Aarhus Universitet

BeerBong Battle

Rapport

Gruppe 5
201711532 Andreas Elgaard Sørensen
201710717 Mathias Tørnes Pedersen
201710709 Mads Stengaard Jørgensen
201710702 Mark Højer Hansen
201710692 Umut Sahin

Vejleder
Jesper Rosholm Tørresø
jrt@ase.au.dk

17. december 2019

AARHUS UNIVERSITY

Abstract & Resumé 1

1.1 Abstract

This paper is made by Information and Communication Technology Engineering students of Aarhus Univeristy, as the project assignment of the 4. semester. The paper describes the system 'BeerBong Battle', which is a system that makes it possible to compete in drinking beer against friends and strangers. It was strived upon making it possible for two people to compete against one another over the internet and teams competing against each other offline. The games takes place on a smartphone application, which is connected to a beerbong and a webserver. This paper covers the development methods, demands, design and implementation of the system. Furthermore it covers the tests of the individual system components, as well as tests for the system as a whole.

1.2 Resumé

Denne rapport er udarbejdet af Information og Kommunkationsteknologi studenrende fra Aarhus Universitet. Rapporten omhandler et semesterprojekt på 4. semester lavet af fem personer. Projektrapporten beskriver systemet 'BeerBong Battle', som er et system der gør det muligt at konkurrere i at drikke øl, igennem en ølbong på tid, uden at være samme sted. Det blev stræbet efter både at gøre det muligt at konkurrere mod venner og fremmede over internettet eller bare en gruppe på samme sytem i offline tilstand. Spillet foregår styres fra en smartphone applikation, som spiller sammen med en ølbong og en webserver. Denne rapport dokumenterer udviklingsmetoder, krav, design og implemenetering, og som det sidste er systemet testet både som indviduelle moduler og en samlet intgrationstest.

Ansvarsområder 2

Initialer	Navn
AS	Andreas Ellegaard Svendesen
MJ	Mads Steengaard Jørgensen
MH	Mark Højer Hansen
MT	Mathias Tørnes Pedersen
US	Umut Sahin

Ledelse	**AS**	**MJ**	**MT**	**MH**	**US**
Mødeleder	X				
Referant				X	X
Modul					
BeerBong Applikation			X		X
BeerBong Server		X			
BeerBong Database	X	X			
BeerBong Controller	X		X		
Rapport					
Resume og Abstract				X	
Forord			X		
Problemformulering	X	X	X	X	X
Krav	X	X	X	X	X
Afgrænsning					X
Metode					X
Analyse	X	X	X	X	X
HW Arkitektur	X			X	
SW Arkitektur	X	X	X	X	X
HW Design	X			X	
SW Design	X	X	X	X	X
Test	X	X	X	X	X
Resultater	X	X	X	X	X
Diskussion	X				
Konklusion				X	
Fremtidigt arbejde	X	X			X

Tabel 2.1: Tabel over ansvarsområder for alle gruppemedlemmer, hvor X indikerer ansvar for område.

Indholdsfortegnelse

1 Abstract & Resumé **2**
1.1 Abstract . . . 2
1.2 Resumé . . . 2

2 Ansvarsområder **3**

Indholdsfortegnelse **4**

Figurliste **6**

Tabelliste **6**

3 Ordforklaring **8**
3.1 Læsevejledning . . . 8

4 Forord **9**

5 Indledning **10**
5.1 Problemformulering . . . 10
5.2 System beskrivelse . . . 10

6 Metode **12**
6.1 SCRUM . . . 12
6.2 Udviklingsværktøjer . . . 12

7 Kravspecifikation **13**
7.1 Overordnede krav . . . 13
7.2 Funktionelle krav . . . 13
7.3 Ikke-funktionelle krav . . . 17

8 Afgrænsning **18**

9 Analyse **19**
9.1 Indledning . . . 19
9.2 BeerBong App . . . 19
9.2.1 Delkonklusion . . . 20
9.3 BeerBong Server . . . 20
9.3.1 Delkonklusion . . . 21
9.4 BeerBong Database . . . 21
9.4.1 Delkonklusion . . . 22

9.5 BeerBong Controller . . . 22
9.5.1 Hardware . . . 22
9.5.2 Software . . . 23

10 Arkitektur **25**
10.1 Overordnet arkitektur . . . 25
10.1.1 Context Diagram . . . 25
10.1.2 Container Diagram . . . 25
10.2 Hardware arkitektur . . . 26
10.2.1 BeerBong Controller . . . 26
10.3 Software arkitektur . . . 27
10.3.1 Indledning . . . 27
10.3.2 Domænemodel . . . 27
10.3.3 BeerBong App . . . 28
10.3.4 BeerBong Server . . . 29
10.3.5 BeerBong Database . . . 32
10.3.6 BeerBong Controller . . . 32

11 Design og Implementering **34**
11.1 Hardware design og implementering . . . 34
11.1.1 BeerBong Controller . . . 34
11.2 Softwaredesign og implementering . . . 35
11.2.1 BeerBong App . . . 35
11.2.2 BeerBong Server . . . 38
11.2.3 Database . . . 39
11.2.4 BeerBong Controller . . . 39

12 Modultest **42**

13 Integrationstest **45**
13.1 Indledning . . . 45
13.2 Integrationstabel . . . 46

14 Accepttest **47**
14.1 Funktionelle krav . . . 47

15 Resultater **49**
15.1 Indledning . . . 49
15.2 Modultest Resultater . . . 49
15.2.1 Hardware Modultest . . . 49
15.2.2 Software Modultest . . . 49
15.3 Integartionstest resultater . . . 50
15.3.1 Accepttest . . . 51

16 Diskussion **52**

17 Konklusion **53**

18 Fremtidigt arbejde **54**

Bibliography **56**

Figurliste

5.1 Systemskitse over BeerBong. 11

7.1 Aktør-kontekst diagram. 14
7.2 Use-case diagram. 15

10.1 Overordnet C4 diagram 25
10.2 Container diagram over systemet 26
10.3 BDD for BeerBong Controller. 27
10.4 IBD for BeerBong Controller. 27
10.5 Domænemodel for BeerBong Battle systemet 28
10.6 Component diagram for applikationen 29
10.7 Component diagram af BeerBong Server 30
10.8 Diagram over unitofwork 31
10.9 ER diagram for database struktur. 32
10.10Componentdiagram for BeerBong Controller 33
10.11State machine diagram for BeerBong Controller 33

11.1 Kredsløb over komponenter tilsluttet Raspberry Pi Zero W 35
11.2 MVVM design mønstrets opbygning 36

13.1 Testplan for integrationstest, hvor top-down metoden anvendes 45

Tabelliste

2.1 Tabel over ansvarsområder for alle gruppemedlemmer, hvor X indikerer ansvar for område. 3

3.1 Ordforklaring 8

7.1 Aktørbeskrivelse af brugeren 14

7.2 Aktørbeskrivelse af brugeren 14
7.3 Aktørbeskrivelse af brugeren 14
7.4 Use Case 2 - Online spil 16

12.1 Modultest af BeerBong Controller Hardware 42
12.2 Modultest af BeerBong Controller Software 43
12.3 Modultest af BeerBong App Software 43
12.4 Modultest af BeerBong Server Software 44

13.1 Modultest af BeerBong Server Software 46

14.1 Accepttest resultater for version 1.2 47
14.2 Accepttest af Use Case 2 - Hovedscenarie 48

15.1 Modultest resultater af BeerBong Controller Hardware 49
15.2 Modultest resultater af BeerBong Controller Software 49
15.3 Modultest resultater af BeerBong App Software 50
15.4 Modultest resultater af BeerBong Server Software 50
15.5 Integrationstest resultater 50
15.6 Accepttest af Use Case 2 - Hovedscenarie 51

Ordforklaring 3

Forkortelse	Forklaring
Analog Discovery	Multiværktøj, med oscilloskop, forsyning, funktionsgenerator og logic analyzer.
GUI	Graphical User Interface.
GPIO	General Purpose Input/Output.
USB	Universal Serial Bus.
WiFi	Standard for trådløst datanet.
MVC	Model-View-Controller.
REST	Representational state transfer.
API	Application programming interface.
MVVM	Model-View-ViewModel
IDE	Integrated Development Environment
XAML	Extensible Application Markup Language
RPI / RPIZW	Raspberry Pi Zero Wireless
JSON	Javascript object notation
HTTP	Hypertext tranfer protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WS	Websocket
O/RM	Object relational mapper

Tabel 3.1: Ordforklaring

3.1 Læsevejledning

Denne rapport er opbygget op efter C4 modellen, dette betyder at der startes med at se på sytemet som en hel context, og der så i den overordnede arkitektur bliver kigget på systemet inddelt i containere. Videre i den individuelle arkitektur zoomes der endnu længere ind, og kigges på hvad de indviduelle containere indeholder af komponeneter. Niveauet dybere end dette er at kigge direkte på klassediagrammer/Source kode, her henvises til billagsrapporten [1].

[1] 8 'Softwaredesign'

Forord 4

Den følgende projektrapport er skrevet i faget I4PRJ4-02. Projektet er udarbejdet af fem IKT-ingeniørstuderende på Aarhus Universitet. Rapporten er lavet af semestergruppe 5, bestående af Mathias, Mark, Mads, Umut og Andreas. Under semestret har gruppen været tilknyttet vejleder Jesper Rosholm Tørresø.
Rapporten er det endelige produkt udarbejdet igennem semesteret. Den består af tre dele: Projektrapport, Processrapport og Billagsrapport. Udover rapport delen, er et fysisk produkt også udviklet, som består af en mobil applikation og en modificeret ølbong.
Rapporten har afleveringsfrist d. 17/12/2019, og efterfølgende bedømmelse med forsvar den 21/01/2020. Den endelige karakter bedømmes ud fra rapporten, og vægtes ud fra den efterfølgende samtale.

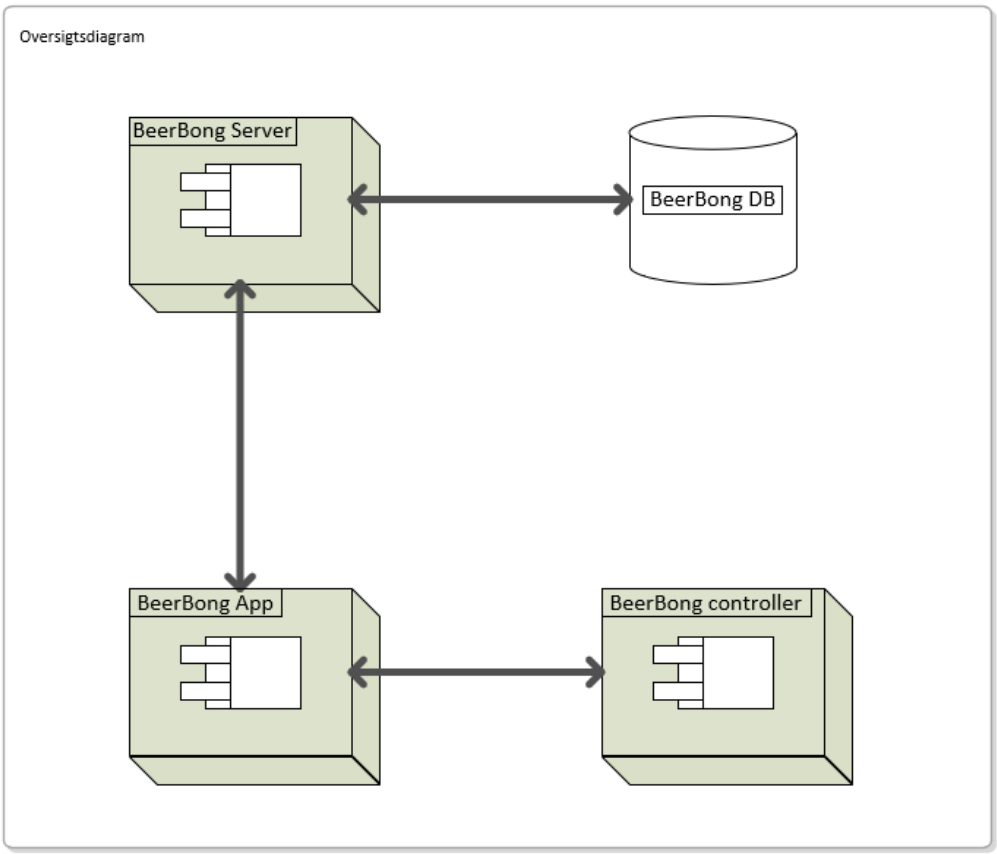
Indledning 5

5.1 Problemformulering

Vi har som projektgruppe valgt, at forsøge at binde fester sammen over hele landet vha. BeerBong Battle. BeerBong Battle skal gøre det muligt for individuelle fester at kunne komme i kontakt med hinanden og dermed konkurrere mod hinanden om hvem der hurtigst kan drikke en ølbong. Festdeltagerne skal kunne dyste mod hinanden vha. en application som er trådløst tilknyttet til en BeerBong. Her er det muligt at ønske en modstander vha. en server, efter begge modstandere har drukket deres øl, skal det være muligt at påbegynde et nyt spil. Det er også muligt at lave et lokalt spil til festen som ikke kører vha. en server, men hvor brugerne manuelt indtaster antal spillere og deres navne, hvorefter det er muligt at se hvem der hurtigst kan drikke en øl.

5.2 System beskrivelse

BeerBong Battle systemet skal med fokus på de festlige elementer gøre det muligt, at udfordre hinanden på kryds og tværs af fester. BeerBong Battle systemet fremgår af systemskitsen 5.1 hvori alle moduler i systemet fremgår. Dette er henholdsvis en mobil applikation, en elektronisk ølbong, en server samt en database. Mobil applikationen skal gøre det muligt for brugere at interagere med en grafisk brugergrænseflade, hvor de kan dyste mod andre på tværs af forskellige fester eller i offline tilstand. Med mobil applikationen kan brugerne se ranglister, spille online og spille offline. Med den elektroniske ølbong (BeerBong controller), kan festens deltagere måle hvor hurtigt det er muligt at drikke en øl. Samlepunktet for brugerne på kryds og tværs af festerne, er serveren som forbinder festerne med hinanden, og databasen som gemmer de relevante resultater for de enkelte fester.



Figur 5.1: Systemskitse over BeerBong.

Metode 6

Dette kapitel omhandler de anvendte metoder i projektforløbet og hvilken effekt de har haft i processens helhed. For yderligere information om dette henvises der til Procesrapporten [1].

6.1 SCRUM

Efter sidste semesters projektforløb har gruppen fået en del erfaring i brugen af SCRUM. Mange af de elementer som virkede fremmede sidste semester kom naturligt i brug dette semester, og alle medlemmer af projektgruppen forstod betydningen af at benytte SCRUM. Retrospektive møder blev styret med inspiration fra et foredrag hos Systematic fra forrige semester, og i sprintplanlægningen fik alle medlemmer mulighed for at præsentere deres planer eller bekymringer for den kommende sprint. Dette gav gruppen mulighed for at optimere sprintplanlægningen. Som backup havde gruppen også standup møder, hvor medlemmer kunne rapportere fremskridt og mulige problemer, hvorefter gruppen i fælleskab ville komme frem til en løsning. Dette kunne være at rykke et medlem fra en anden undergruppe, eller nedprioritere en opgave.

6.2 Udviklingsværktøjer

Gruppen har hovedsageligt gjort brug af Microsoft Visual Studio 2019 til at kode, der er dog også blevet gjort brug af Nano og Atom for at kode til BeerBong Controller, da den bruger en Raspberry Pi. Derudover er Swagger anvendt til at dokumentere API'et samt teste det. En anden måde at teste API'et på har været vha. Xunit, da dette arbejder godt sammen med .Net core platformen. Til applikationen er .NET frameworket Xamarin blevet anvendt, da det har gjort det muligt at skrive i C#/XAML, og programmere dette til Android. Yderligere information om de anvendte udviklingsværktøjer vil kunne findes senere i rapporten, eller i Bilagsrapporten [2]

Værktøjer til projektets proces
Zenhub var et nyt værktøj, men da alle havde erfaring med Trello fra tidligere semestre, var det ikke helt fremmede. Zenhubs indbyggede funktioner, blandt andet automatisk oprettelse af burndown charts, gav gruppen et overblik undervejs i udviklingsfasen. Derudover var de andre værktøjer intet nyt. Google Drev blev anvendt til administrering af dokumenter, og Facebook til kommunikation iblandt gruppemedlemmerne.

[1] 1 'Procesrapport'
[2] Afsnit 9 'Analyse'

Kravspecifikation 7

7.1 Overordnede krav

Til projektet er der blevet stillet en række krav til projektets funktionalitet. Dette er blevet gjort vha. MoSCoW analysen, hvor der overordnet for projektets funktionalitet er opstillet krav i forskellige prioriteringer. De opstillede krav og deres proritering, har givet en retning for projektet, samt afgrænset det i sådan et omfang, at den ønskede funktionalitet er blevet opfyldt. Derudover er der overvejet fremtiddigt arbejde til senere udvikling.
Følgende punkter viser nogle af de opstillede krav, de restende kan findes i bilagsrapporten [1].

