Espergærde Gymnasium og HF

Appudvikling

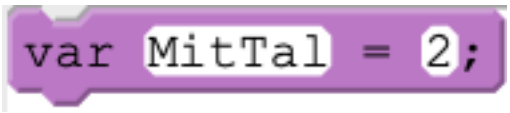
Informatik

Karl Malassé, MA, dele af noterne "Spiludvikling" og "Internettet og websider"Mette Machholm, MQ, bearbejdet til en ny version målrettet mobilapps i 2022

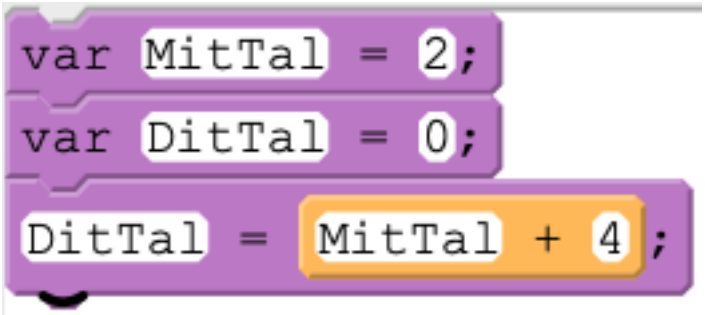
1 Variabler - Programmering i App Lab

Når man skal gemme data i et computerprogram, gemmes de i variable. Man kan tænke på en variabel som en boks, hvor man kan gemme data, indtil man skal bruge det på et senere tidspunkt. Når data gemmes, lægges det i boksen. Når data senere skal bruges, kigger man i boksen for at se værdien af variablen.

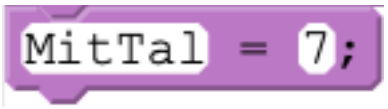
Variablen MitTal er et eksempel på en variabel. I App Lab skal man først oprette variablen inden den kan bruges. Man kan dog tildele variablen en værdi samtidigt med, at den oprettes. Her tildeles variablen værdien 2, samtidigt med at variablen oprettes. Man kalder det at initialisere en variabel



Når man på et senere tidspunkt skal bruge den værdi man har gemt i variablen MitTal, skriver man navnet på variablen på det sted, hvor tallet skal bruges. Her henter man tallet gemt i MitTal, dvs. 2 og lægger 4 til. Derefter gemmes tallet i variablen på venstre side af lighedstegnet, dvs. efter denne blok kode ligger værdien 6 gemt i variablen DitTal.



Hvis man ønsker at ændre værdien i variablen MitTal, kan man gøre det



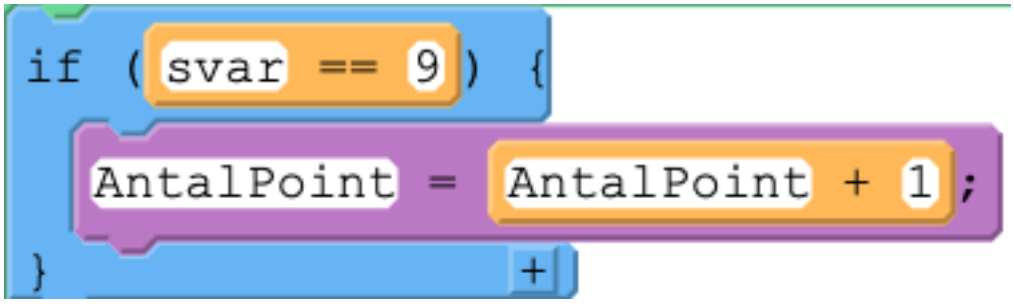
Eller man kan lægge 5 til den værdi som allerede ligger gemt i en variabel

1.1 Brug variabel til point eller liv i spil

Hvis vi skal gemme og vise point opretter vi en variabel til at gemme pointene i. Her variablen AntalPoint.

AntalPoint sættes til 0 fra begyndelsen.

Når vi får et point i spillet, lægges 1 til AntalPoint. Her skal svaret være lig 9, hvis der skal tildeles et point.



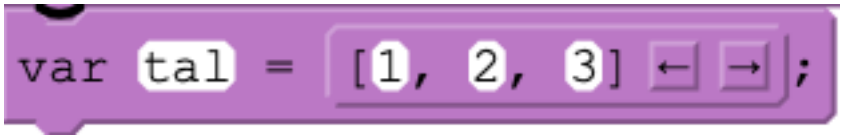
På samme måde kan man tælle ned fra et antal, hvis man fx mister et liv i spillet

1.2 Lister

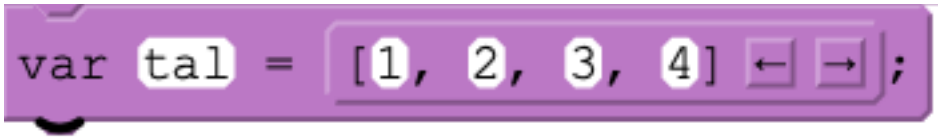
Når man har flere variable af samme slags, er det en fordel af lægge dem i en liste. En liste findes under variable i App Lab.

Der findes lister til tekst. Så sættes værdierne i "citationstegn"

Eller tal, hvor der ikke er citationstegn



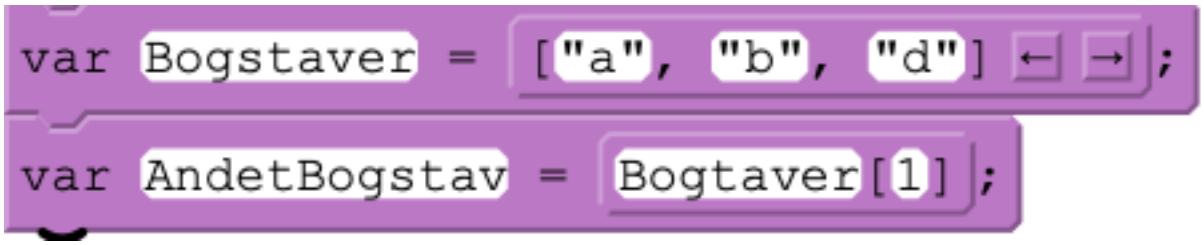
Man kan tilføje flere ved at klikke på højrepil



1.3 Hente værdier i lister

Når man skal hente en værdi i en liste, skal man skrive hvilket nummer element man vil hente i listen.

Man tæller fra 0, så nummer 1 giver det 2. element i listen. Nedenfor sættes AndetBogstav til bogstavet ”b”.

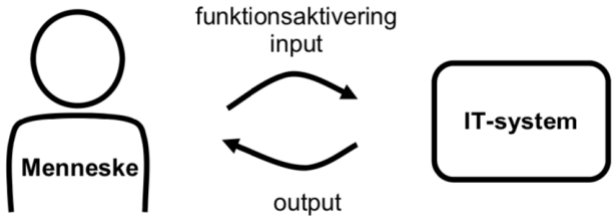


2 Evaluering af design

Når man har lavet en kørende prototype, skal den evalueres. Dette gøres bl.a. for at rette uhensigtsmæssigt design og forbedre produktet f.eks. et spil. Der findes forskellige metoder til at test prototypen. Vi vil her se på test, hvor mulige brugere inddrages i testen af prototypen. Inden testen beskrives skal vi kende definitionerne på Interaktion, Usability og User Experience.

2.1 Interaktion

Interaktion kan defineres bredt som alt samspil mellem en bruger og et it-system. IT-systemet kan være et spil eller en webside. Interaktionen går to veje: 1) Brugeren aktiverer it-systemet med input. Input kan f.eks. ske via tastatur, mus eller skærmen på en mobiltelefon. 2) IT-systemet laver et output som ofte afhænger af brugerens input. Samlet beskrives input og output mulighederne ofte som systemets brugergrænseflade, dvs. grænsen mellem menneske og IT-system.



Figur: Interaktion mellem bruger og IT-system

2.2 Usability

IT-systemer med høj *Usability* er lette af anvende for brugeren. I et spil med høj Usability vil brugeren have let ved at forstå, hvordan spillets regler fungerer, hvordan man styre spillet og hvad målet med spillet er. Reglerne skal være logiske og forståelige. Et spil der minder meget om andre spil og styres på samme måde som andre tilsvarende spil, vil ofte have en høj Usability, da spillet opfører sig som brugeren forventer.

Hvis man byttede om på virkningen af højre og venstre piletast, ville det være svært for brugeren at styre en spilfigur med piletasterne, og brugeren ville opleve at spillet havde en lav Usability.

2.3 Userexperience

Brugerens oplevelse af IT-systemet kaldes ofte Userexperience. Har et spil god Userexperience, vil spilleren ofte motiveres til at fortsætte med at spille. I spil kan det fx være et pointsystem, som understøtter spillerens motivation til at fortsætte med at spille. Oplevelsen af spillet er god og dermed vil spillet formodentlig også have en god Userexperience. Man kan sammenligne Usability og Userexperience i figuren nedenfor.

Figur: Usability vises inderst. Kriterierne for Usability er bl.a. at IT-systemet er effektivt at bruge og let at lære at bruge. Userexperience vises yderst. Kriterierne for Userexperience er bl.a. at IT-systemet skal være motiverende og underholdende at bruge. (Roger et al., 2011)

2.4 Test af Usability og Userexperience

Når man tester Usability, skal man planlægge testen, udføre testen, fortolke resultaterne og til sidst konkludere hvilke ændringer i IT-systemet, man vil prioritere at lave. Når man tester et IT-system er det bedst, hvis man kan få "rigtige" brugere, som ikke kender systemet, til at prøve systemet.

