S2 (måske) vigtige noter

Ting jeg sikkert har glemt onsdag, som er meget godt at huske 😎

# Inception

### Nummer 1.

Formålet med inception er at give input til ellaboration.

I inception læser man systemsbeskrivelsen og finder systemkrav ud fra beskrivelsen.

Vi har i fællesskab med feedback fra lærer fundet gode krav.

Et godt krav er f.eks.:

* Noget testbart
* Utvetydigt
* Testbart
* Målbart

# Ellaboration

### Nummer 2.

I ellaboration begynder vi at kigge nærmere på noget som skal bruges i selve koden.

I ellaboration begynder man at finde entiteter ud fra systemsbeskrivelsen. Disse entiteter kan skabe klasser i vores entitets library i .NET web api’et eller tabeller og kolonner i vores database.

Ellaboration kigger vi også på at analysere og skabe diagrammer over vores kode (**OOA** Object Oriented Analysis), for at bevare overblikket og finde ud af om vi har glemt noget i Inception fasen.

# Construction

### Nummer 4.

Første gang i construction skaber vi infrastrukturen af vores system, så det er skalerbart og nemt at tilføje nye funktioner og krav. (agilt software 🎊🎊)

I construction begynder vi at kigge på at kode vores .NET web api og klienten.

Construction omfatter kodning af systemet

# API

### In depth knowledge

API består af en IIS og Controllers. En controller opretter et endpoint, som har et ansvar i vores system.

IIS’ens ansvar i API’et er at føre kald til vores API til den rigtige controller.

# Fullstack system

### In depth knowledge

Et fullstack system står umiddelbart ud fra 3 programmer. Klienten (Web, WPF, Blazor, Razor), Server (.NET, ExpressJS, Nest.JS) og databasen (MSSQL, PostgresSQL, MySQL, MongoDB)

Klientens ansvar er at skabe et interface til vores bruger, når brugeren foretager en handling bliver vores server så kaldt og udretter noget. Typisk er det at oprette ny data, hente data osv. I vores database.

# Database & SQL

### In depth knowledge

En database er typisk oprettet i SQL, et databaseprogrammeringssprog.

En database består af tabeller som består af kolonner og rækker. En kolonne specificere hvilken type og hvad data der kan være i den celle.

Udover simple kolonner og rækker kan der også være primære nøgler, fremmed nøgler, stored procedures, views osv.

Primær nøgle: En kolonne som typisk består af et tal id. Dette sikre at vi ikke har samme data 2 gange i vores tabel.

Fremmed nøgle: Hvordan vi skaber en reference til en række i en anden tabel. Dette skaber også referentiel integritet.

Stored Procedure: Et stykke gemt SQL kode i vores tabel, som vi hurtigt kan eksekvere og modtage noget data.

# Separation of concerns

### Good to remember

Separation of concerns ligger lidt I ordet. Vi ønsker at separere vores problemer ud i flere forskellige klasser af flere forskellige grundet, b.la.:

* Overblik
* Fejl søgning
* Vedligeholdelse af kode

# Projektstyring

### Good to remember

Projektstyring er super vigtigt for større, komplekse systemer. Der er mange ting at holde styr på, og for at sikre at vi ikke glemmer hvad kunden har bedt om opretter vi det som punkter i en liste.

Vi bruger typisk GitHub projects, da det arbejder godt sammen med vores GitHub repository – men folk der bruger Azure, bruger f.eks. typisk Trello.

# UML Diagrammer

### Good to remember

UML-diagrammer er et all around forstået design sprog, som er nemt at læse sig til. UML bliver brugt til meget design og analyse inden begyndelse af konstruktionen af projekter.

# Access modifiers

### Good to remember

Access modifiers fortæller hvad der kan læse/ændre i medlemmer (metoder, properties, fields, constructors, klasser) i vores kode.

Brugte access modifers er følgende:

* **Public:** Alle kan tilgå denne datatype.
* **Private**: Kun metoder i samme klasse kan tilgå denne datatype.
* **Protected:** Kun brugt i nedarvning. Baseklassen kan læse og ændre denne modifier. Den kan læses af alle klasser som nedarver denne.