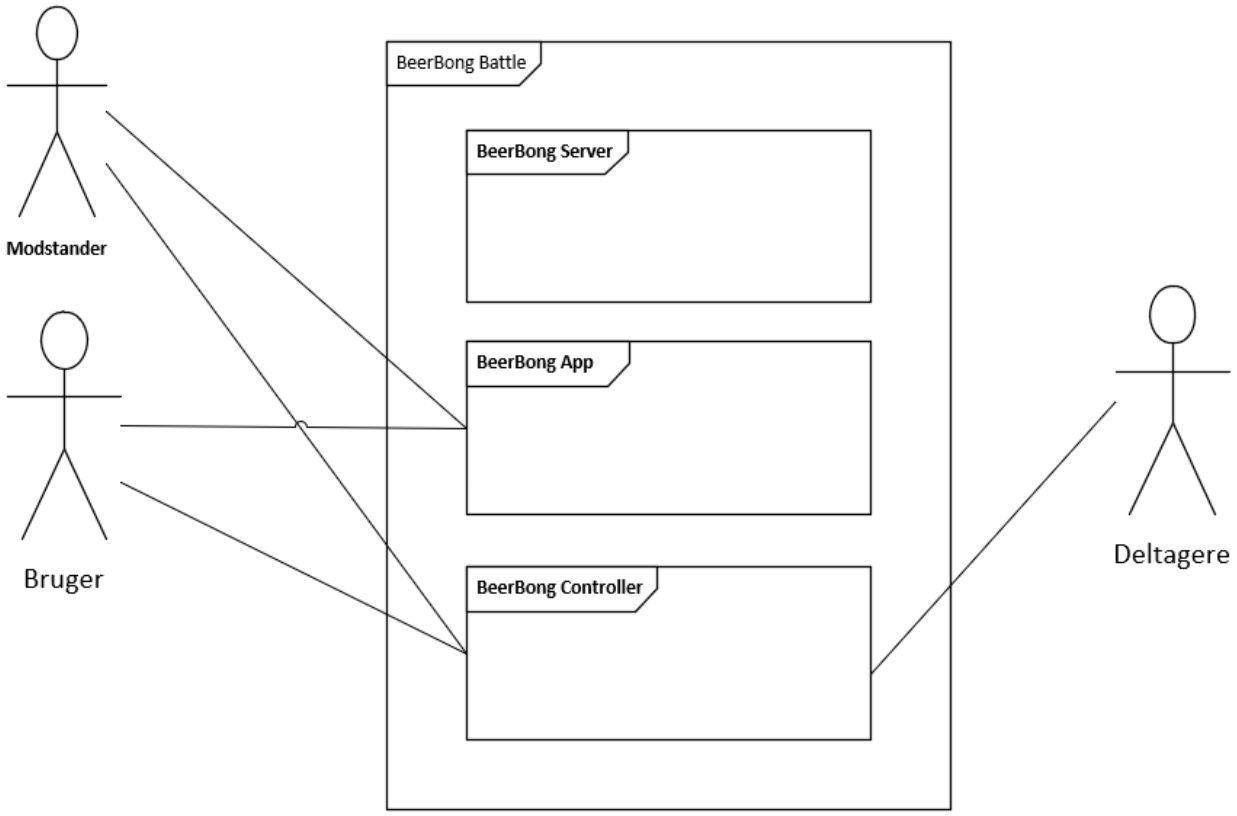
- BeerBong Server **skal** give mulighed for at oprette et personligt login.
- BeerBong App **kunne** visualisere tidsforskellen grafisk mellem spillerne.
- BeerBong Controller **vil ikke** kunne kende forskel på forskellige væsker.

7.2 Funktionelle krav

Dette afsnit indeholder de funktionelle krav som er sat for produktet BeerBong Battle. Dette indebære de overordnede diagrammer herunder aktør-kontekst og use-case diagrammer, samt den primære use case [online spil]. Disse diagrammer samt use casen beskriver de funktionelle krav til BeerBongBattle systemet.

På figur 7.1 fremgår aktør-kontekst diagrammet for BeerBong Battle systemet. Aktørerne der fremtræder på figur 7.1 er beskrevet i tabel 7.1, tabel 7.2 og tabel 7.3

[1] Afsnit 2.4 'Ikke-funktionelle krav'



Figur 7.1: Aktør-kontekst diagram.

Aktørnavn	Bruger
Rolle	Primær
Beskrivelse	Brugeren skal kunne tage en øl i ølbongen, og få taget tid igennem appen.

Tabel 7.1: Aktørbeskrivelse af brugeren

Aktørnavn	Deltager
Rolle	Sekundær
Beskrivelse	Deltageren skal kunne tage en øl i bongen, men tilgår nødvendigvis ikke appen.

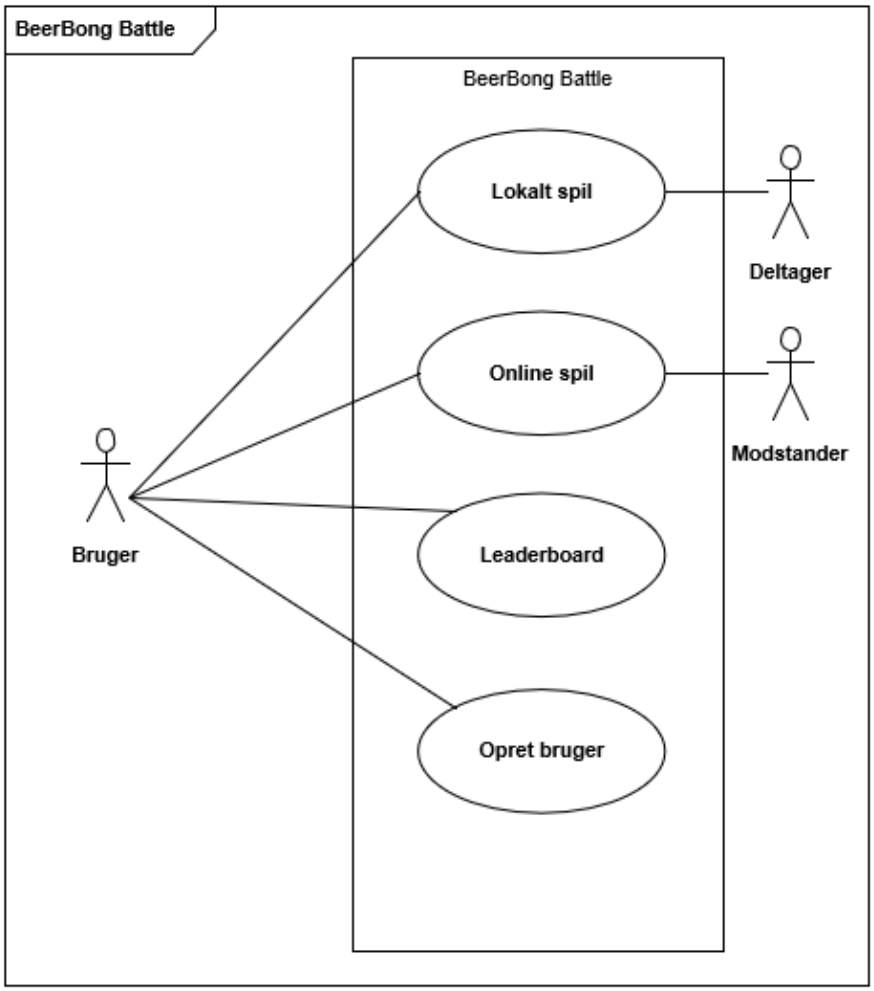
Tabel 7.2: Aktørbeskrivelse af brugeren

Aktørnavn	Modstander
Rolle	Sekundær
Beskrivelse	Modstanderen, som er en anden bruger, skal kunne dyste mod Brugeren, via Smarphone App og BB controlleren.

Tabel 7.3: Aktørbeskrivelse af brugeren

På figur 7.2 fremgår use-case diagrammet for BeerBong Battle systemet. Her fremgår hvilke aktører som anvender de opstillede usecases. For information omkring de andre usecases henvises der til bilagene[2].

[2]Afsnit 2.3 'Funktionelle krav'



Figur 7.2: Use-case diagram.

Use case 2 - Online Spil

På tabel 7.4 fremgår den primære usecase for BeerBong Battle systemet. Af use-case diagrammet 7.2 fremgård det hvilke aktører der anvender denne use case.

Navn	**Online spil**
Mål	At to spillere har konkurreret mod hinanden på forskellige enheder.
Initiering	BeerBongBattle initieres af brugeren.
Aktører	Primær: Bruger. Sekundær: Modstander
Antal samtidige forekomster	1 eller flere
Prækondition	Bruger er logget ind.
Postkondition	Bruger har drukket en øl og fået sin tid, samt modstanders tid.
Hovedscenarie	1. Bruger trykker på 'Start Online Spil'. 2. Bruger trykker på 'Find modstander'. 3. Applikationen søger efter modstander. 4. Applikationen finder modstander. 5. Applikationen viser ny side med fundet modstandernavn. 6. Applikation starter nedtælling på et minut. 7. Bruger fylder ølbong. [Extension 1: Ikke nok øl/Ølbong ikke fyldt inden for et minut] 8. Efter et minut fremkommer en 'Klar' knap på applikationen 9. Bruger trykker 'Klar' 10. Applikationen viser en nedtælling til at brugeren skal være færdig med at drikke en øl. 11. Bruger drikker al øl i ølbong. [Extension 2: Bruger kan ikke færdiggøre øl inden for given tidsramme] 12. App viser tider og vinder af spil.
	[Extension 1: Ikke nok øl/ølbong ikke fyldt inden for et minut] 1. Applikationen meddeler at der ikke er nok øl i ølbongen. 2. Use Case genoptages fra punkt 2. [Extension 2: Bruger kan ikke færdiggøre øl inden for given tidsramme] 1. Spillet slutter og use case starter fra punkt 10.

Tabel 7.4: Use Case 2 - Online spil

7.3 Ikke-funktionelle krav

Nedenstående sektion indebærer de ikke-funktionelle krav til BeerBong Battle systemet opstillet med (F)URPS+ modellen, som opdeler kravene i kvaliteter.

Functionality

- BeerBong Controller **skal** have en kapacitet på minimum 0,67 liter.

Usability

- BeerBong Server **skal** have et leaderboard med plads til minimum 20 tider
- BeerBong App **skal** have et leaderboard med plads til minimum 20 tider
- BeerBong Controller **skal** kunne registrere væske niveau i ølbong op til 33 cl.

Reliability

- Ølbongen **skal** overholde dimensionerne højde på minimum 100 cm og en diameter på maksimum 10 cm.

Performance

- BeerBong Controller **skal** kunne registrere tider i tidsrummet 0,5-60 sek [+/- 0,1 sek].

Supportability

- BeerBong Controller **skal** kunne forbinde til et internet netværk.

Afgrænsning 8

Projektet var allerede i starten ambitiøst, men gruppen var også enige om at produktets funktionalitet ville være det primære fokus. Efter den initielle brainstorm, hvor idéen kom på bordet, var der en del attraktive forslag til projektets funktionalitet, som gruppen måtte tage et mere realistisk blik på. Der var mange kreative idéer angående appliktionens design, såsom at vise tiden i real time, og måske endda illustreret med en animation af en ølbong, som tømmes på brugergrænsefladen. Det blev desuden valgt, ikke at lave endnu en Ølbong, for at teste multiplayer, eftersom gruppen mistede to medlemmer.
Efter disse to gruppemedlemmer forlod gruppen, måtte de resterende genoverveje deres ressourcer. Da meget af serveren og databasen allerede var i gang, besluttede gruppen sig for at fokusere på multiplayer funktionen i produktet, for at få så mange moduler som muligt i brug og nedprioritere det lokale spil, da dette ikke anvendte sig af server/database delen af projektet. Udover dette opstod der også problemer med Bluetooth forbindelsen, hvilket resulterede i anvendelse af en websocket forbindelse i stedet.

Analyse 9

9.1 Indledning

I følgende afsnit vil de initielle overvejelser samt de senere valg til applikationens udførelse forklares og uddybes[1].

9.2 BeerBong App

Med brugervenlighed som det primære fokus til selve spillet i produktet kom der mange attraktive idéer på bordet. Produktet skulle dog ikke bare være simpelt at bruge i hverdagen, men også i en mere specifik situation, nemlig en festlig sammenhæng, hvor sandsynligheden af at en bruger i forvejen var under alkohols effekt ikke kunne undervurderes. Det var derfor yderst vigtigt at gøre brugergrænsefladen, med opsætning samt udsendelse af information, så simpel som muligt. Med alle disse faktorer i tanke, var den generelle konsensus i gruppen at alt ville være nemmere med en mobil applikation. Med denne kunne en bruger forbinde til ølbongen som de ville med en trådløs audio enhed, e.g en højtaler.

Efter valg af brugergrænseflade skulle platform vælges. IOS blev hurtigt udelukket, da ingen i gruppen var besiddelse af en macbook, som er det eneste man kan udvikle native til IOS på. Problemet kunne løses med at køre en VM på sin pc, men der var ikke ønske om at gøre det mere kompliceret end højst nødvendigt.
Da platformen ikke skulle være IOS, var der kun et godt alternativ og det var Android. Native android applikationer udvikles normalt i IDE'en Android Studio, med sprogene Java eller Kotlin. Dette var heller ikke optimalt, da det ikke er programmerings sprog, som indgår på dette studie. Heldigvis findes der alternativer til udviklingsmiljøer af Android applikationer. Dette alternativ er visual studio hvor man kan udvikle native android applikationer med frameworket xamarin, som programmeres i XAML og C#. Da der både var erfaring i C# og XAML fra fagene på semestret, blev Xamarin valgt til udviklingen af applikationen.

Da udviklingsværktøjet også var bestemt, manglede forbindelsen til resten af systemet. To muligheder fremkom som det første, Bluetooth of Wifi. Som tidligere skrevet var brugervenlighed det primære fokus siden idéen til systemet først fremkom. Bluetooth var derfor go-to valget. En bruger skulle derfor kunne forbinde til systemet som de ville med et headset, hvilket var det bedst mulige valg for applikationen. Efter yderligere research viste det sig også at de fleste udviklere gjorde brug af Bluetooth i tilfælde hvor hastigheden af det sendte data ikke var en top prioritet, og da Wifi forbindelser kræver noget mere i form af strøm sammenlignet med Bluetooth. Det skulle så vise sig senere i projektet at valget af Bluetooth var forkert. Yderligere information om dette kan findes i afsnit[2]

[1] Yderligere information om analysen kan findes i afsnit 3.2 'BeerBong Applikation' i Bilagsrapporten
[2] reference til fremtidigt kommunikationsafsnit

Spillet skulle bestå af to dele. Et lokalt spil, hvor en ejer af produktet kan oprette et spil med et brugerdefineret antal spillere, hvorefter disse spillere kan gå på tur til at drikke, og til sidst vil deres tider displayes og en vinder kåres. Den anden del er en multiplayer funktion, hvor en ejer af produktet skal kunne spille imod en anden ejer. Denne game mode var tænkt til at være optimal i festivaller, eller i barer, hvor to personer hurtigt kunne tage et venligt spil mod hinanden. Til det lokale spil var mulighederne at bruge en lokal database, eller en remote. Efter nærmere undersøgelse fremkom det at Xamarin understøttede brugen af SQLite [3] til oprettelse af lokale databaser. Med dettet var det muligt at oprette og gemme data fra et lokalt spil midlertidigt på brugerens egen telefon. Dette var optimalt, da data i de lokale spil ikke skulle gemmes eller uploades nogen steder. Med samme research til online spil var det mest ideelt at bruge en remote server og database, da brugerne også kunne have mulighed for at uploade data til et leaderboard. Udover dette skulle brugerne også logge ind i applikationen for at spille mod hinanden, hvilket også blev gjort gennem disse. Mere om disse teknologier kan læses i afsnit 9.3 'BeerBong Server' og 9.4 'BeerBong Database'

9.2.1 Delkonklusion

Ved at grundigt undersøge flere muligheder for applikationens diverse aspekter, var det en simpel overgang fra initielt design til implementering. På trods af visse uforudsedte begivenheder i projektforløbet, der resulterede i afgrænsning fra de initielt ønskede dele, var det stadig muligt at komme i mål med et produkt der kunne være en stepping stone for fremtidig udvikling.

9.3 BeerBong Server

Nedenstående afsnit omhandler de overvejelser som har været på spil i forhold til hvordan backend for systemet skulle laves. I afsnittet reflekteres der over de forskellige muligheder samt valg der blev taget i forhold til BeerBong serveren. Yderligere information kan findes i bilagsrapporten [4].