2.4.1 Planlægningen af testen af et spil

- Vælg hvem der skal afprøve spillet. Det kan være andre elever i klassen eller andre elever på skolen, hvis de er i målgruppen for jeres spil
- Vælg hvordan testen udføres. Det er bedst, hvis brugeren selv skal finde ud af hvordan jeres spil virker. Så kan I observere, mens brugeren afprøver spillet.
- Vælg hvad I vil fokusere på. Eksempler "Hvor lang tid går der før brugeren finder ud at bruge spillet? (brug stopur)", "Skal brugeren have hjælp for at komme i gang eller videre?"
- Lav nogle afsluttende spørgsmål om Usability, som alle testbrugere svarer på efter testen af prototypen. Eksempel: "Var der noget du ikke forstod, hvordan du skulle gøre?" "Har du forslag til forbedringer at interaktionen med spillet?"
- Lav nogle afsluttende spørgsmål om Userexperience. Eksempler "Fik du lyst til at prøve igen?" "Forslag til tilføjelser til spillet, der kunne gøre spillet sjovere?"

2.4.2 Udførelsen af testen at et spil

- Mange testbrugere giver et bedre og mere brugbart resultat, men det tager tid at lave test med mange brugere, så ofte begrænses antallet af testbrugere
- Det er bedst, hvis en af jer taler med testbrugeren, mens en anden observerer og noterer reaktioner på spillet og svar på spørgsmål.

- Man spørger mulige testbrugere, om de vil være med til at teste spillet og derefter viser man den kørende prototype til testbrugeren.

2.4.3 Fortolkning af testen

Når man har udført testen, skal man tolke resultaterne. Der kan opstå mange problemer og usikkerheder i tolkningen af testresultaterne, og nogle gange er det bedst at lave en ny forbedret test af ens IT-system, når man er blevet klar over, hvilke resultater, man ikke kan tolke, fordi man fik for lidt viden ud af testen.

2.4.4 Konklusion på testen

Til sidst laver man en liste, hvor man prioriterer de forbedringer, man bør lave. Noter gerne alle mulige forbedringer og vurder om de er "lette"/"svære" at lave, samt om de er "meget vigtige" eller "mindre vigtige".

2.5 5-sekunderstest[1]

Da det er vigtigt, at brugere hurtigt kan se, hvad et spil, en webside eller et andet it-produkt kan bruges til eller hvordan det virker, er det også vigtigt at teste brugeres først indtryk af et it-produkt. Til det har man udviklet en 5-sekunderstest, hvor testpersonerne kort ser på produktet i 5 sekunder og derefter skal fortælle om, hvad de lagde mærke til.

2.5.1 Planlægning af testen

Vælg hvilke spørgsmål du vil stille testpersonen. Typiske spørgsmål kan være "Hvad lagde du mærke til?", "Hvem tror du it-produktet henvender sig til?", "Hvad er dit indtryk af designet?"

2.5.2 Udførelsen af 5-sekunderstest

Det er vigtigt, at du fortæller testpersonen, at han/hun har 5 sekunder til at kigge på it-produktet og derefter skal fortælle om det uden at kunne se på det. Derefter vises it-produktet eller prototypen i 5 sekunder (tag tid). Når fremvisningen er overstået, stiller du dine forberedte spørgsmål og noterer testpersonens svar.

2.5.3 Fortolkning og konklusion

Ud fra de svar, du har fået, skal du konkludere og prioritere de ændringer du vil lave. Det kan fx være, at du vil ændre farver på eller placering af de vigtigste elementer i spillet eller på websiden.

[1] https://fivesecondtest.com

3 Spil[2]

3.1 Spilgenre: handling

Spil kan genreinddeles på et utal af måder, og en af de mest brugte kriterier er handling og succeskriterium. Her kan man skelne mellem fire typer af spil: *action*, *adventure*, *strategi* og *proces*.

Action	**Adventure**	**Strategi**	**Proces**
Kamp - reflekser	*Gådeløsning - logik*	*Opbygning - analyse*	*Udforskning - leg*
			

I **actionspil** skal man have hurtige reflekser for f.eks. at undgå skud og forhindringer og gennemføre spillet. Eksempler på actionspil er Pacman, Snake, Skydespil m.m. Spillene kan være ganske enkle og gøres meget korte.

Adventurespillene udfordrer logikken og går ud på at løse gåder og gennemskue træk og konsekvenser. Adventure fra 1976 er det første spil, mens Myst var med til at gøre genren populær.

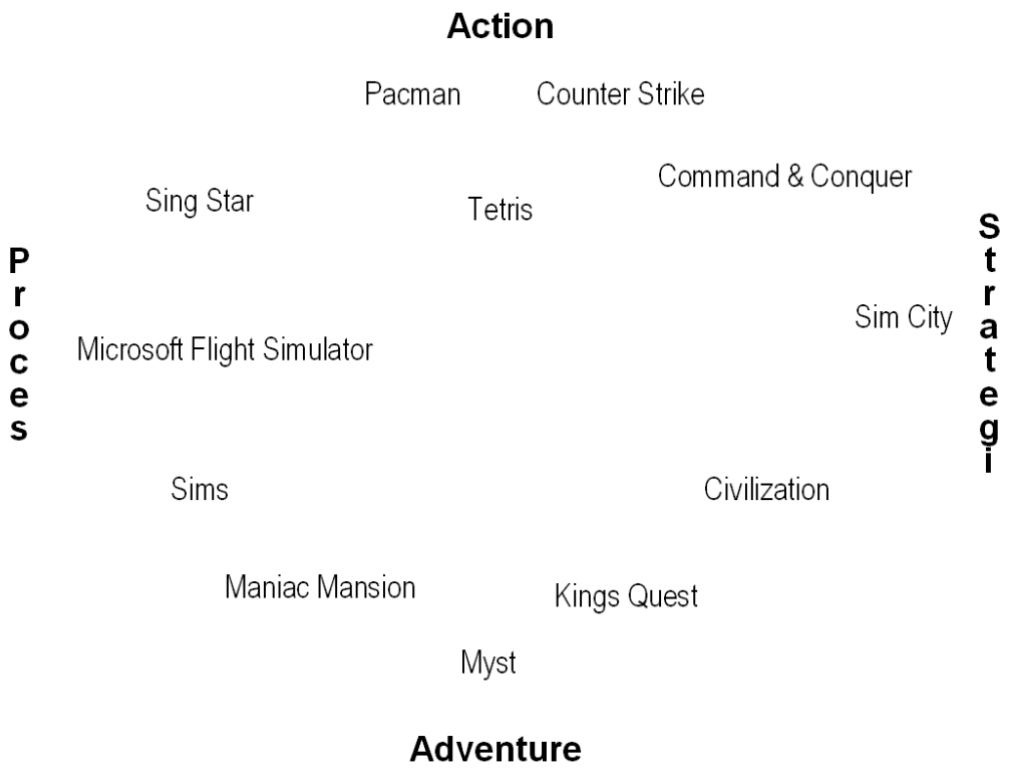
I **strategispillene** er det evne til at analysere, samle og organisere ukendte og kendte faktorer der tæller. Her er Sim-City der klassiske eksempel sammen med Civilization & Dune. - Da genre går ud på at opbygge ressourcer er spillene er oftest meget tidskrævende.

Man kan diskutere om en række **processpil** snarere er leg end spil, da der ikke er noget klart slutmål. Her står selve oplevelsen i centrum og "spillene" er meget forskellige fra flysimulatorer til dukkehusspil som Sims.

3.2 Ét spil - flere genre

Oftest ligger spil dog ikke inden for én genre, men trækker på elementer fra flere. Det kan illustreres ved at sætte genre overfor hinanden i et diagram og placere spillene tættest på den/de genrer de trække mest på. Trækker et spil på flere genre lægges det imellem disse:

[2] (http://iftek.dk)



3.3 Spillertyper

Richard Bartle fra University of Essex har lavet en model, som kan kategorisere computerspillere fra online multiplayer spil i 4 kategorier: **achievers, explorers, killers** og **socializers**. De fleste tilhører flere forskellige kategorier. (F.eks. 25 % Achiever, 80 % Explorer, 35 % Killer og 45 % Socializer).

Stræbere (achievers) fokuserer på at være bedre end de andre spillere, f.eks. ved en highscore. Det er vigtigt for denne type spillere at kunne vise sine opnåede resultater frem og kunne sammenligne sig selv med andre.

Udforskere (explorers) fokuserer på verdenen og historiefortællingen, f.eks. ved at prøve alle baner, inklusive de skjulte ekstra baner. Det er vigtigt for denne type spillere at spilleverdenen er troværdig og har ekstra valgfrit indhold.

Dræbere (killers) fokuserer på destruktion og magtfølelse, f.eks. ved at kunne ødelægge indhold i spillet. Det er vigtigt for denne type spillere, at de har mulighed for at være på den forkerte side af etik og moral. - Vigtigt: Et dræberspil kan virke meget frastødende på de andre spillertyper (især socializers), med mindre at spillet er lavet humoristisk eller har et klart budskab (F.eks. September 12th, der viser at bombning af terrorister medfører flere terrorister).

Sociale (socializers) fokuserer på social interaktion med andre spillere, hvor de kan udtrykke sig og vise tilhørsforhold (ligesom likes på Facebook). Det er vigtigt for denne type spillere, at de kan forbinde spillet til deres verden og sociale netværk.

3.4 Spillets Gameplay

Spiloplevelsen afhænger i høj grad af computerspillets rammer også kaldet **gameplay**. Rammerne (gameplayet) sættes af:

- reglerne
- interaktionsmulighederne.

Det er vigtigt, at spillerne *forstår* **og** *accepterer* rammerne og derfor følges/simuleres ofte regler, som spillerne kender på forhånd enten direkte fra virkeligheden eller fra andre spil. Kendte rammer sikrer, at spillerne hurtigt skal kunne komme i gang med spillet.

