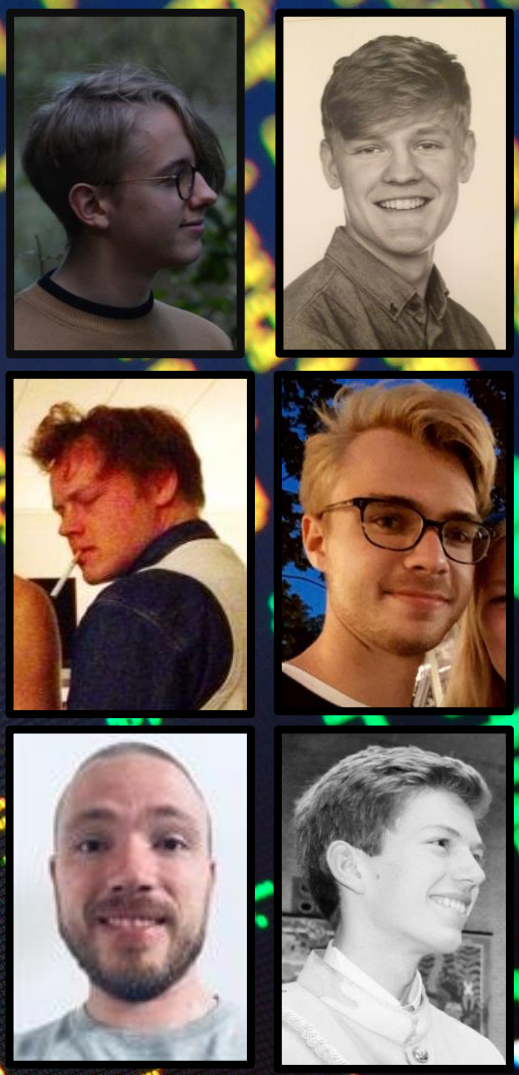
DTU Compute



Pong
CDIO Final

Kursus: 02324 - F19

S183018 Claes J. S. Lindhardt
S130016 Andreas BG Jensen
S185122 Jacob Riis Jensen
S185139 Malte Pedersen
S180945 Simon Woldbye-Lyng
S185125 Kristian Andersen

Batchlor of Engeneering

GitHub: https://github.com/AndreasBGJensen/OnlinePong

DTU Compute
Department of Applied Mathematics and Computer Science

Projekt Titel:
CDIO opgave

Projekt periode:
15/02-2019-02/03-2019

Temaer:
Computer Science,
Software ingeneering

Vejledere:
Christian Budtz
Stig Høgh

Relavante kurser:
42430

Appendix:
A-D

GitHub
link: https://github.com/AndreasBGJensen/OnlinePong

DTU diplom
Lautrupvang 15,
2750 Ballerup
http://www.diplom.dtu.dk

1 Abstract:

This project is to demonstrate our ability to make system architecture diagrams as well as operation diagrams of an IT-System. It is also to demonstrate pratical and theoritical understanding of Roy Fielding REST principles. This is all executed by building a webside where more then one player can play Pong('digital tennis').

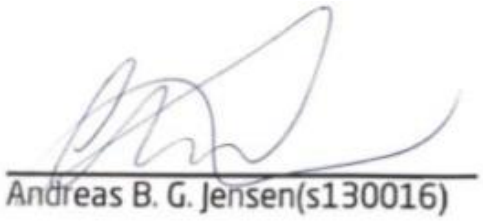
Readers guide:
This report is made in LaTeX, and uses hyperlinks. These however only work in pdf format. It is also written in Danish.

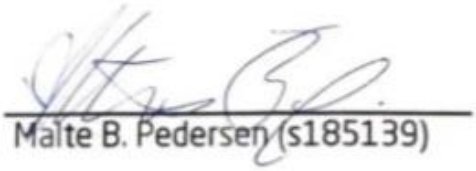
The relevant parts of sources will be given in footnotes as this example [1].

the first thing in the footnote is
a number in [ ], which refers to a number in the bibliography. Here you can find the source described in debt. Then it will be described what part of the source is relevant. Typically it is a page number, but should the source be video it might be a time frame.

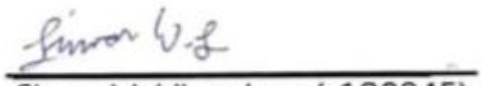
[1] [1]: example source: Google p. 2, l.4-p.5 l.7

Disclaimer: The content of this report is publicly available, but must not be published without permission from the authors.

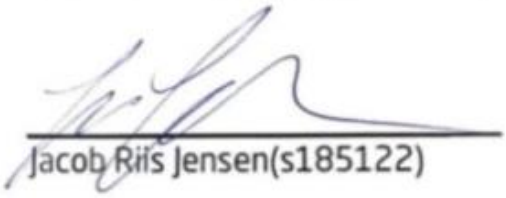

Andreas B. G. Jensen(s130016)


Malte B. Pedersen (s185139)


Claes J. S. Lindhardt(s183018)

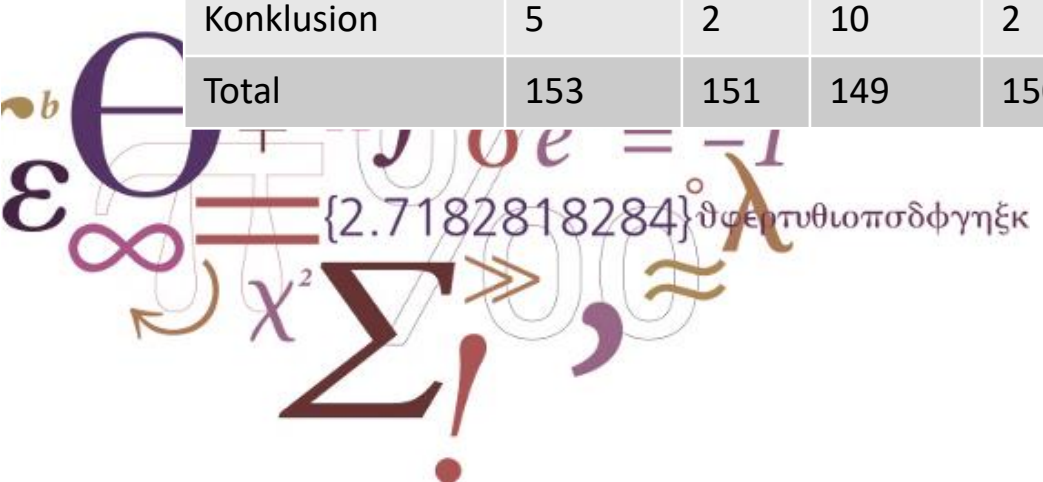

Simon Woldbye-Lyng(s180945)


