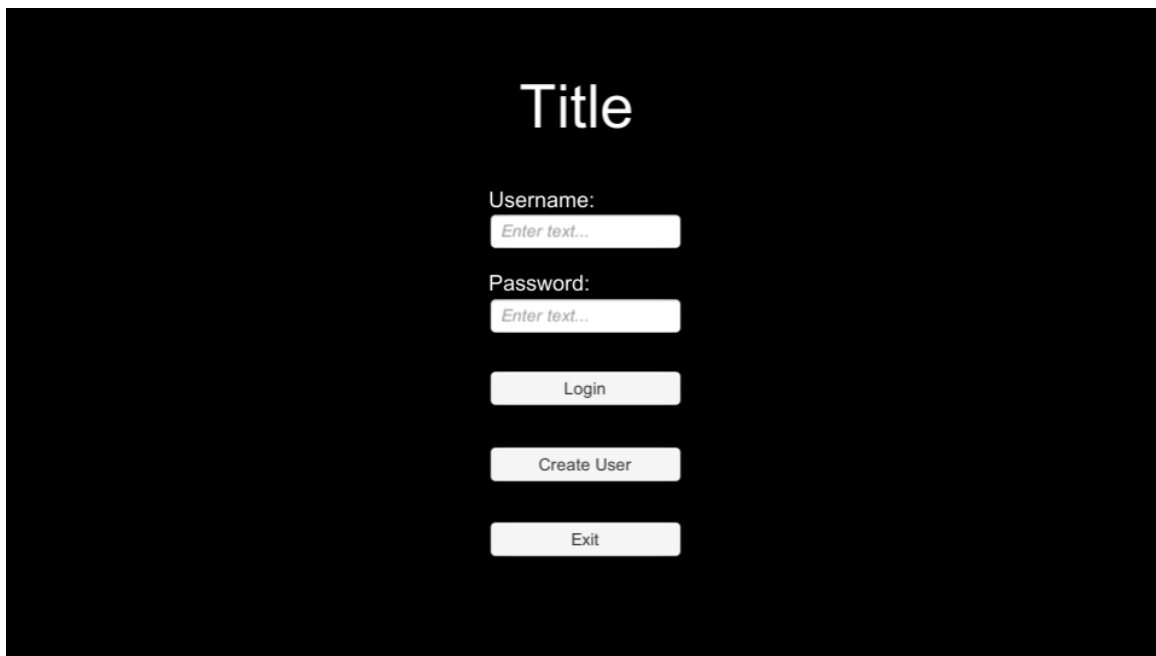


Figure 30: Login view. Bruger kan indtaste sit brugernavn og kodeord for at få adgang til spillet, eller trykke på create user for at komme til et vindue hvor man kan oprette en ny bruger. Det er også muligt at forlade spillet igen.

#### 8.2.4 Settings

Settings vinduet tillader at spilleren kan ændre indstillinger for spillet, og kan tilgås fra hovedmenuen, samt fra selve spillet.

Det er her muligt at ændre f.eks. skærmopløsning og lydstyrke. Det er muligt at gemme de indstillinger som er valgt, forlade menuen uden at gemme de valgte indstillinger, samt gendanne standard indstillingerne for spillet.

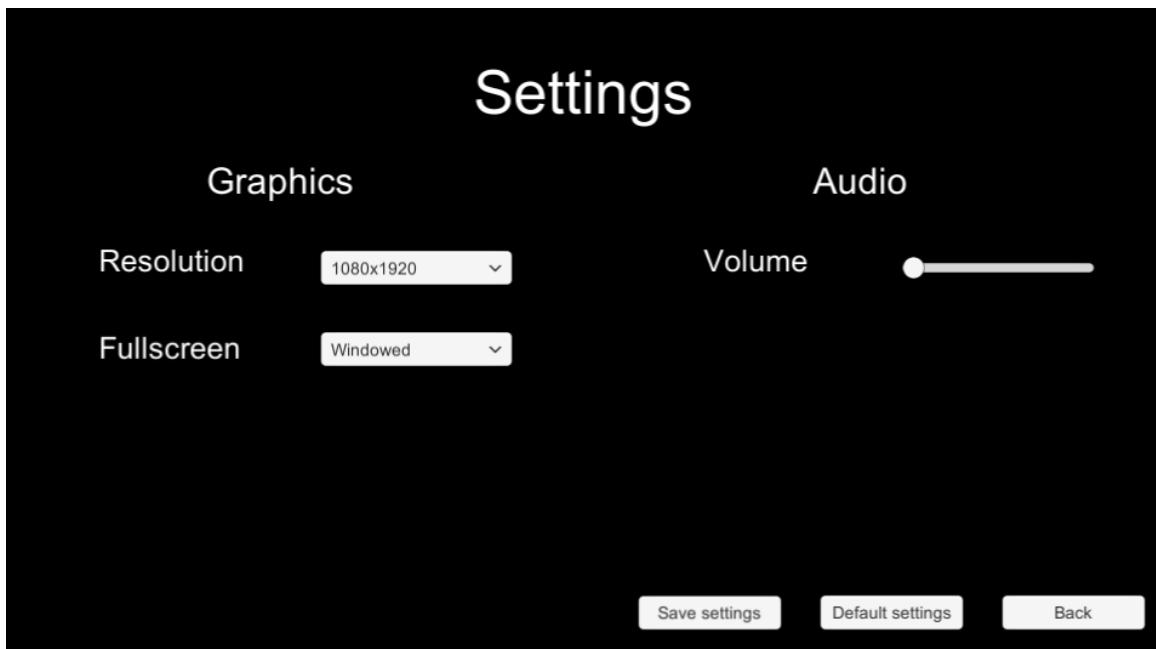


Figure 31: Settings view. Tillader at man kan ændre indstillinger for spillet, så som skærmopløsning og lydstyrke. Det er muligt at gemme indstillingerne, forlade skærmen uden at gemme indstillingerne, samt sætte spillet tilbage til standard indstillinger.



## 8.3 Game Controller Design

Game Controllerens endelige design følger arkitekturen tæt og tager udgangspunkt i de funktioner som skal kaldes i modulet Front-end. I dette afsnit fokuseres der på de features i spillet som gør spillet funktionelt og giver brugeren et reelt gameplay.

### 8.3.1 Map Creation

Når bruger vælger at starte et spil eller loadet et spil fra Front-end siden, skal Game Controlleren indlæse et map så Front End kan fremvise det for bruger. Det ønskes at dette sker ved at Game Controlleren indlæser et tekstdokument(.txt-fil), hvor rummet er repræsenteret med de forskellige veje til et nyt rum. Herfra skal der hentes beskrivelser af de forskellige rum fra databasen via Back-End modulet. Derved når der skiftes rum i Front-end modulet, så sendes beskrivelsen til Front-End modulet som derefter kan vise beskrivelsen for det specifikke rum til brugeren.

### 8.3.2 Player class og Items

Heri skal der implementeres stats til spilleren. I vores implementation kommer der to færdigheder som kan have en indflydelse på spillerens forløb under combat. De to færdigheder er HealthPoints og ArmorClass. HealthPoints skal udgøre hvor meget skade spilleren kan tage inden død. ArmorClass skal udgøre hvor højt fjenden skal slå for at give skade til spilleren. Heri skal implementeres items som kan forøge spillerens chancer for at gennemføre de forskellige fjender. Heri vil der være 'shields' som forøger variabelen ArmorClass for spilleren, så fjenden skal slå højere for at give skade til spilleren. Den anden type item er våben som er med til at øge spillerens 'terning-rul' for at slå højere end fjendens ArmorClass og skade fjenden. Combat-fasen bliver uddybet yderligere i understående afsnittet subsection 8.3.4.

### 8.3.3 Enemies

Fjender skal implementeres på samme måde som spilleren. Deres færdigheder såsom HealthPoints, ArmorClass og Attack(våben)skal dog være forudbestemt. Fjenderne er fordelt over mappet og skal være gradvist stærkere jo længere spilleren kommer i spillet. Dette skal gøres ved at forøge deres færdigheder. Disse fjenders færdigheder og navne skal hentes fra GameControllern af Front-End, når spillet startes af brugeren. Når bruger så går ind i et givet rum, hvor en fjende er til stede, går spillet ind i combat state. Når fjender er blevet bekæmpet, lagres det i databasen via back-end modulet når bruger vælger at gemme spillet. Når bruger dernæst åbner det gemte spil, er fjenden bekæmpet og fjernet fra spillet.

### 8.3.4 Combat

Combat skal initieres når spiller går ind i et rum hvor der er en fjende til stede. Spillet vil derefter gå ind i et combat state hvor spilleren får mulighed for at angribe fjenden eller flygte(flee) og gå tilbage til det forrige rum. Nogle af disse combat interaktioner skal være obligatoriske, så spilleren er nødsaget til at færdiggøre combat for at komme videre i spillet. Når der angribes vil der være implementeret to terninger der bestemmer om spilleren rammer og hvor meget spilleren skader fjenden. For fjenden vil der blive implementeret det samme. Spillerens færdigheder vil have en indflydelse på kampens fremgang(Se afsnit subsection 8.3.2). Herefter vil der først blive tjekket om fjenden har mere HealthPoints tilbage og efterfølgende det samme for spilleren. Der vil være to outcomes for combat state. Enten bekæmpes fjenden af spilleren eller vice versa. Når spilleren bekæmper fjenden, vil rummet som spilleren gik ind i, blive tilgængeligt for spilleren. Hvis spilleren bliver bekæmpet af fjenden, vil spillet gå til end-screen, hvor bruger får mulighed for at lukke spillet eller gå til hovedmenu.

### 8.3.5 Saves

Når bruger vil stoppe spillet, vil der blive implementeret en funktion for brugeren at gemme sine fremskridt i spillet. Brugeren kan dermed gemme spillet i menuen som lagre spillets fremskridt i databasen via Back-End Modulet. Herunder vil dataen der bliver sendt, være rum der er blevet besøgt, fjender der er bekæmpet og genstande der er blevet samlet op og udrustet. Når bruger vil åbne det save igen, vil dataen blive sendt tilbage så bruger kan fortsætte med samme fremskridt.

## 8.4 Backend Design

I det følgende afsnit beskrives design overvejelser i forbindelse med udviklingen af systemets backend Web Api. Afsnittet vil give et overblik over hvilke resources/routes Web api'et stiller til rådighed. Efterfølgende forklares hvordan Authentication og Authorization vil blive håndteret, samt hvordan passwords vil blive hashed. Hertil vil en BackEndController klasse, som skal bruges på client siden blive redegjort for. Tilslut præsenteres applikationsmodeller over controller klasserne for de relevante User Stories, som består af en række sekvens og klasse diagrammer.

### 8.4.1 Analyse konklusion

I dette projekt vil give bedst mening at anvende mulighed 1 baggrunden for denne beslutning kan ses i subsection 6.3 med en backend bestående af ASP.NET og EF Core. Her fås nemlig muligheden for at separere data ressourcerne fra resten af applikationen samtidig med vi for en mere sikker og pålidelig håndtering af brugere samtidig med EF Core stadig understøttes til kommunikation med databasen.

### 8.4.2 Routes

Applikationens nødvendige routes kan inddeles i to under grupper "Save", som indeholder routes relateret til game state og "User", som indeholder routes relateret til brugeren. Alle routes som omhandler game state skal authorize, mens routes, som omhandler brugeren tillader anonyme forespørgelser. Nedenfor er givet en oversigt over de forskellige routes med en tilhørende beskrivelse.

#### Save:

- GET: /api/Save Denne route henter et specifikt gemt game state fra databasen, for den bruger som er logget ind.
- POST: /api/Save  
Denne route sender et specifikt game state til databasen, for den bruger som er logget ind.
- GET: /api/Save/Get List Of Saves  
Denne route henter en liste af game states, for den bruger som er logget ind.
- GET: /api/Save/Get Room Description  
Denne route henter en beskrivelse af det valgt rum i spillet.

#### User:

- POST: /api/User/Register  
Denne route registrerer en ny bruger ved at gemme oplysninger om denne i databasen, og returnerer en JWT token. Når en bruger bliver registreret får den tildelt 5 pladser i databasen til game states.

- POST: /api/User/Login

Denne route logger en bruger ind ved at tjekke bruger oplysninger med dem, som er registreret i databasen, denne route returnerer også en JWT token.

#### 8.4.3 Authentication/Authorization med JWT Token

JWT (Jason Web Token) er et JSON object, som bruges til at validere (Authenticate), og til at afgøre hvorvidt en client har adgang til den givne data (Authorizing). En token består af tre dele en header, en payload og en signature.

##### Header:

Headeren består af en type og en hashing algoritme, typen vil for dette system være "JWT" og algoritmen vil være "HmacSha256".

##### Payload:

Payloaden indeholder såkaldte claims, der findes tre typer af claims reserved, public og private. Reserved er en række prædefineret claims som kan bruges til eksempelvis at sætte en tidsbegrænsning på den gældende. Public og private claims kan indeholde noget information omkring brugeren. I dette system benyttes private claims til at indeholde brugerens username, som så kan bruges til at genkende hvem den aktuelle token tilhører ved en forespørgsel.

##### Signature:

Signature bruges til at verificere hvem som er afsender af JWT'en, og til sikre at den ikke er blevet ændret undervejs. Her samles header, payload og en "secret" som er en hemmelig string af tegn, kun Web Api'et kender. For at generere denne token implementeres en funktion, som opretter en token efter de forhold som er beskrevet ovenfor.

#### 8.4.4 Hashing

Under design fasen blev det besluttet at anvende hashing til at gemme en brugers password i databasen, for at øge sikkerheden. Sikkerhed var ellers ikke prioriteret i vores MoSCoW analyse subsection 5.3, for denne udgave af spillet. Hashing af passwords går ud på at kryptere det gemte password i databasen. Det skal krypteres således at det ikke er muligt at dekryptere, samtidig med at det stadig kan valideres om det er et korrekt password, der modtages af en client. Når et password hashes anvendes et såkaldt "salt", som er en tilfældigt genereret string kobineret med password'et. Til at hashe med anvendes biblioteket Bcrypt [**Bcrypt**]

#### 8.4.5 BackendController på client siden

Til brug af clienten for at tilgå backenden, skal der bruges en BackendController klasse, dennes ansvar vil være at udføre HTTP request/responses, og vil indeholde en funktion for hvert endpoint.

Klassen vil gøre brug af en HttpClient som er en klasse .Net stiller til rådighed til at håndtere HTTP request/responses. Da backenden kræver en JWT for at få adgang til endpoints, hvad angår loading og saving af game state, skal clienten også gemme JWT for den bruger som er logget ind, således at den kan blive sendt med de forskellige requests.

## 8.5 Applikationsmodeller

I dette afsnit gennemgås applikationsmodeller for de user stories som vedrører backend'en, der udarbejdes sekvensdiagrammer for at beskrive funktionalitet og klasse diagrammer til at beskrive indeholdet og sammen spillet imellem klasserne. Applikationsmodellerne deles op efter controllere, en for UserControllereren og en for SaveControlleren. Fokuset vil her ligge på controller klasserne, for en beskrivelse af DAL komponenterne henvises til database design afsnit subsection 8.7.

### 8.5.1 Applikationsmodel User Controller

#### User Story 1: Log in

Når clienten udfører en HTTP request med URL'en : " https://localhost:7046/api/User/Login", udføres følgende aktioner som fremgår af sekvensdiagrammet på Figure 32

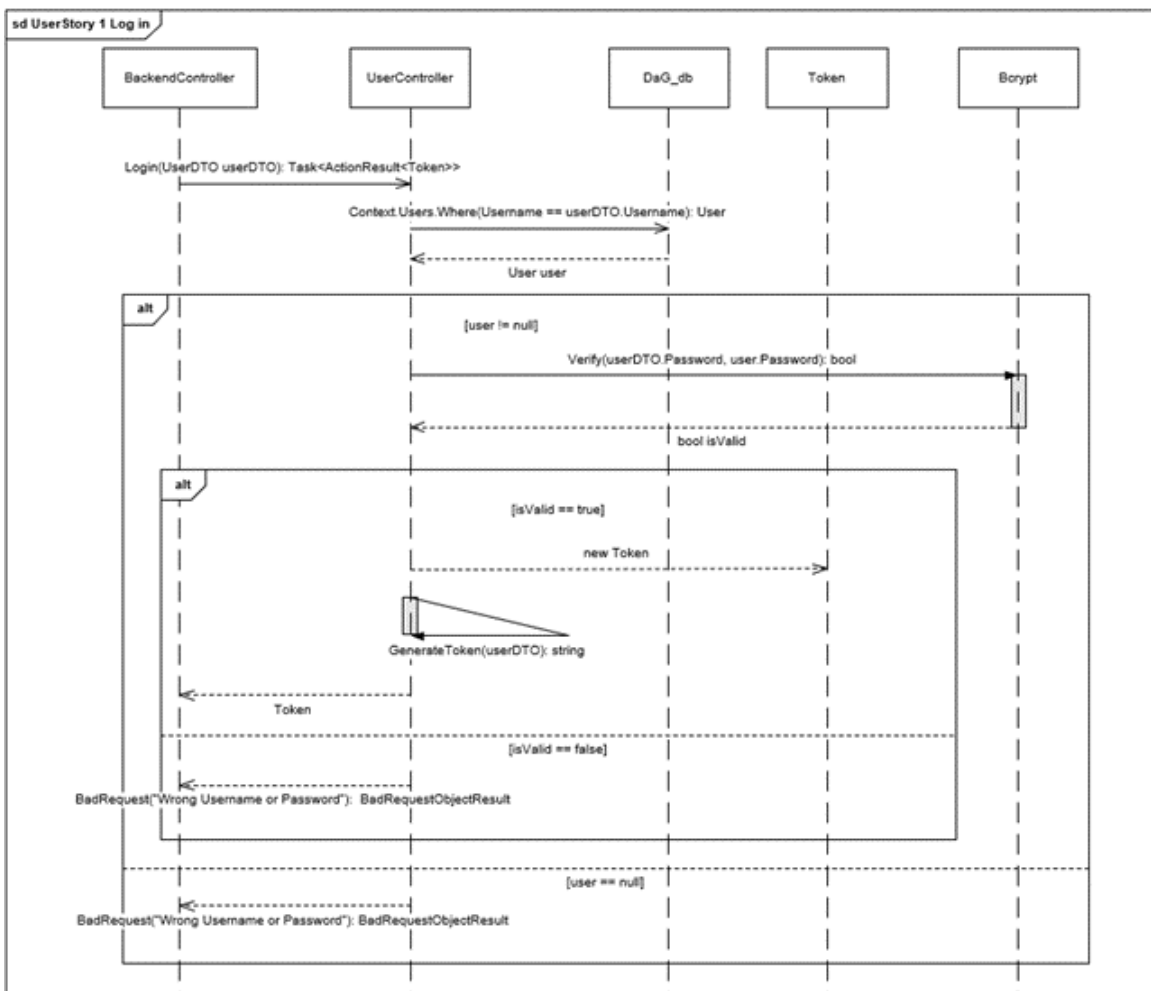


Figure 32: Sekvensdiagram som viser funktionaliteten i User Story 1 Log in

Authentication controlleren fra C3 modellen hedder her "UserController" og har til opgave at gøre det muligt for en bruger at registrere sig og logge ind. Login implementeres i Login() funktionen, som

er det endpoint clienten kalder for at logge ind. Her bliver der tjekket om det indtastede brugernavn findes i databasen. Hvis brugernavnet findes, så bliver adgangskoden tjekket med funktionen verify fra Bcrypt klassen, og passer den, bliver der returneret en JWT token. Hvis brugernavnet derimod allerede eksisterer eller adgangskoden ikke er korrekt returneres en fejlmeddelelse.