Arkitektur

Funktionaliteten af BeerBong serveren skulle være bindeled mellem Presentation Tier(mobil applikationen) og database Tier(SQL databasen). Valget af Business Logic Tier (BeerBong server), bestod af en client-server arikitektur, hvor de forskellige funktionaliteter for BeerBong Battle systemet, var fysisk adskildt. Udfra de forskellige muligheder for at implementere Business Logic Tier for BeerBong Battle systemet, blev det valgt at anvende en REST arkitektur til at definere hvordan hvordan data skulle overføres mellem Presentation tier og database tier. Fordelene ved at anvende et REST arkitektur for BeerBong serveren er at den kan genbruges af alle mulige forskellige Presentation Tiers. Herudover implementeres der et uniform interface, som clienter kan anvende til at kunne tilgå resourcerne på databasen. Dette gøres med identifiers i form af URL's og herefter en specifikation af hvordan serveren håndtere resourcen med HTTP verbs.

Udviklingsværktøjer

Til at udvikle REST API'et var der mange muligheder. Herunder PHP, Node.js og mange andre. Ud af disse teknologier blev der valgt at arbejde med Asp.net core sammen med MVC frameworket. Dette var grundet at der var væsentlig større erfaring med Asp.net core, samt de

[3] Kilde: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/data/databases
[4] 3 'Analyse'

andre teknologier var forholdsvist ukendte. Herudover har Asp.net core platformen med Microsoft nogle suveræne deployment værktøjer, hvor det har været muligt med få klik at kunne deploy BeerBong serveren via microsofts cloud services.

I forhold til udviklingsværktøjer blev projektet udviklet i Visual Studio, grundet dens understøttelse af Asp.net core platformen, samt tidligere erfaringen med dette udviklingsværktøj. Dette kunne dog har været blevet udviklet i et hav af andre udviklingsmiljøer.

Sikkerhed

I forhold til sikkerhed på appen er det blevet valgt at anvende Asp.net cores memebership system Identity, som giver login funktionalitet til BeerBong serveren. Da systemet ikke har været større, har det været mest medgøreligt at implementeree Identity servicen på serveren. Alternativt kunne der have været anvendt tredje parts tjenester til at oprette et dedikeret API til login funktionaliteten på API'et. Dette kunne f.eks. have været 0Auth, der således ville have haft ansvaret for authentication og authorization. Dette ville have været et bedre alternativ hvis systemet havde været større med flere API'er og databaser.

Dokumentation

Til at dokumenterer API'et er det blevet anvendt Swagger. Til at generere dokumentationen baseret på swagger er der anvendt swashbuckle. Dette værktøj har bidraget til at dokumentere alle BeerBong serverens Endpoints. Herudover er Swagger også blevet anvendt til at integrationsteste API'et og se om de forskellige endpoints har fungeret korrekt.

Tests

Til at teste API'et er Xunit og moq frameworkene blevet anvendt, grundet de er nyere og velintegreret sammen med .net core platformen. Moq frameworket er blevet anvendt til at mock de forskellige afhængigheder i unit testene. Alternativt kunne Nunit og Nsubstitute frameworkene have været anvendt. Men da Xunit frameworket anbefales til test af .net core platformen, samt microsoft anvender Xunit internt, har dette framework haft prioritet.

9.3.1 Delkonklusion

Med analysen som fundament for BeerBong serveren, var springbrættet for API'et lagt. Med REST arkitekturen var retningslinerne for strukturen af serveren fastlagt. Herudover med de bevægende dele fastlagt såsom, Asp.net, MVC, Entityframework core, Microsofts identity og swagger var fundamentet for udviklingen BeerBong serveren lagt, og udviklingen kunne påbegyndes.

9.4 BeerBong Database

Dette afsnit vil afdække hvordan det er blevet valgt at udvikle databasen til lagring af data for systemet. Der vil blive gennemgået reflektioner og generelle overvejelser over hvordan det er blevet besluttet at lave databasens struktur. Det vil fremstå som en diskussion hvor alle relevante alternativer bliver opvejet mod hinanden for at finde den bedste løsning.

Til udvikling af databasen var der forskellige alternativer til udviklingen. Først skulle der besluttes hvorledes der skulle bruges en relationel database eller noSQL. Eftersom systemet ikke er stort og kompliceret var en noSQL database lavet vha. MonogoDB ikke nødvendigt.

Derudover kunne en relationel database let og hurtigt laves, da der tidligt blev stiftet bekendskab med denne på i løbet af semestret, modsat noSQL og MongoDB, som først kom senere.

Efter at valget faldt på en relationel database skulle der vælges en server til at køre denne. Eftersom databasen altid skulle være funktionel for alle brugere som måtte spille BeerBong Battle, var det vigtigt at anvende en server der er fejlfri og altid funktionel. Det oplagte valg blev en database hosted af Azure. Valget faldt på denne server, eftersom den er gratis for studerende at anvende, og at den fungerer godt med Visual Studio udviklingsværktøj.
Desuden skulle det besluttes hvorledes databasens entities og queries skulle udvikles. Entity Framework (EF-core) er Microsofts object-relationel-mapping værktøj (ORM) der giver programmøren nem adgang til at mappe database objekter til C# objekt modeller. Det er derfor nyttigt at anvende EF core frem for Direct SQL Query, da det fjerner en masse overflødig programmering.
Det blev derfor valgt at udvikle entities og queries vha. EF core, som herefter kunne mappes til en Azure database.

9.4.1 Delkonklusion

Efter analysen er færdiggjort for databsen til BeerBong Battle har gruppen fået en afklaring i forhold til valg af koponenter. Det er blevet fastlagt, at der skal anvendes Azure til server host og EF-core som udviklingsmiljø.

9.5 BeerBong Controller

Der vil i dette afsnit blive gennemgået en analyse af alt forarbejde til udvikling af BeerBong Controlleren. Valg af komponenter og alternativer hertil vil blive opstillet. Afsnittet er opdelt i to dele - hardware og software. Denne analyse er en kort gennemgang af den fulde analyse, som kan findes i bilagsrapporten [5]. I den fulde analyse, vil en mere dybegående analyse være beskrevet, hvor flere overvejelser er beskrevet.

9.5.1 Hardware

Microcontroller

Analysen af hardwaren startede ud med at opveje fordele og ulemper ved de mulige microcontrollere, som kunne blive anvendt. De oplagte valg var en version af Raspberry Pi eller Arduino. Begge microcontronllere har både fordele og ulemper. Der er tidligere blevet arbejdet meget med Raspberry Pi, hvorimod Arduino miljøet er mere ukendt og derfor indebar flere ukendte variabler. Valget faldt derfor på en version af Raspberry Pi. Herfra skulle det besluttes hvilken version der skulle anvendes. De forskellige versioner af Raspberry Pis kommer i sidste ende an på størrelsen på printboardet. Her var det vigtigt at finde en der mødte vores krav bedst muligt. En lang batterilevetid er vigtigt for projektet. Derudover skulle mikrocontrolleren understøtte enten Bluetooth eller WiFi. Derfor faldt valget på den mindste version der var, nemlig en Raspberry Pi Zero W, som både har et lavt energiforbrug og mulighed for implementering af enten Bluetooth eller WiFi.

Sensor krav

Under analysen af BeerBong Controlleren blev der gjort mange overvejelser omkring om hvilke type sensorer som skulle bruges til registrering af væske i røret. For at gøre hele designprocessen så overskuelig som muligt, skulle systemet have de samme spændingsniveauer på alle sensorer.

[5] 3.1 'BeerBong Controller'

Derfor blev der brugt meget tid på at finde sensorer, som kunne forsynes med 3.3 V fra Raspberry Pi. Denne løsning vil først og fremmest gøre batterilevetiden længere, desuden vil det ikke være nødvendigt at lave en levelconverter til dataoverførelse mellem sensor og Raspberry Pi.

Sensor forslag

Først blev det overvejet om der skulle bruges en flowsensor til at validere hvorvidt at der løb 33 cl igennem mundstykket på bongen. Dette valg ville mindske mulighed for at snyde, fordi brugeren hverken ville kunne hælde for lidt øl i bongen, men heller ikke ville kunne hælde det ud den forkerte vej, så de slap for at drikke det. Derudover er en flowsensor ekstrem præcis til både at måle væske og derfor kan den måle en meget præcis tid. Denne løsning blev desværre udelukket, fordi det var for svært at få en flowsensor som passede på mundstykket og samtidigt var præcis nok.
Den endelige løsning der blev valgt til registrering af væske samt tidstagning, blev en kombination af to løsninger. Til registering af tid blev der monteret en magnet på håndtaget, til at registrere, om ventilen på ølbongen var åben. Dette gør det muligt at tage en forholdsvis præcis tid, når brugeren begynder at drikke fra ølbongen. Til registrering af væske i ølbongen blev der monteret to lasere, der bruges til at tjekke om ølbongen er fuld og om der stadig er væske i ølbongen når brugeren er færdig med at drikke. En mere præcis beskrivelse af dette design valg kan ses i afsnit 11.1 'Hardware design og implementering'.

9.5.2 Software

Dette afsnit afdækker hvilke overvejelser som der har været omkring udviklingen af softwaren til BeerBong Controlleren. Der bliver gennemgået hvilke muligheder der har været omkring bl.a. programmeringssprog, Image til Raspberry Pi og udviklingsværktøjer.

Image

Først i analyse fasen blev der diskuteret, hvilket image som skulle bruges på den RaspberryPi som skulle fungere som controller. Mulighederne stod imellem Raspbian og et image udleveret fra undervisningen på 3. semester. Det image som blev udleveret ved undervisningen var et custom linux image som underviserne havde uarbejdet specifikt til undervisningen. Fordelene ved at bruge dette ville have været at gruppen allerede havde kendskab til dette. Raspbian har imidlertiddigt mange andre fordele, i kraft af at det er testet godt af andre udviklere. Det blev besluttet at udvikle BeerBong Controlleren vha. Raspbian, grundet dokumentation.

Programmeringssprog

Ift. valg af programmeringssprog var der flere muligheder at vælge imellem. De overvejede sprog var C#, C++, C eller python. C og C++ blev hurtigt valgt fra på baggrund af at der fokuseres på højere niveau af programmeringssprog dette semester. Der blev som gruppe taget et valg om at udvikle i C# på alle platforme, primært fordi det er det som bliver brugt i alle fag dette semester, og dermed lever bedst muligt op til læringskravene. Men da det ikke er muligt at bruge alle bibloteker fra .Net framworket på en Raspberry Pi, betød det at eksterne biblotekter ville være en nødvendighed. Forbindelse til GPIO portene, kunne ikke oprettes med .Net Framework standard bibloteker, så et eksterne biblotek Unosquare [6] blev brugt. Eftersom programmet blev udviklet på Windows og i C#, betød det at der var et godt altervativ til

[6] https://github.com/unosquare/raspberryio

crosscompiling. Der findes nemlig et development framework ved navn af Mono, som gør det muligt at køre .exe filer på et Raspbian styresystem.

Kommunikation

Noget andet som skulle overvejes grundigt, var hvordan at BeerBong Controlleren skulle forbindes med BeerBong Applikationen. Under overvejelsen af dette blev 3 muligheder opsat, at forbinde til WebApi'et, så appliaktionen kunne hente dataen derfra. En anden løsning var en direkte forbindelse til BeerBong App vha. Websocket. Begge disse løsninger betød, at controlleren skal have en internet forbindelse, hvilket gør brugervenligheden mindre, eftersom det er besværligt at tilslutte en Raspberry Pi til et netværk. Den sidste mulighed fjernede dette problem, ved at forbinde til BeerBong App vha. Bluetooth. Dette ville gøre det muligt at forbinde direkte til applikationen, uden at Controlleren skulle forbindes til et netværk. Denne løsning viste sig at være besværlig, som bliver nærmere beskrevet i afsnit 11.2.4. Det blev derfor besluttet, at implementere en Websocket server (Raspberry Pi) og en klient (BeerBong App), der er implementeret vha. node.js.

Delkonklusion

Efter analyse fasen var overstået var der blevet lagt et godt fundament for, hvordan BeerBong Controlleren skulle udvikles. Alle nødvendige sensorer var blevet fundet efter grundige overvejelser i forhold til alternativer. Derudover var der lavet et godt kenskab til det nye image (Raspbian) på Raspberry Pi, som vil skabe fundamentet for udviklingen. Yderligere var alle kommunikationsmetoder mellem BeerBong App og BeerBong Controlleren, hvilket viste sig at blive vigtigt i designfasen.

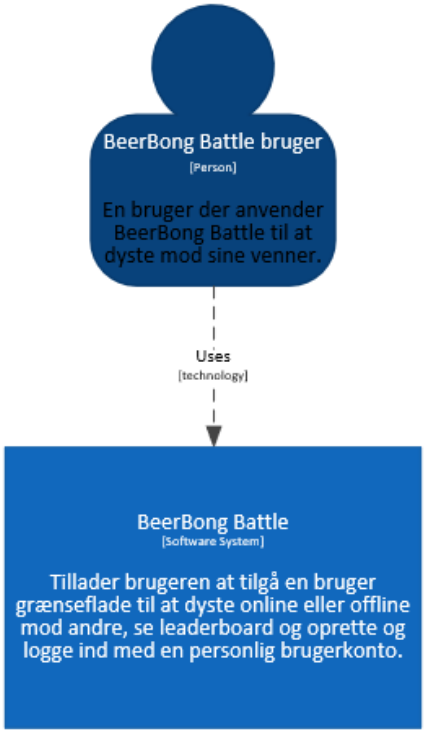
Arkitektur 10

10.1 Overordnet arkitektur

Dette afsnit afdækker den overordnede arktietktur for systemet. Herunder context samt container diagrammerne for BeerBong Battle systemet. Til at dokumentere arkitekturen for systemet er der anvendt C4 diagrammer, samt sysml til at beskrive hardware arkitekturen. Først gennemgås den overordnet arkitektur for systemet, hvorefter der dykkes ned i hver enkel container i systemet[1].

10.1.1 Context Diagram

På figur 10.1 fremgår context diagramet for systemet. Context diagrammet viser det store billede af systemet.



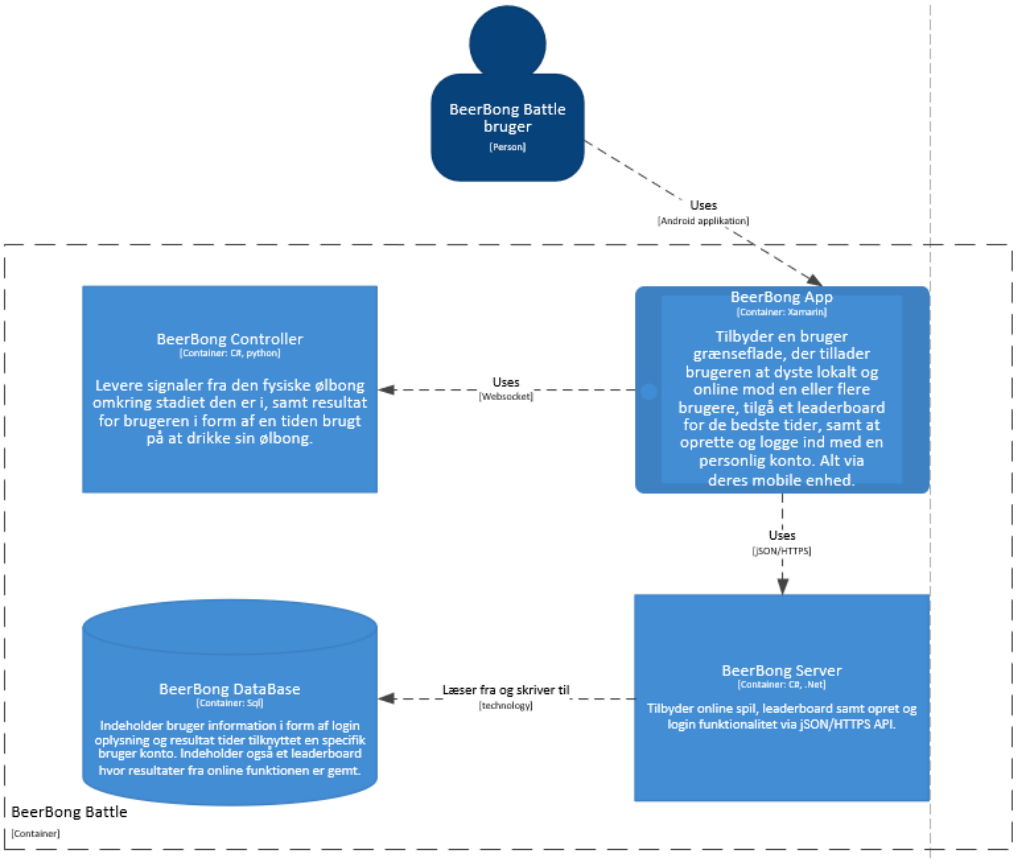
Figur 10.1: Overordnet C4 diagram

10.1.2 Container Diagram

På figur 10.2 fremgår container diagrammet for BeerBong Battle systemet. Efter dette zoomes der ind på BeerBong Battle systemet fra context diagrammet 10.1, fremgår de containere som

[1]Yderligere information kan findes under afsnit 4.2 'Overordnet arktitektur' i Bilagsrapporten

BeerBong Battle systemet består af. Ydermere baserer den overordnet arkitektur sig på et N-tier arkitektur stil. hvilket er en client-server arkitektur, hvor ansvars områderne er fysisk adskilt. Med BeerBong App som presentation tier, BeerBong server som Business logic tier og til sidst BeerBong DB som database tier.