Kristian Andersen(s185125)


Jacob Riis Jensen(s185122)

Time Regnskab:

	Andreas	Claes	Kristian	Simon	Jacob	Malte
Formål	20	2	16	5	24	19
Analyse	40	28	40	27	40	19
Design	40	60	40	60	33	42
Implementering	33	40	33	37	40	49
Test	15	18	10	17	4	19
Konklusion	5	2	10	2	16	5
Total	153	151	149	150	154	153



Indhold

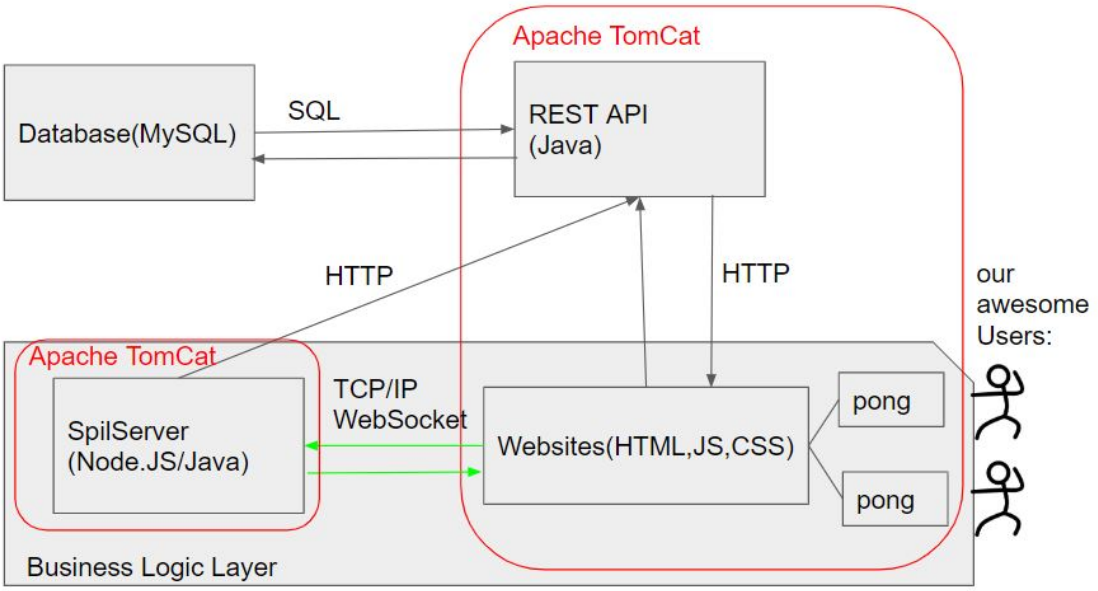
2 Introduction

Formålet med denne opgave er at udvikle et multiplayer Pong spil, som gør brug af REST api arkitektur tilgangen, og gennem denne gør brug af java, JavaSctipt, HTML og CSS. Dette system skal have:

- En database
- En webside
- En API der bygger på REST principper.
- En spilserver

Dette skal gøres ved brug af en tomcat server og Amazon Web Services.
For at holde styr på arkitekturen og implementeringen ynder vi at gøre brug af interfaces i java.

For at give et overblik over programmet har vi lavede vi en skitse med de forskellige centrale artifatcts(elementer) der indgår i systemet.



Figur 1: En skitese med de centrale elementer i IT-systemet, også kaldet Systemakitektur-diagram

De Centrale i problemstillinger for at det her kan lykkedes er at de forskellige elementer i systemet kan sende data "smooth"mellem hinanden.

Del I
Analyse

2.1 Krav og Use cases

Use Case:
For at forstå hvad der er relevant for systemet er det vigtigt at forstå hvad det skal bruges til. Det har vi gjort vha. en Use case:

2.2 Use case

Nedenfor er ses en liste over de use cases der er tænkt til denne opgave.

- En spiller skal kunne oprette sig i en database.
- En spiller skal kunne spille imod en anden spiller.
- En spiller skal kunne hente en high score.
- En spiller skal kunne se sine egne stats.
- Bolden skal flyve med en fart i alle retninger.

Nedenfor er der udarbejdet en fulldressed use case, hvori alle use cases kan identificeres.

2.3 Krav

Ud fra de use cases har vi så opstillet en række krav:

Funktionelle Krav

1. Database (MySQL) (Skal tale med REST API)
    1.1. Brugeroplysninger
    1.2. Match historik [nice to have]
    1.3. Score historik
2. Rest API (Java) (Skal tale med Database og Game server)
    2.1. Godkende en bruger
    2.2. Poste en bruger
    2.3. Hente highscore
    2.4. Poste score
    2.5. Poste et nyt spil
    2.6. Hente spillerinformation
    2.7. Data skal valideres [nice to have]
3. WebServer (Website) (HTML, JS, CSS) (skal tale med REST og SpilServer)

3.1. Loginsystem
3.2. Finde spil
3.3. Se stats [nice to have]

4. Pong Spillet(skal køres af hjemmesiden)
   4.1. Kommunikere til spilserveren.
   4.2. To spillere
   4.3. Bold skal bevæge sig i konstant hastighed i alle retninger.[nice to have]
   4.4. Bolden skal påvirkes af batslag.
   4.5. Spil score(point registrering)
   4.6. Vælge, hvilke taster spillere skal spille med (taster skal være valgfrie).[nice to have]
   4.7. Exit funktion.

5. SpilServer (Node.js)
   5.1. Lytte efter clienter
   5.2. Lave en tråd når der er fundet to clienter
   5.3. Sikre at tråden kan kommunikere med clienterne
   5.4. Holde styr på igangværende spil
   5.5. Sende spildata til andenparter. (Sende afsluttede spil's data til databasen).
   5.6. Vise, hvor mange spil der er igang.