Bag et gameplay ligger der følgende elementer og overvejelser:

- Regler: logiske og forståelige.
- Motivation: Spillet skal give spilleren lyst til at fortsætte.
- Læring: Spilleren skal kunne komme videre i spillet.
- Identifikation: Indlevelse i spilkarakter:

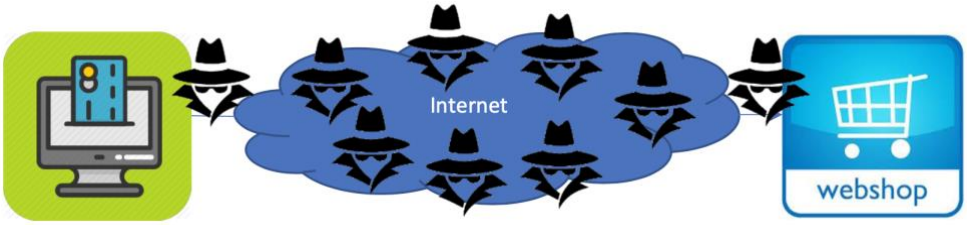
3.5 Modellering af Gameplay

Spillets gameplay - funktionalitet - illustreres storyboards og rutediagrammer inden udviklingen af spillet. Dette gøres for at kunne diskutere og udvikle gameplayet, men også for bagefter at kunne fordele og koordinere udviklingsarbejdet. Storyboards kan bruges til at illustrere sekvenser ved konkrete situationer i spillet og har den fordel, at også brugergrænsefladen illustreres. De er især brugbare til at udvikle centrale passager i spillet som start og forskellige slutudfald. I et rutediagram kan den bagvedliggende funktionalitet beskrives og det kan danne udgangspunkt for selve programmeringen, men det illustrerer til gengæld ikke brugergrænsefladen. Dette skal i se i næste afsnit.

4 Sikkerhed på internettet – Hvad sikrer man ved at gå fra http:// til https:// ?

I dette afsnit skal vi fokusere på, hvorfor der er hjemmesider, der skal tilgås med **https://**, mens andre kan tilgås med **http://**. Og hvorfor er **https://** bedre. De teknikker, som præsenteres i dette kapitel, bruges i https-protokollen til at sikre dine data, men bruges også alle andre steder, hvor vi har brug for sikker og hemmelig kommunikation på nettet, f.eks. i gode beskedapps, men ikke i Messenger og sms'er.

Som eksempel kigger vi på dig, når du er på nettet og handler. Du har lagt nogle varer i din kurv på din foretrukne webshop og har udfyldt dine kort-oplysninger. Du trykker "enter", hvorefter din browser udformer en request-pakke med dine oplysninger/data (adresse, dankort-numre, navn, ...), der skal sendes til din webshops server.



Figur 1: I kommunikationen mellem kunde og webshop kan ondsindede agenter f.eks. stjæle kundes kreditkortoplysninger

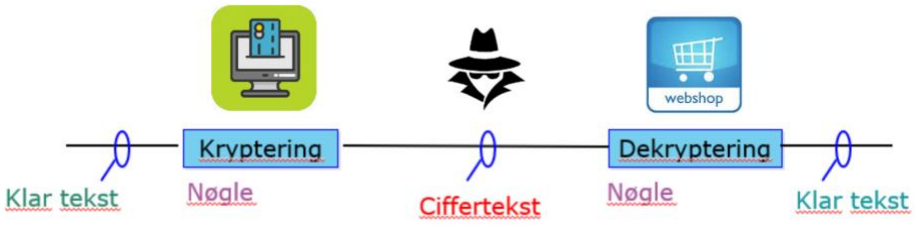
Sikkerheden for dine data i request-pakken kan ødelægges på forskellige måder:

- **Konfidentialitet:** Dine data bliver læst – ubemærket. Dine data når webshoppen som om intet var sket, men andre har læst fx dit kreditkort nummer.
- **Integritet**: Dine data bliver ændret, men fortsætter til webshoppen, uden at du eller webshoppen opdager det. Fx kan modtageradressen for varen være ændret
- **Autenticitet**: Din request-pakke omdirigeres til en anden webserver, der foregiver at være webshoppen – men ikke er det. Dit kreditkort kan blive misbrugt, og du får ikke dine varer.

4.1 Konfidentialitet

At sikre konfidentialitet er at sikre, at data kun kan forstås af afsender og modtager. En tredjepart, der lytter med, eller opsnapper request-pakken på sin vej, må ikke kunne forstå data. Det er det, som kryptering går ud på.

Når dine data **krypteres**, laves dine data, kaldet *klartekst*, om til en *ciffertekst*. Det er cifferteksten, der sendes fra dig og modtages af webshoppen. Webshoppen **dekrypterer** *cifferteksten* og får dermed *klarteksten*, webshoppen dermed kan læse. Der er bare lige det, at de hemmeligheder afsender og modtager skal dele for at kunne kryptere og dekryptere foregår på den samme kommunikationskanal, som den ondsindede også lytter på. Så hvordan undergår man, at den ondsindede også kan dekryptere cifferteksten?

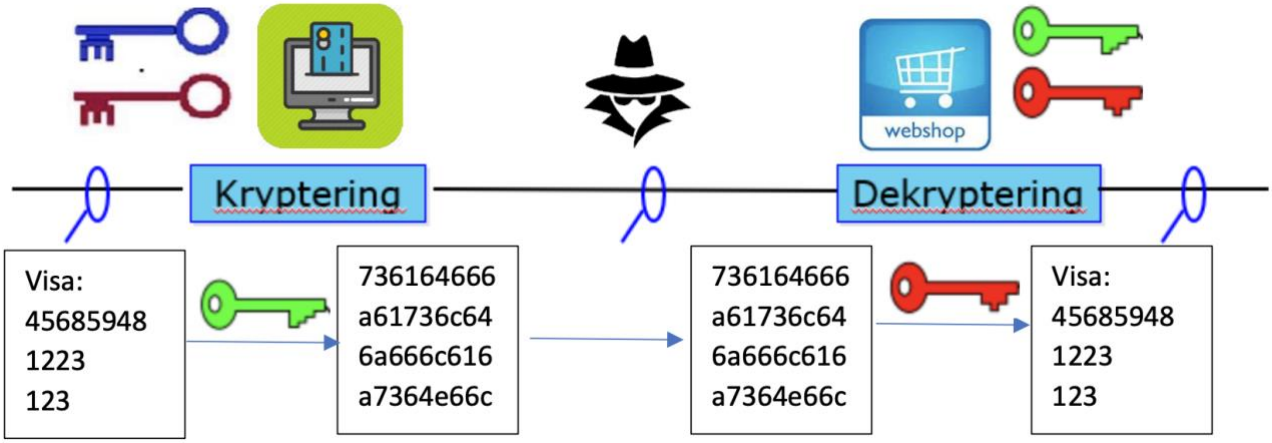


Figur 2: Klarteksten krypteres til en ciffertekst inden den sendes og dekrypteres hos modtageren

4.2 Public-key-kryptosystemer

Den mest udbredte krypteringsmetode i dag er stadig RSA, selvom ECC bliver mere og mere udbredt. De to krypteringsmetoder er forskellige rent matematisk – men de følger samme grundlæggende koncept.

I "publickey-kryptomekanismer" har hver part en **privat (hemmelig)** og **offentlig (offentligt kendt)** nøgle. Kommunikationen indledes med, at begge parter **udveksler** deres **offentlige nøgler** – og de ondsindede, der lytter med på linjen, får også nøglerne – men det gør ingen skade.



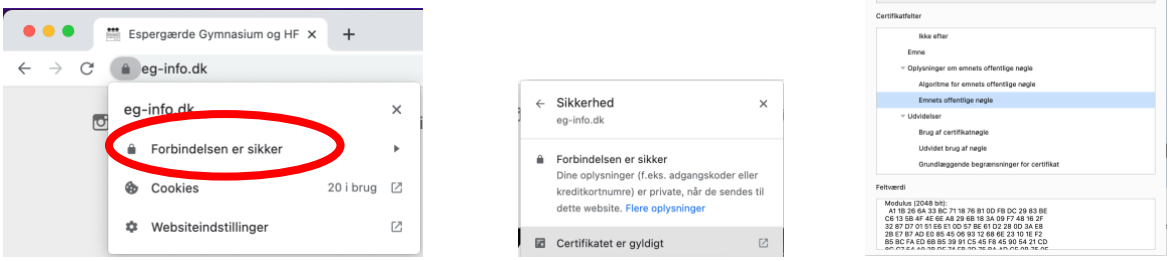
Figur 3: Kryptering med modtagers offentlige nøgle og dekryptering med modtagers private nøgle

Figur 3 viser, at begge parter har hver deres sæt nøgler. Når dine data krypteres, vil din browser bruge webshoppens **offentlige nøgle** til at kryptere med. Webshoppen offentlige nøgle har du (din browser), for kommunikationen startede med at udveksle de offentlige nøgler. Dine krypterede data sendes til webshoppen og webshoppen bruger sin egen private nøgle til at dekryptere med.

Du tænker måske: "Jeg har jo ingen nøgler". Og nej, det har du ikke – men det sørger din browser for at lave, når der er brug for det.

Opgave 1
Figur 3 viser hvordan der krypteres og sendes data fra din computer til webshoppen. Hvilke(n) nøgle(r) bliver anvendt til at kryptere og dekryptere med, når webshoppen skal sende en krypteret besked til dig?

Opgave 2
Webshop, banker, facebook osv... har et fast sæt nøgler (en privat og en offentlig). De laver ikke en ny én for hver gang en ny kommunikation starter. Og deres offentlige nøgle kan du se i din browser. Prøv med http://www.eg-info.dk/. Klik på hængelåsen ved siden af adressen – derefter på "Certifikat".



4.3 Autenticitet

Autenticitet betyder, at man er den, man påstår man er. Fx at websiden, du sender dine data til, faktisk tilhører den webshop, du tror, du kommunikerer med.