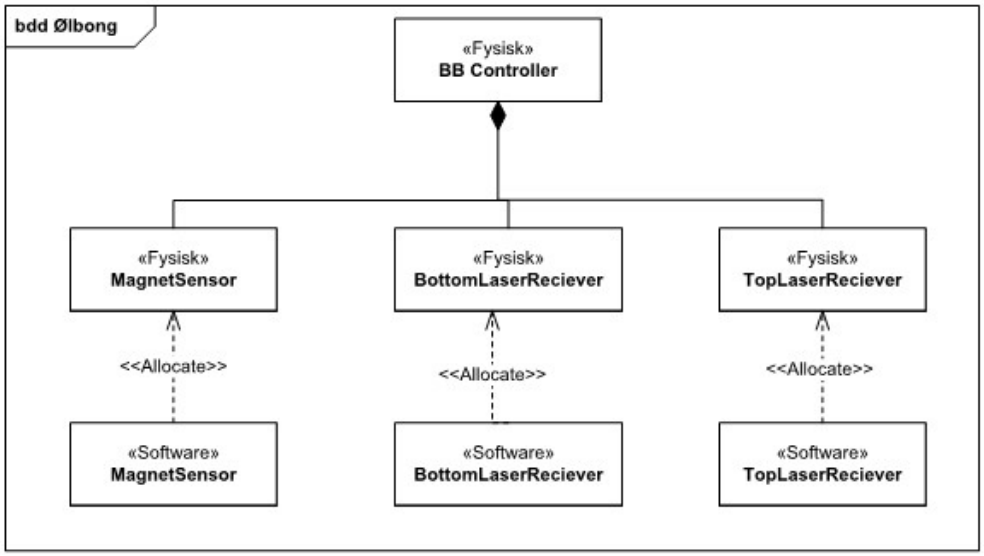
Figur 10.2: Container diagram over systemet

10.2 Hardware arkitektur

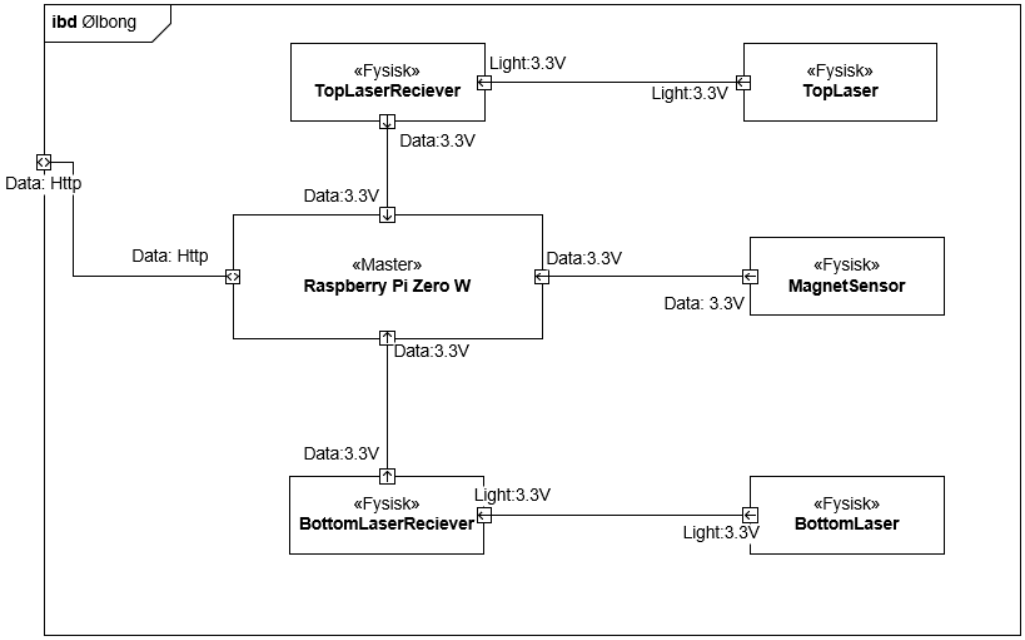
10.2.1 BeerBong Controller

Dette afsnit beskriver hardwaren i BeerBong Controller, som er det eneste der indeholder noget hardware i systemet. Det kan ses på figur 10.3 på BDD'et for Controlleren, at den består af 3 dele. Det er 2 lasersensorer til at detektere væske i ølbongen, en i bunden til at se om bongen er helt tom, og en der tjekker om der er 33 cl i. Den sidste sensor er en magnet sensor som registrer om ventilen er åben eller ej. På figur 10.4 ses det IBD som er udarbejdet ud fra BDD'et. Dette IBD beskriver de interne forbindelser, mellem komponenterne i modulet. Signalbeskrivelsen for dette diagram kan findes i Bilagsrapporten [2].

[2] Afsnit 5.1 'Signalbeskrivelse'



Figur 10.3: BDD for BeerBong Controller.



Figur 10.4: IBD for BeerBong Controller.

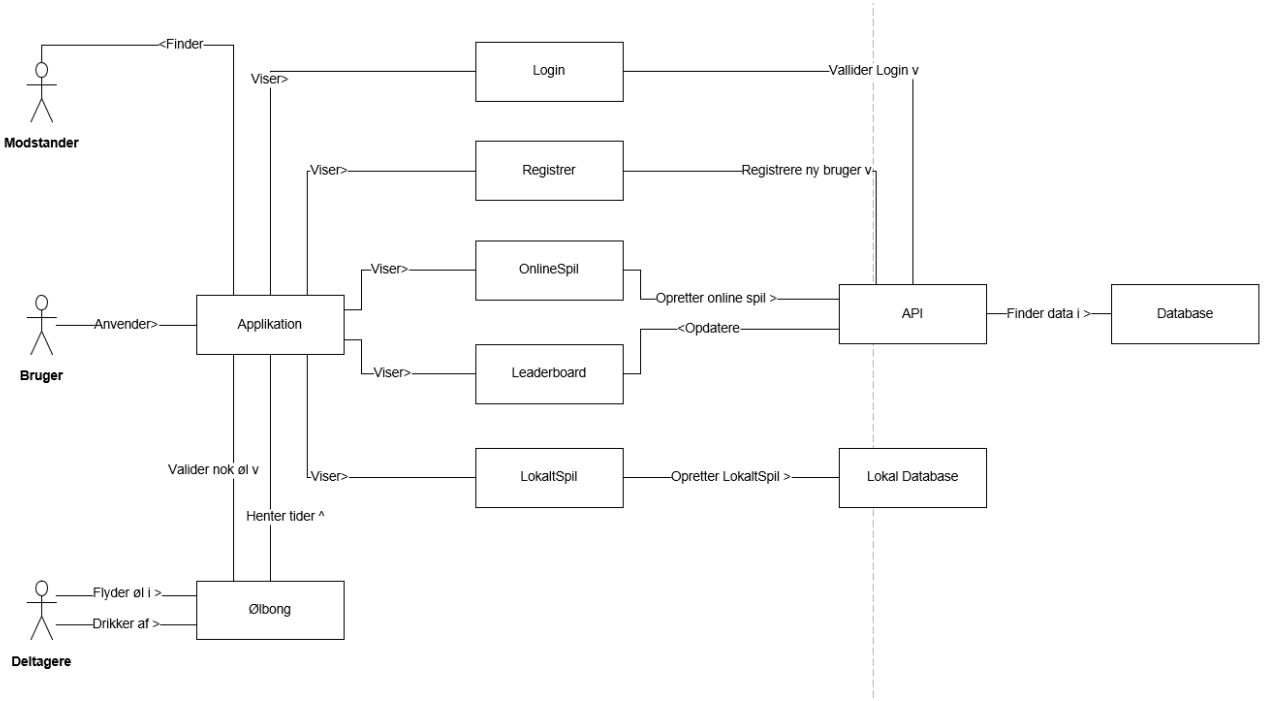
10.3 Software arkitektur

10.3.1 Indledning

Dette afsnit omhandler software arkitekturen for systemet BeerBong Battle. Arkitekturen vil tage udgangspunkt i C4 modellen på figur 10.2, hvor hver containers software arkitektur vil blive gennemgået i mindre afsnit. Hver container for hvert modul vil blive åbnet, og de mest væsentlige komponenterne inden i redegjort for.

10.3.2 Domænemodel

Nedenstående figur 10.5 viser domænemodellen for BeerBong Battle systemet. Herudover illustrerer domænemodellen systemets opbygning, samt dets anvendelse. Domænemodellen afspejler domænerne der fremgår i systemet, som findes i de opstillede usecases.

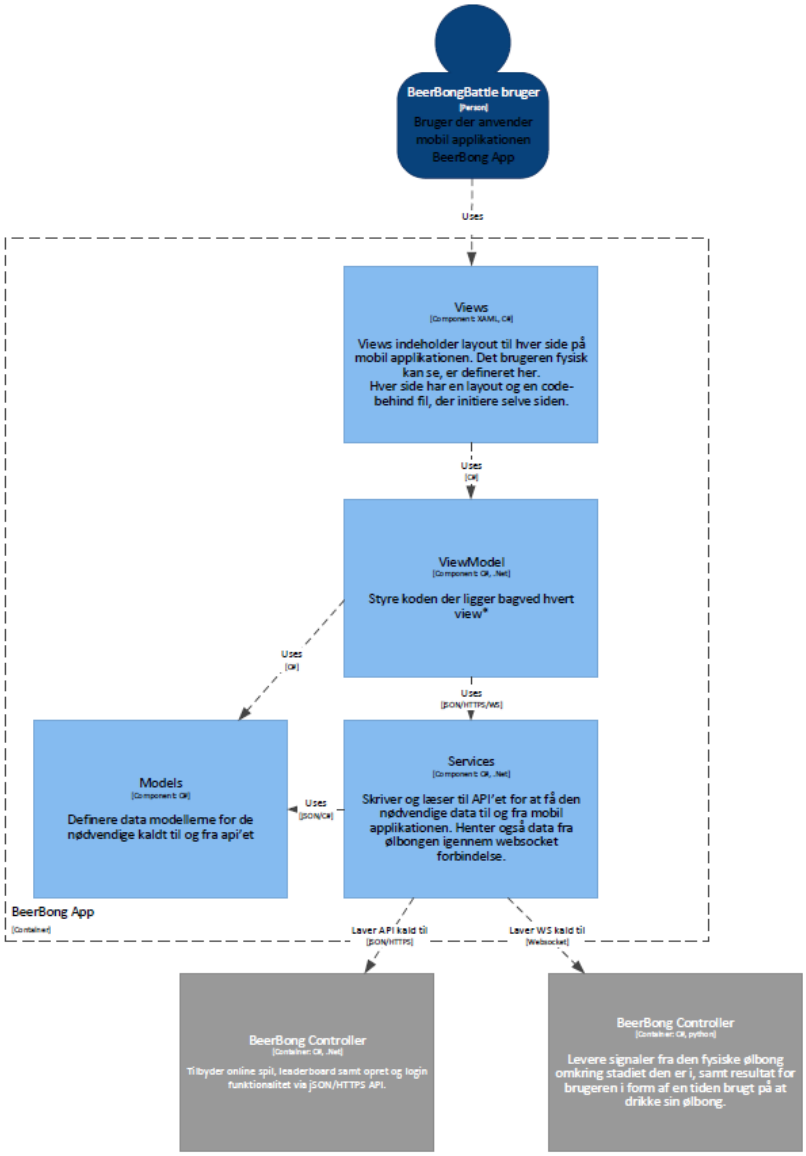


Figur 10.5: Domænemodel for BeerBong Battle systemet

10.3.3 BeerBong App

BeerBong Applikationen er opbygget af fire software komponenter som til sammen udgør containeren BeerBong App på figur 10.2. De fire komponenter består af, Views, ViewModel, Models og Services. De fire komponenter ses på figur 10.6, hvor der ses hvordan komponenter anvender hinanden. Tilsammen udgør de tre komponenter views, viewmodels og models MVVM/MVP arkitekturen som applikationen er udviklet med. Denne arkitektur form har hjulpet med at gøre koden for applikationen skalerbar, testbar og fjernet den tætte kobling som brugergrænsefladen har med den logik der styre grænsefladen. Dette vil uddybes yderligere i afnist 11.2.1 'BeerBong App'.
Overordnet fungerer arkitekturen for BeerBong applikationen således at views komponentet definerer det brugeren bliver præsenteret for, samt funktionaliteten for visse dele af brugergrænsefladen som ikke kan styres fra viewmodel komponenten.
Viewmodel komponenten håndterer brugerens input igennem data binding fra views komponenten til viewmodel komponenten, og udføre de nødvendige service kald igennem service komponenten.
Models komponenten definerer de data modeller som bliver anvendt til at skrive og læse fra api'et/brugergrænsefladen. Modellerne udfører ingen logik, men lader enten viewmodel komponenten eller BeerBong api'et håndterer dette. Service komponenten har ansvar for alt kommunikation til og fra api'et samt BeerBong controlleren. Meget af applikationen funktionalitet består i at lave kald til BeerBong api'et, derfor anvender viewmodels komponenten sig rigtig meget af service komponenten, ved at oprette et objekt af service klassen, og kalde service klassens funktioner igennem dette objekt i viewmodellen. Objektet af service klassen vil så igennem et http kald til api'et modtage en specifik form for data oftest defineret af en af modellerne i model komponenten. For en dybere forståelse for de forskellige kommunikations metoder, henvises der til bilag[3].

[3]9.2 'BeerBong App'