Ikke-Funktionelle Krav

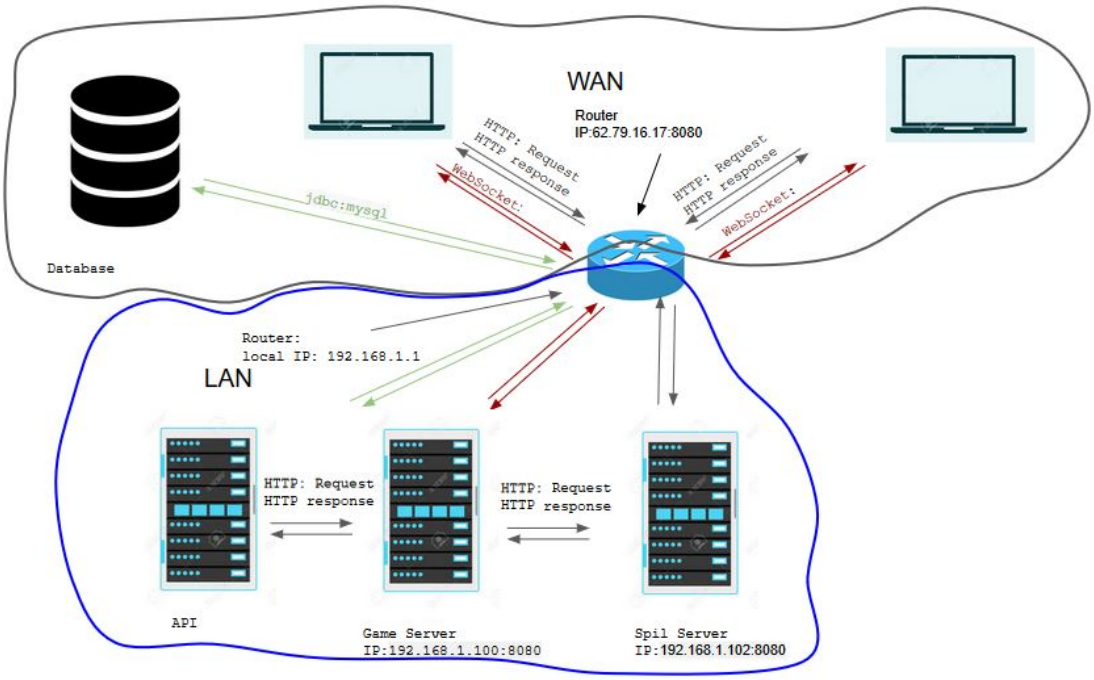
1. Ikke funktionelle
   1.1. Systemet skal opbevare følgende informationer omkring hver bruger: selvvagt brugernavn, password initialer, og brugerens rolle.
   1.2. En bruger i systemet skal *skal* indeholde *alle* informationer.
   1.3. ID og CPR-nummer skal være unikke i systemet.
   1.4. Systemet må ikke gå ned ved forkert brugerinput, og skal give brugeren en fejlmeddelelse ved dette forekommer.
   1.5. Brugeren skal informeres, hvis der sker fejl i systemet, via beskrivende fejlmeddelelse.
   1.6. Man skal kun kunne have rollerne Admin, Operator, Pharmacist og Foreman.

Del II
Design

I dette afsnit bliver systemets overordnede arkitekttur beskrevet. Herunder vil det våre beskrevet, hvorleden databasen er opbygget.

3 IT-Systemet som helhed

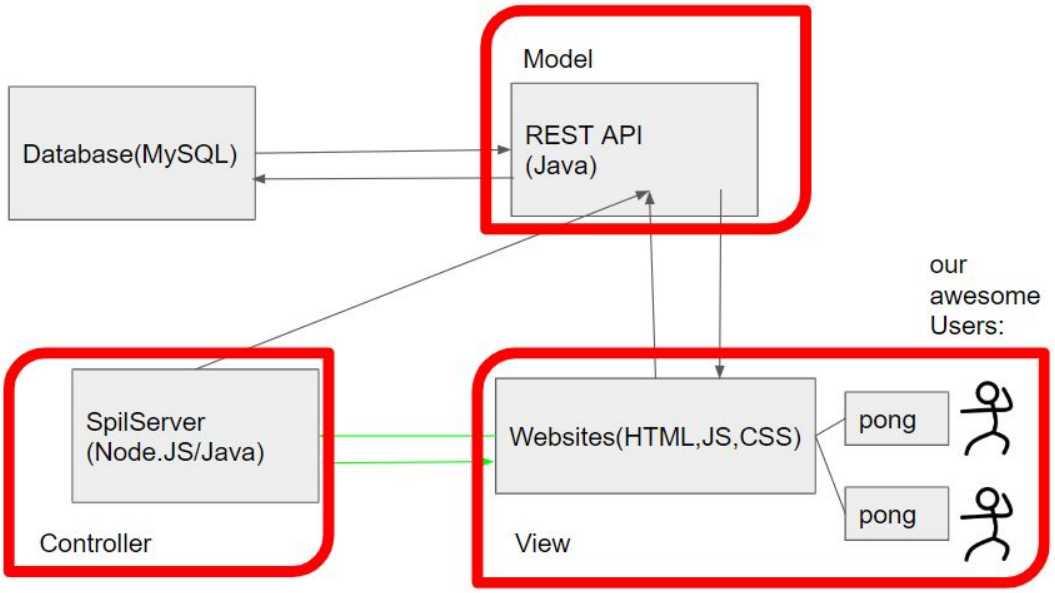
3.1 System-idriftsættelsesdiagram



Figur 2: Netværkets topologi. Figuren viser HUAWAY routeren der der adskiller henholdsvis WAN - og LAN netværket. På WAN siden har routeren IP adressen 62.79.16.17 og på LAN siden har routerne 192.168.1.1.

3.2 MVC model

Man kan se hele systemet indefor en MVC model, eller man kan se delene af systemet som deres eget MVC model. De mest centrale dele for denne opgave er Websiden, spilservern og så API'en. Derfor har de ud over at indgå i denne MVC model fået deres egen.[1]



Figur 3: MVC overblik over helesystemet

1. WEBSIDEN ER VIEW FORDI: Det er den brugen interegere med og i princippet det eneste han ser, den fungere som systemets UI
2. SPILESERVEREN ER CONTROLLER FORDI: Den står formelt set for alt spil kommunikation mellem modellen med databasen og frontented med websiden. Nogle gange er model dog nød til at tale direkte til View, når det kommer til login og ligende. Det er ikke idealt, men bedre end et system der ikke virker.
3. REST API'EN ER MODEL FORDI: det er den der holder alle de data vi bruger. Så det er ligesom vores data layer.