# Forskellen på en klasse og et objekt

### Good to remember

Klassen er et blueprint af objektet.

Objektet er en instans af klassen, som indeholder diverse data og behaviour.

# Indkapsling

### Important to know

Indkapsling er en måde at sikreet objekts tilstand. Dette bør gøres overalt i vores kode for at sikre objektets tilstand.

Vi har et field og en property. Vores field holder vores data, propertien har vi vores indkapsling. Indkapsling er egentligt bare en måde at sikre at vi ikke indsætter ugyldigt eller uønsket data i vores field. Dette gør i vores set del af get/set’eren i propertien.

# Entiter

### Good to know

Entiteter bliver oftest konverteret til datatyper. Entiteter er dele af kerne værdier af et forretningsdomæne. F.eks. en biograf kan være billet, biograf sal osv. – disse 2 entiteter har noget data indeni som typisk forventes udfyldt.

Det er en måde at oprette/sende objekter rundt, med en over all forståelse af hvad dette objekt indeholder.

# Komposition & Aggregation

### Good to know

Komposition & Aggregation er hvordan man samler objekter. Komposition kan sammenlignes med en krop. Hvis hjertet i en krop stopper, stopper hele kroppen også. Derved er de afhængige af hinanden.

Hvorimod at Aggregation er lidt løsere, her kan hvert medlem af gruppen også fungere alene. F.eks. et bibliotek. Et bibliotek kan have 0 til mange bøger, ligesom at en bog kan eksistere udenfor et bibliotek.

# Arv

### Good to know

Vi bruger ofte nedarvning i OOP for at sikre DRY princippet. Nedarvning gør det muligt at have en base klasse. Denne base klasse kan have metoder som tilslut database, udfør sql kode i database osv. (Disse har oftest ”protected” Access Modifier) og så nogle ”sub klasser” – disse sub klasser kan udføre base klassens metoder, så vi kun skal vedligeholde gengående metoder et sted.

Nedarvning er mere teknisk set en måde at sige ”dine fields, metoder, properties, constructors osv.) er nu også mine” – så vi kan udnytte dem. Det er derfor også kun 1 objekt du opretter, ved nedarvning.

# Keywords

### Nice to remember stuff.

* **Dataformat**: En måde at transportere data gennem internet, ved brug af http(s) protokollen.
* **JSON**: (JavaScript Object Notation) er et dataformat.
* **IIS**: IIS har ansvaret for at dirigere URI (alt efter domæne .xx/) til den rigtige action metode i vores API.
* **URI:** Alt efter vores domæne (”www.domæne.xx/**api/Member**”) - det med fed er URI**.**
* **API**: Application Programming Interface.
* **Klient**: Den visuelle del af et fullstack system. Brugeren ser dette.
* **Server**: Hjerne bag et helt system.
* **Database**: Programmet som indeholder alt data til hele systemet.
* **Metode/Method**: En funktion som har et ansvar i en .NET klasse.
* **Field**: Fieldet holder dataen til vores objekt.
* **Properties**: Metoden til et field? som sikre et objekts tilstand via. Indkapsling.
* **Indkapsling**: Sikre at vi ikke gør et objekt’s data ugyldigt/invalidt.
* **Klasse/Class**: En klasse med et ansvar. F.eks. vores Controllers, Entiteter osv.
* **Constructors**: Måden man sikre objektets start tilstand.
* **Argument**: Det der gør at vi kan sende data fra funktion til funktion.
* **Action Method**: En public metode som ligger i en API controller.
* **Regions**: Regions gør udvikler oplevelsen bedre, da vi kan skjule og vise regioner af kode vi (ikke) har brug for, uden at fjerne det.
* **AI**: Min bedste ven, det er brugt meget til svar på diverse spørgsmål omkring objekt orienterede principper.
* **Design pattern**: Hvordan et library er bygget op med tekniske værktøjer.
* **DRY:** Dont Repeat Yourself
* **KIS(S):** Keep It Simple (Stupid)
* **UI:** User Interface
* **UX:** User Experience