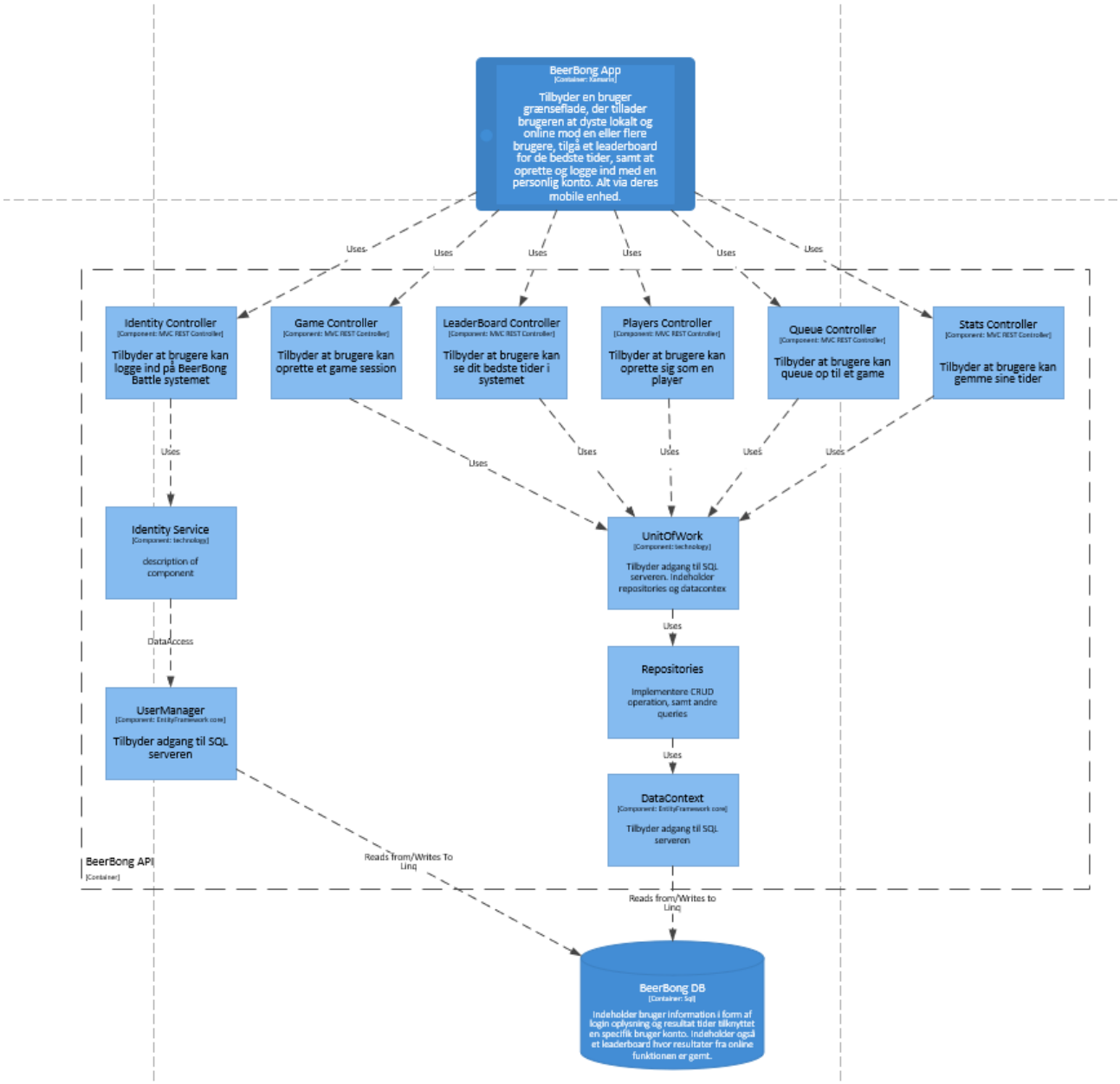


Figur 10.6: Component diagram for applikationen

10.3.4 BeerBong Server

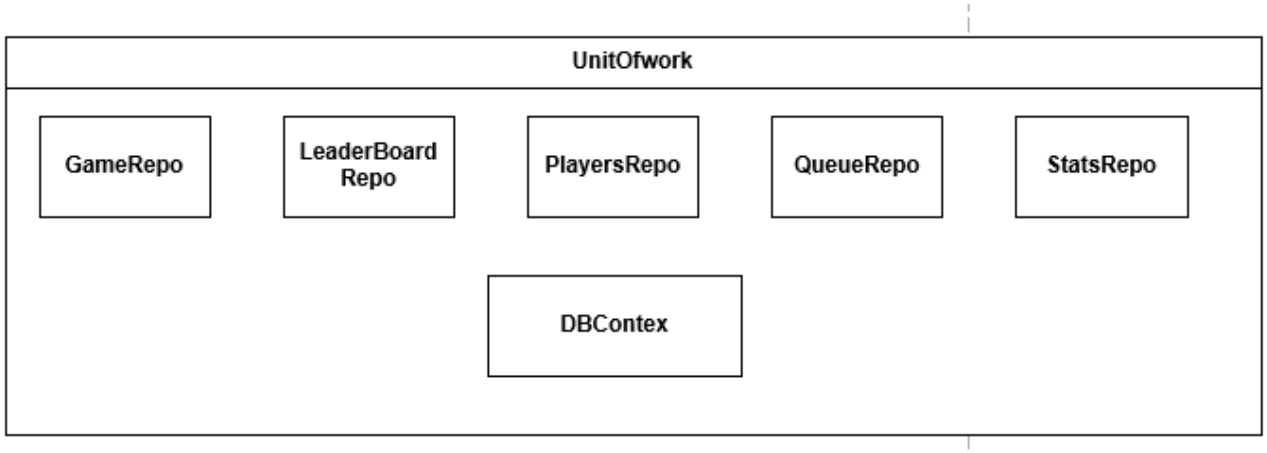
Nedenstående figur 10.7 fremgår komponent diagrammet for BeerBong API'et. Af komponent diagrammet vises de største byggesten som containeren, BeerBong server, består af. Yderligere beskrives komponenternes interaktion med hinanden, samt deres ansvar og implementationsdetaljer. Den arkitektoniske stil for containeren BeerBong serveren har bestået af REST arkitekturen for et API. REST definerer en række arkitektoniske principper, som man kan designe sine web services efter. Dette betyder man bygger sine services op efter HTTP methods eller CRUD operationer såsom POST, GET, PUT og DELETE. Et Restful API er stateless og passivt. Dette gøres ved at serveren reagerer på de endpoints som den tilbyder, som karakteriseret ved REST arkitekturen. Når en request kommer til serveren via de tilbudte endpoints, så loader serveren resourcen fra databasen vha. repositories og sender en repræsentation tilbage til klienten. Dette betyder også at "body" på HTTP requestene bruges til at overføre resourcens state. Serveren kommer aldrig til at gemme noget information om klienten, og er derfor stateless. Til bygge komponenterne BeerBong Server containeren består af, er der blevet anvendt MVC frameworket. Den eneste forskel API'et tager i forhold til anvendelsen af MVC frameworket er, at denne i stedet for at returnere et View, returnere et result i JSON format. Dette uddybes i afsnit 11.2.2.

Overordnet fungerer BeerBong serveren via de endpoints Api'et tilbyder klienterne. MVC frameworket router, model binder og model validere klientens request og sender det pågældende request ned til den controller, som udbyder aktionen [4].
Controllerne anvender unitofwork til at bruge businesslogikken i repostories til at hente data fra databassen igennem et persistence framework. Persistence frameworket (entityframework core) bliver anvendt til at mappe .Net domæne klasserne til SQL tables i databasen, hvilket fremgår af datacontext klassen [5]. Til sidst bliver dataen fra databasen retuneret til klienten i JSON format. Denne process foregår over HTTP protokollen. For mere hvordan HTTP er blevet anvendt i systemet henvises der til bilagsrapporten [6].



Figur 10.7: Component diagram af BeerBong Server

[4] ASP.NET Core IN ACNTION
[5] Entity Framework Core IN ACTION
[6] Kapitel 9 'Kommunikation'



Figur 10.8: Diagram over unitofwork

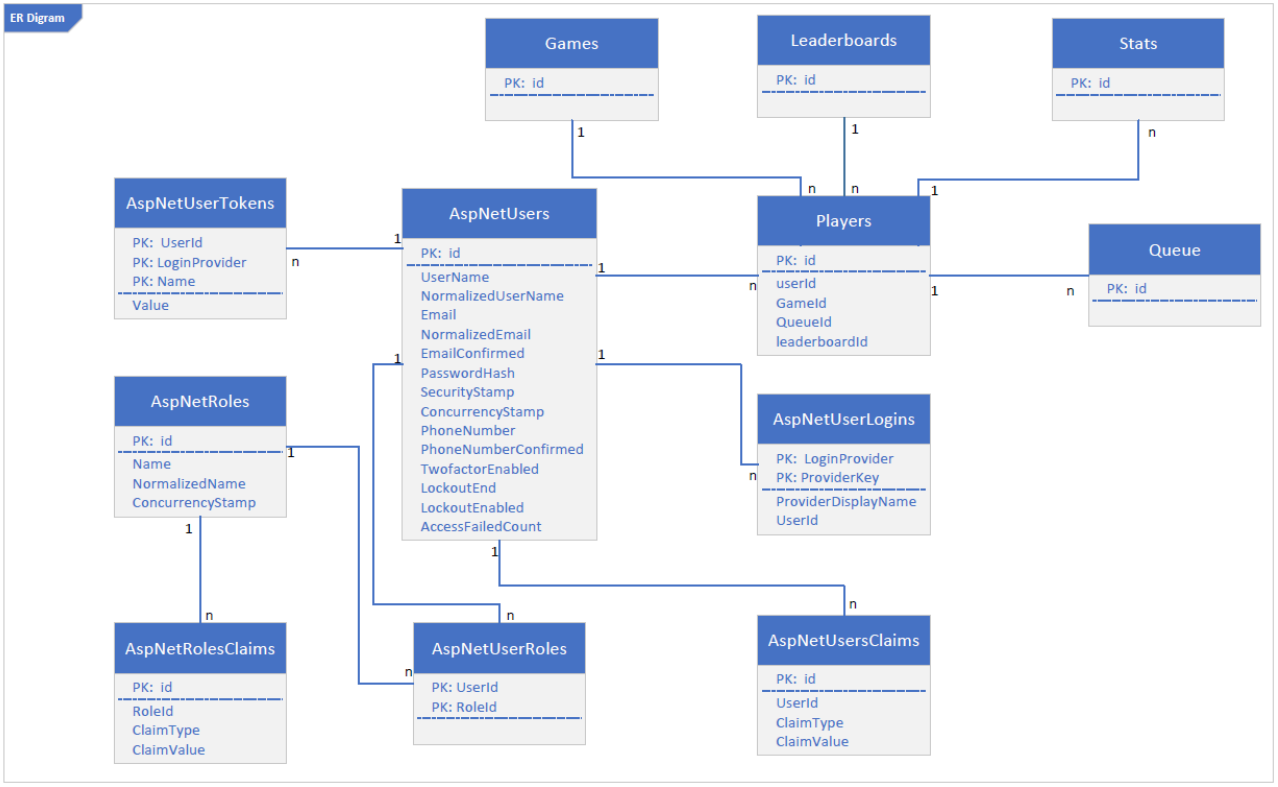
Nedenstående figur 10.8 viser Hvordan Unitofwork holder på de forskellige repositories samt datacontex klassen, og deraf kan de forskellige controllers igennem UnitofWork tilgå dataen i databasen.

10.3.5 BeerBong Database

ER Diagram

BeerBong Database er blevet udviklet vha. SQL. Derfor er der blevet udviklet et ER-diagram til at beskrive databasen bedst muligt. Databasen kører på Azure, hvillket gør at databasen altid er aktiv og er klar til at en bruger kan spille eller se leaderboardet. Derudover kan det ses på figur 10.9 hvordan det er blevet valgt at opbygge databasen. Relationerne mellem de forskellige entities kan ses, og om det er 'en til mange' eller 'en til en'. Databasen er lavet vha. Identity-biblioteket, som gør det muligt for brugere at logge ind på systemet og derefter gemme login.

Til håndtering af spillere er der blevet lavet en entity kaldet 'Players', som har relationer til henholdsvis 'Games', 'Leaderboards', 'Stats', 'Queue' og AspNetUsers, som indeholder alle gemte brugere der er registreret i systemet.

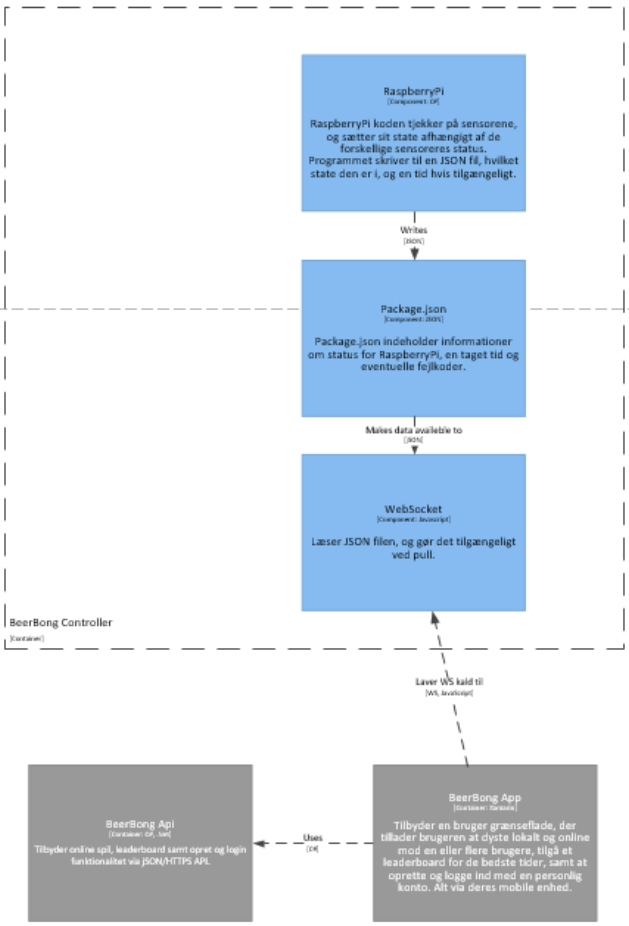


Figur 10.9: ER diagram for database struktur.

10.3.6 BeerBong Controller

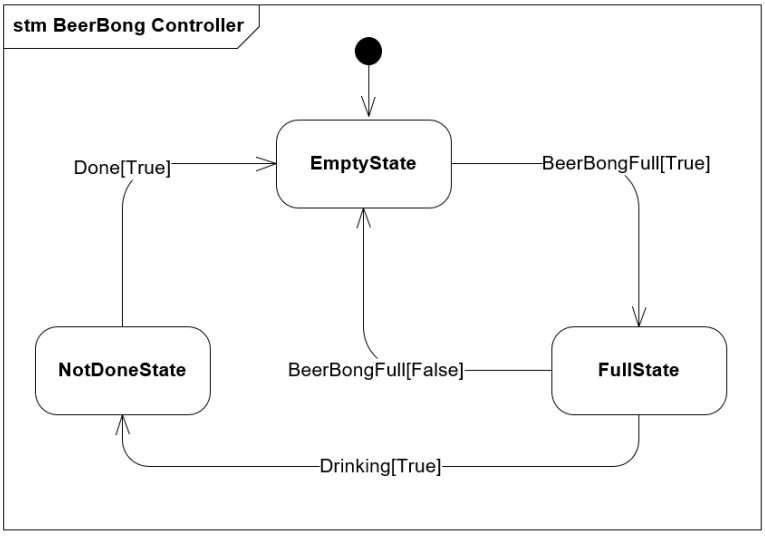
BeerBong Controlleren består af tre containere som sammensat giver BeerBong Controlleren. Figur 10.10 er et komponent diagram, som viser de tre komponenter RaspberryPi, Package og Websocket. Diagrammet beskriver de interne forbindeser for BeerBong Controller samt forbindelsen til andre containere. Det kan ses at den eneste forbindelse ud fra containeren er til BeerBong App. Klassediagram og sekvensdiagram for BeerBong Controlleren kan findes i bilagsrapporten[7].

[7] 6.2 'BeerBong Controller'



Figur 10.10: Componentdiagram for BeerBong Controller

BeerBong Controlleren er opbygget op efter et state pattern, og det er derfor relevant at bruge et state machine diagram til at vise programmets interne opførelse. Figur 10.11 er et state machine diagram som viser hvilke states RaspberryPi komponenten kan befinde sig i, og hvad der skal opfyldes for at der skiftes states. Mere om desigent af denne komponent kan gennemgåes i afsnit 11.2.4. Klassediagram samt sekvensdiagram for denne komponent findes i Billagsrapporten[8].



Figur 10.11: State machine diagram for BeerBong Controller

[8] Afsnit 6.2 'BeerBong Controller'

Design og Implementering 11

Dette afsnit indeholder design og implementations detaljer for de enkelte containere, som Beer-Bong Battle systemet består af. Dette inkluderer design valg for både software og hardware, samt en begrundelse for disse.

11.1 Hardware design og implementering

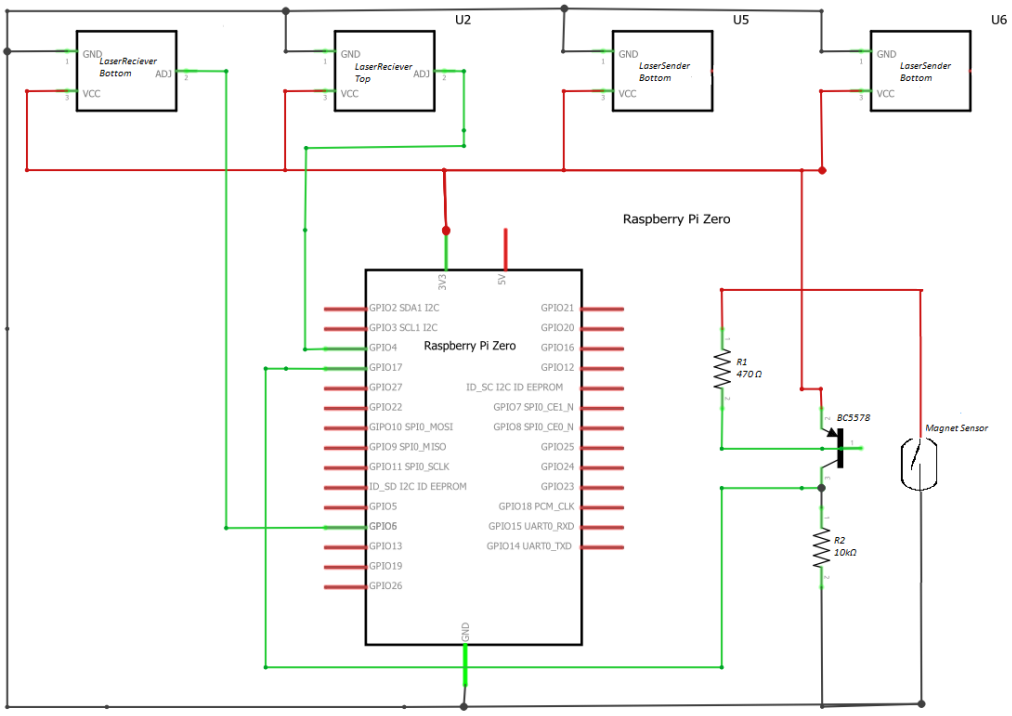
11.1.1 BeerBong Controller

I følgende afsnit vil alt hardware design blive gennemgået, som er lavet på baggrund af konklusioner fra analysen i afsnit 9.5.1. Yderligere infomation kan findes i bilagsrapporten [1].

På figur 11.1 ses kredsløbsdiagrammet for BeerBong Controlleren. Controlleren består af en Raspberry Pi Zero Wireless, som er forsynet af en powerbank. Raspberry Pi forsyner fem komponeneter - to Laser Receivers, to Laser Sendere og en Magnet Sensor. BottomLaserReceiver er forbundet til Raspberry Pi'ens GPIO 6 port, og TopLaserReceiver er forbundet til GPIO port 4. Hvis laseren registreres i disse receivere sendes et højt signal til den givne GPIO port. Den sidste komponent som er er forbundet til denne Raspberry Pi er det mindre kredsløb til magnet sensoren, som er en magnetisk reed switch. Denne del af kredsløbet består af reed switchen, to resistorer og en tansistor.