[1] Se It-Systemets elemter afsnittet

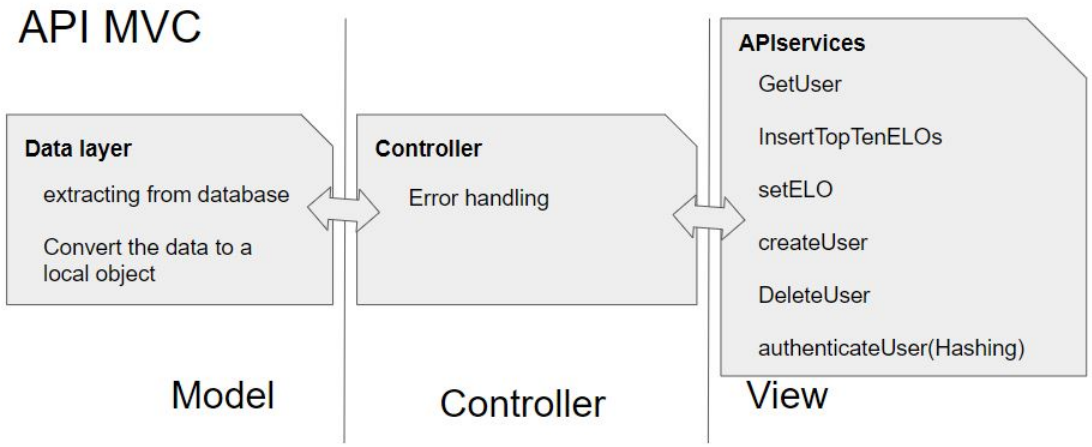
4 IT-Systemets elementer

4.1 API og REST Arkitektur

Ud fra vores krav kom vi frem til at vores API helt konkret skulle kunne

- Poste en ny score til databasen
- Hente Bruger Data, hvis brugeren giver det rigtige password
- oprette en ny bruger

. Ud fra det har vi Designet vores API ud fra MVC princippet



Figur 4: En oversigt over hvordan vi forstillede os at bygge vores api

4.2 Database

De bruger data vores database skulle kunne holde er

- Brugere
- Elo Score Data
- hashede Passwords

.
Det kom vi frem til at gøre ved

Field	Type	Null	Key	Default
username	varchar(15)	NO	PRI	NULL
password	varchar(500)	NO		NULL
elo	int(10)	YES		1000

Figur 5: her er et skemadiagram over databasen

Det er smart fordi så behøver vi ikke scores når vi har elo, fordi scoren kan omregnes til en stigning eller et fald i elo scoren. Alle Passwordsene er hashede så selvom man tilgår databasen kan man ikke bruge den information man får ud til noget.

4.3 Spil

Da vores online spil skal køre i et HTML dokument blev vi nød til at kode det i JavaScript. JavaScript er et imperativt programmerings sprog, som.. to be continued

Hvordan banerne vender

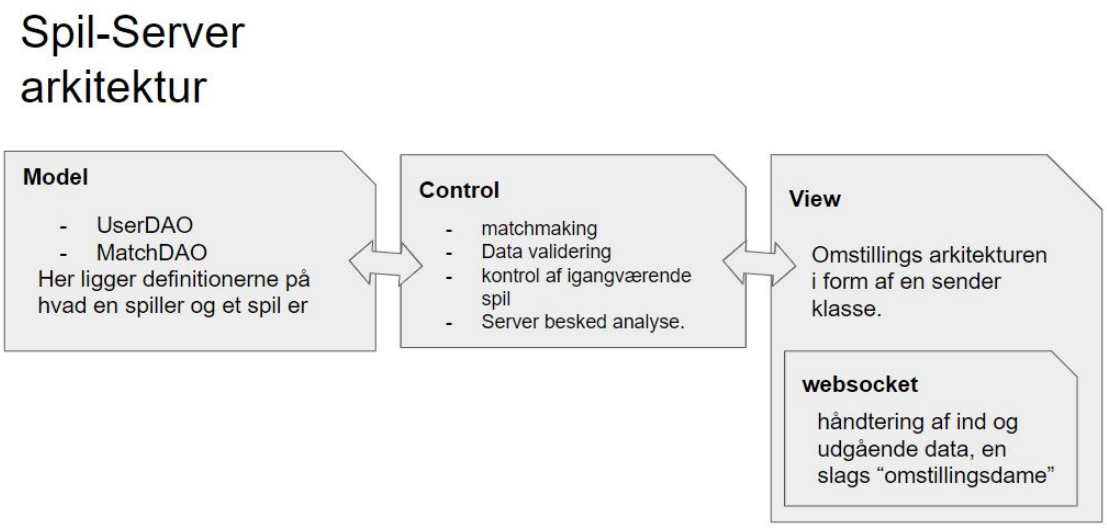
Klassediagram

Flowchart / Aktivitets diagram

4.4 Webserver

Vi vil gerne bruge websockets, og vi skal implementere det i java. Det eneste webserver tjeneste vi kender der tilbyder det er Tomcat, derfor bruger vi tomcat som webserver til at hoste vores website.

4.5 SpilServer

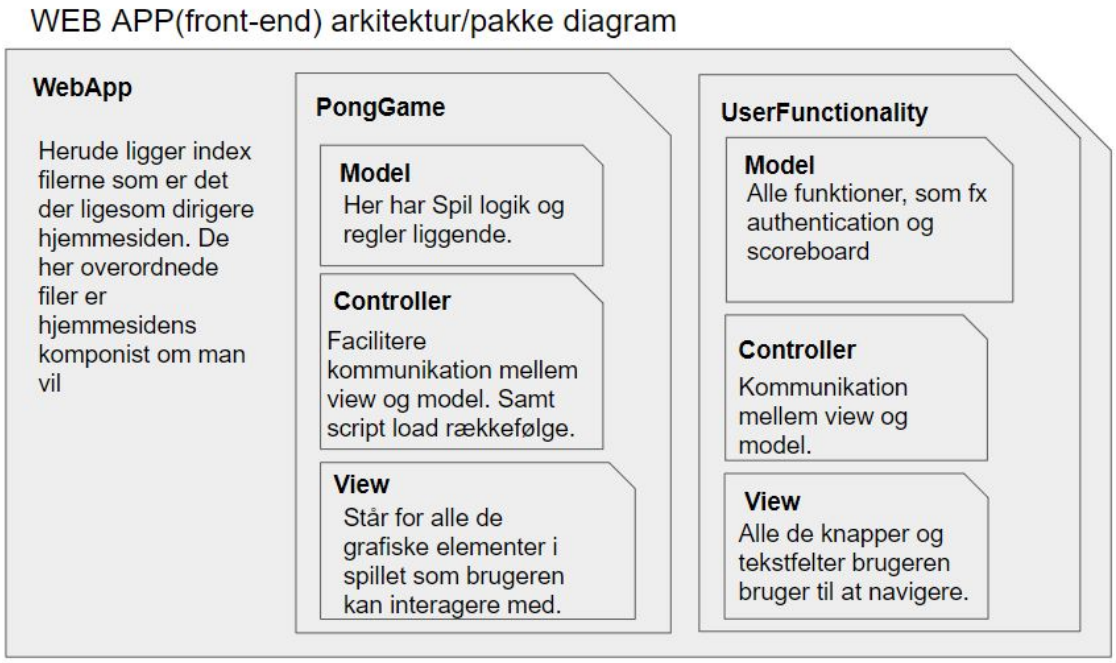


Figur 6: Et overblik og hvad der sker hvor i spilserveren

Spil serveren er logik centert i IT-systemet. Det er her et flertal af 'de tunge beregninger' foregår. Det er også den der agere som systemets kontroller. Den skal derfor kunne facilitere og hånterer kommunikation mellem Rest Api'en og websiten.