### User Story 2: Opret Bruger

Når clienten udfører en HTTP request med URL'en : " https://localhost:7046/api/User/Register", udføres følgende aktioner som fremgår af sekvensdiagrammet på Figure 33

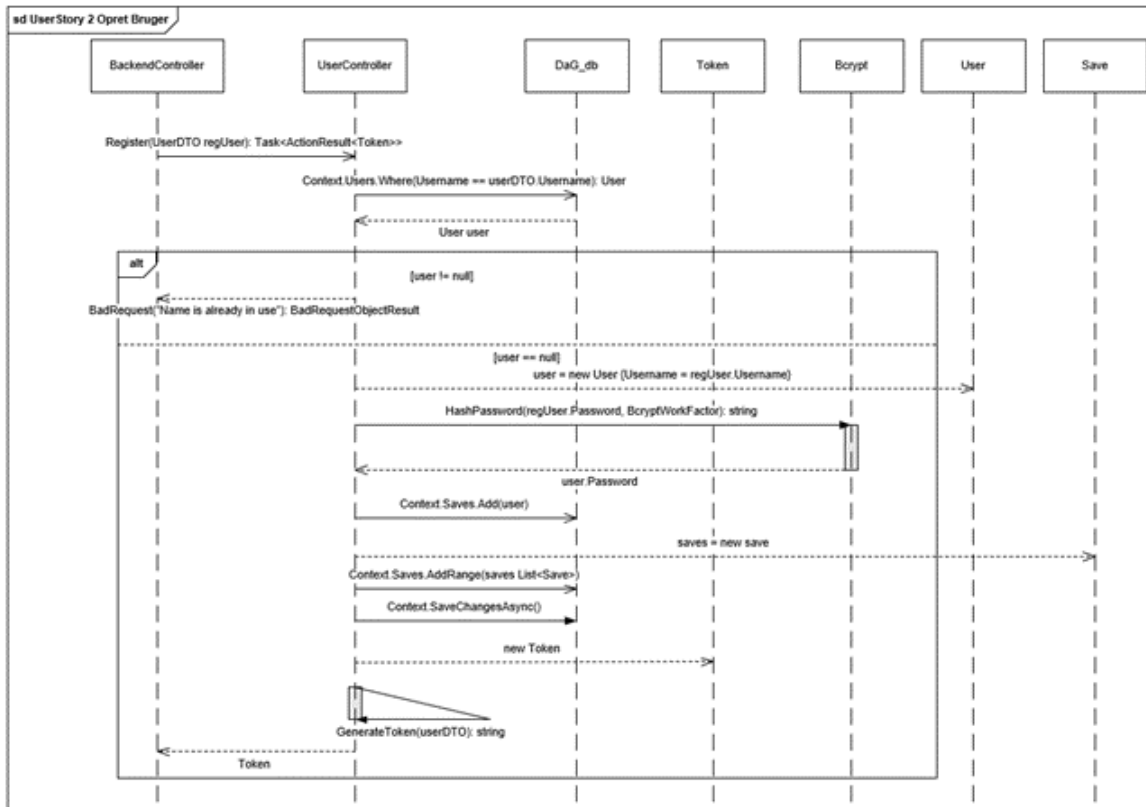


Figure 33: Sekvensdiagram som viser funktionaliteten i User Story 2 Opret Bruger

For at kunne logge ind skal det være muligt at oprette en bruger til dette laves en Register(UserDTO regUser) funktion. Denne starter med at checke om brugernavnet er optaget. Er den det, returneres der en fejlmeddelelse, og ellers registreres brugeren. Dernæst bliver det valgte kodeord "hashed", således at der opnås en vis sikkerhed i systemet. Når der oprettes en ny bruger skal denne have tildelt 5 pladser i databasen til sine gemte spil, dette sker ved at der oprettes en liste af saves der indeholder 5 nye saves, disse gemmes med contexten i SQL databasen. Hvis alt går godt returneres en JWT token.

### Samlet Klasse diagram over User Story 1 og 2

På Figure 34 kan ses en oversigt over de klasser, metoder og attributter, som anvendes i udførelsen af User story 1 og 2. Hertil anvendes også klassen Bcrypt som ikke er taget med, da den blot er brugt udefra, detaljer om Bcrypt kan ses i afsnittet ovenover subsection 8.4.4.

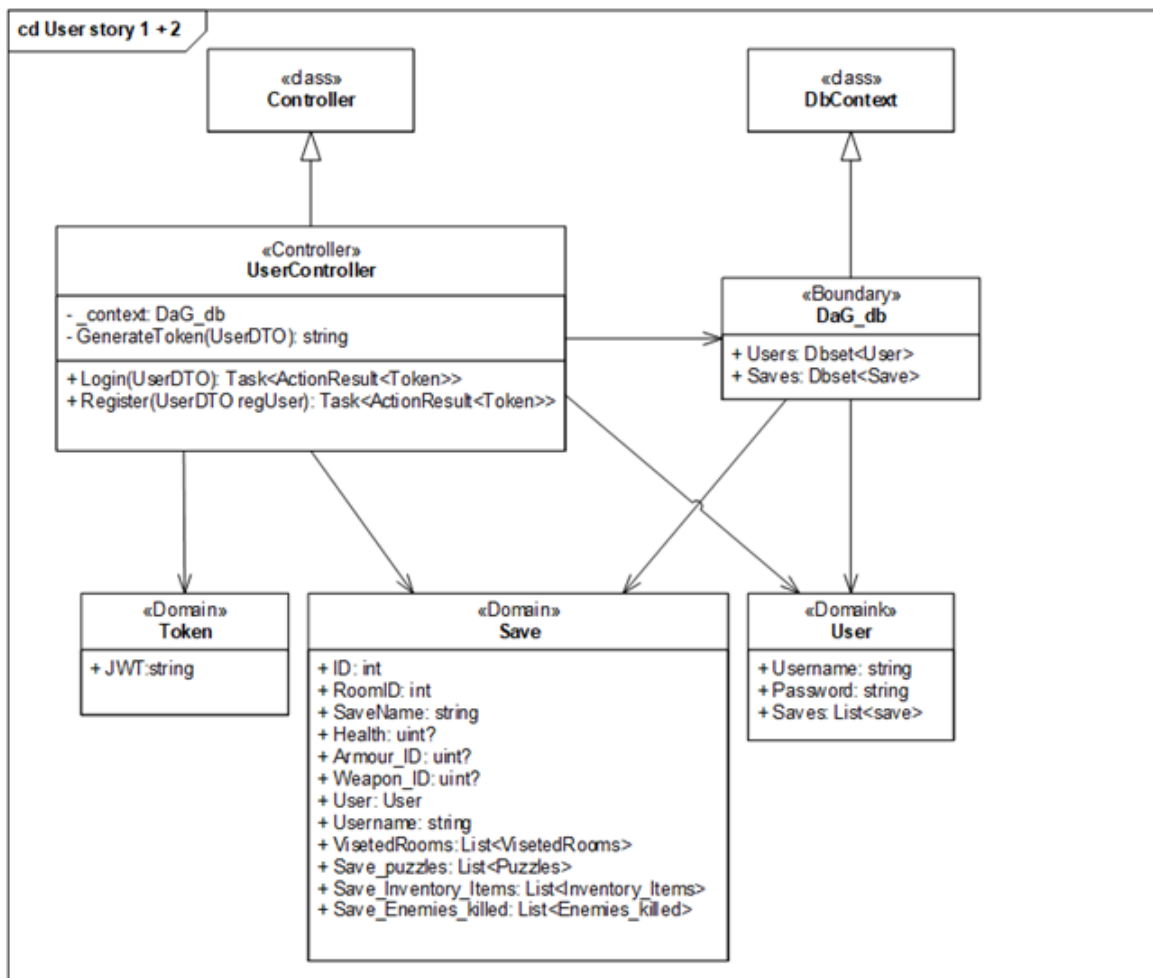


Figure 34: Samlet klasse diagram over User story 1 og 2

### 8.5.2 Applikationsmodel Save Controller

#### User Story 15: Save Menu - Save Game

Load/save controlleren fra C3 modellen er blevet til en enkelt controller, som kaldes for "SaveController". Denne controller har til ansvar at gemme et spil, men samtidig også være i stand til at kunne hente gemte spil, altså at load dem. Når clienten udfører en HTTP request med URL'en : "https://localhost:7046/api/Save", udføres følgende aktioner som fremgår af sekvensdiagrammet på Figure 35

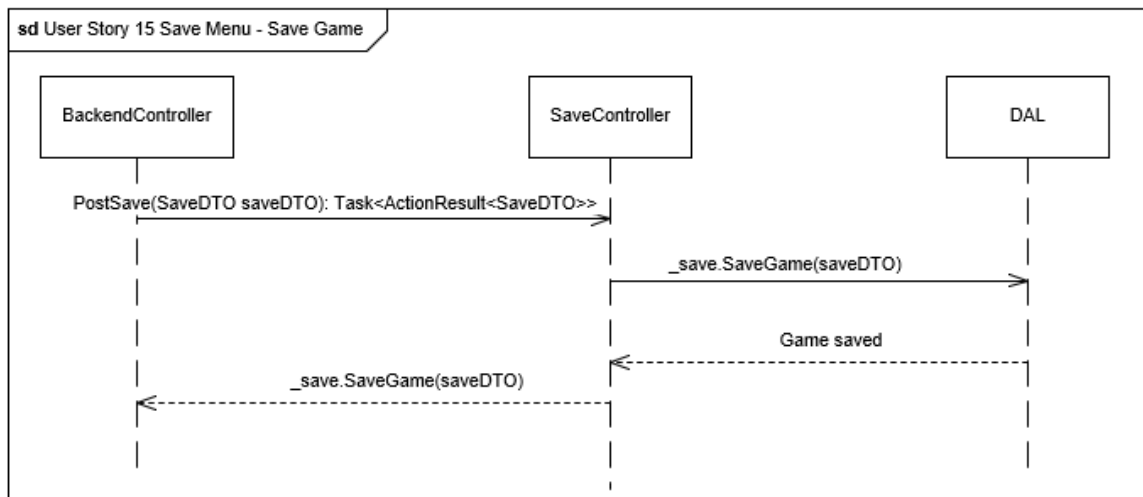


Figure 35: sekvensdiagram over User Story 15 Save and Exit

Der bruges en `HttpPost` funktion kaldet `PostSave` som gemmer et nyt spil i databasen ved at ligge det modtagne gamestate fra parameter listen ind i et funktionskald `SaveGame()` i DAL klassen, som så sørger for at gemme spillet i databasen på den korrekte måde.

#### User Story 17: Main Menu - Load Game

Når clienten udfører en HTTP request med URL'en : " `https://localhost:7046/api/Save/Get List Of Saves`", udføres følgende aktioner som fremgår af sekvensdiagrammet på Figure 36

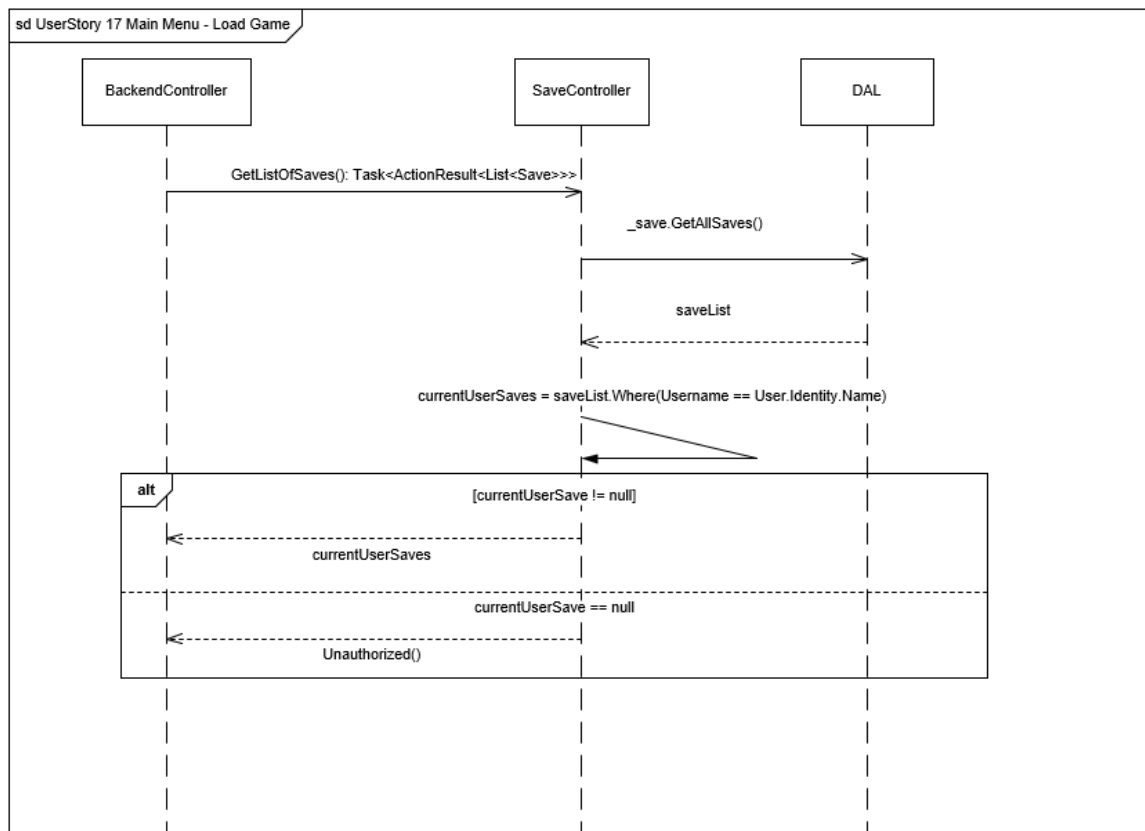


Figure 36: Sekvensdiagram over User Story 17 Main Menu - Load Game

Da vi gerne vil gøre det muligt at have 5 save games pr. profil, skal det naturligvis være muligt at kunne vælge mellem de forskellige saves, til dette laves funktion `GetListOfSaves()`. Clienten kalder her `GetListOfSaves()` som er et `HttpGet` endpoint, denne funktion henter alle saves i DAL klassen ved at kalde `GetAllSaves()`. `SaveController` finder så de gemte spil i listen hvor brugernavnet passer med den bruger, som er logget ind ved brug af `User.Identity` som stilles til rådighed af `ControllerBase` klassen, her i kan claims for den nuværende JWT nemlig findes.

### UserStory 18 : Main Menu - Load Game - Load

Når clienten udfører en HTTP request med URL'en :” `https://localhost:7046/api/Save?id=id`”, udføres følgende aktioner som fremgår af sekvensdiagrammet på Figure 37



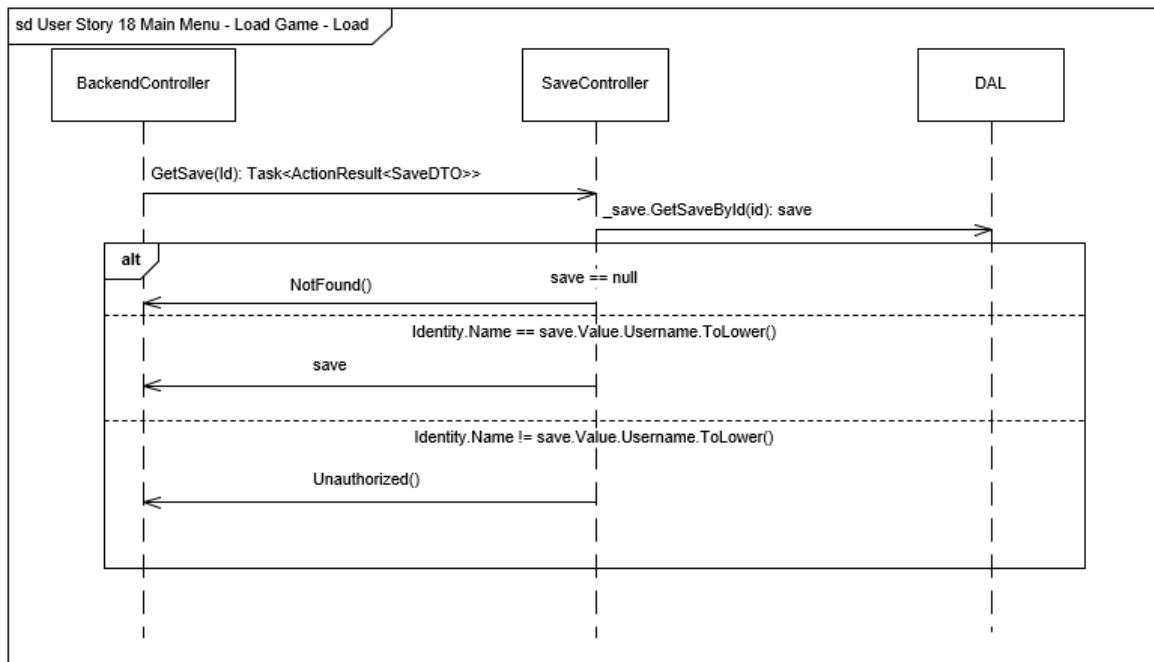


Figure 37: Sekvens diagram over User story 18 Main Menu - Load Game - Load

**Samlet klasse diagram over User story 15, 17 og 18**

Et Samlet klasse diagram for User story 15, 17 og 18 kan ses på Figure 38

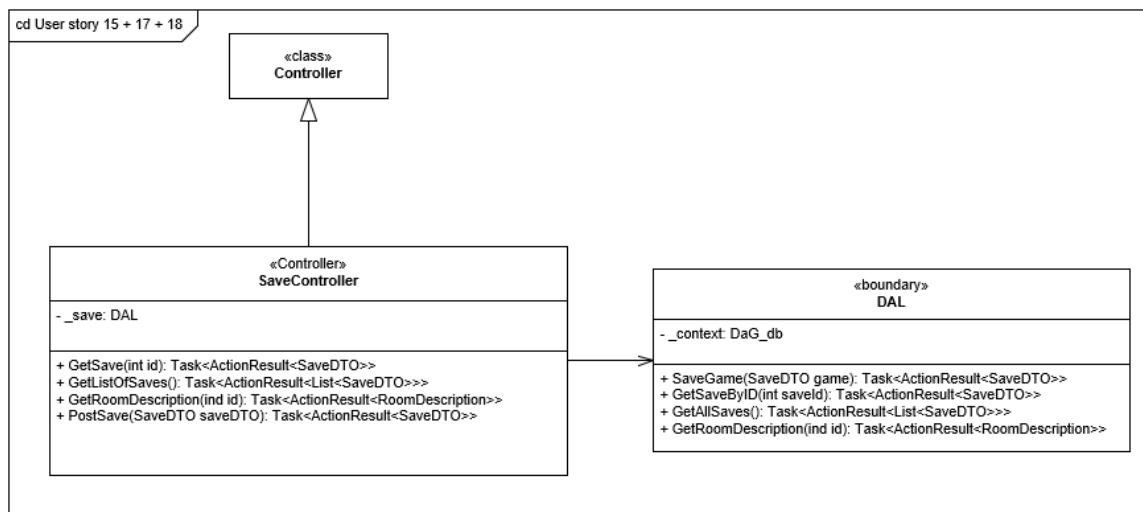


Figure 38: Samlet klasse diagram for User story 15, 17 og 18

Udover de funktioner som bliver brugt i User stories, tilføjes der er en funktion GetRoomDescription, som blot henter en beskrivelse af det rum brugeren befinder sig i.