4.3.1 Man-In-The-Middle-angreb

At sikre autenticitet går ud på at kunne være sikker på identiteten på den man kommunikerer med. Dvs. at du faktisk kommunikerer med webshoppen og ikke en ondsindet organisation, som stjæler dine kreditkort-data.

Overvej følgende. Du sidder på en café og er på et åbent og offentligt netværk. Din browser viser din foretrukne webshop. Forbindelsen er krypteret, så dem, der lytter med på linjen kan ikke læse kommunikationen mellem dig og webshoppen. Du er sikker - tror du.

I virkeligheden har en anden person på netværket overtaget kommunikationen mellem dig og webshoppen. Altså fysisk - dine request-pakker sendes til den ondsindede Man-In-The-Middle. Du tror, at du kommunikerer med webshoppen, men i virkeligheden kommunikerer du med den ondsindede person, som videresender dine beskeder (request-pakker) til webshoppen og videresender beskeder fra webshoppen til dig.

Så selvom dine beskeder er krypteret, kan den ondsindede person sagtens læse al kommunikation, for du krypterer (uden at vide det) med den ondsindedes offentlige nøgle, som vist nedenfor.

Figur 4: Man-In-The-Middle-angreb. Du krypterer med den ondsindedes offentlige nøgle i stedet for webshoppens nøgle, og den ondsindede kan læse alt, hvad du sender til webshoppen.

4.3.2 CA (Certificate authorities)

For at undgå Man-In-The-Middle-angreb har man brug for en måde at kunne identificere den, man kommunikerer med. Problemet løses ved at binde en offentlig nøgle med et domænenavn, en organisation eller en persons identitet, således, at du (din browser) kan sikre dig, at den offentlige nøgle, som du (din browser) bruger til at kryptere med rent faktisk tilhører det domænenavn, som din webshop har. Groft sagt, er der en myndighed, der garanterer, at modtagerens offentlige nøgle tilhører modtageren. Sådan en myndighed kaldes for en CA (Certificate authority).

Opgave 3
Følg fremgangsmåden i opgave 2 og undersøg hvem, der er CA for din bank, Facebook og skolen.

4.3.3 Hvem garanterer for CA'ens identitet?

Når en CA godkender identiteten af en offentlig nøgle, så underskrives den offentlige nøgle med CA'ens private nøgle. Det er der ingen andre en CA, der kan.

Men hvorfor kan vi overhovedet stole på disse CA'ere? Vores browsere kommer med en række godkendte CA'ere – og er et certifikat underskrevet af en af disse godkendte CA'ere, vil vores browser acceptere dem som sikre. Der er ofte en kæde af CA'ere, og den øverste skal være en af browserens godkendte CA'ere – disse kaldes også Rodcertifikatmyndighed.

Opgave 4
Hvem er den øverste CA (rod-CA) for din bank? For Facebook? For skolen?

Opgave 5
Åbn Chrome -> Indstillinger -> Sikkerhed og privatliv -> Administrér certifikater

På windows: vælg fanebladet "Rodnøglecentre der er tillid til"

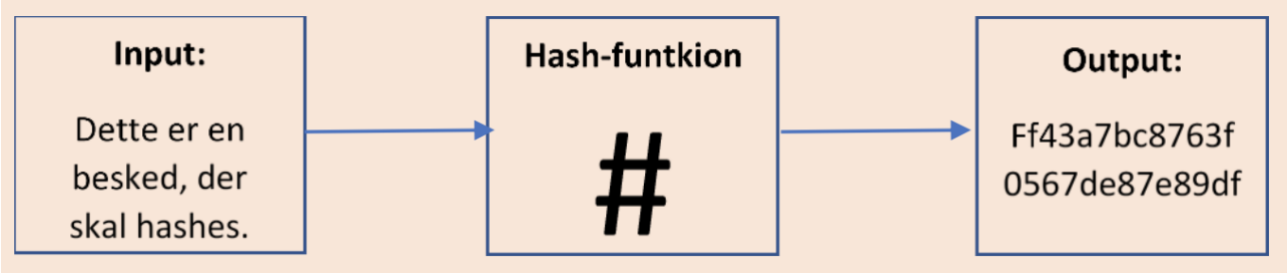
På Mac: vælg "System Roots"

Du få nu en liste over alle de (rod) CA'er, som Chrome stoler på.

Opgave 6
Når din browser støder på en hjemmeside, der bruger et certifikat, som er underskrevet af en myndighed, som din browser ikke stoler på, får du det at vide. Hvordan? Prøv selv!

4.4 Hashing

Hashing er ikke det samme som kryptering, men minder lidt. Det er en metode, hvor man kun kan "kryptere", men ikke kan dekryptere. Det er ofte nyttigt, når man skal sikre, integritet, som beskrives i afsnit 4.5.



Figur 5: Eksempel på hash-værdien (output), når beskeden (input) behandles med en hash-funktion

Hashing går ud på, at hash-funktion ud fra noget data kan udregne en værdi, kaldet en digest (eller hash-værdi). Data kunne være en tekst, en kode eller en sammensætning af forskellige data. Hash-funktioner er specielle på flere måder:

- Uanset hvor stor inputet er, vil hash-værdien være af fast længde (antal tegn)
- Man kan ikke komme tilbage til inputet ud fra en hash-værdi
- To forskellige tekster kan ikke producere den samme hash-værdi

Du kan se mere om hashing i videoen på dette link.

Opgave 7
På denne hjemmeside kan man få hash-værdien for noget input, man selv bestemmer.

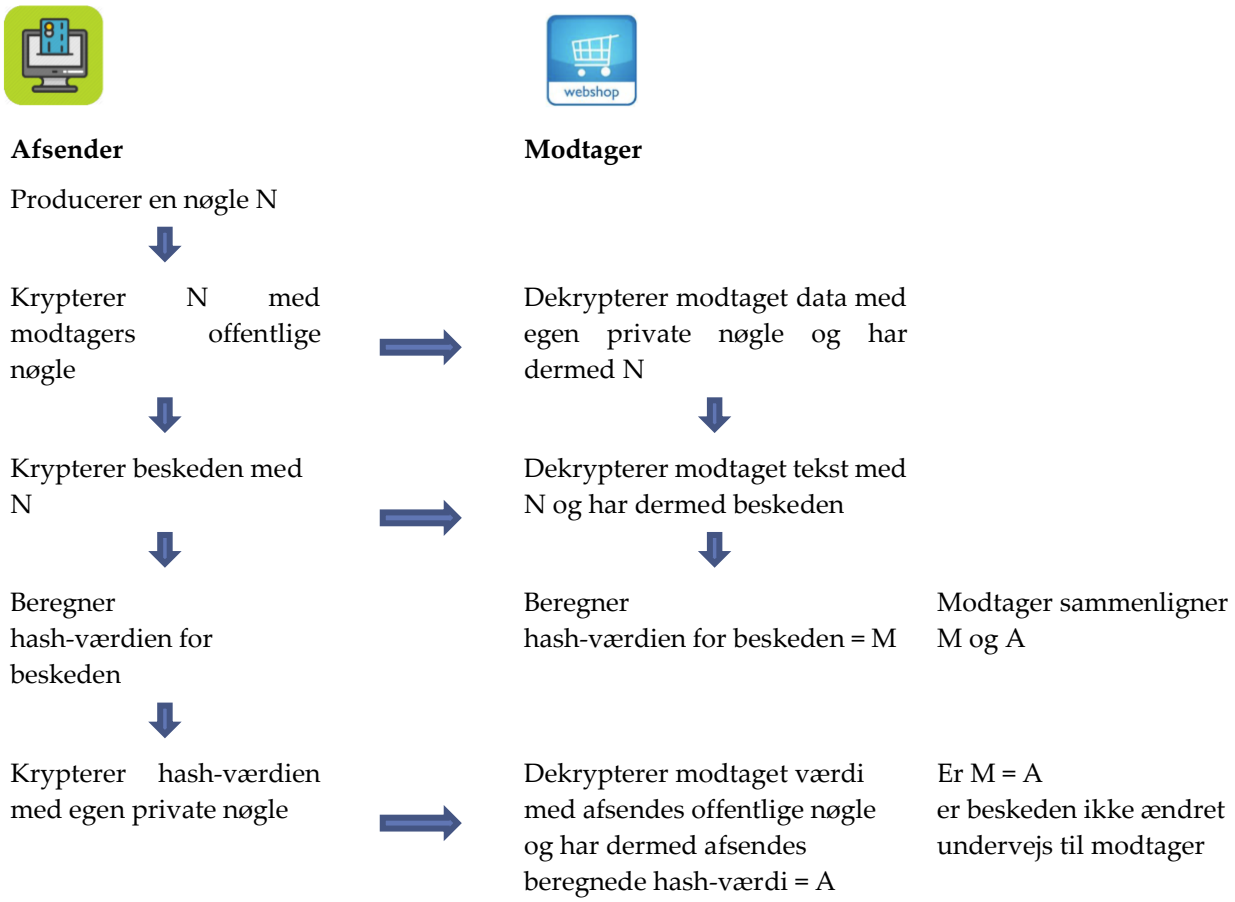
a) Skriv "Jeg er ... år gammel.", hvor du erstatter "..." med din alder. Vælge "MD5" som algoritme (det er en af de mest udbredte og sammenlign din hash-værdi med naboens. Er de meget forskellige?

b) Prøv at fjerne "." i slutningen af din inputtekst og beregn hash-værdien på ny. Hvor stor en forskel er der?

4.5 Integritet

At sikre integritet går ud på at sikre, at beskeden ikke bliver ændret undervejs fx ved et Man-In-The-Middle-angreb.