4.6 Webside og UI

Vi har valgt at have to MVC størrelser på vores webside. Så man kan udskifte pong spillet eller websidefunktionerne uden at det får en effekt for den anden. Det gør også koden mere overskulig. Så har vi delt de to systemer op i hver deres MVC model for at holde kopblingen lav.



Figur 7: Et overblik over websidens arkitektur

4.7 ELO-rating system

Fordele: Det giver points alt efter hvad niveau en spiller er på, så man få flere pointes for at vinde over spillere som er bedre end en, end ved at spille med spillere som er dårligere end en.

Ulemper: Folks rating er baseret på deres spil historik. Så ind til de har en fornuftig lang spil historik, kan man risikere at de bliver matched med spillere af et forkert niveau eller får et forkert antal points ud af deres spil.

Teori(matematikken bag):

implementeringen(skal måske stå under implementering?):

Del III
Implementering

5 System-overblik og Samlet pakke Diagram

Her indsætter vi det pakke diagram der skal være i rapporten.

6 Pong spillet

Der er mange elementer der skal harmonere for at spillet skal fungere som vi gerne vil have det til. Som tidligere nævnt er det kodet i JavaScript, da det er en applikation der kører i en webbrowser og som derfor skal spille sammen med HTML.

6.1 Når man logger ind

Inde i index.html dokumentet har vi lavet et login system, som snakker sammen med spilserveren (Mere om det i afsnittet om "webside og UI"og i "Game server og Websockets").
Her kan der indtastes brugernavn og adgangskode, som skal valideres af spil serveren.

INDSÆT BILLEDE / KODE HER.

Når brugernavn og adgangskoden er accepteret køres metoden createConnection(), som opretter en forbindelse via websocket til spil serveren, og fortæller at brugeren gerne vil finde et spil. Derefter kalder den metoden decodeEvent().

```
function decodeEvent(jsonObject){
    switch (jsonObject.code) {

        case 101:
            findingGame(jsonObject);
            break;

        case 102:
            acceptGame002();
            initializeGame();
            break;

        case 103:
            sendGameState103and010();
            break;

        case 10:
            gameDataUpdatedata(jsonObject);
            break;

        case 201:
            wrongUserNameOrPassword();
            break;
```

```
        case 202:
            userAlreadyLoggedIn();
            break;

        case 203:
            unableToAuthendizise();
            break;

        case 210:
            opponentDisconected();
            break;
    }
}
```

decodeEvent() bliver kaldt i connection.onmessage metoden, som:

```
connection.onmessage = function (event) {
        var obj = JSON.parse(event.data);
        decodeEvent(obj);
    }
```

Som ligger inde i createConnection() metoden. Den bliver kaldt hver gang klienten får en besked via den websocket. decodeEvent() har en switch som har cases der bliver valgt ud fra hvilken kode der står i det JSON object som kommer ind. Først bliver case 101 kaldt efter 2 brugere har sagt at de er klar til at finde et spil, hvorefter case 102 bliver kørt, når et pil er fundet, der accepterer og initialisere spillet.

6.2 Spil kørsel

når spillet initialiseres køres metoden initializeGame().

```
function initializeGame(){
    document.getElementById("loading").innerHTML = "A game has been found...";
    canvas.style.display = 'inline';
    setupGame(chosenScore);
    animate(runGame);
}
```

initializeGame() fortæller at et spil er fundet, sætter canvas til at fylde det ud som spillet er sat til at fylde i pixels, og kalder setupGame() og animate() metoderne.

```
var setupGame = function(chosenScore) {
    maxScore = chosenScore;
    player1 = new Player1();
    player2 = new Player2();
    ball = new Ball(350, 200);
    gameRunning = true;
```

setupGame() sætter scoren man spiller til, opretter objekter af spillere og bolden og sætter gameRunning variablen til true, som hjælper til at slutte spillet også med endGame() metoden.

6.3 Opret og render elementer

6.4 Player bevægelse

6.5 Opdatering af spillere

6.6 Kommunikation mellem spillere

7 API og REST Implementering

I view layer bruger vi @Post Metoder fordi det er sikre, da vi kun behandler JSON data, og man ikke kan skrive noget dirkete i HTML'en. Det eneste sted der stadig er en @get metode er til GetUser for der kan man kun få folks ELO score.[2]. Det er ikke så problematisk for det er alligvel offentligt tilgængeligt. Alt sensitiv information som password forespørges gennem en @post metode.

8 MySQL Database Implementering

Programmet kræver kun lagring af én type bruger. Af den grund nøjes vi med at bruge én MySQL tabel.

9 Game server og Websockets

GameServer er ansvarlig for at uddelegere spiltråde og spil og at spillet forløber hensigtsmæssigt.

9.1 Websockets

Forbindelsen imellem Web Serveren og SpilServeren forgår via Websockets. Websockets er full-dublex og muliggøre at kommunikation imellem server og klient går begge veje. Altså kan GameServeren sende frames til klienten uden at der er foretaget en forespørgsel. En forbindelse imellem klient og server der er etableret via websockets mindsker latency og kræver mindre båndbredde da hver besked ikke er påkrævet en header. For at en forbindelse imellem server og klient kan finde sted skal der foretages et "Websocket Handshake". Dette sket ved at klienten sender en HTTP request, hvor der i requestheaderen indgår "Upgrade"som informere GameServeren om at klientens ønsker at etablere en forbindelse via en websocket. Nedenfor ses et eksempel på, en request for en websocket.

```
GET ws://websocket.example.com/ HTTP/1.1
Origin: http://example.com
Connection: Upgrade
Host: websocket.example.com
Upgrade: websocket
```

Gameserveren svarer ved at sende et svar, hvori der indgår status koden 101 i svarets header. Hvis serveren undestøtter Websockets vil serveren opgradere HTTP forbindelsen til en Websocket protokol der fortsat anvender TCP som dens underlæggende protokol. Et eksempel på en respons header (svar med header) ses nedenfor.