### 8.5.3 Konklusion

Backendten er blevet designet således at den kan håndtere alle de nødvendige kald for at opfylde user stories omkring brugere og gemte spil, dette er beskrevet igennem appilationsmodeller. JWT tokens anvendes til Authentication og Authorization. Til hashing af passwords anvendes Bcrypt. Hertil er BackendControlleren klassen blevet introduceret på clienten, som står for HTTP request/response.

## 8.6 Software Design DAL Design

På Figure 39 herunder ses klassediagrammet over backend DAL, som benyttes til database access. Som det kan ses på diagrammet, indeholder DAL en database context, som benyttes til at forbinde backend applicationen til databasen. Derudover består DAL af fire overordnede funktioner, hvoraf de 3 tilhører en user story. Den sidste funktion benyttes når vi starter spillet, til at lave rumbeskrivelser.

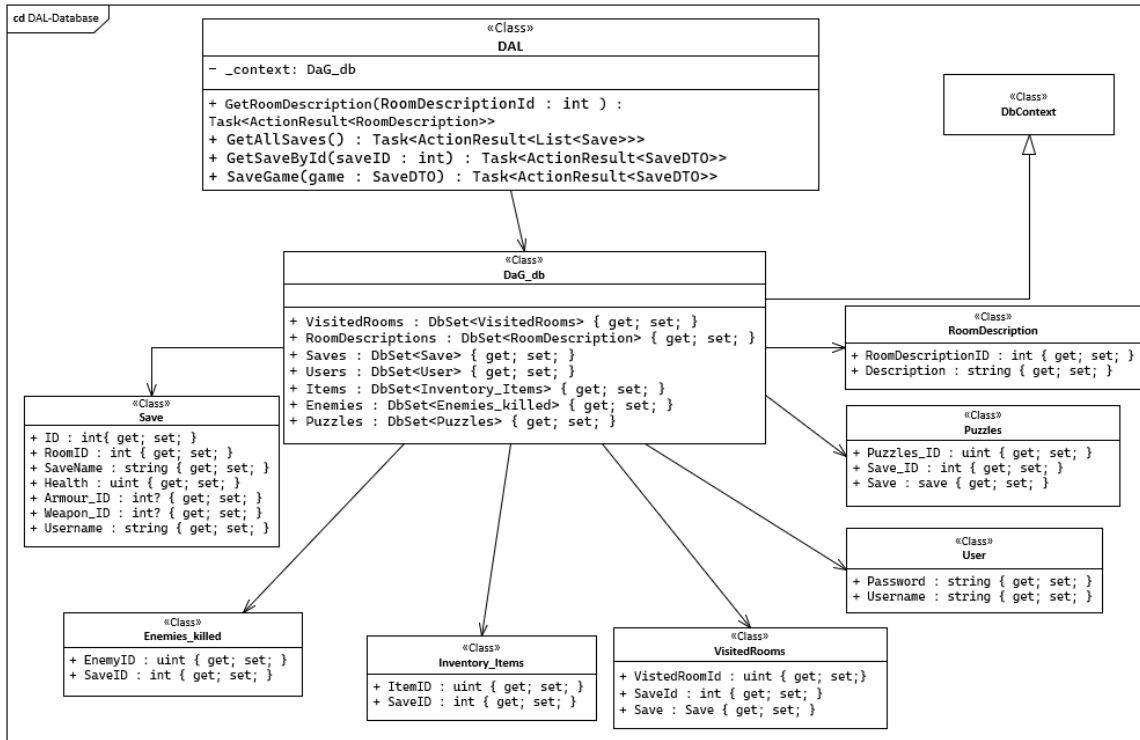


Figure 39: Samlet klasse diagram for User story 8, 15, 17 og 18 med DAL db context og entitetsskasser. For læseligheden er DAL klassen ikke forbundet til forskellige entitetsskasser selvom disse benyttes.

Følgende funktion som beskrives på Figure 40 benyttes når spil klienten startes, da beskrivelser af rum ikke ændre sig gennem spillets levetid, i den nuværende iteration.

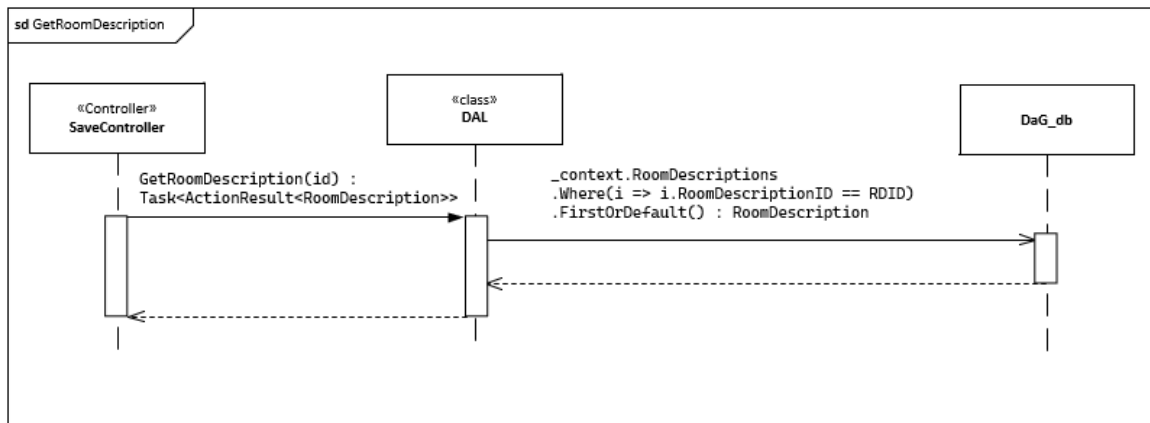


Figure 40: Sekvensdiagram om læsning af en rumbeskrivelse

Navn: GetRoomDescription

Parametre: int RoomDescriptionId

Returtype: Task<ActionResult<RoomDescription>>

Beskrivelse: Denne funktion finder og returnerer en beskrivelse for det valgte RoomDescriptionID. Dette kan også ses på sekvensdiagrammet på figur xxx herunder.

### 8.6.1 User story funktioner

De følgende 3 funktioner benyttes til udførelse af forskellige user stories. Det drejer sig om:

- User story 15 – Save game
- User story 8 - Save - No Combat + User story 17 – Load game list
- User story 18 – Load game

Den første funktion SaveGame benyttes til at udføre user story 15. Her skal der gemmes et spil. Da vi har et krav om kun at holde 5 gemte spil pr. bruger, vælger vi at oprette 5 "tomme" saves, når vi opretter brugeren. Når brugeren så ønsker at gemme et spil, skal vi ikke tilføje et nyt, men istedet overskrive et valgt gammelt save. Dette kan også ses på sekvensdiagrammet på figur Figure 41

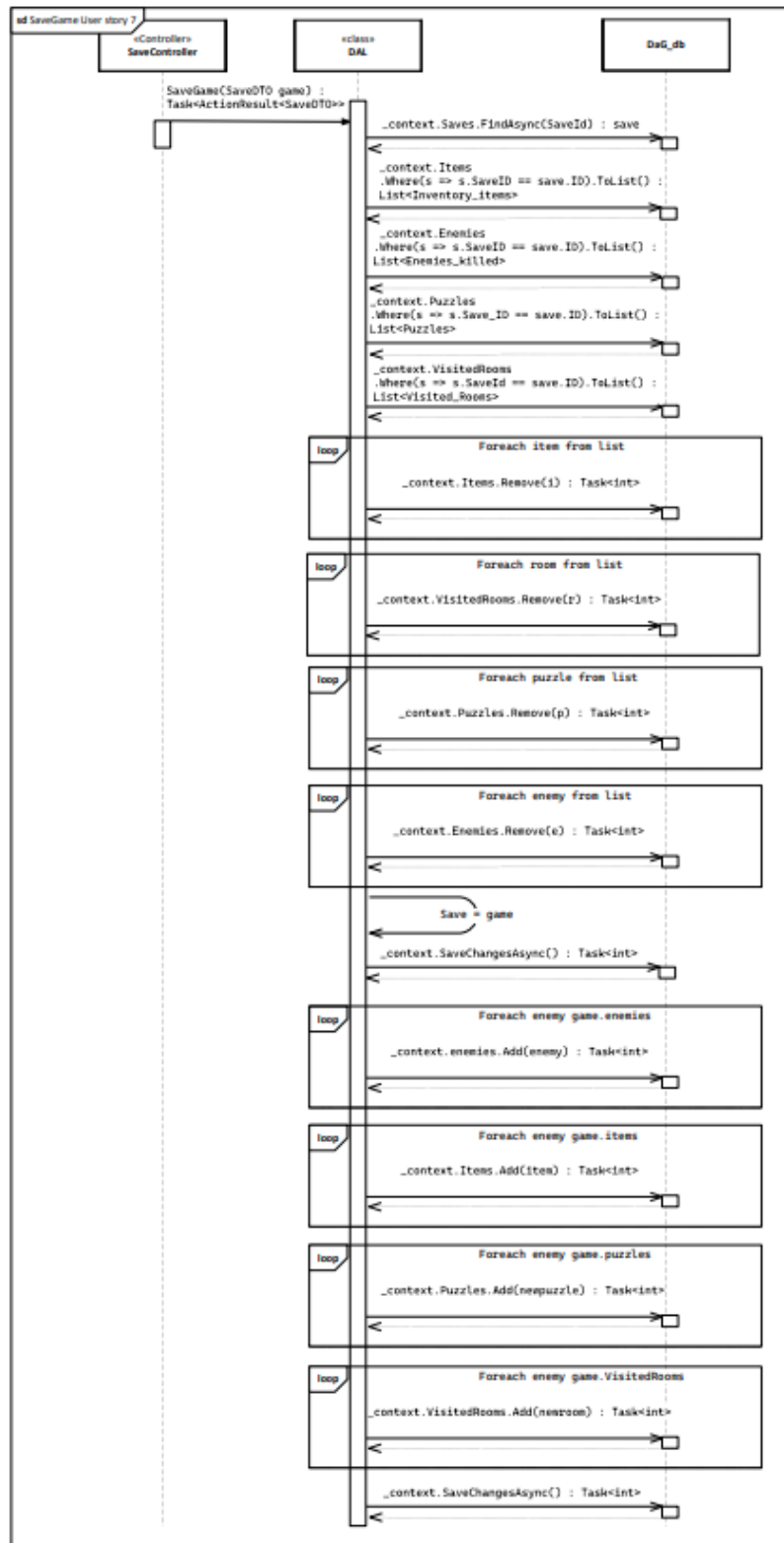


Figure 41: Sekvensdiagram for user story 15 save game som beskriver queries til databasen for at gemme et spil

Navn: SaveGame

Parametre: SaveDTO

Returtype: Task<ActionResult<SaveDTO>>

Beskrivelse: Da der i vores frontend sørges for at en bruger blot kan have 5 saves, starter vi med at finde det save vi gerne vil overskrive. Det gamle save, samt tilhørende lister slettes, hvorefter det nye save gemmes og eventuelle nye lister til fx. items gemmes.

Den anden funktion GetAllSaves benyttes til at udføre user story 8 og 17. Her skal der loades en liste af gemte spil, når brugeren ønsker at se sine gemte spil. Dette kan også ses på sekvensdiagrammet på Figure 42 .

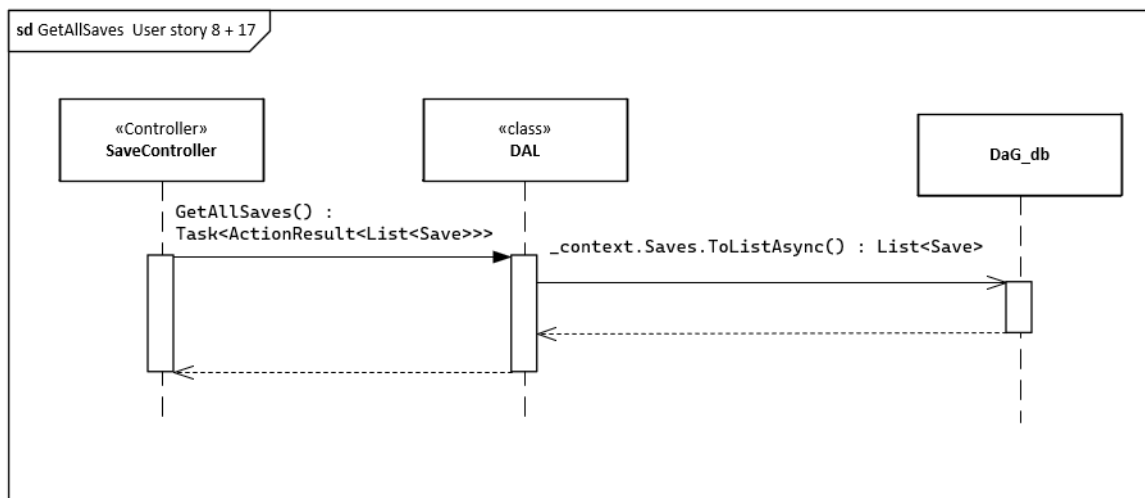


Figure 42: Sekvensdiagram for user story 8 og 17 GetAllSaves som beskriver queries til databasen for at hente alle saves

Navn: GetAllSaves

Parametre: ingen

Returtype: Task<ActionResult<List<Save>>>

Beskrivelse: Her hentes all gemte saves i spillet.

Den tredje funktion GetById benyttes til at udfører user story 18. Her skal der loades et gemt spil, som brugeren nu ønsker at spille. Dette kan også ses på sekvensdiagrammet på Figure 43.

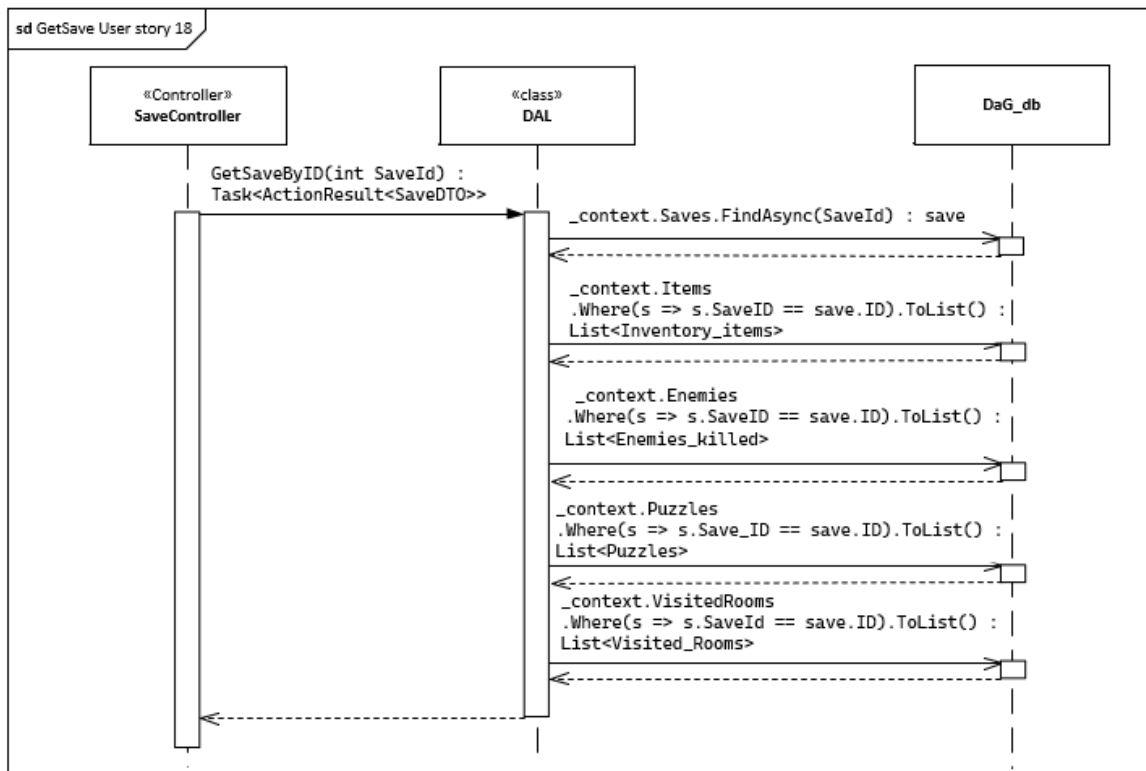


Figure 43: Sekvensdiagram for user story 18 GetSaveById som beskriver queries til databasen for at hente et specifikt save og dens tilhørende information

Navn: GetSaveById

Parametre: int saveID

Returtype: Task<ActionResult<SaveDTO>>

Beskrivelse: Denne funktion finder det save med det medsendte ID, samt tilhørende lister, indsætter værdier i et SaveDTO objekt, hvorefter det returneres.

## 8.7 Database Design

Projektets database har gruppen valgt at hoste i lokal storage. Dette er valgt da der under semesterets forløbet opstod problemer med skolens licens af Microsoft produkter. For at undgå at komme ud for udfordringer med hosting senere i forløbet, blev der valgt at gå med den sikre løsning, at hoste databasen lokalt på enheden. Til dette benyttede gruppen et docker image [**SQL-server-with-docker**], specifikt det samme image som blev benyttet i DAB undervisning, til vores SQL server. Hvis ikke der var problemer microsoft, havde gruppen i stedet valgt at lagre data på en cloud-based storage fremfor lokal storage.

For at kunne udarbejde et ER diagram til modellering af vores sql database skal vi start med at finde ud af hvilke krav vi har til og hvilke attributter vi ønsker at gemme i vores database. Først og fremmest ønskede gruppen at vi kunne gemme beskrivelserne af de forskellige rum, i spillets layout, for at formindske antallet af filer i klienten, og samtidigt gøre eventuelle senere tilføjelser nemmere. Her benyttes rummets id som key, da vi ikke ønsker at man skal kunne oprette flere beskrivelser til samme rum. Diagrammet kan ses på Figure 44.

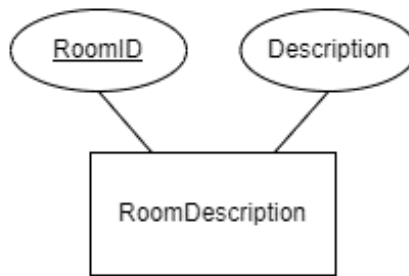


Figure 44: ER diagram for Roomdescription. En beskrivelse består blot af en beskrivende string samt det tilhørende unikke rumid.

Her efter kommer kravene til at kunne gemme et spil for en bruger. Her ønskede vi at man kunne stå et vilkårligt sted i spillet, med undtagelse af en combat, og gemme spillet. Det skulle derefter være muligt for spilleren at loade spillet igen, hvorefter spillet er i samme stadie som man gemte det i. ER diagrammet for at gemme et spil til en bruger på Figure 45.



Figure 45: ER Diagram til at gemme et spil til en specifik bruger. Her ses de forskellige informationer som skal til for at kunne gemme et helt spil

Først og fremmest ønskede gruppen et bruger system, så eventuelle gemte spil kun tilhørte en bruger. Der gemmes derfor en bruger entitet med et unikt brugernavn og et tilhørende password. Sikkerhed på password og versalfølsomhed på brugernavnet håndteres af spillets backend.