En beskeds integritet kan sikres ved hjælp af nedenstående metode, hvor det afgørende punkt er, at afsenderen beregner hash-værdien for beskeden og krypterer hash-værdien med si egen private nøgle. Både beskeden og hash-værdien sendes krypteret til modtageren. Modtageren dekrypterer både beskeden og hash-værdien, og selv beregner beskedens hash-værdi. Nu kan de to hash-værdier beregnes. Er de ens, er beskeden ikke ændret undervejs.



Figur 6: Metode til verifikation af beskedens integritet

5 It-systemers arkitektur

I dette kapitel lærer du, hvordan computere kan arbejde sammen, så man kan dele data og udveksle information mellem mange computere, dvs. du lærer om det man kalder it-systemets **arkitektur**. Vi vil se på 3 typer arkitektur:

- en simpel arkitektur, hvor computeren arbejder alene. Det kunne være et spil, som spilles på en mobiltelefon uden udveksling af data med andre spillere
- en 2-lags-arkitektur kaldet **klient/server-arkitektur**, hvor mange **klienter** henter data fra og sender data til en **server**. Det kunne være et spil, som spilles på mange mobiltelefoner (klienterne) og den fælles highscore gemmes på en server.
- en **3-lags-arkitektur**, hvor der skal gemmes meget forskelligt data og serveren derfor kobles til en **databaseserver**, der egner sig godt til at strukturere store mængder data, som oplysninger om hvem som spiller mod hvem i store online spil.

5.1 Simpel arkitektur

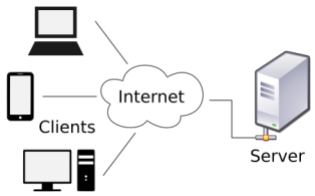
Når du har lavet et spil f.eks. i App Lab, kan man hente spillet ned på mange mobiltelefoner. Derefter kan man spille spillet uafhængigt af alle andre på hver sin mobiltelefon. Figur 7 viser tre udgaver af samme spil, som spillet på tre forskellige mobiltelefoner uafhængigt af hinanden.

Figur 7: Tre personer spiller det samme spil, og har hver deres point og highscore, men kan ikke se, hvor mange point andre spillere har scoret.

I en simpel arkitektur er der den begrænsning, at man ikke kan sammenligne point og beregne, hvem der er den bedste highscore. Der er ingen kommunikation mellem de enkelte mobiltelefoner. Man er derfor nødt til at spørge alle de andre spillere, hvad deres highscore er. Det bliver besværligt, hvis der er mange spillere eller man befinder sig langt fra hinanden. Det problem løses med en klient/server-arkitektur.

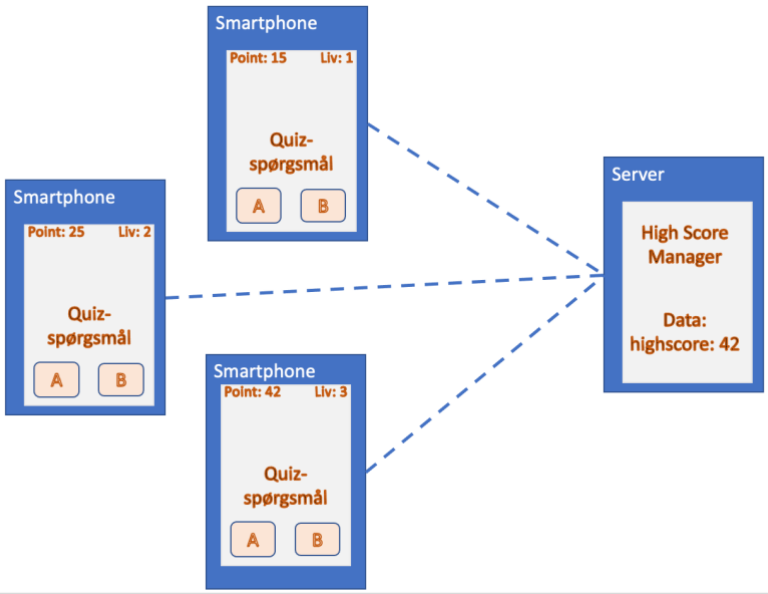
5.2 Klient/server-arkitektur

En klient er et program eller en computer, som benytter tjenester på **en server**. En server er en computer, som sender data eller filer til klienter.



Figur 8: Forskellige typer klienter er kommunikerer med en server over internettet. Klienterne henter data fra serveren og sender data til serveren.

Der findes mange forskellige typer servere. En web-server gemmer websider, som sendes til din computers browser (klienten), når du beder om en bestemt webside. En mail-server bruges i e-mail-systemer, hvor klienterne henter deres mail fra mail-serveren og bruger serveren, når mails afsendes fra klienten. I et spil har vi brug for en server der kan holde styr på highscore, vi kan kalde serveren en "High Score Manager"-server, som vist på nedenfor.



Figur 9: 2-lags-arkitektur med tre klienter (her spillere) og en server, som gemmer highscore for alle spillerne og kommunikerer med klienterne, når klienterne sender en ny score for at få den sammenlignet med den aktuelle highscore.

Det er klinten som begynder kommunikationen med serveren, fx. når klienten har opnået en ny score og skal have den sammenlignet med den aktuelle highscore. Klienten sender en forespørgsel til serveren, fx. med sin nye score. Klienten er den **aktive**, og kommunikationen starter typisk fordi klient-computerens bruger ønsker en service, her at få sin nye score sammenlignet med den highscore serveren har registreret.

Kommunikationen med serveren begynder med, at klienten udformer en besked til serveren (**request**). Beskeden skal følge bestemte regler, så den ikke kan misforstås. Reglerne kaldes

en **protokol** (en måde at kommunikere på). Hver servertype bruger en bestemt protokol. Når serveren modtager en korrekt request, vil den fortolke den og derefter sende sit svar (**response**) tilbage til klienten.

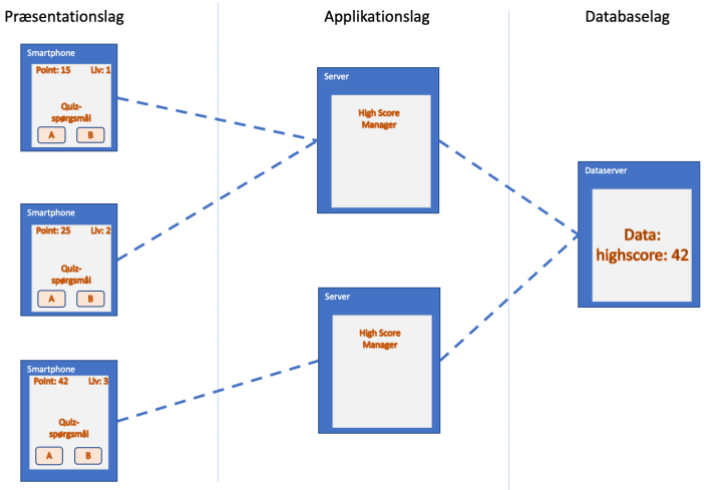
En server er en computer (typisk en kraftig computer som altid er tændt), der servicerer klienter. Den venter på (lytter efter) indkommende forespørgsler (request) og holder sig for det meste passivt. Først når serveren modtager en request, læser serveren beskeden ift. den protokol, der benyttes, og sender sit response tilbage til klienten.

Figur 9 viser tre klienter og en server. Klienterne er de aktive, som starter en kommunikation med et request, når en et spil er færdigt og scoren skal sammenlignes med den highscore, som ligger på serveren. Når serveren modtager en request, sammenlignes med den aktuelle high score, som opdateres, hvis den nye score er højere. Derefter sender serveren et respons til klienten, og klienten modtager information om den nye highscore. De andre klienter modtager først information om en ny highscore, når de selv sender et request til serveren.

5.3 Tre-lags-arkitektur

Tre-lags arkitekturen er en udvidelse af klient-server arkitekturen. Den bruges ofte, når meget data skal håndteres.

- **Klienten** også kaldet **præsentationslaget** er det, som brugeren ser.
- **Applikationslaget,** som er den programmering, der forbinder datalaget med klienten – ligger på en **server**
- **Databaselaget**, som indeholder de data (informationer), som klienten skal bruge. Det kan være nogle loginoplysninger, high-scores, oplysninger om grupper af spillere i online spil, etc.



Figur 10: Tre-lags-arkitektur med præsentationslaget (klienter), applikationslaget med servere, som modtager forespørgsler fra klienter og kommunikere med databasen. I databaselaget er databasen, som gemmer data tilsendt fra applikationslaget og sender data til applikationslaget, når der kommer en forespørgsel fra dette lag.

Hvis vi ønsker at udvide vores spil, så spillere kan udfordre hinanden og man derved kan invitere vennerne til at konkurrere mod hinanden, får man brug for mere end en simpel server. Man splitter serveren op i et applikationslag, som modtager request fra klienterne, og et databaselag, hvor en database lagre oplysninger om brugerne og om hvem som spiller mod hvem, samt highscore i de enkelte grupper af spillere. Serveren i applikationslaget fortolker request fra brugeren fx login, oprettelse af en ny gruppe eller forespørgsel om en highscore. Serveren i applikationslaget kontakter databaselaget, når der er behov for data og behandler data, så det kan sendes til klienterne i den rigtige form.

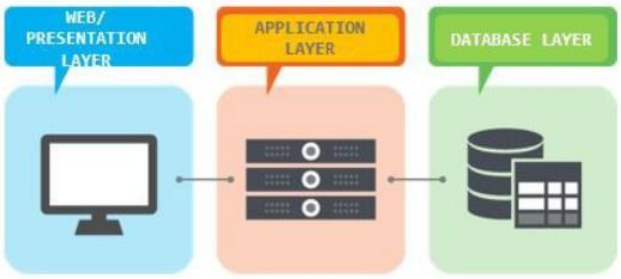
3-lags-arkitekturen er fleksibel og kan nemt udvides til at håndtere mange brugere. Hvis et spil bliver en stor succes, og derfor får mange spillere, kan man dele applikationslaget op på flere servere, der modtager request fra hver sin gruppe af klienter. Disse serveren kan derefter hente data fra den samme database, se Figur 10.

