Arkitektur og løst koblet kode

Agenda

- Arkitektur
- Tett koblet kode
  - Monolittisk arkitektur
- Løst koblet kode
- Typiske arkitekturer

Hva er viktig når vi utvikler programvare?

- Opprettholdbarhet
  - Hvor lett er det å bugfikse, oppdatere/bytte funksjonalitet og/eller teknologi?
  - Trenger vi det?
- Utvidbarhet
  - Hvor lett er det å legge til helt ny funksjonalitet eller teknologi?
  - Trenger vi det?
- Resiliance - Skalerbarhet
  - Må vi kunne dynamisk skalere opp systemet for å kunne håndtere alle forespørsler innen en rimelig tid?

- Forskjellige arkitekturer benyttes for å balansere disse aspektene

Arkitektur

- Arkitektur er ett litt flertydig begrep
  - Systemets struktur vs. Prosessen av å strukturere
  - Scope: Helhetlig vs. kodenivå
  - Vårt utgangspunkt: Hvordan systemet er strukturert i komponenter, forholdet mellom dem og prinsippene bak deres design og videreutvikling (IEEE)
- "Komponenter"
  - Litt flertydig dette også...
  - Kan egentlig være alt fra et komplett program til et enkelt objekt eller funksjon
  - «Definisjon»: Et element som implementerer et klart definert sett med funksjonalitet eller features (Sommerville)

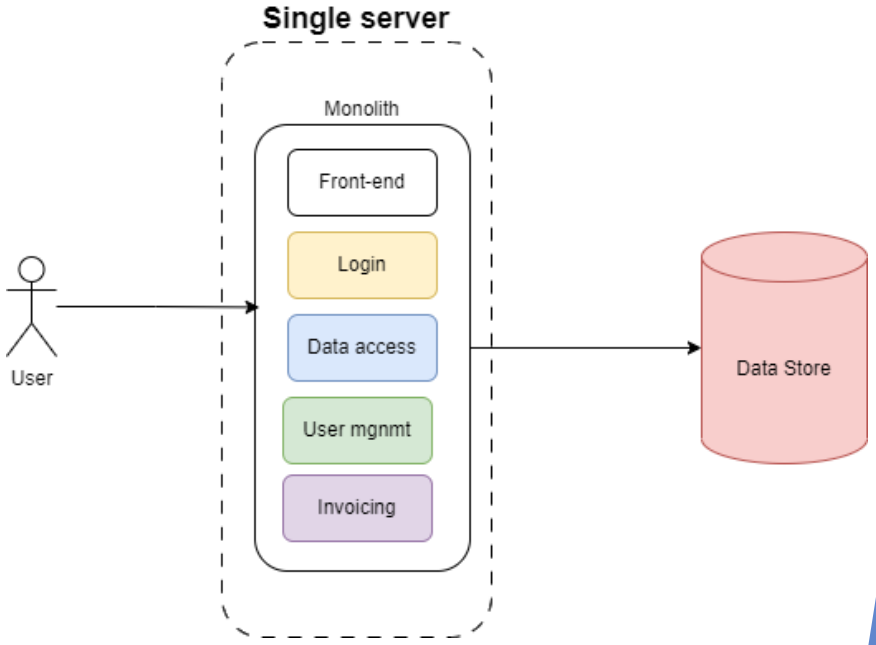
Generelle og vanlig brukte arkitekturer

Tett koblet kode

- Tett koblet kode er når forskjellige «komponenter» i programmet er sterkt avhengige av hverandre
- Hva skjer hvis vi må oppdatere/bytte ut en komponent tett koblet med andre?
  - I beste fall må vi også oppdatere de komponentene som er avhengige
  - I verste fall må vi kaste programmet og skrive det fra bunnen av...
- Altså: Kan bli skummelt i lengden
  - Ny funksjonalitet kan også bli vanskelig å implementere

Monolittisk arkitektur

Monolittisk arkitektur: All kode henger tett sammen og kan tenkes på som én "ting"	•Alle komponenter er nødvendige for å kunne kompilere og kjøre •Resulterer i én kjørbar fil (Ett prosjekt) •De fleste software starter som en monolittisk arkitektur •Alle komponenter deler samme database
Fordeler	•Enkelt å utvikle •Enkelt å feil**søke** (men ikke nødvendigvis fikse) •Gir i prinsipp god hastighet
Ulemper	•Vanskelig å gjøre store endringer •Vanskelig å oppgradere teknologi •Kan være tungt å slippe nye versjoner
Best egnet for	•Små enkle applikasjoner •Kort levetid •Rask utvikling



Løst koblet kode

Løst koblet kode

- Komponentene kan ofte bygges og kjøres individuelt
- Endringer i én komponent skal IKKE påvirke andre komponenter
- Komponenter kan typisk byttes ut enkelt