En spiller skal derefter kunne gemme 5 unikke spil med forskellige oplysninger. Restriktionen med max 5 forskellige spil pr. Bruger, håndteres ved at oprette 5 "tomme" gemte spil ved oprettelsen af



en bruger. Et af disse gemte spil overskriver derfor når vi gemmer. Der kan på denne måde ikke oprettes mere en 5 gemte spil pr. bruger.

I et gemt spil ønskede vi at gemme en række forskellige attributter for spilleren. Første og fremmest får hvert spil et unikt id som vi benytter til identifikation og at lave forhold mellem de forskellige tabeller. Et gemt spil får et navn, valgt af brugeren, som gør det nemmere for brugeren at differentiere mellem de forskellige spil.

Dette navn skal være forskelligt fra de 4 andre gemte spil som tilhører samme bruger.

En spillers Health gemmes også, da man kan have taget skade efter en kamp.

Det gemmes også hvilket rum, spilleren står i når spillet gemmes, så vi loader korrekt tilbage. En spiller kan derudover også holde genstande, som armor og våben, i hånden eller i sit inventory. Dette gemmes også henholdsvis som en del af et spil og i en inventory liste tilhørende spillet.

Tabellen med inventory har 2 attributter, et ID, som svarer til en bestemt genstand, og en reference til et SaveID. Denne parring er unik, da man ikke kan holde 2 af den samme genstand.

Tabellerne med Enemies og puzzles fungerer på samme måde. Her har hver enemy og puzzle i spillet et unikt id. Id'et gemmes i kombination med et saveId, som et unikt par, da man ikke kan vinde over samme enemy og løse samme puzzle flere gange.

Til slut ønskede vi at kunne vise spilleren de rum som allerede er blevet besøgt. Derfor gemmes der i path tabellen, for hvert spil, en unik kombination af saveID og besøgte rum id. Denne parring er unik da man blot behøver at besøge et rum en gang, før det er synligt på kortet.

Der er i projektet oprettet klasser tilsvarende ER diagrammerne. Forholdene mellem disse, samt keys, er opsat i DaG-db klassen og er skrevet ved hjælp af fluentAPI. Dette kan ses i Implementationsafsnittet subsection 9.6.

## 9 Implementering

### 9.1 System Implementering

Under implementeringen af nogle funktioner i samtlige moduler, er funktioner blevet lavet til funktioner der følger følgende opstilling:

```
public async Task <T> Foobar();
```

Funktionerne returnerer en Task af typen T og den kan gøres asynkront. Dette er specielt vigtigt når man kontakter databasen, da programmet ikke må køre videre før den asynkrone funktion er færdig med sit database kald. Når en funktion i f.eks. Game Controlleren kalder: `public async Task <SaveDTO> GetSaveAsync(int id)` i backend controlleren, i sin egen funktion `SaveGame()` skal `SaveGame()` også erklæres `async` og returnerer en Task af typen T og derfor er nærmest alle funktioner der kontakter databasen erklæret for `async` kald, som returnerer typen Task.

### 9.2 Frontend Implementering

Det endelige design af vinderne(vinduerne?) følger tæt det oprindelige mockup design. Der benyttes MVVM (Model, View and View Model) designpatterns. Det har undervejs i implementeringen vist sig et behov for væsentlig flere menuer end først antaget, og disse er implementeret efter samme overordnede design, som resten af systemet, så derved følges stilen og følelsen af spillet.

Det endelige design har følgende views: Login, Register, Main, Settings, Ingame, Load, Save, Inventory, Character, Victory, Defeat, Room og Combat.

Til at skifte views uden at oprette et nyt vindue bruges et mediator design, hvor hver knap som skifter view giver en besked til mediatoren. Dette virker fordi alle views er oprettet som WPF user controls, og ikke views, hvilket tillader at et main view kan skifte mellem viewmodels, derved skifter alt indhold på vinduet, men uden at selve rammen skifter. Dette resulterer i et mere flydende User Interface for brugeren og derved en alt i alt bedre oplevelse. [Mediator]

Et eksempel på hvordan mediatoren kaldes ved et tryk på en knap kan ses i følgende kode eksempel:

```
private DelegateCommand _loadGame;

public DelegateCommand LoadGame => _loadGame ?? (
    _loadGame = new DelegateCommand(
        ExecuteLoadCommand, CanExecuteLoadCommand));

async void ExecuteLoadCommand()
{
    await GameController.Instance.LoadGame(SelectedSave.ID);
    Mediator.Notify("GameStart", "");
}

bool CanExecuteLoadCommand()
{
    return SelectedSave != null;
}

void LoadCommand()
{
    LoadGame.RaiseCanExecuteChanged();
}
```

Der er implementeret at nogle af spillets funktioner kan tilgås via key-bindings, hvilket har nødsaget et brud på MVVM-designet. Når mediatoren skifter mellem views bliver fokus ikke sat til indholdet af det nye view, og keybindings virker derfor ikke. For at løse dette sættes top-elementet på den nye side som fokus. Dette kan dog ikke gøres i MVVM, så her er det pattern brudt, da det er nødvendigt at gå ind i code-behind filen for at sætte fokus.

De følgende afsnit viser et udvalg af view, samt en beskrivelse af hvordan de afviger fra det oprindelige design og en beskrivelse af interessante programmerings-tekniske beslutninger.

### 9.2.1 Login view

Den overordnede struktur af login (Figure 46) og register menuerne følger meget tæt det oprindelige design. Alle elementer er blevet stiliseret så de matcher det ønskede look. Dette er gjort ved brug af globale resources og styles i app.xaml filen (**INDSÆT REFERENCE TIL HVOR FILEN KAN FINDES HER**). Dette gør det nemt at oprette nye views og ændre udseende af hele spillet. Selve login håndteres af backenden som kaldes af login og register knapperne via command bindings.

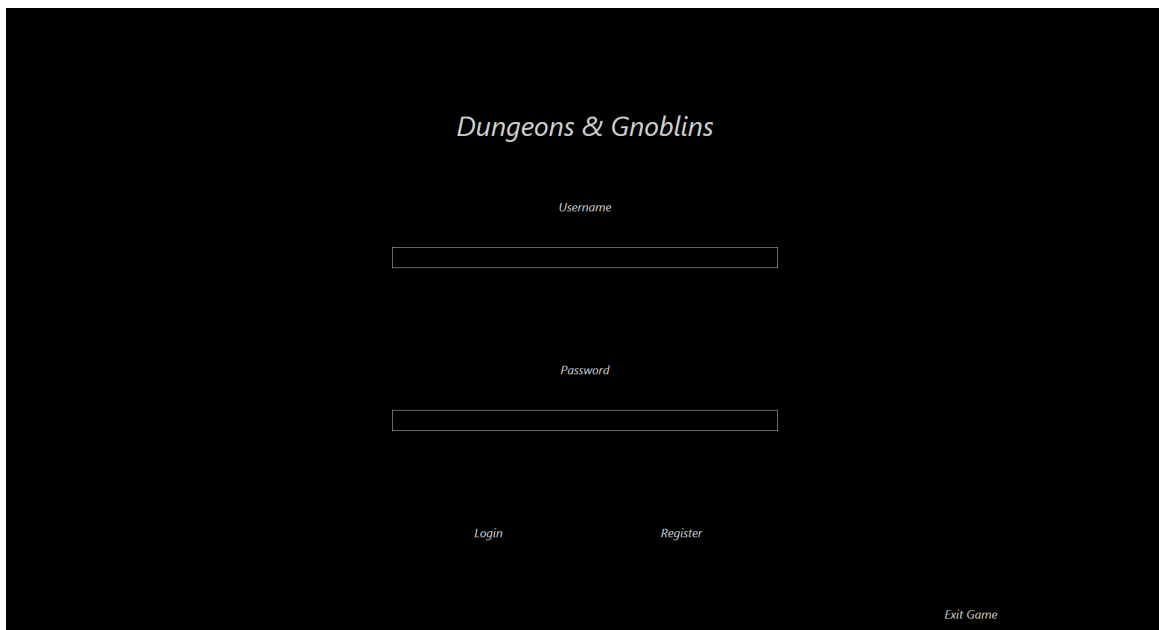


Figure 46: Endelig login skærm. Brugenavn og kodeord kan indtastes i de to felter. Register knappen fører til et nyt view, hvor man kan oprette en bruger, mens login knappen fører spilleren til main menu, hvis deres login er korrekt.

### 9.2.2 Room View

Visuelt er room view (Figure 47) ikke ændret betydeligt fra det oprindelige design. Der er ændret lidt på placering og antal af knapper så det passer til antallet af interaktioner tilgængelig til brugeren. Kortet er lavet så det opdateres når spilleren går ind i et nyt rum, ved at ændre på synligheden af elementerne i kortet. Det er yderligere sat op så det kan skaleres til de skærmopløsninger som understøttes.

Ved at trykke på interact knappen kan spilleren flytte et valgt 'item' fra listen nederst til venstre over i sit inventory (et separat view), som kan tilgås ved at trykke på Inventory knappen.

Alt tekst er vist med data binding.



Figure 47: Endeligt udseende af room view. Generelt er der ikke ændret meget i forhold til det oprindelige design. Kortet er lavet så det skalerer med skærmopløsningen.

### 9.2.3 Combat View

Combat view (Figure 48) er bygget med room view som skabelon, så de fleste elementer er ens. Knapperne antal og funktion er ændret, og sammenlignet med det oprindelige design er knapperne for items og character fjernet, da det blev besluttet at det ikke skulle være muligt at tilgå sit inventory under en kamp. I stedet for en beskrivelse af rummet og en liste af items vises der nu en beskrivelse af hvordan kampen går nederst i venstre side af skærmen. Tal-værdierne hentes fra gameengine, og sættes ind i en tekststreng, som gør det nemt for brugeren at forstå hvordan det skal fortolkes. Der er yderligere tilføjet en healthbar, som giver en visuel indikation af, hvor tæt spilleren er på at tabe spillet og skifter farve fra grøn til gul til rød, som spilleren tager mere skade. Baren giver derfor lidt farve til et ellers meget gråtonet spil.



Figure 48: Combat view er baseret på room view. Her præsenteres spilleren for info om en fjende og hvordan kampen mod fjenden går.

### 9.2.4 Settings view

I settingsmenuen (Figure 49) er det muligt at vælge lydstyrke for musikken i spillet (samt slukke helt for musikken) og vælge mellem tre skærmopløsninger (1280x720, 1920x1080 og 2k). De tre valgmuligheder er valgt til at teksten på skærmen stadig er læselige. Kortet i Room og Combat view skalerer med skærmopløsningen, men sætter en nedre grænse på 300x300 pixel. Tekststørrelsen gør dog at den praktiske nedre grænse, hvor spillet ser ud som det skal, er 1280x720. Ved skærmopløsning større end 2k bliver teksten og kortet for småt til at kunne ses ordentligt, hvorfor 2k er en naturlig øvre grænse for skærmopløsningen. Alle opløsninger imellem de to grænser burde være i orden (så længe det bare nogenlunde følger en 4:3 eller 16:9 ratio).

For at sørge for at den valgte skærmopløsning er brugt i alle views er der oprettet et objekt til at holde informationer om blandt andet indstillinger, samt andre informationer, som det ønskes at kunne tilgå fra flere forskellige views uden at være nødsaget til at sende data med rundt, når der skiftes mellem views. Dette objekt følger et singleton design pattern, og den samme instance af objektet kan derfor tilgås fra alle de view som skal bruge informationer derfra, på den måde opnås det at der kan sættes globale indstillinger som kan bruges på alle views uden at informationen skal sendes med i et skærmskift.

Settings menuen kan tilgås fra både main menu og ingame menu, og det er derfor nødvendigt at spillet ved hvilken menu brugeren kommer fra, sådan at brugeren kan komme tilbage til den rigtige menu, når settings menuen forlades, ved at der trykkes på back-knappen. Til dette bruges singleton objektet fra tidligere afsnit også, da det gør det nemt at bringe informationen mellem views. Dette bruges også i ingame menu, da denne kan tilgås fra både room view og combat view og skal kunne returnere brugeren til det rigtige view, når brugeren vælger at starte spillet igen.

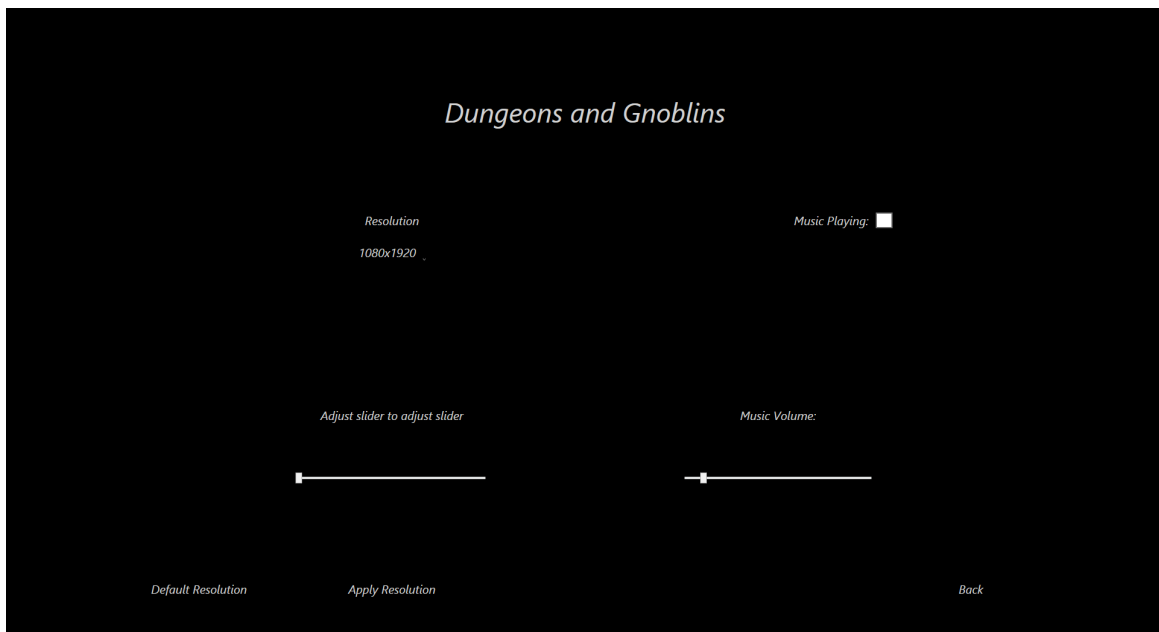


Figure 49: Menu til at ændre spillets indstillinger. Det er her muligt at vælge skærmopløsning og lydstyrke, samt tænde og slukke for musikken. Back knappen fører tilbage til enten main menu eller ingame menu, afhængig af hvilke menu man tilgik settings menuen fra.

### 9.2.5 Load view

Load (Figure 50) og Save menuerne præsenterer spilleren med en liste af gemte spil, som brugeren kan hente, eller gemme. Dette opnås ved at der hver gang spilleren åbner en af de to menuer, hentes en liste af tilgængelige 'save-games', som via databinding, præsenteres for brugeren. Når der trykkes på Save/Load sendes den fornødne kommando til backenden og backenden tager fat i databasen og udfører enten save eller load kommandoen.

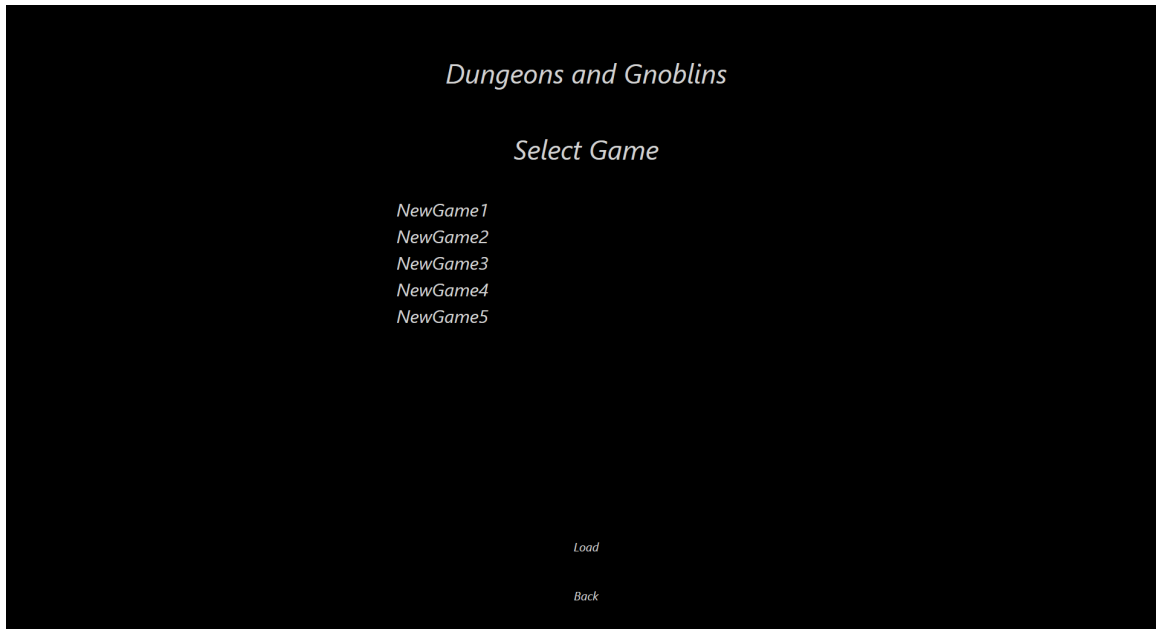


Figure 50: Load menu view. Her præsenteres spilleren for alle gemte spil. Disse hentes fra databasen hver gang spilleren går ind i load menuen.

### 9.2.6 Note om Baggrundsfarver

Da den generelle baggrundsfarve i spillet er sort er alle spillets interface elementer oprettet så baggrundsfarven matcher. Dette er for det meste nemt opnået ved brug af WPF styles, f.eks.:

```
<Style x:Key="textBoxStyle" TargetType="TextBox">
    <Setter Property="Background" Value="{StaticResource backgroundColor}"/>
    <Setter Property="Foreground" Value="{StaticResource textColor}"/>
    <Setter Property="FontSize" Value="25"/>
    <Setter Property="FontStyle" Value="Italic"/>
    <Setter Property="BorderThickness" Value="1"/>
</Style>
```

men enkelte elementer (så som resolution dropdown menu, brugt i settings menuen) viste sig at være betydeligt mere omfattende. Det viser sig at det valgte element (WPF combobox) ikke tillader at baggrundsfarven for dropdown elementerne ændres i Windows 8 eller senere. Løsningen er at lave en kopi af hele templatens (ca. 300 linjer kode) for combobox og ændre 3-4 linjer. [comboBox]

### 9.3 Game Engine

Game Enginen er skrevet i C#, et objektorienteret programmering sprog med stærkt library support der tillader udvikleren at fokusere på udvikling af applikationer istedet for udvikling af libraries til at støtte projektet.

Nedstående præsenteres et diagram over de mest kritiske komponenter Game Enginen interne spil logik og deres relationer til hinanden. Der er ikke medtaget interfaces, eller klasser som er map, items, BackendController og logs som er mere specifikke til spillet og ikke til den interne logik.



Figure 51: Viser det mest definerende elementer af Game Engine og deres relationerne til hinanden. Der er her ikke medtaget interface, abstract klasser, item, logs og map.