Når magneten kommer tæt på, bliver de to elementer inde i reed switchen trukket sammen, og det medfører at transistoren skifter til at være 'on'. Når de så er trukket sammen sker der det at R1 trækker basen af transistoren til lavt, så transistoren tænder. Så løber der spænding fra emitteren til collectoren som er forbundet til GPIO14, og det kan læses som et højt signal på Raspberry Pi.

[1] Kapitel7 'Hardwaredesign'



Figur 11.1: Kredsløb over komponenter tilsluttet Raspberry Pi Zero W

Delkonklusion

Det var vigtigt at lave hardwaren så hurtigt som muligt, så tiden kunne prioriteres på andre måder. Den software der er blevet lavet passer derfor godt med det forventet resultat. De sensorer som er anvendt, opfylder alle de krav som er opstillet i afsnit 7.

11.2 Softwaredesign og implementering

11.2.1 BeerBong App

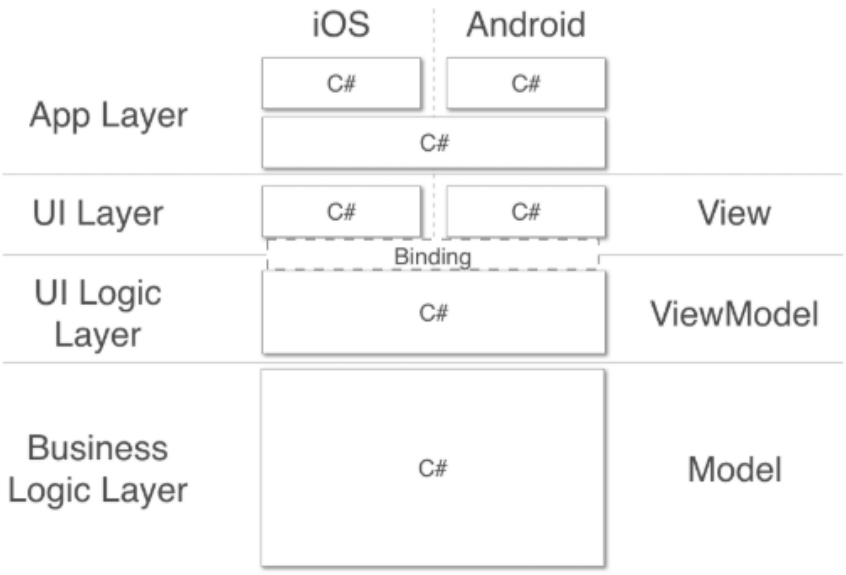
Dette afsnit vil gennemgå de designmæssige valg foretaget for BeerBong applikationen. Dette indebærer design mønstre anvendt i implementeringen, og dets betydning for applikationen. Herudover vil der blive gået i dybden med særlig vigtige komponenter/klasser, og forklaret særligt vanskelige metoder og deres funktioner.
Der vil også blive forklaret designmæssige mangler, og hvordan implementeringen kunne blive optimeret.

BeerBong applikationen er udviklet i IDE'en Visual Studio, i Xamarin frameworket, som er en forlængelse af .NET platformen der giver adgang til en række værktøjer og biblioteker til app udvikling. Xamarin's XAML sprog er anvendt til at designe layout'et brugeren bliver præsenteret for, og design mønstrene MVVM og MVP som er typiske mønstre at bruge til udvikling af applikationer er anvendt til implementeringen af BeerBong applikationen.

MVVM/MVP

Model View View-Model design mønstret er anvendt til implementering af tre views i BeerBong applikationen, som også kort er beskrevet i arkitektur afsnittet 10.3.3. I implementeringen af MVVM, indeholder view komponenten, view klasserne, som definere brugergrænsefladen. View klasserne fungere som aktive grænseflader, der behandler begivheder og håndterer data-binding, som kræver kendskab til model og viewmodel klasserne. Dette kendskab opnås ved

at mappe og lave referencer igennem data-binding fra view klasserne til viewmodel klasserne. Eksempelvis er knappen på brugergrænsefladen som opretter en bruger i opretbruger use casen data-bundet til en kommando i viewets tilhørende viewmodel. Ved tryk på knappen, vil der igennem data-bindingen blive kaldt kommandoen defineret i viewmodellen, og denne kommando vil så udføre sin interne logik/funktionalitet. I dette tilfælde anvendes også model komponenten, der definerer modellen som data'en fra view'et bliver bundet til når kommandoen opretbruger bliver kaldt. Viewmodellen laver så et service kald igennem et objekt af service komponenten, og skriver den data bundet fra viewet til modellen til api'et, hvorefter api'et opretter brugeren hvis alt er som det skal være. Sådan fungerer MVVM designet overordnet for applikationen når man opretter en bruger, logger ind og tilgår leaderboard. Se figur 11.2 for et overblik over MVVM designets opbygning i applikationen. Ønskes der et mere detaljeret indblik i komponenternes indbyrdes kommunikation henvises der til bilagrapporten [2], der viser sekvensdiagrammer for de forskellige use cases.
De resterende views, Forside, Multiplayer og Lokalspil er implementeret med MVP design mønstret. Forskellen imellem de to design mønstre er at MVP designet ikke indeholder en viewmodel klasse, men istedet en presenter klasse. Denne presenter klasse er i implementeringen for BeerBong applikationen den klasse som er tilhørende til hvert view, da hvert view består af en xaml og en cs fil. Denne presenter klasse får input direkte fra dens tilhørende xaml fil, og håndterer alt logikken. Dette skaber en meget tæt kobling imellem brugergrænsefladen og koden der styrer funktionaliteten, hvilket gør koden, i dette design mønster, utroligt svært at skrive automatiseret tests til. Til gengæld er dette design nemmere at implementere da der ikke gøres brug af data binding, og fordi funktionaliteten ligger så tæt på selve brugergrænsefladen. Derfor er multiplayer udviklet i dette design, da det var vigtigt at denne use case fungerede. En dybere forståelse for designet og koden kan ses i bilag.[3]



Figur 11.2: MVVM design mønstrets opbygning

At alle views ikke er implementeret med mvvm design mønstret skyldes tidsmangel og at funktionalitet var en utrolig vigtig faktor for projektet.
Ønskes en dybere forståelse for designet se bilag. [4]

[2] Afsnit 6.3.3 'Sekvensdiagram'
[3] 8 'Bibliography'
[4] Afsnit 8.1 'Softwaredesign og implementering'

Singleton

BeerBong applikationen anvender utrolig meget data der skal deles imellem komponenterne. Dette er i implementationen løst med brug af singleton design mønstret, hvor der er en klasse App.cs som definerer en række statiske variabler. De statiske variabler gør det muligt for komponenterne at dele data imellem dem, ved at lade en klasse i en komponent sætte variablens data, og en anden komponent kan så oprette et objekt af app klassen og tilgå denne variable. Dette bruges eksempelvis i applikationen når der i online spil use casen skal sættes en tid til en spiller. En spiller bliver tildelt et id når brugeren logger ind, og dette id bliver gemt i en statisk variable i app klassen. Når der i onlin espil så skal sættes en tid til spilleren, bliver spiller id'et fra da spilleren loggede ind, sammenlignet med de spiller id, som ligger i spilresultatet, og er der et match imellem de to, ved applikationen hvilken tid denne spiller skal have.
Der er visse ulemper som brugen af singelton har bragt med sig. Det er bla. svært at lave unit tests når man anvender singleton, for hvis en test anvender og sætter en statisk variable, så skal de efterfølgende tests der anvender samme variable, sættes i den korrekte rækkefølge. Hvis der eksempelvis først laves unit tests for onlinespil use casen, så vil disse fejle da playerid'et ikke er blevet sat da der ikke er faket et bruger login. Derfor er det nødvendigt at unit tests af bruger login køres inden test af onlinespil kan køres.

SOLID

Med mangel på ressourcer efter gruppen mistede to medlemmer, har brugen af SOLID principperne ikke været en top prioritet, hvilket kan være problematisk i fremtiden, hvis appen skal udvides og forbedres. Flertallet af applikationens moduler har i øjeblikket mere end et ansvar, hvilket gør disse moduler sværere at vedligeholde i tilfælde af ændringer.

Hvis det nu skulle ske at appen skulle videreudvikles, kan det heller ikke udelukkes at nogle ting skulle opdateres. Dette kunne være en visuel ændring såsom at tilføje farver til de enkelte spillere for at bedre se forskellen på hvem der er hvem, eller gøre det muligt at spille gruppe mod gruppe i multiplayer, som vil blive uddybet senere. På nuværende tidspunkt skal der ændres i eksisterende kode. Ved at gøre brug af arv, ville det være muligt at override den metode der opretter en enkelt spiller, og i stedet gøre det muligt at oprette en gruppe af spillere.

Som applikationen er nu, er der ikke et tilfælde hvor Liskovs substitutions princip kan bidrage meget, men hvis spillet skulle udvikles til at håndtere forksellige typer alkohol, såsom shots, ville det hjælpe at have en overklasse der eksempeltvis hed 'Alkohol' hvorefter det forskellige typer af drikkevarer tilføjes som subtypes af denne. Hvis denne implementering ikke bruges ville det også bryde open-close princippet.

Brugen af interfaces er også begrænset i applikationen. Dette er et dependency problem, da det kan ende ud i at visse moduler er afhængige af metoder som de slet ikke gør brug af.

Delkonklusion

Anvendelsen af MVP design mønstret har gjort det nemt at implementere applikationens funktionalitet, men svært at skrive unit tests til. Derfor blev MVVM mønstret implementeret for visse views, og gjorde det testbart og fjernet den tætte kobling imellem brugergrænsefladen og funktionaliteten som MVP bringer. Havde der været et større kendskab til udvikling af mobil applikationer og design mønstre, var MVP mønstret blevet helt undgået og kun anvendt MVVM, men grundet tidsmangel skete dette ikke.

11.2.2 BeerBong Server

Nedenstående afsnit gennemgår de designmæssige valg for BeerBong serveren. Dette indebærer arkitektur og valgte design mønstre, og deres betydning for en vedligeholdig, skalerbar og testbar kode, som opretholder SOLID principperne for god kode. Herudover vil alternativer til designet også gennemgåes, samt de fordele og ulemper de valgte designløsninger har givet.
Det overordnede design for BeerBong API'et fremgår af figur 10.7. På figuren fremgår det at der er valgt at implementere repository pattern sammen med unitofwork. BeerBong serveren er udviklet i Asp.net core med MVC frameworket. Hvilket giver mulighed for en række værktøjer og biblioteker.

Generelt

Med Asp.net core platform kommer en lang række af nyttige værktøjer. En af disse værktøjer er egenskaben til at anvende denpendency injection. Dette betyder at vi kan tilføje klasserne til IoC containeren, hvorefter vores afhængigheder automatisk vil blive injected i f.eks. controllernes constructor. Dette betyder at Interface segregation princippet fra SOLID bliver nemmere at anvende i den forstand at klassernes afhængigheder afhænger af interfaces, og disse bliver injected i controllernes constructor. Således afhænger controlleren kun af de metoder som bliver defineret i de enkle repositories interfaces. Herudover er der valgt at implementere en mapnings mekansime med hjælp af automapper frameworket. Dette har betydet at det har været simpelt at kunne mappe mellem DTO'er og domæne objekter.

Repository pattern

Funktionaliteten for de forskellige repositories er at medierer mellem domain- og data-laget. Data-lags mapping bliver håndteret af objekt-relational mapper (O/RM) Entityframework core i form af datacontext klassen, således der kan mappes mellem SQL-databasen og de implementerede .Net klasser. I denne sammenhæng fungerer de implementerede repositories som en in-memory kollektion af domæne objekter. Dette betyder at det er muligt at arbejde på kollektion som var det en simpel kollektion af objektet, hvor man kan slette, redigere og tilføje data.

I sin essens virker et repository som abstraktion oven på Entityframework core, hvor man kan encapsulere sine queries og genanvende disse. Dette betyder også at BeerBong API'et er uafhængigt af objekt-relational mapperen, og hvis det ønskes at anvende en anden (O/RM) end entityframework core vil dette være muligt. Der implementeres et repository per domæne [5]. Dog er der blevet implementeret et generisk repository som implementerer CRUD operationerne for samtlige domæne klasser, herunder GET, POST, PUT og DELETE. De enkle repositories arver således de basale CRUD-operationer fra det generiske repository, og den mere use case specifikke forretningslogik, bliver implementeret i de enkle repositories. Det skal pointeres at alle repositories er implementeret med et interface. For at se hvordan klasserne intergere i kontekst med en usecase, henvises der til de opsatte sekvensdiagrammer i billagsrapporten [6], samt source koden for at se de implementerede repositories defination.

Et design med repository pattern bidrager med at implementere SRP og OCP i SOLID principperne. Single responsibilty princippet ses ved hvert repository har et ansvar til at mediere et enkelt domæne mellem data-laget og domæne. Open close princippet ses ved at de enkle repositories er "open" for at der kan tilføjes funktionalitet igennem deres interfaces, samt at de

[5] Repository Pattern Done Right, Mosh Hamedani
[6] 6.4 'BeerBong Server'

enkle repositories er "closed" i form af det veldefinerede interface som er blevet defineret, som controllerne anvender. For at se den specifikke implementering af repositories henvises der til bilagene [7]

Unitofwork

UnitofWorks funktionalitet er at holde styr på de in-memory updates der forekommer når vi anvender de enkle repositories, og sender disse updates til databasen i én transaktion. Dette vil sige at når en bruger-specifik aktion som f.eks. at kunne oprette sig om bruger på systemet, som er specificeret i use case 4. Se bilagsrapporten for specifikationerne om use case 4 [8]. Her bliver alle transaktioner såsom crudoperationerne POST, GET eller DELETE i denne aktion færdig i én transaktion istedet for flere database transaktioner [9].
På denne måde bliver Unitofwork lag mellem vores controllers og repositories i systemet. Unitofwork står for at holde på alle de forskellige repositories, som skal modtage datacontext klassen fra entityframework core. Dette forsikre at en transaktion som strækker over flere repositories, bliver enten alle enititeter gemt i databasen ellers fejler alle, da det er den samme instans af datacontext klassen fra entitityframework core. For at se hvordan UnitofWork er opbygget henvises der til source konden, samt figur 10.8. For at se den specifikke implementering af Unitofwork henvises der til bilagene [10]

Delkonklusion

Ved at avende repository pattern og unitofwork får man alt funktionaliteten forklaret i de tidligere afsnit, samt endnu en abstration til sit object-relational mapper (O/RM) hvilket i dette projekt har været Entityframework core. Ved at putte en abstraktion mere på mellem data og domæne laget, er API'et gjort uafhængigt af O/RM'en hvilket har gjort designet mere testbart. Dette betyder at ved unit tests kan der anvendes en Mock af repostory og unitofwork til at kunne teste de enkle controllers.

11.2.3 Database

Der vil i dette afsnit argumenteres for valg af komponenter til databasen og i mindre grad komme med alternativer til de løsninger som er blevet taget. Forskellige designmønstre vil blive gennemgået og hvilke fordele og ulemper de havde.
Til udvikling af databasen blev der lavet en context klasse, der arver fra biblioteket 'IdentityDbContext', som gør det muligt for en bruger at logge ind på API'et. I denne klasse bliver der også oprettet de entities, som bliver mappet til databasen hosted på Azure.
Der er både blevet anvendt fluent api og data annotation til udvikling af entities. Alt hvad der kan laves i fluent api kan laves som data annotation, men i gruppen var der enighed om, at en kombination af de to gav en god udnyttelse af de værktøjer og gjorde koden mere læsbar. Desuden var det ikke muligt at lave samtlige relationer som data annotation, derfor var det nøvendigt at lave fluent api til dele af database udviklingen.

11.2.4 BeerBong Controller

Nedestående afsnit gennemgår de designmæssige overvejelser og valg for BeerBong Controlleren. Dette indebærer valg af designmønstre og kode strukturering for at opnå nem vedligeholdelse og god testbar kode. Derudover vil der blive gennemgået hvilke fordele der har været ved at

[7] 6 'Bibliography'
[8] 2.6 'Use Case 2 - Start online spil'
[9] Repository Pattern Done Right, Mosh Hamedani
[10] 5 'Bibliography'

følge det valgte design.
BeerBong Controlleren er udviklet i Visual Studio i C# i .Net Framworket. Da det ikke var muligt at bruge .Net Framworket alene til alle dele af koden, blev der taget brug at et bruger lavet biblotek, Unosqaure. Unosqaure blev brugt til at forbinde til GPIO portene på raspberrypi.[11]

State Pattern

Programmet på RaspberryPi[12] er designet som et state pattern.[13] Det betyder at afhængigt af sensorene sættes programmet i et specifikt state. Som det kan ses i afsnit 10.3.6 'BeerBong Controller' figur 10.11, er programmet opbygget med tre states. Alle tre states implementerer den samme funktion IsFull. Funktionen er implementeret anderledes i hvert state, da det er forskellige sensorere den skal kigge på i de forskellige states. Fordelen ved at designe koden efter et state pattern er at det gør det muligt at ændre funktionaliten uden af ændre selve klassen. Main programmet står konstant og kigger på IsFull funktionen for dens Context.