Eksempel 1

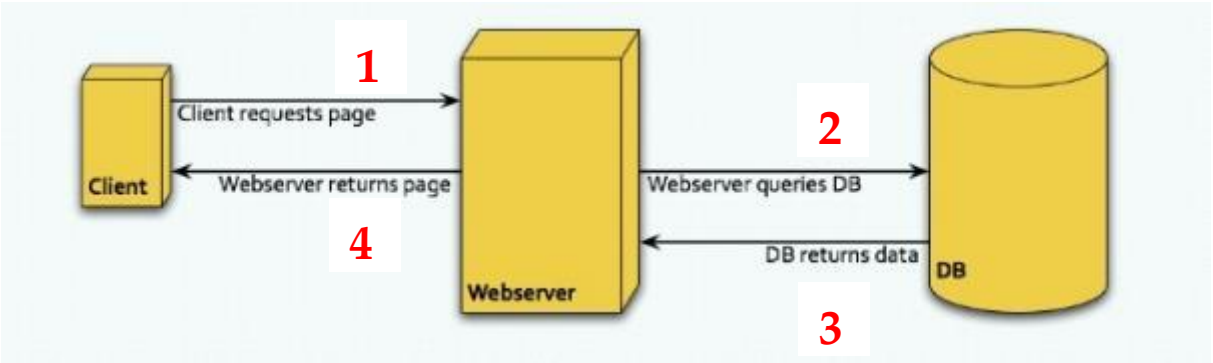
Facebook holder styr på dine login-oplysninger, navn, tlf, hvem du er venner med, hvad du har liket, hvilke begivenheder du er "interesseret i" eller "deltager i". Disse informationer ligger i **datalaget** og er lageret i en database-server. Når du logger ind på facebook i din browser (klient), tilgår browseren webserveren og det er denne webserver, der sørger for at indhente de nødvendige oplysninger om dig fra databasen og før webserveren sender data videre til browseren.

A**pplikationslaget** er den kode der forbinder jer og database-serveren.

Præsentationslaget er så jeres personaliserede facebookside, som I ser, når I logger på.



Dette kommunikationsflow kan skematiseres på følgende vis:



6 Databaser

6.1 Eksempel: online-spil

Har man et online-spil, skal spiludbyderen holde styr på hvilke spillere, der har et login til spillet og f.eks. hvor mange point hver spiller har. Den simpleste måde at gemme disse data kunne være i et regneark, hvor man gemmer oplysninger om den enkelte spiller.

BrugerID	BrugerNavn	Email	HighScore	Adresse	Etage	By	Land
101	Mikkel	Mikkel123@mails.dk	1246	Storegade 13	1. th	Lilleby	Danmark
102	Daniel	Daniel345@mails.dk	1490	Fjeldtoppen 3		Store Fjell	Norge
103	Frederik	Frederik153@mails.dk	2516	Rådshuspladsen 2	3. tv	København	Danmark
104	Julius	Julius123@mails.dk	68	Storetorget 4	2. th	Stokholm	Sverige
105	Camilla	Camilla245@mails.dk	1276	Carl Johan Gata 35		Oslo	Norge
106	Gustav	Gustav246@mails.dk	1719	Kiddesvej 12		Vejle	Danmark
107	Marius	Marius135@mails.dk	1971	Gymnasievej 2		Espergærde	Danmark

Figur 11: Når man har flere oplysninger om en bruger er det en fordel at strukturere oplysningerne i en tabel – lidt ligesom i et regneark (Excel)

Hvis man driver et større online-spil, vil man hurtigt opdage, at man har brug for at gemme flere oplysninger (data), f.eks. hvem spiller mod i hinanden. Da den enkelte spiller kan deltage i flere forskellige grupper af spillere samtidigt, bliver det svært at organisere data i ét regneark, og man får brug for en database.

En **database** er groft sagt en samling af tabeller, hvor forskellige typer data samles i forskellige tabeller. Hver tabel er organiseret ud fra relation mellem oplysningerne i tabellen. Det kan være en tabel med spillere og deres adresse, samt en tabel med navne på grupper af spillere, og den aktuelle highscore i gruppen.

For at hente data, ændre data eller tilføje data til databasen skal man bruge et databasesprog, ofte er det SQL. Dette sprog kan også bruges til at oprette nye tabeller, når man laver en ny database. Når man først har lavet databasen, skal man helst ikke ændre databasen. Derfor skal man tænke sig godt om, så man ved hvilke data databasen skal indeholde, dvs. hvilke tabeller skal databasen indeholde, og hvilken relation er der mellem de forskellige typer data.

6.2 Nogle fagudtryk

Når man organiserer data i en tabel, har vi en række i tabellen for hvert sæt data og en kolonne for hver type oplysning, se Figur 11. Der er forskellige ting at sige om denne tabel:

- Hver række indeholder oplysninger som hører sammen, f.eks. om data om den enkelte spiller. Hver række kaldes i databasesprog for en **tuppel**.
- Hver kolonne har en generel overskrift, der beskriver indholdet af cellerne i kolonnen. Hver type data kaldes en **attribut**. Og fordi strukturen af tabellen skal være så bred som muligt, er der celler, der ikke nødvendigvis skal bruges for alle spillere. **Attributten** "Etage" i Figur 11 skal eksempelvis kun bruges til spillere, som bor i en etageejendom, så

værdien af **attributten** "Etage" for alle andre spillere er ikke specificeret - eller rettere sagt - specificeret til ingenting (NULL).

- Den allerførste **attribut** i Figur 11 hedder "*BrugerID*". Vi vil gerne have data, der kan defineres entydigt, og skal derfor tage højde for, at man kan have flere spillere med samme navn. Der skal også være mulighed for at man skifter sin email-adresse, og derfor opretter man et burgerID, man ved er **entydigt i eget system**. Det kalder man for en **primær nøgle**. Ingen andre i en tabel må have samme primær nøgle. Det er lidt det samme som CPR-numre – ingen andre i Danmark har samme CPR-nummer som dig.
- Endeligt, så har denne tabel også et navn: "*Brugere*".

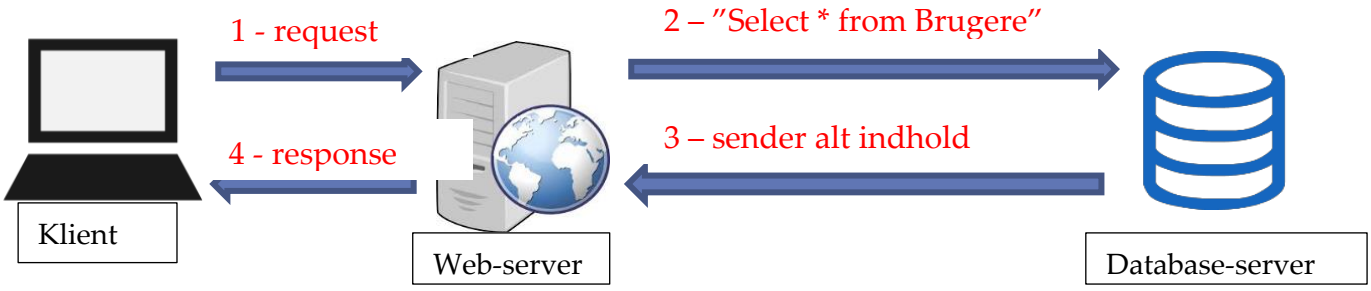
Opgave 8
Brug tabellen "Brugere" i Figur 11 og besvar følgende spørgsmål:

a) Hvilken primære nøgle har spilleren "Marius" og hvilken by bor spilleren i?
b) Hvilken spiller har primær nøgle 104 og hvad er denne spillers highscore?
c) Hvor mange spillere har adresse i Danmark og hvad er værdien af attributten "By" for hver af disse?

6.3 Grundlæggende forespørgsler – hent data i databasen

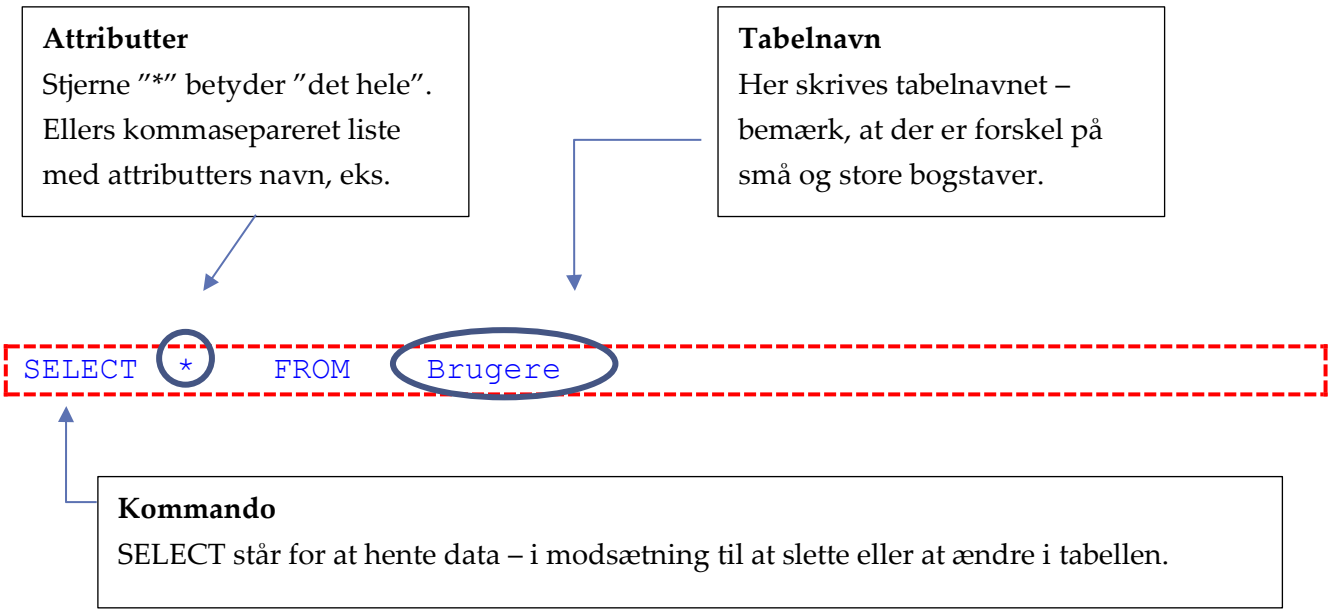
Den forrige opgave har I løst ved selv at slå op i tabellen. Det er lige præcis det, vi gerne vil undgå – vi vil gerne automatisere systemet, så man ikke behøver selv at slå op. Men for at gøre det har vi brug for at vide, hvordan man kommunikerer med databasen (som foreløbigt kun indeholder én tabel).