### 9.3.1 Gamecontroller Implementering

GameControlleren er det centrale komponent i game engineen, denne er ansvarlig for kommunikation til frontend del af applikationen. I denne implementering af GameControlleren har den adgang til alle spillet funktionaliteter gennem dennes association til Combatcontroller, se Figure 51, og BackendController, som ikke er vist på Figure 51.

Fra et implementering perspektiv er dette en nem løsning for en lille applikationen som denne men fra et design perspektiv er dette en dårlig løsning. GameControlleren har alt for mange grunde til at ændre sig og følger næppe SOLID principperne.

GameControlleren saver og loader spillet gennem sin relation til BackendController. Desværre er load funktionen ret kompliceret og burde være delt op i mindre funktioner. Source code for load kan ses på Figure 52

Koden bliver kompliceret idet den forsøger at gennemløbe alle Room objects i spillet for på korrektvis at fjerne enemies og items, som allerede er blevet samlet op tidligere. Her gennemløbes alle Room objects og i hvert room gennemløbes alle items i Room chest objectet for at krydsreferer alle items med inventory listen for at se om de skal fjernes som en del af save gamet.

Ligeledes håndtere den enemies og krydsreferer alle enemies med slainEnemies listen for at se hvilke enemies spilleren allerede har besejret. Disse enemies fjernes herefter fra spillet.

```

2 References
public async Task LoadGame(int id)
{
    await Reset();
    SaveDTO Game = await backEndController.GetSaveAsync(id);
    CurrentLocation.RemovePlayer();
    CurrentLocation = GameMap.Rooms[Game.RoomId];
    VisitedRooms = Game.VisitedRooms;
    SlainEnemies = Game.SlainEnemies;
    Inventory = Game.Inventory;
    CurrentPlayer.HP = Game.Health;
    GameMap.Rooms[CurrentLocation.Id].AddPlayer(CurrentPlayer);
    List<(uint, Item)> temparray = new List<(uint, Item)>();

    foreach (ILocation room in GameMap.Rooms)
    {
        if (room.Chest != null)
        {
            foreach (Item item in room.Chest)
            {
                if (Inventory.Contains(item.Id))
                {
                    if (Game.WeaponId == item.Id)
                    {
                        CurrentPlayer.EquippedWeapon = (Weapon)item;
                    }
                    if (Game.ShieldId == item.Id)
                    {
                        CurrentPlayer.EquippedShield = (Shield)item;
                    }
                    CurrentPlayer.Inventory.Add(item);
                    temparray.Add((room.Id, item));
                }
            }
        }

        foreach ((uint TempRoom, Item TempItem) tempval in temparray)
        {
            GameMap.Rooms[tempval.TempRoom].Chest.Remove(tempval.TempItem);
        }

        if (room.Enemy != null)
        {
            if (SlainEnemies.Contains(room.Enemy.Id))
            {
                room.RemoveEnemy();
            }
        }
    }
}

```

Figure 52: Load funktionen, resetter game, setter de rigtige states og gennemløber alle Room instances og sætter deres state til den korrekte state.

### 9.3.2 Generering af Map

Map klassen generer et map til spillet. Denne består af en simple liste af lister over, hvilket rooms hvert room er forbundet til.

Denne Process er kompliceret og kræver en uddybende forklaring. Problemet består i at afgøre hvordan man sikre at det samme map bliver genereret hver gang spillet loades.

#### Gem en map layout fil

Denne løsning viste sig at være den bedste løsning for at sikre sig, at map layoutet kun skulle eksistere i en enkel folder i projektet. Men det viste sig vanskelig at læse en sådan fil uafhængigt af pc'en programmet blev kørt på.

#### Lav en klasse som generer map layout filen

Ultimativt blev dette løsningen, som benyttes i projektet. En MapCreator klasse danner en map layout fil når dens konstruktor bliver kald.

En map klasse som store map layoutet kan nu læse layout filen og danne et map udfra denne fil. Filen består af linjer med formen *"leftRoomId, TopRoomId, RightRoomId, BottomRoomId"*. Den først linje dækker room 1, næste linje dækker room 2 osv.

Hver linje i map layout filen mappes til en liste af roomId'er, der kan insertes i Map klassens mapLayout liste. Denne Liste danner grundlaget for spillets map og diktere hvordan spilleren kan navigere i spillet. Denne mapping mellem strings og int array er vist i Figure 53.

```
1 reference
private LinkedList<int> CreateRoomLink(string linkingString)
{
    int[] link = linkingString.Split(",").Select(int.Parse).ToArray();
    return new LinkedList<int>(link);
}
```

Figure 53: illustrer hvordan en linje fra map layout filen omdannes til en liste af room id'er som rummet er forbundet med. Dette er kerne mekanismen for hvordan en spiller kan navigere rundt i spillet, da en spiller ikke kan bevæge sig fra et Room til et anden, der ikke er i denne liste af forbindelser mellem det nuværende room og den ønskede bevægelses retning.

#### Items og Enemies

Som det sker for rooms, ligeledes sker det for enemies og items. MapCreator klassen generer separate filer for enemy positions og item lokationer. Map klassen kan herefter læse filen og mappe hver linje i filen til, en enemy eller item og placere det i det korrekte room.

### 9.3.3 Log

Kommunikation med frontend fra GameControllern og CombatControlleren gør brug af en log. Logen består af et dictionary datastruktur som GameControllern og CombatControlleren benytter til at logge vigtige information, som frontend kunne have brug for at vise til spilleren.

Dette kan være information om spillerens bevægelse fra et Room til et andet. Det kan også være information omkring spillerens eller enemies tilbageværende livsmængde osv. Dette isolere frontend fuldt fra Game Engine, da frontend ikke har andre accesspunkter til information om hvad der sker i spillet.

Derved opnår frontend separation of responsibility da frontend nu kun er ansvarlig for at display information som det er givet af game engine.

## 9.4 Kommentar til implementeringen

Gennem udviklingen af Game Enginen er der brugt interfaces eller abstrakte klasser til implementering af alle funktionaliteter. Dette gør det både nemt at teste implementeringerne samt at udvide spillet. Men for scopet af opgaven er dette nok "Overkill". Det er godt at bruge, men for en applikation så lille som denne, der ikke kommer til at udvide sig er det nok for meget boilerplate kode.

## 9.5 Backend Implementering

I det følgende afsnit gennemgås implementerings detaljer omkring Web api'ets controller klasser, samt for BackEndController klassen på client siden. Disse klasser gør alle brug af nogle model DTO'er som ligeledes præsenteres.

Alle controller klasserne er implementeret i C#, da dette er sproget som anvendes i Asp.net Core.

### 9.5.1 Data Transfer Object:

De data objekter som gemmes i SQL databasen vil indeholde nogle navigational properties, som bruges til at query data'en og til at opretter relationer mellem objekterne. Da disse properties ikke har nogen relevans for clienten oprettes der følgende DTO'er for modellerne User og Save som ses på Figure 54

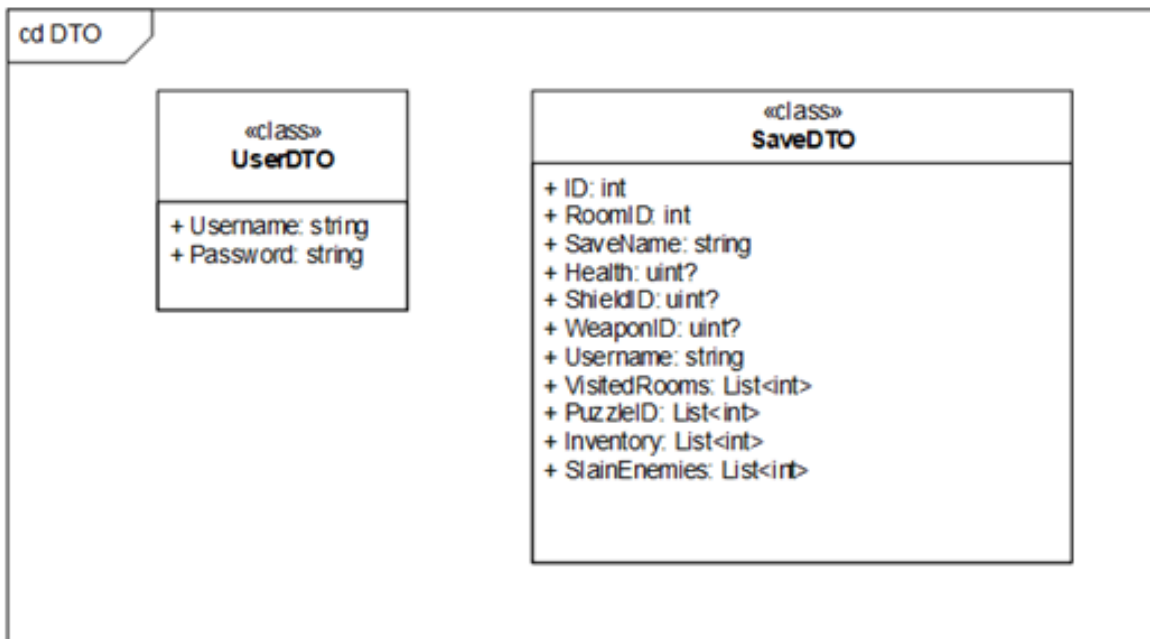


Figure 54: klasse diagram over DTO'er for User og Save

På den måde kan man sikre at kun den nødvendige data sendes til clienten.

### 9.5.2 SaveController

Alle routes som omhandler game state skal authorize, mens routes, som omhandler brugeren tilader anonyme forespørgelser.

SaveController klassen inkluderer biblioteket Microsoft.AspNetCore.Mvc, dette gør det muligt, at anvende attributter til at definere hvilke Http metoder som funktionerne anvender eksempelvis HttpGet se Figure 55.

```
// GET: Save
[HttpGet]

0 references | sulunicul, 4 days ago | 2 authors, 4 changes
public async Task<ActionResult<SaveDTO>> GetSave(int id)
{
    var identity = User.Identity;

    var save = await _save.GetSaveByID(id);

    if (save == null)
    {
        return NotFound();
    }

    if (identity.Name == save.Value.Username.ToLower())
    {
        return save;
    }
    else
    {
        return Unauthorized();
    }
}
```

Figure 55: Code snippet af HttpGet metode som modtager et id og returnere et SaveDTO object til klienten

Biblioteket gør det ligeledes muligt at returnere ActionResult, som kan indeholde objekter og en status kode eller blot en status kode, hvis noget går galt. Alle funktionerne er async og returnerer en Task, hvilket gør at der er tale om asynkrone funktioner, der først returnerer en værdi når dataen er klar. Dette gør også at klienten kan klare andre opgaver indtil dataen er klar. SaveController inkluderer også Microsoft.AspNetCore.Authorization biblioteket, som bidrager med funktionalitet til kun at tillade kald, fra en korrekt bruger med attributten [Authorize]. Da der kun findes en type af identiteter nemlig en helt normal bruger, som har adgang til alle endpoints, skal der findes en måde at sikre at brugeren kun kan tilgå sine egne data. Derfor gøres brug af User.Identity, kan ses på Figure 56. Dette anvendes til at hente Claims for User'eren tilhørende den nuværende action, hvilket så kan bruges til at tjekke om brugeren har lov til at hente det pågældende gemte spil.

### 9.5.3 UserController

UserController klassen minder meget om SaveController klassen, den gør ligeledes brug af Libraries Microsoft.AspNetCore.Mvc og Microsoft.AspNetCore.Authorization. På Figure 56 ses Login funktionen.

```
[HttpPost("Login"), AllowAnonymous]
0 references | magnusblaa, 1 day ago | 2 authors, 2 changes
public async Task<ActionResult<Token>> Login(UserDTO userDTO)
{
    userDTO.Username = userDTO.Username.ToLower();
    var user = await _context.Users.Where(u => u.Username == userDTO.Username).FirstOrDefaultAsync();
    if (user != null)
    {
        var isValid = BCrypt.Net.BCrypt.Verify(userDTO.Password, user.Password);
        if (isValid == true)
        {
            return new Token {JWT = GenerateToken(userDTO)};
        }
    }
    ModelState.AddModelError(string.Empty, "Wrong Username or Password");
    return BadRequest(ModelState);
}
```

Figure 56: Code snippet af HttpPost metode som modtager et UserDTO object og returnerer en JWT hvis brugeren findes og password'et er korrekt

Her tilføjes endnu en attribut AllowAnonymous, da login skal være et anonymt kald, fordi brugeren endnu ikke er logget ind på clienten. Hvis brugeren findes og password'et er korrekt returneres en ny JWT.

Til at Hashe passwords med anvendes Bcrypt [**Bcrypt**]. Der findes en C# udgave kaldet Bcrypt.Net som vil blive anvendt. Helt præcist er det funktionen HashPassword("password", BcryptWorkfactor), denne funktion skal have en BcryptWorkfactor, hvilket er et tal der siger noget om hvor mange iterationer Bcrypt vil bruge på at generere et "salt" til at hashe password'et med. Her anbefales det at bruge en værdi på 11, da det generelt set anses som tilstrækkeligt niveau hvad angår sikkerhed. Funktionen verify("password", hashedpassword) bruges til at verificere om et givent password svarer til dets krypterede udgave.

### 9.5.4 BackEndController Client

BackendController indeholder som det blev nævnt i backend Design afsnittet subsection 8.4 et HttpClient objekt til at håndtere http request/response, og et Token objekt til at holde på den modtagne JWT. Et klasse diagram over BackendController kan ses på Figure 57

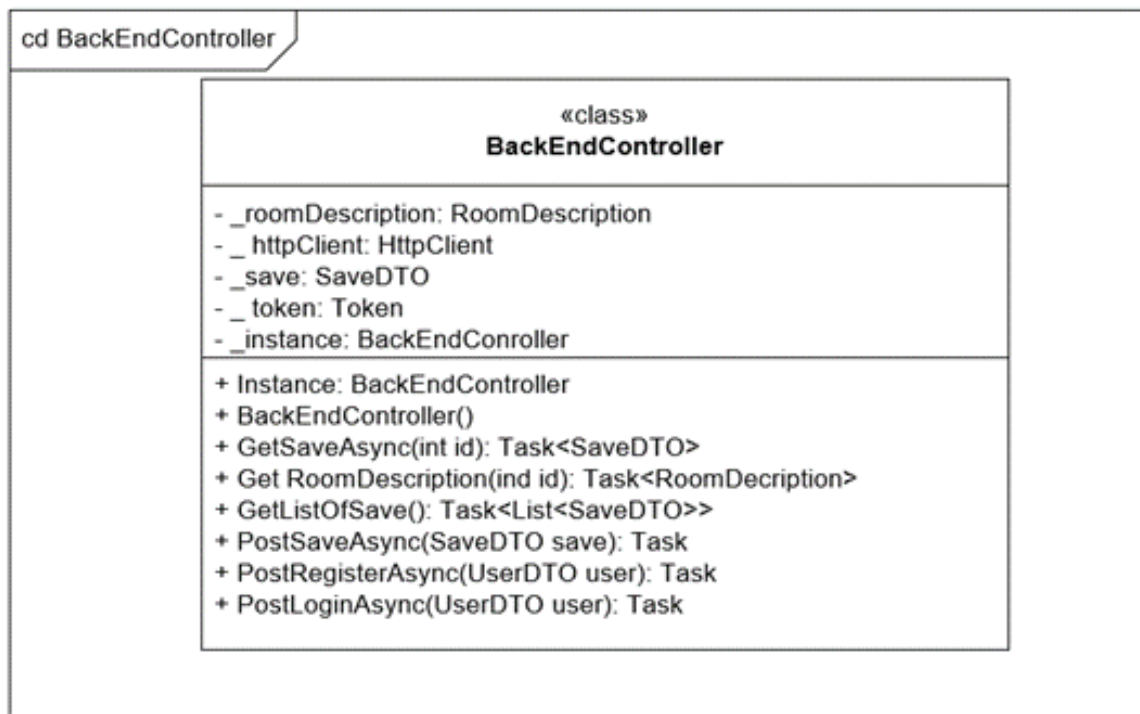


Figure 57: klasse diagram af BackendController

For at sikre at der kun findes en JWT i programmet ad gangen, skal BackendController klassen implementeres som en singleton, hvilket vil sige at der kun eksisterer en enkelt instance i hele programmet, som der er global adgang til. På den måde sikres det nemlig at BackendController altid indeholder den korrekte JWT, uanset hvor henne i programmet der laves en request fra. Uden dette ville JWT nemlig kun være sat inde i det scope, hvor der blev logget ind eller en bruger blev registreret fra, altså der hvor den nuværende instance af BackendController eksisterer. Funktionerne her er ligeledes asynkrone.

#### 9.5.5 Konklusion

Funktionerne i backend controller klasserne samt funktionerne på klientens BackendController er implementeret som asynkrone funktioner, der først returnerer en værdi når ressourcen som efterspørges er klar. Client klassen er implementeret som en singleton for at holde styr på den aktuelle JWT.

## 9.6 Database Implementering

Til implementering og håndtering af databasen i .NET har gruppen valgt at benytte Entity framework core. EF core gør det nemt at oprette og håndtere objekter C# som skal gemmes eller hentes i databasen.

Til modellering af databasen benyttes det udarbejdede ER-diagram hvorpå keys og relations er bestemt.

Disse keys og relations opsættes i projektets backend ved hjælp af fluent api, hvori man nemt kan specificere keys, foreign keys, relations, hasData og meget mere.

Ved ændringer af databasens udseende og form kan EFcore migrations hjælpe med at oprette de korrekte queries således at databasen bliver ændret korrekt, samt at man ved forkerte ændringer hurtigt kan hoppe tilbage til en tidligere migration ved eventuelle fejl. I EF core benyttes Language Integrated Query (LINQ) til at skrive ensartede queries til databasen.

Der er i projektets backend oprettet klasser, models, tilsvarende ER diagrammernes entiteter. Udover det valgte attributter er der også oprettet navigationals i de nødvendige klasse så der kan laves relationer. På Figure 58 herunder ses opsætningen af keys for de forskellige entiteter. Disse er opsat med fluent api efter ER diagrammerne på Figure 44 og Figure 45.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    /***** KEYS *****/
    //User
    modelBuilder.Entity<User>()
        .HasKey(x => x.Username);

    //Save
    modelBuilder.Entity<Save>()
        .HasKey(x => x.ID);

    modelBuilder.Entity<Save>()
        .HasIndex(n => new { n.SaveName, n.Username }).IsUnique();

    //RoomDescriptions
    modelBuilder.Entity<RoomDescription>()
        .HasKey(x => x.RoomDescriptionID);

    //Rooms
    modelBuilder.Entity<VisitedRooms>()
        .HasKey(x => new { x.SaveId, x.VistedRoomId});

    //Puzzles
    modelBuilder.Entity<Puzzles>()
        .HasKey(i => new { i.Save_ID, i.Puzzles_ID, });

    //Inventory
    modelBuilder.Entity<Inventory_Items>()
        .HasKey(k => new { k.SaveID, k.ItemID });

    //Enemies
    modelBuilder.Entity<Enemies_killed>()
        .HasKey(k => new { k.SaveID, k.EnemyID });
}
```

Figure 58: Opsætning af keys og unikke indexes for de forskellige entiteter i backends DbContext

Udover de forskellige keys, skal der også opsættes relationer mellem de forskellige entiteter, som vist på ER-Diagrammerne Figure 44 og Figure 45, samt referencer til foreign keys. Opsætningen kan ses på Figure 59 herunder:



```

// one to many relationship save vistedRooms
modelBuilder.Entity<VisitedRooms>()
    .HasOne<Save>(x => x.Save)
    .WithMany(y => y.VisitedRooms)
    .HasForeignKey(x => x.SaveId);

//many to one
modelBuilder.Entity<Save>()
    .HasOne<User>(s => s.User)
    .WithMany(s => s.Saves)
    .HasForeignKey(i => i.Username);

//1 save to many Puzzles
modelBuilder.Entity<Puzzles>()
    .HasOne<Save>(i => i.save)
    .WithMany(i => i.Save_Puzzles)
    .HasForeignKey(s => s.Save_ID);

//1 Save to many Items
modelBuilder.Entity<Inventory_Items>()
    .HasOne<Save>(i => i.save)
    .WithMany(s => s.Save_Inventory_Items)
    .HasForeignKey(s => s.SaveID);

//1 Save to many Enemies
modelBuilder.Entity<Enemies_killed>()
    .HasOne<Save>(s => s.save)
    .WithMany(s => s.Save_Enemies_killed)
    .HasForeignKey(i => i.SaveID);

```

Figure 59: Opsætning af relations og foreign keys for de forskellige entiteter i backends DbContext

Databasen er seedet ved hjælp af hasdata funktionen. Her oprettes en enkelt bruger, "Gamer1", med password "123", som hashes ind i databasen. "Gamer1" får derudover også 5 tilhørende "tomme" saves og til slut er der også indsat rumbeskrivelser for hver af de 20 rum. Dette ses på Figure 60 herunder.