Ved start sættes 'Context' til emptystate, og derefter står programmet kun og kigger på Context.isFull. Statet som Context er i, bliver sat inde i de indviduelle states, afhængigt af sensorens input. Context.SetState bliver kaldt hver gang at ølbongen skal skifte state, når der er øl i bongen skiftes der fra empty til full, når der begyndes at drikke skiftes der fra full til notdone, og når brugeren er færdig skiftes der til empty. Timeren startes fra der skiftes fra full til notdone, og stoppes når der skiftes fra notdone til empty. Undervejs skrives state til en JSON fil, hver gang der skiftes til et nyt state, og når der skiftes fra NotDoneState skrives tiden også til denne fil.

Nogle fordele ved at følge state pattern, er at der følges S og O i SOLID principperne. Det følger Single Responsibility, da koden bliver organiseret således at hver stykke kode til det specifikke klasse bliver enkapsuleret i sin egen klasse, og kun rørt hvis at programmet er i dette state. Open/Close princippet da det er muligt at introducere et nyt state uden ændringer i Context eller de andre states. Dette er meget positivt i forhold skal scalability, da hvis nye features blev lavet til Controlleren, ville det være simpelt at implementere i et nyt state.
Der er overodnet fulgt disse to design principper i designet af Controlleren. Alle tre sensorere har hver deres klasse, som implementerer et interface, dette følger også Single Repsonsibility og Open/Close princippet, da hver klasse kun har et ansvar, og der kan sagtens tilføjes flere sensorer uden at ændre i andre klasser.

Bluetooth

Der blev forsøget med at bruge Bluetooth[14] som kommunikationsform imellem Raspberry og Smartphone applikationen. Mere specifikt blev der forsøgt med Bluetooth Low Energy. Bluetooth LE er en 'mindre' version af bluetooth. Der kan overføres med mindre hastigheder, men kræver også mindre energi. Det er optimalt hvis der kun skal overføres mindre mængder data, bluetooth bliver gennemgået mere dybdegående i bilagsrapporten [15]. I controlleren blev bluetooth forbindelsen implementeret som en serielport. Hver gang Controller programmet skiftede state ville dette blive skrevet til den port så bluetooth var forbundet til, samt en tid hvis dette var aktuelt. Der blev ikke fulgt nogen specifik protokol, men det specifikke state samt

[11] 10 'Bibliography'
[12] 12 'Bibliography'
[13] 11 'Bibliography'
[14] 14 'Bibliography'
[15] 9.2.1 'Bluetooth'

tid blev sendt som en string. I sidste ende blev Bluetooth ikke anvendt som kommunikation mellem BeerBong App og Beerbong Controller, grundet problemer under implementeringen på BeerBon App[16].

Websocket og PM2

Brugervenlighed havde stor betydning igennem hele udviklingen af projektet. Det var bl.a. en af grundende til, at anvende bluetooth i begyndelsen fremfor en websocket. Efter denne metode blev udelukket, ville vi gøre det muligt, at anvende BeerBong Controlleren uden brug af en computer. Derfor skulle alle nødvendige programmer startes ved opstart. Til dette blev der anvendt production process manager (PM2), som kan bruges til Node.js applikationer med en indbygget load balancer. Det gør det muligt at holde et program konstant kørende og give dem admin tilgængeligheder.
En websocket protokol gør det muligt at lave en to vejs kommunikation mellem klient(er) der er godkendt af server. Dvs. at klienterne kører ikke godkendt kode, i et kontrolleret miljø over HTTP. Protokollen starter med et 'Opening Handshake'. Det er klienten som skal initiere forbindelsen til server, ved at kontakte den og herefter spørge om tilladelse til at abonnomere på websocket serveren. Når serveren modtager en forespørgelse på en forbindelse, giver den et svar tilbage hvori at protokollen ændres fra HTTP til Websocket.

Websocket serveren[17] holder styr på hvilke klienter der er lavet et 'Handshake' med, så den ikke lavet et handshake når der bliver sendt data mellem de to. Det endelige formal med denne type kommunikation er give en sikker og pålidelig kommunikationsform, hvori klienter får skubbet (push) alt data fra server til klienter istedet for en pull funktion ligesom på WebApi [18].

Delkonklusion

Controlleren er designet i to dele, der er Controller programmet kaldet RaspberryPi.exe og Websocket severen. Controllerprogrammet er opbygget efter state pattern, så main programmet hele tiden kalder samme funktion på context, men implementeringen ændrer sig efter hvilket state den er i. Det enkapsulerer funktionaliteten, i tre indviduelle klasser, så de kun har ansvaret for det som skal gøres i det specifkke state. Ved at designe programmet således følges S og O i SOLID principperne, og skaber dermede en mere struktureret kode. Desuden er der lavet interfaces til alle klasser, for at gøre det muligt at lave fakes til alle klasserne, for at få en mere tesbar kode. Websocket koden er lavet som en node.js Websocket server, efter at forbindelse med bluetooth fejlede. Denne server gør det muligt for appliaktionen at modtage data automatisk (push). [19]

[16] Se bilagsrapporten for yderligere information9.2.1 'Bluetooth'
[17] 13 'Bibliography'
[18] https://javascript.info/websocket
[19] Sourcekode for BeerBong Controlleren kan findes i bilag/sourcekode/BeerBong Controller

Modultest 12

Indledning

Dette afsnit gennemgår modultest for de individuelle moduler i systemet. Dette indebærer en forklaring for hver modultest og hvad testen har omfattet. Resultaterne for de enkle moduler ville blive holdt op mod de forventede resultater. Ligeledes gennemgås de tests der er anvendt til at dokumenter softwarens funktionalitet, i form af unit tests. For flere detaljer se [1]

Modultest af hardware

I tabel 12.1 ses en opsummering af modultest af hardware komponenterne. Der er beskrevet hvilket modul der testes, hvordan det testes og hvad resultatet af testen er. Mere dybdegående beskrivelser af disse test kan findes i Billagsrapporten [2].

Modul	**Test beskrivelse**	**Forventede Resultater**
Laser Receiver	Receiver kobles op til Analog Discovery, med oscilloskop på udgangen af receiveren.	Nå lys registreres kommer der spænding på udgangen af receiveren.
Laser Emitter	Emitter kobles op til Analog Discovery, og der sendes 3,3V ind i emitteren.	Når forsyningen er aktiv, lyser laseren.
Magnet Reed Switch	Magnet sammen med tilhørende kredsløb kobles op til til Analog Discovery med 3,3V og 5V. Der testes med en magnet der sættes op til switchen, og der registreres om der kommer spænding på udgangen afkredsløbet.	Når magneten holdes tæt på switchen, kommer der spænding på udgangen af kredsløbet.

Tabel 12.1: Modultest af BeerBong Controller Hardware

Modultest af software

Dette beskriver modultests af de software moduler som systemet består af. Det er opdelt i tre tabeller, som beskriver hvilken del af modulet, en beskrivelse af den test som modulen er under, og de resultater der kom ud af testen. Hver tabel dækker et modul, og de dele som der er blevet testet i modulet. Tabel 12.2 beskriver modultest for BeerBong Controlleren, tabel 12.3 beskriver modultests for BeerBong App, og tabel 12.4 beskriver modultest af BeerBong servere. Disse tabeller er kun opsummeringer af de fulde tests, som kan findes i Billagsrapporten [3]

[1] Afsnit 10 'Modultest'
[2] Afsnit 10.2.1 'Modultest af hardware'
[3] Afsnit 10.2.8 'Modultest af software'

BeerBong Controller

Modul	Test beskrivelse	Forventede Resultater
Unit tests[4]	Der er lavet en række unit test som tester controllerens funktionalitet. Der testes på om der skiftes state korrekt, og om programmet bliver i de korrekte states, når de korrekt forudsætninger er sat. Der er ikke testet på kode som interegerer direkte med hardware eller for den kode om skriver til json filen.	At der nåes en forsvarlig % på code coverage, at al den kode som kan testes bliver testet.
Websocket	Der er til test af Websocket blevet lavet en testclient. Der kigges så på om denne client modtager den korrekte data.	Det forventes at testclienten modtager den korrekte data.
Bluetooth	Der er blevet testet med en terminal client på en android telefon. Telefonen bliver parret med Raspberry Pi'en, og der testes ved at sende og modtage fra RaspberryPi, til/fra telefonen.	Det forventes at der mmodtages korrekt data både på telefon og Controller.

Tabel 12.2: Modultest af BeerBong Controller Software

BeerBong App

Modul	Test beskrivelse	Forventede Resultater
Unit tests	Til beerbong applikationen er der laver en række unit tests som tester serivce komponentets kald til api'et, og en enkelt test til viewmodel komponentet. Der testes på fem af service komponentets kald til api'et, hvor interfacet IRestService bliver substitutet (mocket), og metoderne under test bliver kaldt igennem den. Samtidig bliver et objekt af RestService oprettet og samme metode kaldt igennem der. Viewmodellen er unit testet ved at kalde en kommando i viewmodellen.	Der forventes at kaldene til api'et returnere det samme som defineret i mocken, og at viewmodellens kommando og dens tilhørende metoder bliver kaldt. For at se de implementerede unit tests henvises der til bilagene [5]
Test af brugergrænseflade	Brugergrænsefladen er testet igennem visual studio's android emulator, hvor hver side af applikationen er blevet gennemgået og hver funktionalitet for brugergrænsefladen testet.	Der forventes at layoutet for hver side ser ud som kodet i xaml filerne, og knapper osv. fungere og udføre den logik som de er programmeret til.

Tabel 12.3: Modultest af BeerBong App Software

[4] 15 'Bibliography'
[5] 7 'Bibliography'

BeerBong Server

Modul	Test beskrivelse	Forventede Resultat
Unit tests	For BeerBong serveren er blevet anvendt unit test til at teste de fem controllere på serveren. Dette er blevet gjort ved at Mock controllernes afhængigheder. Til dette er unitofwork, det pågældende repository og mapper blevet Mocket, og på den måde er controllerne blevet isoleret således deres metoder kunne testes. Den genrelle fremgangsmåde har været om den valgte controllers metoder herunder GET, POST, PUT og DELETE har retuneret det korrekte actionresult efter de mocket afhængigheder har retuneret et opsat result	Det forventes at controllernes forskellige metoder retunere det samme som de opsatte resultater Mockene retunerede.
Swagger	Swagger er blevet anvendt til at teste alle BeerBong Serverens komponenter sammen. Dette er blevet gjort med den genereret Swagger UI, hvor controllernes URLS kan testes	Det forventes at når Den valgte URL for en controllers request korrekt, så retuneres det korrekte element fra databasen. For at se de implementerede unit tests henvises der til billagene [6]

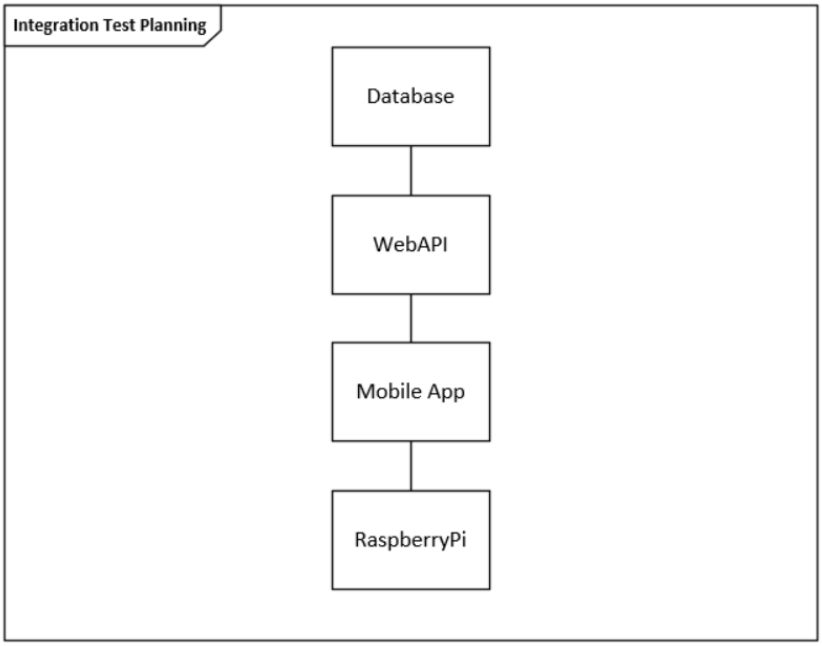
Tabel 12.4: Modultest af BeerBong Server Software

[6] 4 'Bibliography'

Integrationstest 13

13.1 Indledning

Dette afsnit gennemgår integrationstest for systemet. Der anvendes topdown metoden til at teste moduler, på figur 13.1 ses hvordan systemet bliver testet. Da det er topdown startes der med at teste databasen og WebAPi'et sammen alene. I næste test, testes der også med mobil applikationen, og i den sidste test bliver alle moduler testet sammen.



Figur 13.1: Testplan for integrationstest, hvor top-down metoden anvendes

13.2 Integrationstabel

I tabel 13.1 opsummeres hvert scenarie kort med en beskrivelse af hvilke moduler der indgår i testen og en beskrivelse af testen. For en mere dybdegående beskrivelse af integrationstesten henvises der til billagsrapporten[1].

Scenarie	**Enheder under test**	**Beskrivelse af test**
Scenarie 1	API og Database	Denne test består af at skrive noget specifikt data til databasen fra serveren. Der postes en spiller og et spil til databasen, og det tjekkes om dette kan findes i databasen gennem Microsoft SQL Management Studio.
Scenarie 2	Database, API og Applikation	Denne test har til formål at teste om API'et og applikationen kan kommunikere korrekt. Der bliver testet om applikationen kan oprette en bruger, og om brugeren så kan findes i databasen.
Scenarie 3	Applikation og Controller	Der testes om applikationen kan oprette forbindelse og kommunikere korrekt med controlleren. Dette bliver gjort ved at controlleren startes op, der forbindes til controlleren og så kontrolleres der i VS debugger, hvad tekstestrengen som modtages er korrket.

Tabel 13.1: Modultest af BeerBong Server Software

[1] Afsnit 11 'Integrationstest'

Accepttest 14

14.1 Funktionelle krav

Dette afsnit dækker den endelige accepttest for systemet. Der vil først blive gennemgået en kort gennemgang af alle accepttest for at vise hvilke der blev gennemført. Den fulde accepttest for hele systemet kan findes i bilaget [1]. Den korte gennemgang af den fulde accepttest kan ses i tabel 14.1. Efterfølgende vil et eksempel på en accepttest af systemet blive vist.

Unit under Test	**Observation**
Use Case 1	Use case fejlede, eftersom den ikke møder de krav beskrevet i kravspecifikation i bilaget [2]
Use Case 1: Extension 1	Extension virker ikke efter hensignet og er derfor fejlet.
Use Case 1: Extension 2	Extension virker efter hensigten beskrevet i krav specifikationen i bilaget.
Use Case 1: Extension 3	Extension virker ikke efter hensignet og er derfor fejlet.
Use Case 2	Use case virker efter hensigten og møder alle krav specificeret i kravspecifikationen i bilaget.
Use Case 2 Extension 1	Extension virker efter hensigten og møder alle krav specificeret i kravspecifikationen i bilaget.
Use Case 2 Extension 2	Extension virker efter hensigten og møder alle krav specificeret i kravspecifikationen i bilaget.
Use Case 3	Use case virker efter hensigten og møder alle krav specificeret i kravspecifikationen i bilaget.
Use Case 4	Use case virker efter hensigten og møder alle krav specificeret i kravspecifikationen i bilaget.
Use Case 4 Extension 1	Extension virker efter hensigten og møder alle krav specificeret i kravspecifikationen i bilaget.
Use Case 4 Extension 2	Extension virker efter hensigten og møder alle krav specificeret i kravspecifikationen i bilaget.
Ikke-funktionelle krav	Alle ikke funktionelle krav er efterlevet, som er beskrevet i bilaget.

Tabel 14.1: Accepttest resultater for version 1.2

Tabel 14.2, viser accepttesten af Usecase 2 hovedscenarie. Den fulde acceptest kan findes i Billagsrapporten.[3].