Men inden vi kaster os ud i det, skal vi lige se på kommunikationsflowet. Det er webserverens opgave at sørge for at hente de data hos databasen, som klienten har brug for i responsen.



Figur 12: Kommunikation mellem klient, web-server og database i en tre-lags-arkitektur

Web-serveren bruger et specielt database-sprog for at fortælle database-serveren, hvilke data, den gerne vil have – det sprog hedder SQL (**S**tructured **Q**uery **L**anguage). Hvis det er hele tabellen man gerne vil se, så ser forespørgsel sådan ud:



Figur 13: database forespørgsel, som henter hele indholdet i tabellen "Brugere"

Eksempel 2

Forespørgslen

```
SELECT HighScore FROM Brugere
```

Vil give en lang søjle bestående af highscore for alle spillere.

Forespørgslen

```
SELECT brugerNavn, Adresse, By, Land FROM Brugere
```

Vil give fire kolonner med de efterspurgte attributter for alle spillere.

Vi kan nu også være mere specifikke, hvis vi tilføjer et krav med "WHERE".

```
SELECT * FROM Brugere WHERE By="Espergærde"
```

Ovenstående forespørgsel giver os alle oplysninger på spillere, som bor i Espergærde. Dette giver også rækken, hvor vi kan aflæse svaret på a) i Opgave 8.

Opgave 9
På adressen kan du selv indtaste en SQL-forespørgsel og se, hvad svaret er fra serveren. Indtast følgende forespørgsler. Får du svaret, som du ønskede at få?

a) SELECT contactLastName, contactFirstName, country FROM customers
b) SELECT * FROM customers WHERE customerName="Dragon souveniers, Ltd."
c) SELECT customerName, contactLastName, contactFirstName FROM customers WHERE customerNumber="348"
d) SELECT phone FROM customers WHERE country="Norway"

Opgave 10
På den samme adresse skal du nu prøve at skrive dine egne forespørgsler og afprøve dem.

a) Du skal hente alle oplysninger for alle kunder i Tyskland (Germany).
b) Du skal hente alle oplysninger for alle kunder med postnummer 44000.
c) Du skal hente kundenummer på kunden " Enaco Distributors".

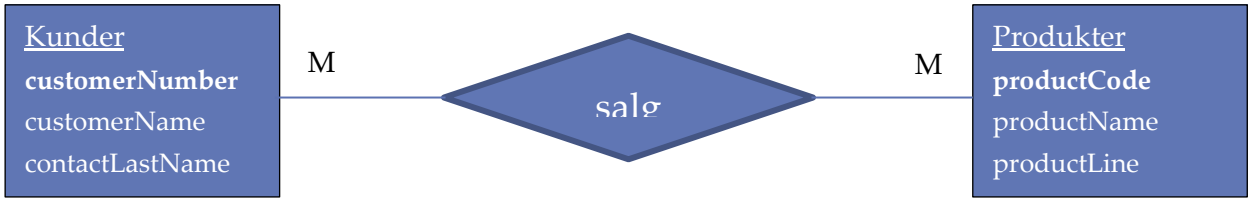
Opgave 11
En anden tabel har navnet "products", som indeholder alle produkter i webshoppen.

a) Hvilken SQL-forespørgsel skal du skrive for at få alle attributter frem for alle produkter? Afprøv det på denne adresse.
b) Hvilke attributter har tabellen?
c) Hvilke værdier kan attributten "productLine" tage?

6.4 Entiteter, attributter og relationer

Foreløbigt har vi to tabeller i vores database – de kaldes også for "**entiteter**". Entiteter er typisk "fysiske" ting. Entiteterne har **attributter**. En attribut kan forstås som en egenskab. For et produkt kan egenskaben være en farve, f.eks. hvis man har både grønne og røde udgaver af det samme produkt.

I en webshop vil vi gerne bruge databasen til at holde styr på, hvad vi sælger, dvs. vi vil registrere hver gang, der for sker et salg af et produkt. Salget er den **relation,** der findes mellem vores to entiteter og lad os kalde relationen for "salg". Entiteter, attributter og relationer repræsenteres i et såkaldt **E/R diagram** (Entity/Relation diagram), hvor entiteter står i firkantede bokse og relationer står i parallelogram-bokse. Attributter skrives typisk inde i boksen. Den attribut, som bruges som **primære nøgle**, markeres med fed



Figur 14: E/R-diagrammet for relationen salg

E/R diagrammer vil indgå i designet af databasen allerede inden, at vi har besluttet hvilke tabeller vi vil lave, i eksemplet har vi tabellerne *customers* (kunder) *og products* (produkter). Diagrammet skal forstås således, at "kunder" og "produkter" forbindes med relationen "salg". Forbindelsen går begge veje: ved et salg skabes 1) en relation fra en kunde til et produkt og 2) en relation fra et produkt til en kunde.

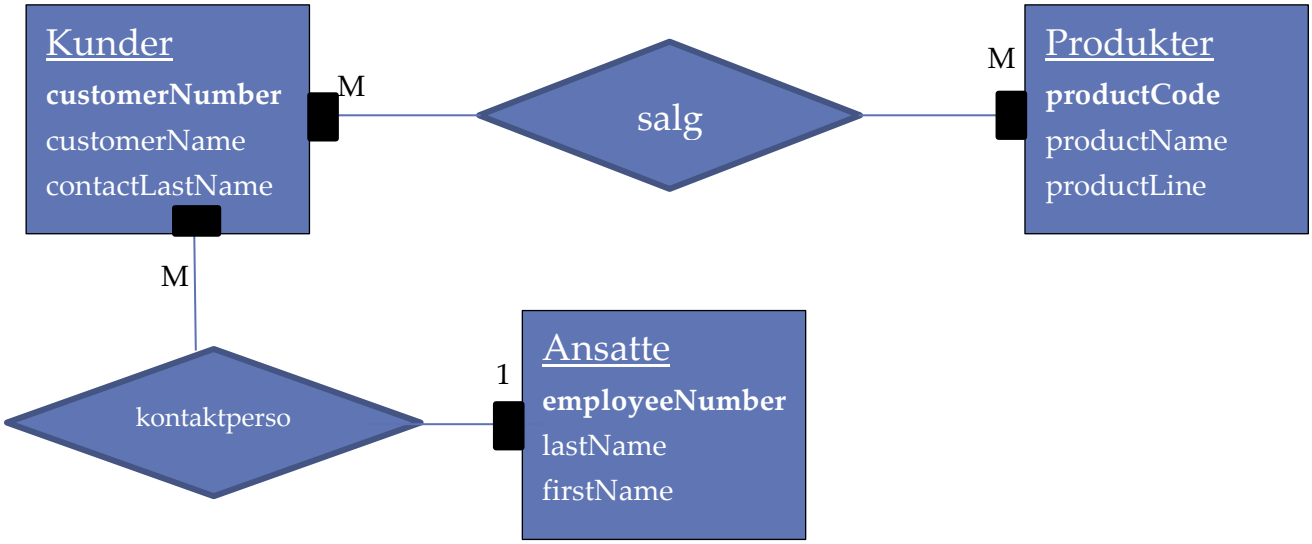
M'erne i diagrammet angiver **relationsgraden** mellem entiteterne. En relationsgrad kan enten være 1 (én) eller M (mange). For at finde ud af om det er det ene eller det andet, skal man undersøg hvilke af nedenstående sætninger, der er sande:

~~En kunde kan kun købe ét produkt, dvs. kun indgå i én salgsrelation~~
En kunde kan købe **MANGE** produkter, dvs. indgå i MANGE salgsrelationer

~~Et produkt kan kun købes af én kunde, dvs. kun indgå i én salgsrelation.~~
Et produkt kan købes af **MANGE** kunder, dvs. indgå i MANGE salgsrelationer

Der skal stå "M" i begge ender af diagrammet, da relationen er Mange til Mange med mange i begge ender af relationen.

Lad os antage, at vi har en tabel "employees" (Ansatte) og attributten "salesRepEmployeeNumber" i tabellen "customers" peger på en ansat i tabellen. Relationen mellem tabellerne "ansatte" og "kunder" kunne være "kontaktperson". Der er kun en person, der er kontaktperson for en kunde. Men en ansat vil godt kunne have flere kunder. Så her har vi en én-til-mange relation og vil udtrykke sig som følgende i et E/R-diagram.



Figur 15: E/R-diagram med 3 entiteter (Kunder, Produkter og Ansatte) og 2 relationer (salg og kontaktperson)

Vi skal nu sætte "**medlemstypen**", som bestemmer en "kan eller skal" forhold i en relation.