```

modelBuilder.Entity<Save>()
    .HasData(
        new Save { RoomID = 0, ID = 2, SaveName = "NewGame2", Username = "gamer1", Health = 10 },
        new Save { RoomID = 0, ID = 1, SaveName = "NewGame1", Username = "gamer1", Health = 10 },
        new Save { RoomID = 0, ID = 3, SaveName = "NewGame3", Username = "gamer1", Health = 10 },
        new Save { RoomID = 0, ID = 4, SaveName = "NewGame4", Username = "gamer1", Health = 10 },
        new Save { RoomID = 0, ID = 5, SaveName = "NewGame5", Username = "gamer1", Health = 10 }
    );

modelBuilder.Entity<User>()
    .HasData(
        new User { Username = "Gamer1", Password = BCrypt.Net.BCrypt.HashPassword("123", 11) });

modelBuilder.Entity<RoomDescription>()
    .HasData(
        new RoomDescription { RoomDescriptionID = 1, Description = "1. The king has died of a magical curse o"},
        new RoomDescription { RoomDescriptionID = 2, Description = "2. You hear a strange rumble. What is hap"},
        new RoomDescription { RoomDescriptionID = 3, Description = "3. That was a strange encounter, but the"},
        new RoomDescription { RoomDescriptionID = 4, Description = "4. The door revealed a tunnel that led to"},
        new RoomDescription { RoomDescriptionID = 5, Description = "5. This room seems empty and dark. There"},
        new RoomDescription { RoomDescriptionID = 6, Description = "6. The whelps corpse has a weird smell to"},
        new RoomDescription { RoomDescriptionID = 7, Description = "7. What is this? It looks like a cellar t"},
        new RoomDescription { RoomDescriptionID = 8, Description = "8. This badly-lit corridor leads to a rus"},
        new RoomDescription { RoomDescriptionID = 9, Description = "9. The room is empty, but leads to anothe"},
        new RoomDescription { RoomDescriptionID = 10, Description = "10. It seems you've stumbled upon a dini"},
        new RoomDescription { RoomDescriptionID = 11, Description = "11. Wait, a well? A rope is attached to"},
        new RoomDescription { RoomDescriptionID = 12, Description = "12. The room is lit by a mysterious item"},
        new RoomDescription { RoomDescriptionID = 13, Description = "13. The dead naked Gnoblin looks horrend"},
        new RoomDescription { RoomDescriptionID = 14, Description = "14. Ew. The room is filled by a stench o"},
        new RoomDescription { RoomDescriptionID = 15, Description = "15. The door locks as you enter the room"},
        new RoomDescription { RoomDescriptionID = 16, Description = "16. The corridor is split between two wa"},
        new RoomDescription { RoomDescriptionID = 17, Description = "17. The brute-goblin put up a good fight"},
        new RoomDescription { RoomDescriptionID = 18, Description = "18. A door is present and you need a key"},
        new RoomDescription { RoomDescriptionID = 19, Description = "19. The Gnoblin king has been slayed. I"},
        new RoomDescription { RoomDescriptionID = 20, Description = "DB says: This is room 20" }
    );

```

Figure 60: Seeding af databasen med en enkel bruger, "Gamer1", 5 tilhørende "tomme" saves og beskrivelser af historien til de 20 forskellige rum

## 10 Test

Testing er en grundsten i ethvert succesfuldt softwaresystem. Uden testing kan der ikke stilles garanti for et systems opførsel. Nedenstående afsnit dækker alle test foretaget af "Dungeons and Goblins" spillet, for at sikre den korrekte opførsel i henhold til kravspecifikationerne. Her dækkes test af alle de største komponenter, Frontend, Game Engine, Backend og database. Testene inkluderer automatiseret unittests, integrationstest og accepttest.

### 10.1 Modultest Frontend

Modultesten af Frontenden er lavet i meget tæt samarbejde med Game Engine holdet. Dette er valgt sådan, at når Game Engine lavede en ny funktion eller funktionalitet, gik frontend holdet igang med at implementere en visuel repræsentation af denne nye funktion/funktionalitet. Således var det ikke kun en visuel test af at views så godt ud eller at man kunne trykke på en knap, men derimod kunne både frontenden og Game Enginen testes i samarbejde, hvor den reelle funktionalitet for systemet blev testet.

#### 10.1.1 Test metoder

Hver gang der er blevet lavet små eller større ændringer i frontenden, er der blevet lavet både en funktionel og visuel test af de nye ændringer. Dette blev gjort for æstetiske ændringer ved at kigge i preview vinduet i WPF for det view der blev ændret. Var det derimod en ændring der inkluderede

databinding til Game Enginen, blev det nødvendigt at teste hele programmet ved brug af compileren og derved lave en runtest, som inkluderede at det rigtige data blev hentet fra Game Enginen eller at den rigtige funktionalitet blev kaldt ved et tryk på en knap.

### 10.1.2 Eksempel på frontend test i forbindelse med Game Engine

Et godt eksempel på hvordan frontend testene blev kørt er ved implementeringen af Room-viewets map element og mere specifikt bevægelsen fra et rum til et andet, altså de implementerede "Go North,East,South,West" knapper, som krævede både en visuel og funktionel del.

Den visuelle del bestod i at kortet skulle opdateres, spilleren flyttes og rum-beskrivelsen opdateres. Den funktionelle del bestod i at Game Enginen skulle kontaktes på korrekt vis med rigtige parametre. De rigtige databindings skulle opdateres, således at den visuelle del blev opdateret korrekt og derefter skulle informationen om det rum gemmes korrekt i Game Enginen.

For at være sikker på at alt data blev opdateret korrekt blev programmet kørt i debug mode, hvor der kunne sættes break-points i koden og værdierne på diverse variabler, såsom roomdescription og currentroom kunne undersøges. Samtidig blev der kigget på den visuelle del af selve viewet, hvor der blev tjekket om beskrivelsen blev korrekt displayet, mappen opdateret korrekt og at spilleren flyttes korrekt.

### 10.1.3 Eksempel på frontend test

Ikke alle moduler krævede at Game Enginen blev kontaktet. Eksempelvis Mediatoren, som er beskrevet i Frontend implementings afsnittet som kan findes her: subsection 9.2. Med dette modul blev der lavet mocks, hvor vi satte en dummy knap op til at kunne kalde notify() funktionen i Mediatoren og der blev tjekket visuelt om viewet blev skiftet uden at hele applikationsvinduet blev skiftet og at det korrekte view blev vist.

## 10.2 Modultest Game Engine

Game Engine styrer spillets indre logik. Dette dækker over alt fra spillerens bevægelse gennem spillet til "save" og "load" af nye og gamle spil. Det er derfor yderst nødvendigt at denne er fuld testet.

For at sikre Høj kvalitet er Game Engine skrevet med Robert C. Martins "Three Laws of TDD" [**CleanCode**] i baghoved hvilket medfører til at Game Engine er eksklusivt testet ved hjælp af black-box testing. Alle testene tester kun det offentlige interface, som er gjort tilgængelig.

### 10.2.1 GodkendelsesTabel Game Engine

Nedestående præsenteres en fuld tabel for alle classes testet som led i Game Engine. De vigtigste er GameController, CombatController, Player/Room og DiceRoller. Dice danner "Core Mechanics" for spillet.

Interessant nok ses her at GameController fejler sine test grundet at der ikke er skrevet test til mange af GameControllerrens ansvars punkter.

Table 11: Her Ses en komplet liste over alle test foretages på Game Engine komponenter, med kommentar til deres resultater og en endelig vurdering af test resultaterne.

Game Engine GodkendelsesTabel			
Komponent under test	Forventet Adfærd	Kommentar	Test Resultat
Game Controller	<ol style="list-style-type: none"> <li>1. Kan skifte Player til Nyt Room</li> <li>2. Kan samle Item op fra Room</li> <li>3. Kan save Game</li> <li>4. Kan loaded Games</li> <li>5. Kan eliminere Enemy fra Game</li> <li>6. Kan reset Game</li> <li>7. Kan anskaffe Room description</li> </ol>	<p>Game Controller er kun test for at skifte til nyt Room, dette betyder at der ikke kan stilles garanti for at resterende implementeringer af load- og save game osv. fungere som ønsket.</p> <p>Disse ting er svagt testet gennem visuelt trial and error test, men da der ikke er skrevet nogen specifikke test til dem Fejler GameControllern sin komponent test.</p>	FAIL
Combat Controller	<ol style="list-style-type: none"> <li>1. Kan Håndtere Combat Rounds</li> <li>2. Kan håndtere at Player løber væk fra Combat</li> </ol>	<p>Combat controller kan håndtere at spilleren løber fra combat og at Player indgår i combat. CombatController kan stille garanti for at combat sker i den rigtige orden og at hverken spiller eller enemy kan angribe hvis denne er død. Ydmere stiller den garanti for at både enemy og spiller kan lave "critial hits" hvis dice rolleren slår 20.</p>	OK

DiceRoller	<ol style="list-style-type: none"> <li>1. Kan emulerer et kast med en N siddet terning</li> <li>2. Kan emulerer N kast med en M siddet terning</li> </ol>	DiceRoller Kan emulere et eller flere terninge kast med samme antal sidder. Denne kan ydmere stille krav for at fordelingen af disse terningekast har en normal distribution og dermed er alle udfald lige sandsynlige.	OK
BaseMapCreator	<ol style="list-style-type: none"> <li>1. Kan generere et map layout file</li> </ol>	BaseMapCreator kan på korrekt vis generere et map layout file, den kan ydmere generer item layout files og Enemy layout files, der hjælper Map klassen med at genererer spillet Map. Der ikke skrevet test for eksistensen af item og enemy layout Map og derfor kan der ikke stille garanti for at disse bliver genereret på korrektvis. Testen Fejler derfor.	FAIL
BaseMap	<ol style="list-style-type: none"> <li>1. Kan anskaffe Rooms ud fra en Direction</li> </ol>	Map kan anskaffe Rooms ud fra en given Direction. Map kan på korrektvis generer et map ud fra mappets layout file. Den kan også på korrektvis finde ud af hvilke Rooms har adgang til hinanden. Der er ikke skrevet test for at bekræfte at enemy og items er i de korrekte lokationer baseret på enemy og item layout filerne. Derfor fejler denne sin Test.	FAIL

Player	<ol style="list-style-type: none"> <li>1. Kan angribe Enemy</li> <li>2. Kan tage skade</li> <li>3. Kan skade Enemy (Ikke Kritisk)</li> <li>4. Kan skade Enemy (Kritisk)</li> </ol>	Player kan angribe, skade og tage skade fra enemy.	OK
Enemy	<ol style="list-style-type: none"> <li>1. Kan angribe Player</li> <li>2. Kan tage skade</li> <li>3. Kan skade Player (Ikke Kritisk)</li> <li>4. Kan skade Player (Kritisk)</li> </ol>	Enemy kan angribe, skade og tage skade fra Player.	OK
Log	<ol style="list-style-type: none"> <li>1. Kan log et event</li> <li>2. Kan anskaffe et event</li> <li>3. Kan merge to logs</li> </ol>	Log kan logge et event, anskaffe eventet og merge to forskellige logs.	OK
Room	<ol style="list-style-type: none"> <li>1. Kan tilføje Player</li> <li>2. Kan tilføje Enemy</li> <li>3. Kan fjerne Player</li> <li>4. kan fjerne Enemy</li> </ol>	Room kan fjerne/tilføje Player og Enemy som forventet	OK
Items		Items så som sword, shield, axe osv. indeholder kun data og har derfor ikke nogen testbar adfærd andet end deres constructor. De kan alle konstrueres på korrekt vis.	OK

### 10.2.2 Mock testing

I nogle tilfælde kan det være umuligt at lave troværdige tests, når en klasse f.eks. Player er dybt afhængig af DiceRoller klassen. Dette skyldes at DiceRoller danner pseudo-random outputs og derfor er test med den ikke altid troværdige.

Her benyttes mock tests og dependency injections til at sikre at DiceRoller klassen danner de samme outputs hver gang en test køres. Derved kan alle scenarier for Player klassen testes, hvor man kan regne med at DiceRoller giver et bestemt output.

```
[TestFixture]
0 references
public class PlayerTest
{
    private Player uut;
    private Enemy enemy;
    private DiceRoller basicDiceRoller;
    private Weapon weapon;

    [SetUp]
    0 references
    public void SetUp()
    {
        basicDiceRoller = Substitute.For<DiceRoller>();
        weapon = Substitute.For<Sword>(((uint) 8, (uint) 1), (uint) 2, (uint) 2);
        uut = new Player(10, 16, weapon, basicDiceRoller);
        enemy = new Enemy(10, 10, (10, 1), 2, "test");
    }
}
```

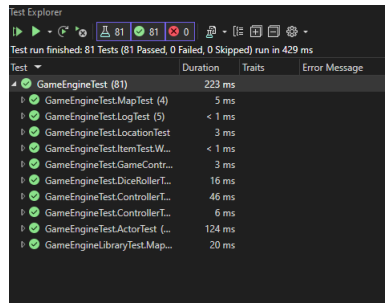
Figure 61

```
[Test]
0 references
public void Hit_IfPlayerEquippedWeaponNull_DefaultHitIsFalse()
{
    basicDiceRoller.RollDice(20).Returns((uint)10000);
    uut.EquippedWeapon = null;
    bool expectedHitResult = false;
    (bool Hit, bool) actualHitResult = uut.Hit(enemy);
    Assert.That(actualHitResult.Hit, Is.EqualTo(expectedHitResult));
}
```

Figure 62: Mocks benyttes her til at sikre at dependencien, her DiceRoller, returner den ønskede værdi for situationen, der er under test. Alle test tildeles informative navne, for at sikre læsbarhed i forhold til testenens formål.

### 10.2.3 Test Resultater for Game Engine

Nedestående vises kort resultatet for Game Engine test, når de alle køres via visual studio. Som det kan ses så er alle testene succesfulde, hvilket giver hvis garanti for at game engine opfører sig som specificeret i kravspecifikationerne og i de implementerede interfaces.



The screenshot shows the Visual Studio Test Explorer window. At the top, it indicates 'Test run finished: 81 Tests (81 Passed, 0 Failed, 0 Skipped) run in 429 ms'. Below this, a table lists the tests and their durations. All tests are marked with a green circle, indicating they passed.

Test	Duration	Traits	Error Message
GameEngineTest (81)	223 ms		
GameEngineTest.MapTest (4)	5 ms		
GameEngineTest.LogTest (5)	< 1 ms		
GameEngineTest.LocationTest	3 ms		
GameEngineTest.ItemTest.W...	< 1 ms		
GameEngineTest.GameCont...	3 ms		
GameEngineTest.DiceRollerT...	16 ms		
GameEngineTest.ControllerT...	46 ms		
GameEngineTest.ControllerT...	6 ms		
GameEngineTest.ActorTest (...)	124 ms		
GameEngineLibraryTest.Map...	20 ms		

Figure 63: Alle skrevne test til Game Engine passer, hvilket hjælper med at give vished om at Game Engine udfører dens funktionalitet, som det er beskrevet i kravene. Dette siges da alle testene er skrevet på baggrund af kravene som black-box tests og ikke som white-box test efter implementeringen.



### 10.3 Modultest database

Der er som gruppe taget en beslutning om at hoste databasen lokalt i en docker container. Dette blev valgt på baggrund af en samtale med vejleder, hvor vi blandt andet, grundet sikkerhedsforanstaltning på au's netværk og problemer med microsoft licenser, kunne løbe ind i nogle problemer. Den lokale hosting medførte at vi kunne holde øje med databasens udformningen, samt den gemte data, gennem Microsoft azure datastudie.

For at teste DAL funktioner og dens forbindelse til databasen, oprettes først et DAL objekt med en context som indeholder en korrekt connectionstring.

```
var DataHelper = new DAL(context);
```

Figure 64: Oprettelse af DAL objekt med context indeholdende connectionstring

Til test hvor der skal indsættes data i databasen oprettes der objekter af den korrekte type hvorefter DAL funktioner til indsættelse kaldes. Korrektheden af de indsatte data kan herefter tjekkes med datastudie. Et eksempel på dette kan ses på Figure 65 herunder. Her oprettes et GameDTO objekt, gamesave.

Gamesave opsættes til "Gamer1", med navnet "My First Run", hvorefter der tilføjes yderligere data. Det overskrevne save er "NewGame1", som her har id 1.

```
var gamesave = new GameDTO();
gamesave.Armour_ID = 1;
gamesave.Username = "Gamer1";
gamesave.SaveName = "My First Run";
gamesave.RoomID = 13;
gamesave.Health = 69;
gamesave.itemsID.Add(12);
gamesave.enemyID.Add(1);
gamesave.PuzzleID.Add(1);
gamesave.VisitedRooms.Add(1);

DataHelper.SaveGame(gamesave);
```

Figure 65: Kode til test af SaveGame, hvor der oprettes et nyt save som overskriver det gamle save med saveID 1

På Figure 66 herunder ses et screenshot fra datastudie hvor de 5 saves til "Gamer1" kan ses. Det noteres at alle saves er "tomme" og starter i rum 1.

Results		Messages					
	ID	RoomID	Armour_ID	Health	Weapon_ID	Username	SaveName
1	1	0	NULL	10	NULL	gamer1	NewGame1
2	2	0	NULL	10	NULL	gamer1	NewGame2
3	3	0	NULL	10	NULL	gamer1	NewGame3
4	4	0	NULL	10	NULL	gamer1	NewGame4
5	5	0	NULL	10	NULL	gamer1	NewGame5

Figure 66: Screenshot fra datastudie hvor de 5 ens "tomme" saves kan ses

Herefter køres programmet fra Figure 65, og vi kan nu se ændringerne i databasen. Det noteres at SaveName og de andre attributter i Figure 67 nu er opdateret korrekt efter koden.

Results		Messages					
	ID	RoomID	Armour_ID	Health	Weapon_ID	Username	SaveName
1	1	13	NULL	69	NULL	gamer1	My First Run
2	2	0	NULL	10	NULL	gamer1	NewGame2
3	3	0	NULL	10	NULL	gamer1	NewGame3
4	4	0	NULL	10	NULL	gamer1	NewGame4
5	5	0	NULL	10	NULL	gamer1	NewGame5

Figure 67: Screenshot af datastudie efter overskrivning af save 1. Her er saveID 1 opdateret til nu at hedde "My First Run" og health og roomID er ændret korrekt, så det passer med det indsatte

De øvrige tilhørende lister til det gemte save er også opdateret med korrekte værdie. Dette kan også ses på Figure 68 herunder.

ItemID	SaveID
13	1
EnemyID	SaveID
2	1
VistedRoomId	SaveId
1	1
Puzzles_ID	Save_ID
3	1

Figure 68: Screenshot af datastudie efter overskrivning af save 1 med ekstra tabeller som referer til det rigtige saveID

Ved test af funktioner hvor der skal læses fra Db, kaldes DAL funktionen hvorefter den fundne information udskrives til konsolen ved hjælp af console writeline. Korrektheden af data i konsolen dobbelttjekkes med datastudie. Et eksempel på dette ses på Figure 69 herunder. Her henter vi det gemte save fra tidligere fra "Gamer1" ved navn "My First Run", hvorefter data fra dette save udskrives i konsolen.