[2]Referer til den den af rapporten hvor vi snakker om ELO

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Wed, 16 Oct 2013 10:07:34 GMT
Connection: Upgrade
Upgrade: WebSocket
```

For en mere detaljeret beskrivelse at Websockets og dens forbindelse kan findes i skrivelsen, "RFC6455".

En Websocket betragtes som et endpoint i en forbindelse. Når en Websocket skal programmeres anvendes der følgende annotationer.

- @ServerEndpoint: En annotation for en klasse der garanterer at en WebSocket er til gængelig når en specifik URI forespørges.
- @ClientEndpoint: En annotation for en klasse der garanterer at en pågældende klasse bliver betragtet som en Websocket.
- @OnOpen: En annotation for en metode. annotationen sikrer at en metode bliver iværksat når en Websocket bliver etableret.
- @OnMessage: En annotation for en metode. En metode med denne annotation modtager en besked fra en WebServercontainer når en besked er sendt til et endpoint.

9.2 Klassediagram

9.3 Flowchart

10 Webside og UI

10.1 Når man skifter webside

10.2 CSS

11 Interface

Et interface er en reference type i Java. Man refererer til den når man skal bruge Dens metoder i andre klasser. Der er dog ikke implementeret noget i metoderne i en interface. De er abstrakte. Et interface er en slags klasse med en samling af abstrakte metoder, som en anden klasse kan implementere vha. implements nøgleordet.
Når en klasse implementerer et interface så laver den en slags kontrakt med interfacet om at implementere funktionaliteten på metoderne, som interfacet har defineret.

Når man skal lave en interface, skriver man interface i stedet for class.
En klasse beskriver attributter og adfærd som et objekt har, mens en interface indeholder adfærd som en klasse implementerer.
Medmindre klassen som implementerer interfacen er abstrakt, så skal alle metoderne fra interfacet implementeres i klassen.

En interface er forskellig fra en klasse på følgene måder:

- Man kan ikke instansiere en interface.

- En interface har ingen constructor, defineret eller ej.
- Alle metoderne i interfacet er abstrakte. De har ingen tuborg klammer men ender på semikolon.
- Interfaces kan kun indeholde konstanter, som er pr. definition public, static og final.
- En klasse kan implementere flere interfaces, men kun extende en anden klasse
- Et interface kan ikke implementere andre interfaces eller klasser, men kan extend x antal interfaces dog ingen klasser.

Interfaces er en vigtig mekanisme til at få lav kopling (GRASP) fordi at ens kode kun referer til interfacet, og ikke til den konkrete implementering af metoderne i interfacet.
Det gør også koden mere skalerbar.
Det gør at man kan implementere anden kode ud fra den interface uden at det har nogen effekt på ens kode.

11.1 Vores valg af interface

BLA BLA

BLA BLA

BLa BLa

11.2 Eksempel på interface

```
public interface IUserDAO {
    /**@author Claes, Simon
     * The purpose of this class is being able to access the UserData.
     * As is also stated in the name Data Access Object (DAO)
     * @return Userdata
     * @throws SQLException
     */

    /**
     * Retrieves a user by ID
     *
     * @param username  The username of the user
     * @return     User object
     * @throws DALException
     */
    IUserDTO getUser(String username) throws DALException;

    /**
     * Inserts a user into the database
     *
     * @param username    Users username
     * @param password    Users password
```

```
     * @return            String confirmation
     * @throws DALException
     */
    String createUser(String username, String password) throws DALException;

    /**
     * Controls the password is correct through hashing
     *
     * @param username      username of User
     * @param password Password of User
     * @return        True if password is correct, else false
     * @throws DALException
     */
    String checkHash(String username, String password) throws DALException;

    /**
     * updates the elo of a player in the database
     * @param username
     * @param elo
     * @return String: error message
     * @throws DALException
     */
    String setElo(String username, int elo) throws DALException;

    /**
     * @author Simon
     * Get the 10 users with highest elo rating.
     * @return IUserDTO List.
     * @throws DALException
     */
    List<IUserDTO> getTopTen() throws DALException;

    /**
     * @author Simon
     * Deleting users in the DB.
     * This function is exclusively for testing purposes.
     * @param username
     * @throws DALException
     */
    public void deleteUser(String username) throws DALException;


    /**
     * Customizable exception for explaining Database Access Layer exceptions
     */
    class DALException extends Exception {

        //Til Java serialisering...
        private static final long serialVersionUID = 7355418246336739229L;