Skal eller kan en kunde have (mulighed for) kontakt til en ansat? **Skal**.
Skal eller kan en ansat have (mulighed for) kontakt til en kunde? **Kan**. (for det bestemmer vi ikke, kunne dog også være "skal", men for eksemplets skyld, lad det være "kan").
Skal eller kan en kunde købe et produkt? **Kan**.

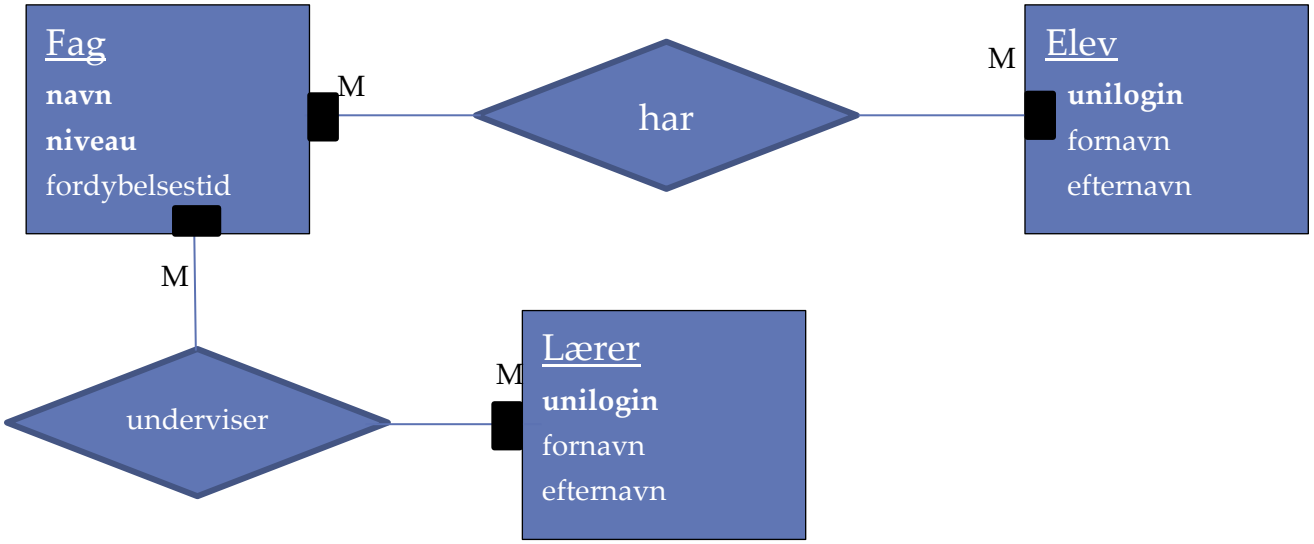
Medlemstypen repræsenteres som "sorte bokse" ved udgangen en entitet. Er boksen indenfor, forstås det som "skal". Er boksen udenfor entiteten forstås boksen som "kan". Kun én sort boks indgår i beskrivelsen af relationen og man udtaler den sorte boks, der kommer først.

Opgave 12
Lad os kigge lidt på facebook i en forsimplet version. Vi betragter to entiteter: user og post. Entiteten user indeholder en bruger

a) Hvilke attributter kan en user have?
b) Hvilke attributter kan en post have?
c) Tegn et E/R diagram for disse to entiter med relation, relationsgraden og medlemstypen. Du kan bruge erdplus.com til at tegne E/R-diagrammet.

Opgave 13
E/R diagrammet nedenfor viser dele af en skoles database. Bemærk, at i boksen "fag" er der to attributter, der er markeret med fed. En primær nøgle behøver ikke være blot en attribut. Et fag med navn og et bestemt niveau – eks. **(informatik, C)** definerer entydig faget.

Aflæs relationer, relationsgraden og medlemstype og skriv sætningerne ned.



Figur 16: E/R-diagram for dele af en skoles database

7 Mere om udvikling af spil[3]

Når der er lavet mockups af interaktionsdesignet, samt evt. storyboards og rutediagrammer over gameplay, så kan selve produktionen af spillet begynde.

7.1 Valg af udviklingsmiljø

Når man laver spil, udvikles de typisk i et programmeringsmiljø, som understøtter udvikling af spil. Simple systemer er f.eks. Scratch, mens Unity er et mere avanceret udviklingsmiljø, der f.eks. understøtter spil i 3 dimensioner.

Den ønskede spiltype, samt den økonomiske og tidsmæssige ramme, er afgørende for valget af udviklingsmiljø. Vigtige faktorer for valg af

- Omfattende og/eller velproducerede spil kræver økonomi til et dyrt udviklingsmiljø og proces og tid til at gennemføre den.
- Sociale spil kræver at, det er enkelt at håndtere og dele data mellem brugere.
- Spil til mobiler medfører et valg mellem mobiltyper, man vil udvikle til.
- Spil til organisationer med få midler medfører, at der kun kan benyttes gratis eller billige udviklingsmiljøer

7.2 Udviklingsprocessen

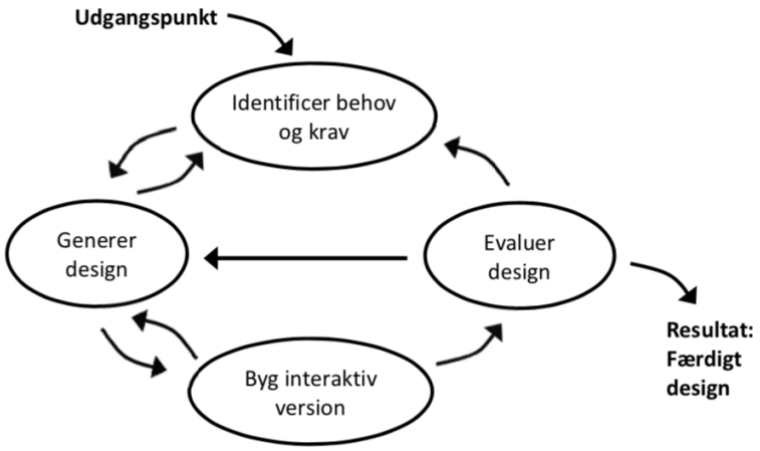
De mockups, diagrammer m.m. der er udviklet under konceptudviklingen kan også bruges til at opdele og styre udviklingsprocessen. Her har man et overblik over ønskede elementer og muligheder, som hver udgør enkelte dele, der kan udvikles og testes, hver for sig for til sidst at sammensættes til en helhed.

Processen med at udvikle it-produkter trin for trin beskrives bl.a. i udviklingsmodellen *'trinvis forbedring'*. Her opdeler man opgaven i delmål og i hvert af disse:

- **konkretiserer** et udkast i form af kravspecifikationer, mockups, datamodeller, rutediagrammer
- **implementerer** udkastet ved at lave den mest basale del og derefter lave udvidelser trin for trin.
- **omstrukturerer** og revider udkastet, hvis det viser sig mere hensigtsmæssigt at følge en ny strategi

Undervejs i processen kan det være nyttigt at teste spillet ved at lade potentielle brugere prøve spillet eller evaluere prototyper af spillet. Udviklingsprocessen er iterativ, dvs. man kan gå igennem de samme faser i udviklingsprocessen flere eller mange gange. Dette illustreres af pilene i figuren:

[3] (http://iftek.dk)



Figur 17: Aktiviteter i interaktionsdesign[4]. Pilene viser hvordan man bevæger sig fra en aktivitet til den næste. I princippet kan man blive ved og tage turen rundt mange gange.

På figuren starter man processen med "identificere behov og krav". Nogle gange skal man lave en ny udgave af et tidligere spil eller andet it-system. Så begynder man ofte med aktiviteten "Evaluer design" og evaluerer det gamle interaktionsdesign, før man går til "identificer behov og krav" til det nye.

7.2.1 Persona og scenarier[5]

Kravene til systemet kan beskrives på mange måder. To typiske teknikker er personas og scenarier. En **persona** er en beskrivelse af en konkret person på en form, så man kan forestille sig, at det var en virkelig person. Ideen med at bruge personas er at tvinge designere af it-systemer til at tænke på rigtige brugere, når de beskriver kravene til systemet, så de ikke kun fokuserer på de tekniske krav.

Når man beskriver en persona, finder man alle de karakteristika man finder relevante. Det kan være alder, beskrivelse af hvor og hvordan personaen typisk bruger it-systemer, personaens behov for et it-system af denne type, og hvilke krav og forventninger personaen kan have til systemet.

Et scenarium er en beskrivelse af et menneskes aktivitet. Den beskriver overordnet men trin for trin, hvad en bruger gør med et it-system. Det beskriver også den sammenhæng, brugeren er i, når systemet anvendes.

7.2.2 Prototyper

Storyboards og rutediagrammer kan være en del af aktiviteten "Generer design" i Figur 17. Når idéen er udviklet, skal den beskrives mere detaljeret i aktiviteten "Byg interaktiv version". Den interaktive version kan bygges på mange måder. Det mest almindelige er at lave en form for prototype, som har nogle af det færdige systems egenskaber. Der kan skelnes mellem tre typer:

[4] fra Rogers et al. 2011 og iftek.dk
[5] Iftek.dk - Interaktionsdesign

- Papirprototype - typisk lavet på papir. Den illustrerer de enkelte skærmbilleder i systemet og måske også detaljer i interaktionen, f.eks. hvad sker der, når spilleren trykker på en knap.
- Fysisk prototype – er en ting som minder om it-systemet. Det er sjældent vi har behov for en fysisk prototype af spil. Ved udviklingen af it-systemer som f.eks. Rejsekortet, har man brug for at lave fysiske modeller af checkin-stander andet udstyr brugeren skal anvende
- Kørende prototype - En kørende prototype fungerer mere eller mindre som det færdige system.

Da vi ikke har meget tid i informatik, vil det endelig produkt i nogle af jeres projekter være en kørende prototype, hvor I kun har lavet dele af et spil eller andet it-system.