```
var save = DataHelper.GetSaveByID(1);

Console.WriteLine("Username: " + save.Username);
Console.WriteLine("Current Room: " + save.RoomID);
Console.WriteLine("SaveName: " + save.SaveName);
Console.WriteLine("Health: " + save.Health);
Console.WriteLine("Armour id: " + save.Armour_ID);
Console.WriteLine("Weapon id: " + save.Weapon_ID);
Console.WriteLine("Items in inventory: ");
foreach (var i in save.ItemsID)
{
    Console.WriteLine(i + " ");
}
Console.WriteLine("Enemies killed: ");
foreach (var i in save.Enemies_killed)
{
    Console.WriteLine(i + " ");
}
Console.WriteLine("Puzzles solved: ");
foreach (var i in save.PuzzleID)
{
    Console.WriteLine(i + " ");
}
Console.WriteLine("Visited rooms: ");
foreach (var i in save.VisitedRooms)
{
    Console.WriteLine(i + " ");
}
```

Figure 69: Kode til test af GetSaveByID, hvor der hentes et save med saveID 1, hvorefter det og tilhørende info udskrives på konsolen

Når kodestykket fra Figure 69 ovenfor køres, får vi resultatet i konsol som set på Figure 70 herunder. Det noteres at det udskrevne data stemmer overens med det indsatte data fra den tidligere funktionstest.