        public DALException(String msg, Throwable e) {super(msg,e);}
        public DALException(String msg) { super(msg);}
    }
```

12 Bruger guide

Del IV
Test

12.1 GameServer

DTU har tildelt os 2 stk. trådløse routere, med internetforbindelse. Forbindelsen imellem client og GameServer er tested ved at to laptops logger på routerens netwærk, hvorefter WebSocketen bliver deployed på GameServeren, der i dette tilfælde er en Tomcat server v. 8.5.35. En anden laptop spiller rollen som client og der etableres en forbindelse imellem client og webserver ved at client sender en request til GameServeren. WebSocketen på serversiden er configureret således at der sendes en besked tilbage til clienten når der er etableret en forbindelse. Hos klienten ses det at der kommer en response message fra WebSocketen på serversiden, med den besked som WebSocketen er konfigureret til.
Derved kan det bekræftes at forbindelsen er etableret, WebSockets kan anvendes og at roterens konfigurationer er korrekte.

Routeren konfigurationer

Virtual Servers List

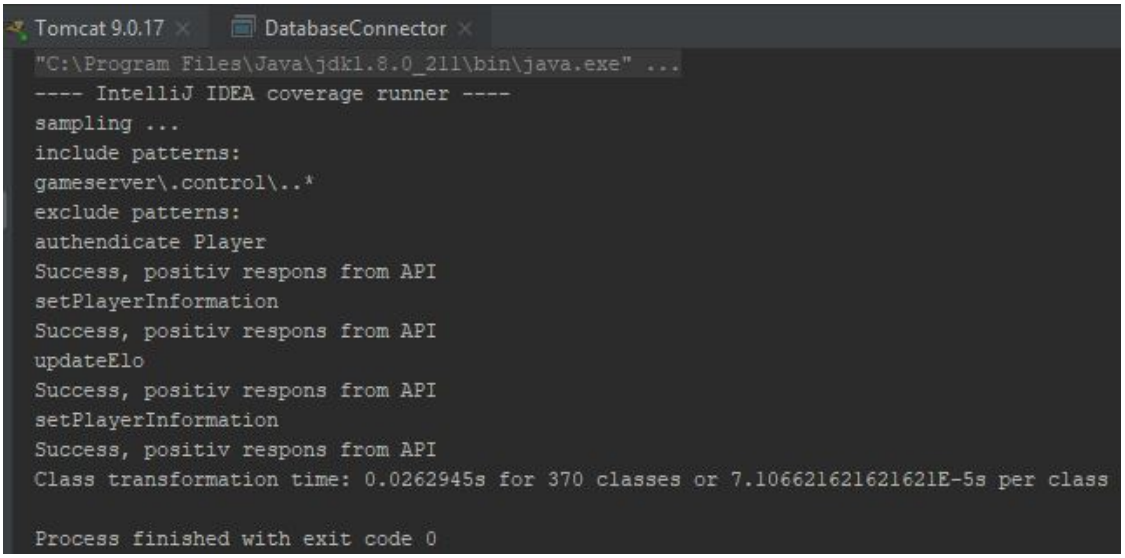
Name	WAN Port	LAN IP Address	LAN Port	Protocol	Status	Options
WebServer	8887	192.168.1.100	8887	TCP/UDP	On	Edit Delete
TomCat	8080	192.168.1.101	8080	TCP/UDP	On	Edit Delete
GameServer	9876	192.168.1.101	8080	TCP/UDP	On	Edit Delete
GameServer2	80	192.168.1.100	80	TCP/UDP	On	Edit Delete

Figur 8: IP adresser der anvendes på router

12.2 Forbindelse imellem Gameserver og Database API

For at teste forbindelsen imellem GameServeren og databasens API er der ikke lavet unittests. Istedet er der lavet en main klasse der eksekverer alle de metoder, hos GameServeren der anvendes til at indhente data fra databasen. Allemetoderne findes i src /GameServer/controle".
Forbindelsen er testet ved at API'en er deployed til Tomcat og der er lavet en main metode i klassen "DatabaseConnector". Efter at Tomcat metoden er startet op bliver main metoden eksekveret resultatet af de metoder der eksekveres printes o consollen.

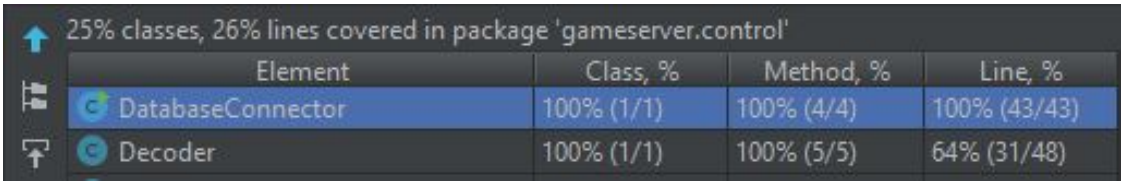
Billederne nedenfor viser et print i consollen. **Test af Gameservers forbindelse med database API**



Figur 9: Resultat af forbindelsestest imellem GameServer og database API

Af figuren ovenfor ses det at alle metoder som gameServern anvender til at indhente database fra databasen har haft success, hvilket understreger at der er oprettet en forbindelse imellem API og GameServeren.

Af figuren nedenfor fremgår det via codecoverage at allemetoder i databaseconnector er blevet testes.



Figur 10: Code coverage af testet kode i

12.3 Random stuff- Tidligere rapport måske??????

Der er løbende blevet lavet tests af programmet. Her er det blevet testet om et nyt stykke kode virker. De primære tests er blevet udført efter at de forskellige lag er blevet samlet (view-, controller- og model-lagene). Dette er blevet lavet som en systemtest, hvor vi blot har testet på det hele det færdige system. Derudover er det blevet testet med en JUnit test hvorvidt metoden *validateUserInfo()* i *CRUD*, en givne informationer korrekt.

13 Diskussion

Hvad gør det at der anvendes en WebSocket istedet for en Rest API. En REST API (asynkron forbindelse, hvor den sendes en forespørgsel, hvorefter der modtages en HTTP respons). Når der

er tale om et real time spil er dette ikke gunstigt, idet at en asykron forbindelse kan give en relativ lang ventetid. WebSockets der kan give en syntkon forbindelse sikrer at ventetider kan blive noget kortere.
Grundet at computere kører med forskellig fps, vil et loop blive eksekveret med forskellig hastighed. Dette kan resultere i et vist lack der gør at en de positioner der bliver sendt imellem de to computere ikke er i overenstemmelse. Pong spillets spillogik er baseret på, at hver "spiller"beder om at få opdateret fps. Jo hurtigere en computer er til at sende- og modtage fps fra serveren jo hurtigere vil den kunne eksekverer spillet. Dette kan skabe et problem eftersom at de to computere afvikler spillet i en forskellig hastighed, hvilket vil skabe en form for "lack"i spillets eksekvering.
Som applikationsnet
Lav en tegning af, hvordan det laptops er forbundne.
Hvorfor er det at en spiller spiller på samme banehalvdel.
Hvordan er forsinkelsen blevet håndteret og hvad kunne der have været gjort anderledes.
Fejlhåndtering
Om brugen af error codebla bla bla

Roller
Der er intet sted i programmet, hvor de forskellige roller man kan have bliver håndteret. Når man bla bla bla

Unikke datatyper i databasen
Databasen er, på nuværende tidspunkt, organiseret således at, det kun er primær nøglen og en bla bla bla

14 Konklusion

Der er blevet udarbejdet et system som kan virke som et modul for et bla bla bla

Del V
Kilder

something smart[1]

Referencer

[1] Albert Einstein. “Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]”. I: *Annalen der Physik* 322.10 (1905), s. 891–921. DOI: http://dx.doi.org/10.1002/andp.19053221004.

15 Billag

Disclaimer: Dele af disse billag kommer fra tidligere afleverede raporter der kan derfor forekomme selvplagiat:

16 Forklaring af Exceptions som vi anvender

Når et program kører, kan det ske at det støder ind i et problem der vil stoppe dets normale flow eller helt crashe det. Dette problem som programmet løber ind i kaldes en 'exception'.
For at undgå at programmet crasher kan man tage og 'catche' disse exceptions for derefter at behandle dem og/eller melde tilbage til brugeren at der er sket en fejl og om muligt hvilken. Denne praksis kaldes for "exception handling".
Helt teknisk, når der sker en fejl i en metode, så bliver der skabt et exception objekt, med informationer, som bliver afleveret til runtime systemet. Som derefter vil lukke programmet og skrive hvad der er galt.
Når en metode har kastet en exception, så vil systemet automatisk prøve at finde en exception håndtering i call stakken (de metoder koden er gået igennem indtil fejlen kom).

16.1 Try-catch

Try-catch er en måde, hvor der bliver afprøvet en stump kode, i en try blok, og som derefter fanger en eller flere exceptions, hvis de er der, hvorefter man kan håndtere dem.
Kort sagt bruges try-catch til at omkranse noget kode som man tror vil smide en exception.
Her er nogle punkter man skal huske på:

- Man skal kun omkranse det kode som man tror vil smide en exception, da al resterende kode i try-blokken ikke vil blive kørt, hvis der bliver smidt en exception. Kun koden efter catch-blokken bliver kørt.
- Man behøver ikke nødvendigvis specificere den exception man tror vil blive smidt i catch. Man kan også bare skrive en af dens parent klasser, helt op til den kaldet ´Exception´.
- Man kan både skrive en custom besked i catch blokken, eller bare printe exception variable navnet for at se hvad det er for en exception og hvad der sker:
  - System.out.println("custom message");

  - System.out.println(exception);

- Man skal kun bruge try-blokken til at afprøve kode for exceptions, og man skal kun bruge catch-blokken til at håndtere exceptions.

- Man kan have flere catches til en try-blok. Start med de nederst prioriterede exceptions og gå op mod dens parent exceptions, sluttende med Exception klassen. Der bliver dog kun printet 1 exception og det er den der bliver fanget først i try-blokken. Hvis der ikke er den specifikke exception i catchen printer den parent Exception catchen, der burde fange alle exceptions.

- Man kan også lave nested try-catch.

Try-catch syntax:

```
try
{
    //Udsagn der kan smide en exception
}
catch (exception(type) e(objekt))
{
    //Fejlhaandterings kode
}
```

16.2 Throw

'Throw' nøgleordet bruges til at kaste en exceptions. Man kan både kaste 'Checked' og 'Unchecked' exceptions.
Det bliver primært brugt til at kaste custom exceptions:

```
    static void fun() {
        try{
            throw new NullPointerException("Custom besked");
        }
        catch(Exception e){
            System.out.println("Fanget i fun().");
            throw e; // kaster igen
        }
    }

    public static void main(String[] args){
        try{
            fun();
        }
        catch(Exception e){
            System.out.println("Fanget i main.");
        }
    }
```

Dette ville printe begge de printede sætninger

16.3 Throws

Med ´throws´ nøgleordet, deklarerer man at der måske vil ske en exception under metodens kørsel, og der nok skal tages forholdsregler i form a try-catch senere i call stakken.
Når den forventede exception sker, vil metoden stoppe og den vil blive kastet tilbage til hvor metoden blev kaldt fra. Her er det så vigtigt at man har en try-catch klar, hvis man ikke thrower den videre.
Med throws nøgleordet håndterer man kun checked exceptions, da unchecked exceptions, er under ens kontrol og kan rettes ved at ordne koden, og errors er uden for ens kontrol.

Throws syntax:

```
void eksempel() throws ArithmeticException{
//Udsagn
}
```

16.4 Throw vs Throws

Throw	Throws
Throw bruges kun til at kaste en exception	Throws bruges til at deklarere en exception
Man kan ikke propagate exceptions	Checked exceptions kan propagates med Throws
Throw er fulgt af et exception objekt	Throws er fulgt af en klasse
Throw bruges inde i en metode	Throws bruges i en metodes signatur
Man kan ikke 'Throw' flere exceptions samtidig	Man kan deklarere flere exceptions samtidig

16.5 Typer af Exceptions

I Java kendes exceptions under klassen Exception og mere specifikt de andre klasser, der arver fra Exception.

Checked exceptions
Checked exception bliver tjekket når koden compiles. Det betyder at hvis en metode smider en checked exception så skal den håndteres i en try-catch blok, eller metoden så skal metoden deklarere den exception med et throws nøgleord.

- SQLException - Fejl der har med en database at gøre.
- IOException - Fejlslagne eller afbrudte I/O operationer.
- ClassNotFoundException - Når der bliver brugt en klasse som ikke eksisterer.

Unchecked exceptions
Unchecked exceptions bliver ikke tjekket når koden compiles. Det betyder at der ikke kommer en compilation fejl, også når den ikke er håndteret, når den exception bliver smidt.
De kommer tit når en bruger laver input som ikke dur. Som programmør skal man derfor huske at håndtere forkerte input i koden så de ikke kan opstå.

- NullPointerException - Det kan ske når du prøver at interagere med en tom reference.
- OutOfBoundsException - Hvis man prøver at interagere med et index længere end arrayet er langt.
- ArithmeticException - Fx at dividere med 0.
- NumberFormatException - En String, uden rette format, prøver at blive konverteret til en nummer type (int, long etc.)
- IllegalArgumentException - En metodes argument er ikke brugtbart.

Errors
Errors er ikke til at redde. Det kan fx være OutOfMemoryError, VirtualMachineError og AssertionError.

16.6 DALException

I det afleverede system er der blevet lavet en custom exception, DALException, der ligger i klassen IUserDAO.
Dette er en exception der, ligesom de predefinerede exceptions såsom NullPointerException, arver fra Exception klassen.
Denne er lavet for at håndtere de fejl der kan ske med SQL databasen, i dette tilfælde SQLExceptionʼs. Hvis der bliver fanget sådan en exception bliver der smidt en 'error code' som kan give information til brugeren om hvad der er sket. For dette system vil DALException smide en "ERROR_DALERROR", der er kendt under en systemvalgt int, så der kan blive kommunikeret til brugeren at der er sket en fejl.

16.7 Error code

Inde i systemets V_IUI klasse kan man finde såkaldte "ERROR_something"der er en kendt int værdi. Denne int værdi kan man referer til når der skal give en fejlmelding til brugeren. Dette kan ses blive brugt i 'showError()' metoden, hvor at der bliver printet en meddelelse til kunden, alt efter hvilken identificerende int den har fået tilsendt. Her kan man enten få meget præcise errors printet ud til brugeren eller have en generel fejlbesked som den førnævnte "ERROR_DALERROR", der printer meddelelsen "System have run into a problem with the database". Vi har valgt at implementere dette således, for lade det være op til klassen der implementerer V_IUI hvordan disse fejl skal vises til brugeren.