[1] Kaptiel 12 'Accepttest'
[2]Kapitel 2.3.2 ''
[3] Kaptiel 12 'Accepttest'

Use Case 2

Use case under test	Online			
Scenarie	Hovedscenarie			
Prækondition	Applikationen på smartphonen er startet, og startside vises.			
No.	**Handling**	**Forventet resultat**	**Faktisk resultat**	**Vurdering** (+/-)
1	Bruger trykker på 'Start Online Spil'.	Applikationen viser 'Find modstander' knap.		
2	Bruger trykker 'Find modstander'.	Applikationen finder en modstander, og viser en ny side med modstandernavn.	.	
3	Bruger fylder øl i bongen	Applikationen viser 'Klar' knap efter 1 minut.		
4	Bruger trykker på 'Klar'.	Applikationen starter en tre minutter lang nedtælling		
5	Bruger drikker ølbong.	Efter tre minutter viser applikationen tid for spiller, modstander og en vinder.		

Tabel 14.2: Accepttest af Use Case 2 - Hovedscenarie

Resultater 15

15.1 Indledning

Dette kapitel beskriver resultater for de 3 forhenværende kapitler Modultest, Integrationstest og Accepttest. Alle test er beskrevet i overstående

15.2 Modultest Resultater

15.2.1 Hardware Modultest

Modul	Resultater
Laser Receiver	Når laser lyses på modtageren, kommer 3,3V ud fra receiveren.
Laser Emitter	Når forsyningen er aktiv, lyser laseren.
Magnet Reed Switch	Ved test med 3,3V kom 3,3V ud af udgangen når magnet kom indenfor 10 mm af switchen. Hvis spændingen øges til 5V ville rækkeviden stige med yderligere 5 mm.

Tabel 15.1: Modultest resultater af BeerBong Controller Hardware

15.2.2 Software Modultest

BeerBong Controller

Modul	Resultater
Unit tests	Der er opnået en code coverage på 54%, hvor koden som ikke er dækket hovedsageligt består af kode, som interegerer direkte med hardware eller for den kode om skriver til json filen.
Websocket	Den kan ses at test clienten først laver et handshake, og derefter modtager clienten den korrekte data.
Bluetooth	Der kan sendes og modtages korrekt på Raspberry Pi til/fra bluetooth terminalen på Andriod telefonen.

Tabel 15.2: Modultest resultater af BeerBong Controller Software

BeerBong App

Modul	Resultater
Unit tests	Alle test virker som de skal, og opnår tilsammen en code coverage på 19%. Testene dækker hovedsageligt kaldende til api'et, og så længe api'et fungere, fungere testene.
Test af bruger-grænseflade	Alt visuelt fungere som det skal, og knapper og anden funktionalitet fungere, og kan navigere rundt i applikationen som ønsket.

Tabel 15.3: Modultest resultater af BeerBong App Software

BeerBong Server

Modul	Resultat
Unit tests	Der er kun opnået en code coverage på 22%, og unit testene dækker kun controllerne. Dette skyldes hovedesagligt mangle på tid og prioritering af funktionalitet. Det der dog er blevet testet har dog været med til at afklare funktionalitet af de enkle units.
Swagger	Til at teste komponenternes funktionalitet sammen i BeerBong serveren, har swagger været en kæmpe hjælp. Dette har betydet API'ets endpoints nemt kunne testes sammen grundet swagger UI.

Tabel 15.4: Modultest resultater af BeerBong Server Software

15.3 Integartionstest resultater

Unit under Test	Observation
Scenarie 1	Der blev postet en bruger med navnet "MathiasTP" til databasen fra API'et. Det kunne bekræftes i Microsoft SQL Management Studio, at der under 'User' collection findes en bruger ved navn "MathiasTP".
Scenarie 2	Testen blev udført ved at oprette en bruger på applikationen, og derefter bekræftes i Microsoft SQL Management Studio, at der fandtes en specifik bruger med navnet "Test2" under User collectionen. Det var muligt at finde den specifikke bruger i User collectionen.
Scenarie 3	Testen blev udført ved at oprette forbindelse til Controlleren. Efter der var oprettet forbindelse blev applikationen kørt i VS debugging og der blev tjekket at den korrekte tekststring blev modtaget. Der blev modtaget en json fil, som tekststreng, med navn, state, tid og comment.

Tabel 15.5: Integrationstest resultater

15.3.1 Accepttest

Use case under test	Online			
Scenarie	Hovedscenarie			
Prækondition	Applikationen på smartphonen er startet, og startside vises.			
No.	**Handling**	**Forventet resultat**	**Faktisk resultat**	**Vurdering (+/-)**
1	Bruger trykker på 'Start Online Spil'.	Applikationen viser 'Find modstander' knap.	Applikationen viser en 'Find modstander' knap	OK
2	Bruger trykker 'Find modstander'.	Applikationen finder en modstander, og viser en ny side med modstandernavn.	Applikationen finder en modstander, og viser en ny side med modstandernavn.	OK
3	Bruger fylder øl i bongen	Applikationen viser 'Klar' knap efter 1 minut.	Applikationen viser 'Klar' knap efter 1 minut.	OK
4	Bruger trykker på 'Klar'.	Applikationen starter en tre minutter lang nedtælling	Applikationen starter en tre minutter lang nedtælling	OK
5	Bruger drikker ølbong.	Efter tre minutter viser applikationen tid for spiller, modstander og en vinder.	Efter tre minutter viser applikationen tid for spiller, modstander og en vinder.	OK

Tabel 15.6: Accepttest af Use Case 2 - Hovedscenarie

Diskussion 16

Efter endt modultest og integrationtest for samtlige moduler og resultaterne hertil er blevet gennemgået, kan det ses at den overordnede funktionalitet beskrevet i projektformuleringen er opnået. Der er blevet udviklet en prototype, som løser den beskrevne projektformulering i et tilfredsstillende omfang. Alle use cases udover usecase 1 blev succesfuldt gennemført i acceptesten.

Disse resultater har vi opnået vha. god kommunikation, god planlægning og omstrukturering af arbejdsområder indbyrdes i gruppen. Til kommunikationen indbyrdes i gruppen er der blevet gjort brug af Scrum, som givet et godt overblik over arbejdsopgaver samt faldgrupper. Det har været et vigtigt værktøj til at kunne færdiggøre projektet med et produkt der er funktionsdygtigt. Derudover har sprintplanlægningen vha. Zenhub givet et overblik over de enkelte gruppemedlemmers arbejdsopgaver og hvor langt hver især er. Der er i nogen grad blevet gjort brug af stand-up meetings, men i et begrænset omfang. En mere konstant brug af stand-up meetings kunne have medført et projektforløb der var mere struktureret. Ydermere at der fra starten i projektet blevet lavet en tidsplan, der løbende er blevet lavet ændringer i. Tidsplanen har hjulpet gruppemedlemmer til at holde deadlines og holde overblikket over fremtiddige opgaver. Med det sagt, er der blevet lavet ændringer løbende.

I begyndelsen af projektforløbet, blev der udarbejdet en risikoanalyse, der tog forbehold for, at gruppemedlemmer kunne finde på at stoppe på uddannelsen. Derfor blev det besluttet, at alle moduler, som var under udvikling skulle være dækket af minimum to personer, så det blev sikret af modulet blev færdiglavet. Det viste sig senere, at to gruppemedlemmer stoppede på studiet midt på semestret. Det skabte i begyndelsen nogle problemer i forhold til arbejdsbyrde og dermed planlægning af projektet. Det betød at dele af projektet skulle afgrænses i sådan en grad at projketet passede i størrelse til 5 personer, der til sidst medførte at funktionaliteten på projektet blev ændret. Det har været en vigtig beslutning, da det gav mulighed for at færdiggøre projektet med en tilfredsstillende funktionalitet. Hvis dette ikke var blevet gjort, ville et produkt formentligt ikke blive færdiggjort og gjort fuldt funktionsdygtigt.

Under udviklingen af de fire forskellige moduler, har det hele tid været vigtigt, at lave software, som var testbart. At lave en testbar kode, har gjort det muligt for de individuelle gruppemedlemmer, at lave dybdegående unit test. Alle modultests har fundet de fleste fejl i modulet, inden integrationstesten fandt sted. De fejl som opstod heri har været forholdsvis lette at lokalisere, eftersom modulerne er blevet tilføjet i en systematisk rækkefølge.

Projektet har været et betydningsfuldt fag på 4. semester, eftersom der er blevet anvendt teori fra fag brugt på både 4. semster, men også tidligere semestre. Disse fag har givet en basis forståelse for kommunikationsformer, design patterns og meget mere, så der kunne blive taget en oplyst beslutning om hvilke komponenter og metoder der skulle anvendes i de forskellige moduler.

Konklusion 17

Dette projekt har haft fokus på implementeringen af BeerBong Battle systemet, som gør det muligt at konkurre mod andre i at drikke en øl i en ølbong hurtigst. Der er til protjektet blevet udarbejdet en prototype, som delvist lever op til de krav som er sat i begyndelsen af projektforløbet. Prototypen lever kun delvist op til systemets krav, da færdiggørelsen af Usecase 1[1] ikke kom i mål. Udover dette lever prototypen op til en overordnet funktionalitet, som gør det muligt at forbinde fester med hinanden, over internettet.
Der er under designet og implementering af systemet taget højde for læringsmålene for projektet, fordi koden er opbygget efter SOLID principperne, specifikt efter Single Responsibility og Open/Close. Sammen med dette er der taget højde for at lave et skalerbart og testbart system, ved at lave interfaces til alle implementerede klasser i systemet. Det kan konkluderes at system lever op til alle "Must" kravene i de ikke-funktionelle krav som er sat i begyndelsen af udviklingsprocessen, og dermed er alle grundstene for systemet på plads. Under accepttesten fremgik det at alle Usecases og Extensions, udover Usecase 1, bestod testen hvilket viser systemet overordnet overholder de funktionelle krav.

Det overordnede problem som problemformuleringen skulle løse, var at det skulle være muligt for mindre fester at konkurrere mod hinanden og evt. kommunikere med hinanden. Systemet lever op til at det gør det muligt for to parter på hver sin lokation at konkurrere over BeerBong Battle mod hinanden i at drikke, øl, dog ikke at kommunikere med dem man spiller imod.
Udarbejdelsen af dette projekt har overordnet fungeret godt, men der har været behov for at afgrænse projektet størrelse samt funktionalitet, på baggrund af at to ud af de syv gruppemedlemmer droppede ud af gruppen efter projektets start. Ser man bort fra dette har både projekt og process arbejdet fungeret godt. Der har været en agil arbejdstilgang til projektet, så løbende været muligt at lave ændringer i løsningerne for systemets krav.

[1] Bilagsrapport afsnit 2.3.2 ”

Fremtidigt arbejde 18

Applikationen åbner døre for en masse viderudvikling. Nogle idéer der kom på bordet i produktets fødsel, men ikke blev implementeret, ses stadig som højt prioriterede idéer.

Det første er Bluetooth forbindelsen mellem applikationen og controlleren. Som nævnt i afsnit 9.2 'BeerBong App' var brugervenlighed førte prioritet helt fra dag et, og at en bruger kunne forbinde til enheden som de ville en højtaler befinder sig stadig højt på ønskelisten. Da der desværre opstod problemer med brugen af Bluetooth i Xamarin, og da deadline var ved at nærme sig, var der desværre ikke tid til at sætte sig ind i det til denne prototype, hvilket resulterede i brugen af WebSocket.

Planen i starten var at multiplayer spillet også skulle have en mulighed for en team battle. Dette ville fungere som to instanser af lokalt spil, hvor en gruppes tider ville sammenlignes og sættes imod en anden gruppe, i stedet for at tage de enkelte deltageres tider. Denne game mode var planlagt med festivaler og festivalsdeltagere som målgruppe, da produktet kunne reklameres på disse lokationer med en camp mod camp eller telt mod telt. Efter gruppen mistede to medlemmer måtte dette desværre nedprioriteres, så multiplayer er på nuværende tidspunkt en mod en, men gruppe mod gruppe er stadig en høj prioritet til multiplayer.

Udover dette er der selvfølgelig en masse design mæssigt der kan gøres for at få appen til at se pænere ud. Planen i starten af dette projektforløb var at lave et produkt som faktisk kunne være noget ude på markedet og ikke bare et midlertidigt semesterprojekt. Dette kræver noget bedre præsentation i brugergrænsefladen, hvilket vil sige der skal gang i noget UX design for at forbedre denne aspekt af applikationen.
En anden idé der lød særlig attraktiv var at vise tiden i real time. Dette ville både videregive informationen til brugeren på en smart måde, og samtidig bidrage til noget design mæssigt, da dette kunne visualiseres i for af en ølbong der tømmes samtidig med at spilleren drikker. Der var dog ikke nogen løsing til dette i starten af forløbet, og SignalR blev desværre først præsenteret sent i forløbet, hvor gruppen allerede havde bestemt at der var kodestop. Der kan dog tages inspiration fra den seneste opgave i NGK om at hente vejrdata, da denne også gjorde brug af en liveopdatering.

Det kan heller ikke siges at koden overholder alle SOLID principperne. Xamarin tilbyder visse løsninger som kan hjælpe på vej, men generelt er appens funktionalitet prioriteret frem for dette. Dette var selvfølgelig ikke optimalt for tests, hvilket også kunne mærkes senere, da der heller ikke er meget fokus på code coverage i applikationen. Dette var sevfølgelig ikke planen, men da gruppen stødte på ressourceproblemer, i form af to medlemmer der forlod gruppen, var valget at prioritere leveringen af et funktionelt produkt. Hvad SOLID kunne have gjort for applikationen kan læses i afsnit 11.2.1 'SOLID'.

BeerBong Controlleren har mulighed for videreudvikling. Der er allerede implementeret bluetooth, dermed kan mobiltelefoner i fremtiden forbinde over bluetooth og dermed øge brugervenligheden. Derudover har den nuværende ølbong en kapacitet på 70 cl, hvilket gør det muligt at købe flere sensorer, og dermed gør det muligt at spille med mere end en øl.

BeerBong serveren med det nuværende design giver mulighed for en masse videreudvikling af projektets funktionalitet. Herudover i forhold til designet kunne der også mens der tilføjes ekstra funktionalitet til BeerBong Battle systemet blive videre udviklet. Dette kunne f.eks. gøres ved at tilføje endnu en abstraktion i form af et service layer. Service layeret ville skulle enkapsulere funktionaliteten af det servicen udbyder (buisness logic), istedet for at denne buisness logic tilfalder de implementerede repsitories. Database skemaet kunne udbydes således der i fremtiden gemmes nye former for data, som API'et skal kunne håndtere. Dette kunne f.eks. være kravet under MoSCoW analysen se bilag [1] hvor det blev specificeret at BeerBong Battle systemet ikke ville implementere et point system. Med designet for API'et ville denne tilføjelse nemt kunne implementeres via nye controllers og repositories.

Herudover kunne det også i fremtiden være muligt at implementere en push funktion på serveren til klienterne. Dette kunne f.eks. blive implementeret med SIGNALR, således det er muligt at for server kunne pushes beskeder til alle klienterne eller specifikke klienter. Dette kunne f.eks. være en fordel hvis der skulle implementeres chat funktionalitet mellem klienterne der anvender BeerBong Battle systemet.

[1] Afsnit 2.4 'Ikke-funktionelle krav'

Bibliography

[1] PRJ4, GR5 *Bilagsrapport.*
`Dette dokument indeholder Krav, Arkitektur, Design og Test i detaljer.`

[2] *PDF: ASP.NET Core IN ACTION, Forfatter : Andrew Lock* `Bogen indeholder en gennemgang af ASP.NET core platformen.`

[3] *PDF: Entity Framework core IN ACTION, Forfatter : Joe P Smith*
`Bogen indeholder en gennemgang af ASP.NET core platformen.`

[4] *SourceCode/WebApiProjekt4/WebApiProjekt4_UnitTest*

[5] *SourceCode/WebApiProjekt4/WebApiProjekt4/Data/EFCore/UnitOfWork*

[6] *SourceCode/WebApiProjekt4/WebApiProjekt4/Data/EFCore/Repository*

[7] *SourceCode/BeerBong App - Online/BeerBongTestProject*

[8] *SourceCode/BeerBong App - Online/BeerBong*

[9] *Link: https://programmingwithmosh.com/net/common-mistakes-with-the-repository-pattern/, Forfatter : Joe P Smith*
`Linket indeholder en beskrivelse af repository pattern med UnitofWork.`

[10] *https://github.com/unosquare/raspberryio*
`Linket er dokumentation for det brugerskabte biblotek Unosquare raspberrypiio`

[11] *Head First Design Patterns, Kapitel 10.*
`Bogen indeholder en gennemgang af state pattern.`

[12] *Sourcecode/BeerBong Controller/RaspberryPi*

[13] *Sourcecode/BeerBong Controller/Websocket*

[14] *Sourcecode/BeerBong Controller/RaspberryPi/Bluetooth*

[15] *Sourcecode/BeerBong Controller/RaspberryPi/UnitTest*

[16] *Datablad: BC5578, Philips, 1997.*

[17] *Datablad: LaserSensorModule, Waveshare*

[18] *Datablad: LeverSwitches170.*

[19] *Datablad: TIM-201 diode.*

[20] *Samarbejdskontrakt, Underskrevet*

[21] *Risikoanalyse,*