```
Username: gamer1
Current Room: 13
SaveName: My First Run
Health: 69
Armour id:
Weapon id:
Items in inventory:
13
Enemies killed:
2
Puzzles solved:
3
Visited rooms:
1
```

Figure 70: Konsoleudskrift efter læsning af spil fra databasen, hvor det noteres at informationen stemmer overens med det indsatte i Figure 65

I Table 12 herunder ses en tabel over udførte test, forventede resultater, den faktiske observering

og vurderingen af observeringen. Alle test er udført og vi observerer det forventede resultat. Vi vurderer derfor alle test som OK.

Table 12: Tabel over modultest af DAL forbindelse til databasen. Alle test observeringer stemmer overens med forventningerne og markeres derfor OK

DAL-Database Godkendelses Tabel				
Test	Funktion	Forventet resultat	Observering	Vurdering (OK/Fail)
1	GetRoomDescription(id)	Her forventes det at funktionen henter beskrivelsen fra den valgte rumId.	Funktionen henter beskrivelsen korrekt og vi kan udskrive denne på konsolen.	OK
2	GetAllSaves	Det forventes at alle spillets saves, med tilhørende info, hentes fra databasen	Alle saves hentes fra databasen og information om disse kan udskrives på konsolen.	OK
3	GetSaveById(id)	Det forventes at der hentes alle oplysninger om et enkelt save fra databasen med tilsvarende id, som den medsendte parameter	Det korrekte save samt tilhørende info hentes og kan udskrive til konsolen	OK
4	SaveGame(Game)	Det forventes at det valgte save med samme id som det nye, overskrives, og at tilhørende info opdateres	Det observeres at det gamle save med samme id, nu er ændret og har korrekte nye værdier	OK

### 10.3.1 Diskussion af database modultest

Den ovenstående modultest er generelt godkendt da alle funktioner opfører sig som forventet, i og med at der hentes og gemmes korrekt i databasen. Med alle funktioner testet og godkendt er DAL-database forbindelsen klar til at blive integreret med resten af systemet.

## 10.4 Backend Modultest

Dette afsnit omhandler modultesten af backenden, denne deles op i følgende to dele, der fortages først en modultest af DAL som kan findes her subsection 10.3 , hvorefter DAL bruges i modultesten af Web api'et, da Web api'et ikke vil blive testet uden DAL modulet.

Til at udføre modultests af web api'et benyttes udviklingsværktøjet Swashbuckle [**Swagger**]. Swashbuckle bruges til at bygge SwaggerDocument objekter på baggrund af de routes, controllers og modeller som er blevet udviklet. Hertil tilbyder Swashbuckle et Swagger User interface, hvor man kan teste sine http funktioner og se om man modtager den forventede response. Derfor er dette et oplagt værktøj at gøre brug af. Et billede af Swagger User interfacet kan ses på Figure 71.

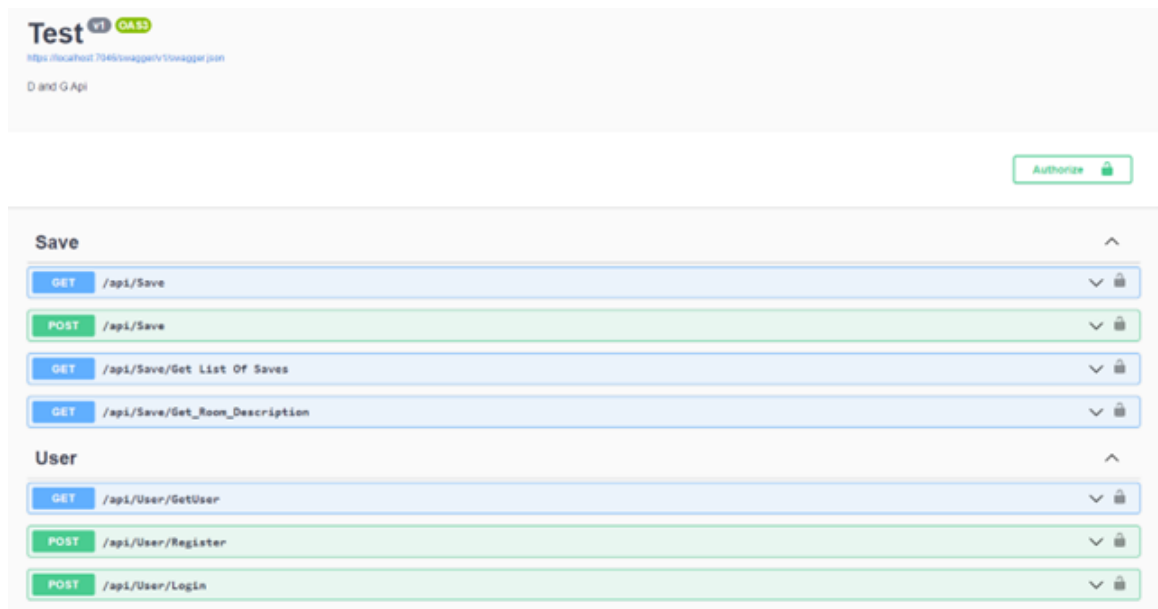


Figure 71: Swagger User interface, som viser de forskellige http metoder

For at aktivere swagger tilføjes følgende middleware til program.cs `AddSwaggerGen()`, `UseSwagger()` og `UseSwaggerUI()`. Til `SwaggerGen` tilføjes også security med JWT tokens så `Authentication` og `Authorization` kan testes.

Der er løbende blevet foretaget modultest, da det er forholdsvis enkelt at benytte swagger. På den måde undgås det at skulle sidde og rette store mængder af kode på en gang, men i stedet blevet gjort opmærksom på fejl løbende. I Table 13 kan ses en tabel oversigt over tests af succes scenarie, for de enkelte funktioner samt resultater.

Table 13: Tabel over modultest af Backendens Web Api, her vises testes af succes scenarier for alle Web Api http funktioner.

Test	Funktion	Forventet resultat	Observering	Vurdering (OK/Fail)
1	PostSave(SaveDTO saveDTO)	Der bliver sendt et specifikt gamestate, for brugeren der er logget ind.	Det er muligt at sende game statet, og de korrekte værdier bliver sendt med.	OK
2	Register(UserDTO regUser)	Der registreres en ny bruger, ved at gemme oplysninger omkring denne. Der returneres en JWT-token.	Brugeren bliver registreret, og kan findes blandt de andre brugere. Det ses at der returneres en JWT-token.	OK
3	Login(UserDTO userDTO)	Der tjekkes om oplysningerne passer med en registreret bruger, og passer det logges der ind. Der returneres en JWT-token.	Det lykkedes at logge ind, og der returneres en JWT-token.	OK
4	GetSave(int id)	Der hentes et specifikt save for den bruger der er logget ind.	Det lykkedes at hente et specifikt game state, uden errors	OK
5	GetListOfSave()	Der hentes en liste af game states, for brugeren der er logget ind.	Det lykkedes at hente en liste af game states, for den specifikke bruger.	OK
6	GetRoomDescription(int id)	Denne route henter en beskrivelse af det valgt rum i spillet.	Det ses, at der bliver hentet en beskrivelse af det valgte rum.	OK

I nedstående tabel testes om funktionerne håndtere fejlscenarier korrekt.

Table 14: Tabel over modultest af Backendes Web Api, her vises testes af fejlscenarier for alle Web Api http funktioner

Test	Funktion	Forventet resultat	Observering	Vurdering (OK/Fail)
1	Register(UserDTO regUser)	Brugernavnet er allerede i brug, og der sendes en fejlmeddelelse	Status: 400. Besked: "Name is already in use".	OK
2	Login(UserDTO userDTO)	Oplysningerne stemmer ikke overens med en registreret bruger, og der bør sendes en fejlmeddelelse.	Status: 400. Besked: "Wrong Username or Password"	OK
3	GetSave(int id)	Der er ikke logget ind, så der kan ikke hentes et game state.	Status: 401, Unauthorized	OK
4	GetListOfSave()	Der er ikke logget ind, og derfor kan der ikke hentes et liste.	Status: 401, Unauthorized	OK
5	PostSave(SaveDTO saveDTO)	Der er ikke logget ind, og derfor kan der ikke gemmes et spil.	Status: 401, Unauthorized	OK

Med alle funktioner testet med et godkendt resultat, er backenden klar til at blive integreret med de andre moduler for systemet.

## 11 Integrationstest

### 11.1 Tidlig Integrationstest

For gruppen var det et mål at komme igang med at lave integrationstest så hurtigt som muligt, da vi ønskede at der hurtigt kom styr på kommunikationen mellem de forskellige moduler i systemet. Det ville derefter være nemmere at implementere nye features.

Før den samlede integrationstest, er der lavet mindre test af forskellige dele af systemet. Frontend og Game engine er testet for sig og her kan man spille spillet, som i den tidlige integrationstest, blot er vores minimum viable product. subsection 3.1

Backend og database forbindelsen er også testet og der kan gennem swagger gemmes og hentes et save gennem api'ets get og post requests.

For at køre systemet har vi først startet den oprettede docker container til spillets database. Derefter startede vi spillets server, i form af backend api. Til slut kunne spillets klient åbnes og systemet testes.

I den tidlige integrationstest løb vi som gruppe ind i en række forskellige problemer.

Vi havde som gruppe arbejdet for opdelt i forskellige dokumenter. Blandt andet var DAL og backend api'et delt ud i forskellige projekter, som først måtte sammensættes og rettes før vi kunne gå videre. Derudover opdagede vi at de forskellige projekter benyttede forskellige versioner af .net, dette var dog hurtigt løst.

Til slut opdagede vi at der, når man loader et spil, ikke blev vist for brugeren hvilken rum man havde besøgt. Dette blev diskuteret og tilføjet til de næste iterationer.

I **INDSÆT REF TIL VIDEO HER** ses en demovideo af integrationstesten af MVP. Her ses det at man kan spille spillet, gemme det og derefter loade det korrekt ind igen, dog igen uden at man kan se hvilke rum man har besøgt.

### 11.2 Fulde Integrationstest

Med den indledende integrationstest, er kommunikationen igennem systemet på plads, samt de basale funktionaliteter. Nu udvides spillet med flere features, generelt drejer det sig om følgende features: Login, Register, Enemies, Items, Health, EquippedItems, Shield, Combat, Map Visibility og Room Descriptions. Dette er features, som kræver yderligere udvikling indenfor alle tekniske områder af projektet.

Feature udviklingen i Frontend og Game Engine Modul udvikles sammen, derfor er de moduler allerede integreret med hinanden, det samme gælder for Backend og Databasen.

Der laves endnu en integrations test med alle de nye features. Her opstod en række nye problemer hvad angår Http kommunikationen imellem backend'en og frontend'en. Dette bundede i overensstemmelsen af data modellerne på backend og frontend siden. Det drejede sig helt specifikt omkring de nyeligt tilføjede properties til modellen "Save", på frontend'en blev brugt datatypen uint imens der på backend blev brugt int, samtidigt med at properties hed noget forskelligt. Derfor blev de nye properties ignoreret når data'en blev seraliseret og deseraliseret. Dette problem var ikke noget som debuggeren gjorde opmærksom på derfor tog det relativt lang tid at få det løst.

Et andet problem som opstod, var angående Room Description, her blev det besluttet under implementering at gemme dem i databasen fremfor lokalt, grundet problemer med resources og deres



specifikke stier på forskellige pc'er.

Med disse problemer løst, resulterede integrationstesten i vores færdige produkt, som blev anvendt i accepttesten. Accepttesten samt en demonstrations video af produktet kan ses her. **REF TIL VIDEO AF ACCEPTEST VIDEO**

## 12 Accepttestspecifikation

For accepttestspecifikationen er der opstillet tests for dette projekt, og på baggrund af disse kan det betragtes som et funktionelt og acceptabelt produkt. Denne accepttestspecifikation er delt op i Funktionelle krav og ikke funktionelle krav.

### 12.1 Funktionelle

#### Test af User Story 1 - Login 1. Scenarie - Succesfuld login

*Givet at brugerens profil er oprettet på databasen og at serveren er oppe.*

- Når bruger indtaster sit login(Brugernavn og password)
- og trykker "Log in" på UI
- Så skifter skærmen til hovedmenu

Resultat: Godkendt

Kommentar:

#### 1. Scenarie - Fejlet login

*Givet at brugerens profil er oprettet på databasen og at serveren er oppe.*

- Når bruger indtaster et forkert login(Brugernavn og password)
- og trykker "Log in" på UI
- Så signaleres der om forkert login-oplysninger til bruger

Resultat: Godkendt

Kommentar: Viser korrekt error besked

#### Test af User Story 2 - Opret profil 2. Scenarie - Bruger opretter profil

*Givet at brugerens profil er oprettet på databasen og at serveren er oppe.*

- Når bruger vælger at oprette profil
- Så gemmes data for brugerens profil på databasen
- Og skærmen skiftes til log ind skærmen

Resultat: Godkendt

Kommentar:

#### 2. Scenarie - Bruger opretter samme profil

*Givet at brugerens profil er oprettet på databasen og at serveren er oppe.*

- Når bruger vælger at oprette en allerede-eksisterende profil
- Så signaleres der om at profil allerede eksisterer

Resultat: Godkendt

Kommentar: Viser korrekt error besked

### **Test af User Story 3-5 - Main Menu 3. Scenarie - Tilgå settings**

*Givet at brugeren er logget ind og har adgang til spillet.*

- Når bruger trykker settings i UI
- Så går spillet til settings menuen

Resultat: Godkendt

Kommentar:

### **4. Scenarie - Start spillet**

*Givet at brugeren er logget ind og har adgang til spillet.*

- Når bruger trykker "New Game"
- Så vises game-interface for brugeren

Resultat: Godkendt

Kommentar:

### **5. Scenarie - Exit game**

*Givet at brugeren er logget ind og har adgang til spillet.*

- Når bruger trykker "Exit game"
- Så lukker spillet

Resultat: Godkendt

Kommentar:

### **Test af User Story 6-14 - Exit Menu 6. Scenarie - Exit spil**

*Givet at brugeren har trykket "New Game"*

- Når bruger trykker "Escape" på keyboardet
- Så popper et menuvindue op med mulighederne "Resume Game", "Save Game", "Main Menu" og "Settings"

Resultat: Godkendt

Kommentar:

### **7. Scenarie - In game menu - Resume game**

*Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard*

- Når bruger trykker "Resume Game" i In game menu

- Så forsvinder menuvinduet og spillet fortsætter

Resultat: Godkendt

Kommentar:

#### **8. Scenarie - In game menu - Save – No Combat**

*Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard*

- Når bruger trykker på "Save Game" i In game menuen
- Så vises save menuen
- Og en liste af gemte spil

Resultat: Godkendt

Kommentar: Der vises maks. fem spil

#### **9. Scenarie - In game menu - Save – Combat**

*Givet at brugeren er i combat*

- Når bruger trykker "Escape" på keyboardet
- Så kan man ikke se en "Save Game" knap i game menuen

Resultat: Godkendt

Kommentar:

#### **10. Scenarie - In game menu - Main Menu**

*Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard*

- Når bruger trykker "Main Menu" i In game menu
- Så vises hovedmenuen

Resultat: Godkendt

Kommentar:

#### **11. Scenarie - In game menu - Settings**

*Givet at brugeren har trykket "New Game" og derefter har trykket "Escape" på keyboard*

- Når bruger trykker "Settings"
- Så åbnes et vindue med mulighed for konfiguration af spil for bruger

Resultat: Godkendt

Kommentar:

#### **12. Scenarie - Exit menu - Settings - Apply**

*Givet at brugeren har trykket "Settings" og ændret på resolution indstilling*

- Når bruger trykker "Apply"
- Så ændres indstillinger som brugeren har ændret

Resultat: Godkendt

Kommentar: Skærm opløsning ændres korrekt

### **13. Scenarie - Exit menu - Settings - Back**

*Givet at brugeren har trykket "Settings"*

- Når bruger trykker "Back"
- Så vises In game menuen

Resultat: Godkendt

Kommentar:

### **14. Scenarie - Exit menu - Settings - Default**

*Givet at brugeren har trykket "Settings" og har ændret mindst 1 indstilling*

- Når bruger trykker "Default"
- Så ændres alle indstillinger tilbage til default settings

Resultat: Godkendt

Kommentar: Indstillinger ændres til Default indstillinger korrekt.

### **Test af User Story 15 og 16 - Save Menu 15. Scenarie - Save Menu - Save Game**

*Givet at brugeren har trykket "Save game" fra Settings menu og ikke er i combat.*

- Når bruger vælger et gemt spil
- Og sætter navnet til det ønskede
- Og trykker på "Save Game" knappen
- Så genoptages spillet

Resultat: Godkendt

Kommentar:

### **16. Scenarie - Save Menu - Back**

*Givet at brugeren har trykket "Save game" fra Settings menu*

- Når bruger trykker "Back" i save game menuen
- Så vender spillet tilbage til In game menuen

Resultat: Godkendt

Kommentar:

### **Test af User Story 17-19 - Main Menu 17. Scenarie - Main menu - Load Game**

*Givet at brugeren er logget ind og har adgang til spillet.*

- Når bruger trykker "Load game"
- Så vises en liste af gemte spil på profilen

Resultat: Godkendt

Kommentar:

### **18. Scenarie - Main menu - Load Game - Load**

*Givet at brugeren er logget ind og har adgang til spillet og trykket "Load Game"*

- Når bruger vælger et gemt spil
- Og trykker "Load Game"
- Så loader spillet det gemte spil med det valgte game state

Resultat: Godkendt

Kommentar: Bruger starter i korrekt rum med korrekt inventory

### **19. Scenarie - Main menu - Load Game - Back**

*Givet at brugeren er logget ind og har adgang til spillet og trykket "Load Game"*

- Når bruger trykker "Back" i load game-menuen
- Så vender spillet tilbage til hovedmenuen

Resultat: Godkendt

Kommentar:

### **Test af User Story 20-26 - Spil spillet 20. Scenarie - Spil spillet - Start spillet**

*Givet at brugeren har trykket "New Game"*

- Når bruger anvender piltasterne på keyboard eller trykker på knappen på interfacet til at bevæge sig sydpå fra startrummet
- Så bevæger brugeren sig ind i rummet "Syd" for det rum de står i

Resultat: Godkendt

Kommentar:

### **21. Scenarie - Spil spillet - Enter nyt room**

*Givet at brugeren har trykket "New Game"*

- Når bruger går ind i et nyt rum ved brug af keyboard
- Så giver UI en beskrivelse af rummet spilleren befinder sig i

Resultat: Godkendt

Kommentar: Rum beskrivelser ændres korrekt når spiller går i nyt rum

### **22. Scenarie - Spil spillet - Combat**

*Givet at brugeren møder en fjende i et rum*

- Når bruger trykker "Fight!"
- Så ruller brugeren et tal mod fjenden om at skade fjenden
- Og derefter ruller fjenden et tal om at skade brugeren

Resultat: Fejl

Kommentar: Kravet er testes af unit tests, så der vides at det er korrekt, men det kan ikke bekræftes visuelt

### **23. Scenarie - Spil spillet - Combat - Flygt**

*Givet at brugeren møder en fjende i et rum*

- Når bruger trykker "Flee"
- Så flyttes brugeren tilbage til det rum han kom fra
- Og får ikke sit mistede liv tilbage igen

Resultat: Godkendt

Kommentar:

### **24. Scenarie - Spil spillet - Inventory**

*Givet at brugeren har trykket "New Game"*

- Når bruger trykker "Inventory"
- Så vises genstande som bruger besidder

Resultat: Godkendt

Kommentar:

### **25. Scenarie - Spil spillet - Interact**

*Givet at brugeren har trykket "Start spil" og at der ike er fjender i rummet*

- Når bruger trykker "Interact"
- Så kan bruger tage potentielle genstande i rummet til brugerens "Inventory"

Resultat: Godkendt

Kommentar:

### **26. Scenarie - Spil spillet - Level klaret**

*Givet at brugeren har fuldført det næstsidste rum*

- Når bruger bevæger sig ind i sidste rum
- Så får bruger en besked om at spillet er klaret og får mulighed for at gå til "Main Menu"

Resultat: Fejl

Kommentar: Bruger kan gå til main menu, men sidste beskrivelse giver en uklar besked.

## 12.2 ikke-funktionelle

### GUI

Table 15: Fuldført ikke funktionelle tests for GUI

Beskrivelse	Verificering	Resultat	Kommentar
GUI'en skal tilbyde valget mellem 3 resolutions	Visuel	Godkendt	
GUI'en skal kunne være fullscreen	Visuel	Fejl	Window bar kunne ikke fjernes i fuld-skærms opløsning
GUI'en skal kunne være windowed	Visuel	Godkendt	

### SOUND

Table 16: Fuldført ikke funktionelle tests for SOUND

Beskrivelse	Verificering	Resultat	Kommentar
Lyden skal kunne justeres mellem 0-100% relativt til PC'ens lyd niveau.	Auditorisk/Visuel	Godkendt	

### DATABASE

Table 17: Fuldført ikke funktionelle tests for DATABASE

Beskrivelse	Verificering	Resultat	Kommentar
Skal kunne gemme maksimalt 5 save games	Visuel	Fejl	Front-end begrænser antal gemte spil, databasen kan indeholde flere gemte spil end fem.
Skal kunne loade et spil indenfor maksimalt 5s	Visuel	Godkendt	
Skal gemme hvilke genstande man bruger lige nu	Visuel	Godkendt	
Skal gemme hvor meget liv man har tilbage.	Visuel	Godkendt	
Skal gemme hvilke fjender man har slået ihjel.	Visuel	Godkendt	
Skal gemme hvilke puzzles man har løst	Visuel	Fejl	Puzzles er ikke implementeret
Skal gemme hvilke rum man har været i.	Visuel	Godkendt	

**GAMEPLAY**



Table 18: Fuldført ikke funktionelle tests for GAMEPLAY

Beskrivelse	Verificering	Resultat	Kommentar
Spillets kort skal holde styr på hvilke rum man kan komme til for et givet rum.	Visuel	Godkendt	
Spillets kort skal kun vise de rum som spilleren har været i.	Visuel	Godkendt	
Spillets kort skal, hvis spilleren har været i alle rum vise alle rum.	Visuel	Godkendt	
Et rum kan have maksimalt 4 forbindelser til andre rum.	Visuel	Godkendt	
Et rum skal have mindst 1 forbindelse til andre rum.	Visuel	Godkendt	
Alle Rum skal kunne nås fra ethvert andet rum, måske ikke direkte, men man skal kunne komme dertil.	Visuel	Fejl	Tutorial rum er ikke tilgængeligt, når bruger har forladt det.
Spillerens rygsæk skal kunne indeholde alle spillets genstande.	Visuel	Godkendt	
Spilleren skal have mulighed for at bruge ét våben og ét skjold af gangen.	Visuel	Godkendt	
Spilleren skal have mulighed for at skifte hvilket våben og hvilket skjold der bruges.	Visuel	Godkendt	

## COMBAT

Table 19: Fuldført ikke funktionelle tests for COMBAT

Beskrivelse	Verificering	Resultat	Kommentar
Når spilleren/fjenden prøver at slå, rammer man kun hvis man på sit angreb slår højere end modstanderens rustningsværdi. Dette afgøres af et simuleret 20 sided terninge kast, hvortil der lægges en værdi til, korresponderende til spilleren/fjendens våben bonusser.	Visuel	Godkendt	Et angreb går også igennem hvis slaget er lige på PC's rustningsværdi
Hvis spilleren/fjenden rammer, bliver skaden bestemt af et/flere simulerede terningekast, afhængigt af hvilket våben der bruges.	Visuel	Godkendt	
Hvis spilleren/fjenden rammer, bliver skaden bestemt af et eller flere simulerede terningekast, afhængigt af hvilket våben der bruges.	Visuel	Godkendt	
Hvis spilleren, når nul liv inden fjenden, så dør spilleren og spillet er tabt.	Visuel	Godkendt	
Hvis fjenden, når nul liv inden spilleren, så dør fjenden og spilleren kan nu frit udforske rummet, som fjenden var i.	Visuel	Godkendt	
Hvis spilleren drikker en livseleksir bliver spillerens nuværende liv sat til fuldt	Visuel	Fejl	Livseliksir er ikke implementeret

**PERFORMANCE**

Table 20: Fuldført ikke funktionelle tests for PERFORMANCE

Beskrivelse	Verificering	Resultat	Kommentar
Spillet skal respondere indenfor maksimalt 5s	Visuel	Godkendt	
Spillet må ikke have mere end én kommando i aktionskøen af gangen	Visuel	Fejl	Aktions kø er ikke blevet implementeret

**STABILITY**

Table 21: Fuldført ikke funktionelle tests for STABILITY

Beskrivelse	Verificering	Resultat	Kommentar
MEANTIME BETWEEN FAILURE 1Time + 10Min?	Visuel	Fejl	MTBF er ikke testet

**DOCUMENTATION**

Table 22: Fuldført ikke funktionelle tests for DOCUMENTATION

Beskrivelse	Verificering	Resultat	Kommentar
Spillet skal have en manual/help page til hvordan alting virker.	Visuel	Fejl	Der er et tutorial rum i stedet for

**SECURITY**

Table 23: Fuldført ikke funktionelle tests for SECURITY

Beskrivelse	Verificering	Resultat	Kommentar
Username skal være mindst 6 characters	Visuel	Fejl	Der er ikke checks på Username længde
Username skal være unikt	Visuel	Godkendt	
Password skal være mindst 8 characters	Visuel	Fejl	
Password skal have store og små bogstaver	Visuel	Fejl	
Password må ikke indeholde username	Visuel	Fejl	Der er ikke checks på Password

## 12.3 Diskussion af testresultater

Projektets implementering og kravspecifikationer har produceret adskillige tests, som projektet i overvejende grad har bestået. De fleste features er blevet testet i henhold til kravspecifikationerne se section 5 og har produceret resultater, der indikerer at hver feature fungerer som ønsket.

Alle funktionelle tests er beskrevet med user-stories og er evalueret med et “Godkendt” eller “Fejl” Table 11 og evt. en forklarende kommentar. Ikke funktionelle test har fået en evaluering “OK” eller “FEJL”. Langt størstedelen af alle tests, for produktet, er bestået med få tests som fejler. Nogle tests fejler, da implementeringen ikke blev som forventet, mens andre ikke er blevet implementeret på grund af tidspres.

To eksempler er “puzzles” og “Delete Save Game”, der skulle have været implementeret, som en del af det færdige spil. Grundet tidspres er disse features ikke blevet implementeret, men i stedet for, er de blevet nedprioriteret til fordel for andre features, såsom “Combat” der er blevet vurderet mere essentiel. “Delete Save Game” er ikke blevet implementeret som specificeret i kravspecifikationerne men eksisterer i stedet for, som evnen til at overskrive allerede eksisterende save games. Her er kravene ikke blevet opdateret til at reflektere den anderledes implementering. Yderligere er MTBF heller ikke blevet testes da dette vil være besværligt at teste for denne type projekt.

Den stores success rate skyldes en insistens på tidlige integration mellem alle store komponenter for at sikre, at alt kommunikation mellem frontend, game engine, backend og databasen har fungeret fra et tidligt tidspunkt i projektet. I stedet for at integrere og teste alting samtidigt, er hvert komponent blevet gradvist integreret i projektet. Det har vist sig nemmere at integrerer mange små ændringer end at lave en stor integration til sidst i udviklingsfasen.

Lad der dog ikke herske tvivl om, at der er store huller i projektets test suit, det betyder at store features ikke er testet i tilstrækkeligt omfang. Features som load- og save game er implementeret og testet ved visuel bekræftelse, men der er en betydelig mangel på modultest til yderligere at bekræfte, at disse feature fungerer som ønsket (se section 5).

## 12.4 Funktionsbeskrivelse

### 12.4.1 Game Controller

**public async Task** Reset(void)

**Return Type:** Task

**Parameter:** void

**Comment:** Resetter game state til default tilstand.

**public async Task** GetRoomDescription(void)

**Return Type:** Task

**Parameter:** void

**Comment:** Initializere alle room descriptions fra databasen til de korrekte Rooms.

**public async Task** SaveGame(int id, string Savename)

**Return Type:** Task

**Parameter:** int id, string Savename

**Comment:** Gemmer game state og sender de via backend til databasen hvor det bliver gemt med det parameter id og navn.

**public async Task** LoadGame(int id)

**Return Type:** Task

**Parameter:** int id

**Comment:** Resetter game state til default, skaffer save game med det givet id og initializer game state til at matche Save game statet.

**public ILog** Move(Enum Direction)

**Return Type:** ILog

**Parameter:** Enum Direction

**Comment:** flytter spilleren i den angivet retning hvis og kun hvis en forbindelse eksistere mellem det nuværende room og den ønskede bevægelses retning.

**public void** EliminateEnemy(void)

**Return Type:** void

**Parameter:** void

**Comment:** Fjerner en enemy fra game state.

**public void** PickupItem(Item)

**Return Type:** void

**Parameter:** Item

**Comment:** Tilføjer et item til spilleren inventory og fjerner denne fra room.

#### 12.4.2 Log

**public void** RecordEvent(string key, string value)

**Return Type:** void

**Parameter:** string key, string value

**Comment:** Gemmer et vent i dictionaryen

**public string** GetRecord(string key)

**Return Type:** string

**Parameter:** string key

**Comment:** Udskriver et event ud fra den givene key string.

**public static ILog** operator +(ILog a, Log b)

**Return Type:** ILog

**Parameter:** ILog a, Log b

**Comment:** Tilføjer alle events fra en log til en anden log.

#### 12.4.3 DiceRoller

**public uint** RollDice(uint numoSides)

**Return Type:** uint

**Parameter:** uint numoSides

**Comment:** Simulere et terninge kast og returnere et tal mellem 1 and numoSides (Incl.)

**public uint** RollDice((uint NumOfSides, uint NumOfDice))

**Return Type:** uint

**Parameter:** (uint NumOfSides, uint NumOfDice)

**Comment:** Simulatore flere terningslag og returnere summen af disse kast.

#### 12.4.4 Room

**public void** AddPlayer(Player player)

**Return Type:** void

**Parameter:** Player player

**Comment:** Tilføjer en spiller til et room. smider MemberOverwrite Exception hvis spiller allerede er i room.

**public void** RemovePlayer(void)

**Return Type:** void

**Parameter:** void

**Comment:** fjerner en spiller fra et room.

**public void** AddEnemy(Enemy enemy)

**Return Type:** void

**Parameter:** Enemy enemy

**Comment:** tilføjer en enemy til et room.

**public void** RemoveEnemy(void)

**Return Type:** void

**Parameter:** void

**Comment:** fjerner en enemy fra et room.

#### 12.4.5 Combat Controller

**public ILog** EngageCombat(ref Player player, ref Enemy enemy)

**Return Type:** ILog

**Parameter:** ref Player player, ref Enemy enemy

**Comment:** Emulere en runde i combat, hvor spilleren slår først og hvis enemy dør, stopper combat ellers slår enemy på player.

**public void** Flee(void)

**Return Type:** void

**Parameter:** void

**Comment:** Stopper combat.

#### 12.4.6 Backend Controller

**public async Task<SaveDTO >** GetSaveAsync(int id)

**Return Type:** Task<SaveDTO >

**Parameter:** int id

**Comment:** de-serialisere save game og returner spillet til game controller.

**public async Task<RoomDescription >** GetRoomDescription(int id)

**Return Type:** Task<RoomDescription >

**Parameter:** int id

**Comment:** De-serialisere Room og returner room description.

**public async Task<List<SaveDTO >>** GetListOfSave(void)

**Return Type:** Task<List<SaveDTO >>

**Parameter:** void

**Comment:** De-serialisere alle save games og returner en liste af save games.

**public async Task** PostSaveAsync(SaveDTO save)

**Return Type:** Task

**Parameter:** SaveDTO save

**Comment:** Serialisere et save game og sender det til databasen Serialisere et save game og sender det til databasen.

**public async Task** PostLoginAsync(UserDTO)

**Return Type:** Task

**Parameter:** UserDTO

**Comment:** Serialisere en userDTO og sender login til databasen. Returner en token hvis login er succesfuld.

#### 12.4.7 Base Map Creator

**public void** GenerateMapLayoutFile(void)

**Return Type:** void

**Parameter:** void

**Comment:** Genererer en Map Layout File i current dictory.

**public void** GenerateEnemyLayoutFile(void)

**Return Type:** void

**Parameter:** void

**Comment:** Genererer en Enemy Layout File i current dictory.

**public void** GenerateItemLayoutFile(void)

**Return Type:** void

**Parameter:** void

**Comment:** Genererer en Item Layout File i current dictory.

#### 12.4.8 Enemy

**public virtual (uint hasHit, uint HasCrit)** Hit(Player player)

**Return Type:** (uint hasHit, uint HasCrit)

**Parameter:** Player player

**Comment:** simulere et angreb på spiller og returnere om angrebet rammer og om det er et crit.

**public virtual uint** Attack(ref Player player)

**Return Type:** uint

**Parameter:** ref Player player

**Comment:** udregner skade og updater player HealthPoint.

**public virtual uint** AttackCrit(ref Player player)

**Return Type:** uint

**Parameter:** ref Player player

**Comment:** udregner skade for et crit angreb og opdatere spillerens liv.

**public virtual void** TakeDamage(uint damage)

**Return Type:** void

**Parameter:** uint damage

**Comment:** Sænker enemies liv med en mængde lig damage, men kan ikke blive lavere end 0.

#### 12.4.9 Player

**public virtual (uint hasHit, uint HasCrit)** Hit(Enemy enemy)

**Return Type:** (uint hasHit, uint HasCrit)

**Parameter:** Enemy enemy

**Comment:** simulere et angreb på enemy og returnere om angrebet rammer og om det er et crit.

**public virtual uint** Attack(ref Enemy enemy)

**Return Type:** uint

**Parameter:** ref Enemy enemy

**Comment:** udregner skade og opdatere enemy HealthPoint.

**public virtual uint** AttackCrit(ref Enemy enemy)

**Return Type:** uint

**Parameter:** ref Enemy enemy

**Comment:** udregner skade for et crit angreb og opdatere enemys liv.

**public virtual void** TakeDamage(uint damage)

**Return Type:** void

**Parameter:** uint damage

**Comment:** Sænker player liv med en mængde lig damage, men kan ikke blive lavere end 0.

## 13 Fremtidigt Arbejde

Sammenlignes det færdige produkt med det produkt der blev diskuteret under idefasen, er der nogle funktionaliteter som ikke er blevet færdiggjort, men kunne have været implementeret. Spillet kunne have haft flere udvidelser såsom Puzzles, flere niveauer, character udvikling og quest items som kunne have givet bruger en mere underholdende oplevelse og mere avanceret gameplay. Dette høre ikke under et specifikt modul i systemet, men tilføjes over alle moduler, hvis nogen af de overstående features blev tilføjet.

F.eks. hvis der blev tilføjet flere niveauer, skulle der implementeres et nyt map i Game Controller og Front-end siden. Derudover skal der tilføjes en udvidelse af load og save game, så Back-end og database kunne gemme specifikt for det nye niveau, hvilket niveau er spilleren på, hvor langt brugeren er på det nuværende niveau og til sidst hvor meget spilleren har udforsket af tidligere niveauer.

Specifikt for Database og Back-End, kunne der tilføjes en lagring af data på en cloud-based storage fremfor lokal storage. Der blev valgt under forløbet at lagre dataen i en local storage, da der i midten af semestret var problemer med skolens licens af Microsoft. For ikke at komme ud for udfordringer senere i forløbet, blev der valgt at gå med den sikre løsning, at hoste databasen lokalt på enheden. Resultatet af et cloud-based lagring ville resultere i at spillet ville være mere tilgængelig for brugere



på forskellige enheder eller platforme.

Derudover kunne spillet laves til at kunne køre på andre styresystemer som f.eks. IOS eller Unix. Dette ville lade en bredere målgruppe spille spillet og derved give en større kundegruppe. Denne ændring ville dog resultere i at det valgte framework til Frontenden skulle skiftes til noget andet, såsom Unity. Dette ville betyde at frontenden og backenden skulle omskrives helt, men det kunne også give en generel bedre spil oplevelse.