Fordeler

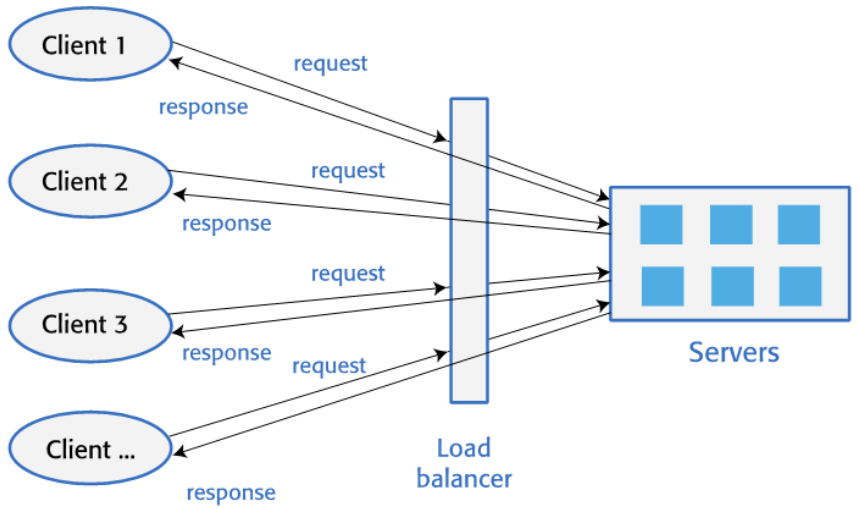
- Enkelt å opprettholde og videreutvikle
- Systemet kan tilpasses ved å bytte ut komponenter
- Komponenter kan enkelt gjenbrukes
- Meget skalerbart

Ulemper

- Tar lengere tid å utvikle
- Kan øke kompleksiteten
- Produktet kan potensielt ikke leveres som én «pakke»
- Øker potensielt prosesseringstid

Klient-server arkitektur

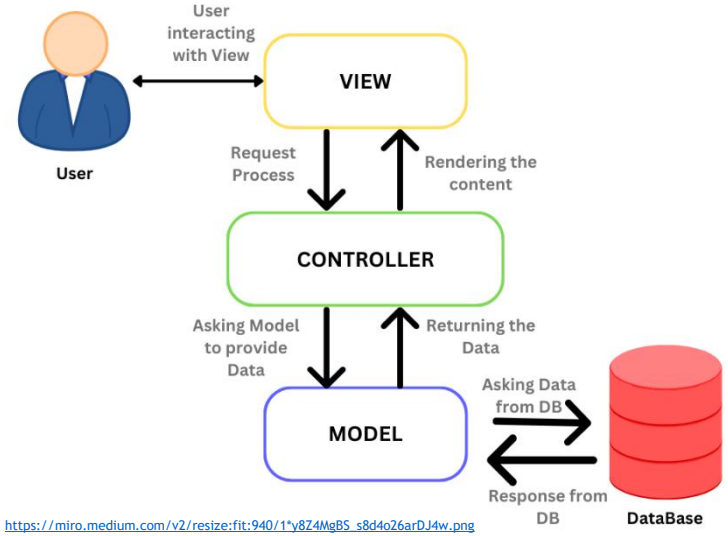
- Klient-server arkitektur: produkt-funksjonalitet skilles i klienter som snakker med delte servere
  - Klient – Er hovedsakelig ansvarlig for brukerinteraksjon og presentasjon av data
  - Servere – Er hovedsakelig ansvarlig for håndtering av data
  - Annen mer spesifikk logikk kan distribueres på enten klient eller servere
- Veldig vanlig i slikt som web-applikasjoner og mobil-applikasjoner
  - ... Egentlig generelt i applikasjoner med en eller flere delte databaser / servere



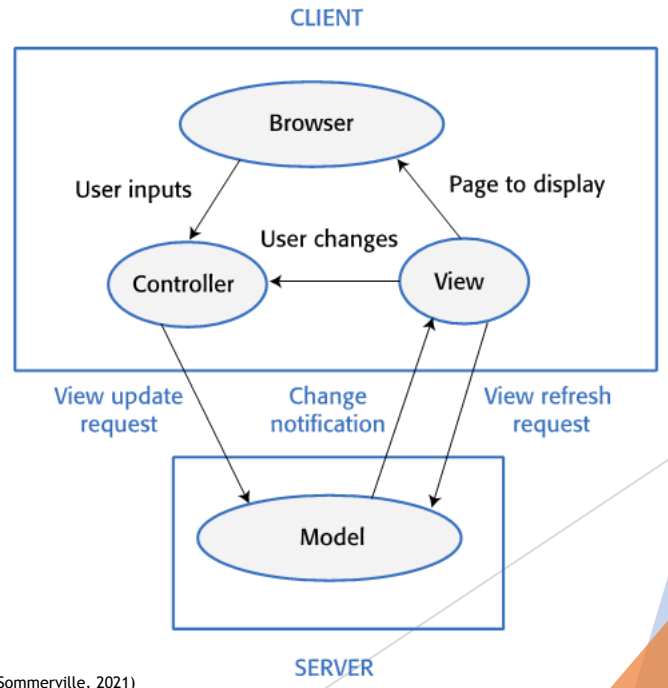
(Sommerville, 2021)

Model-View-Controller arkitektur

- Model-View-Controller (MVC) arkitektur: Ansvar deles i forskjellige lag
  - Model – Data / struktur av data
  - View – Visuell presentasjon av data
  - Controller – Ansvarlig for å håndtering av handlinger (events)
- Finnes flere varianter
  - Lagdelt
  - Trekant
  - Men felles: Model håndteres på serveren
- Brukes typisk sammen med klient-server arkitektur



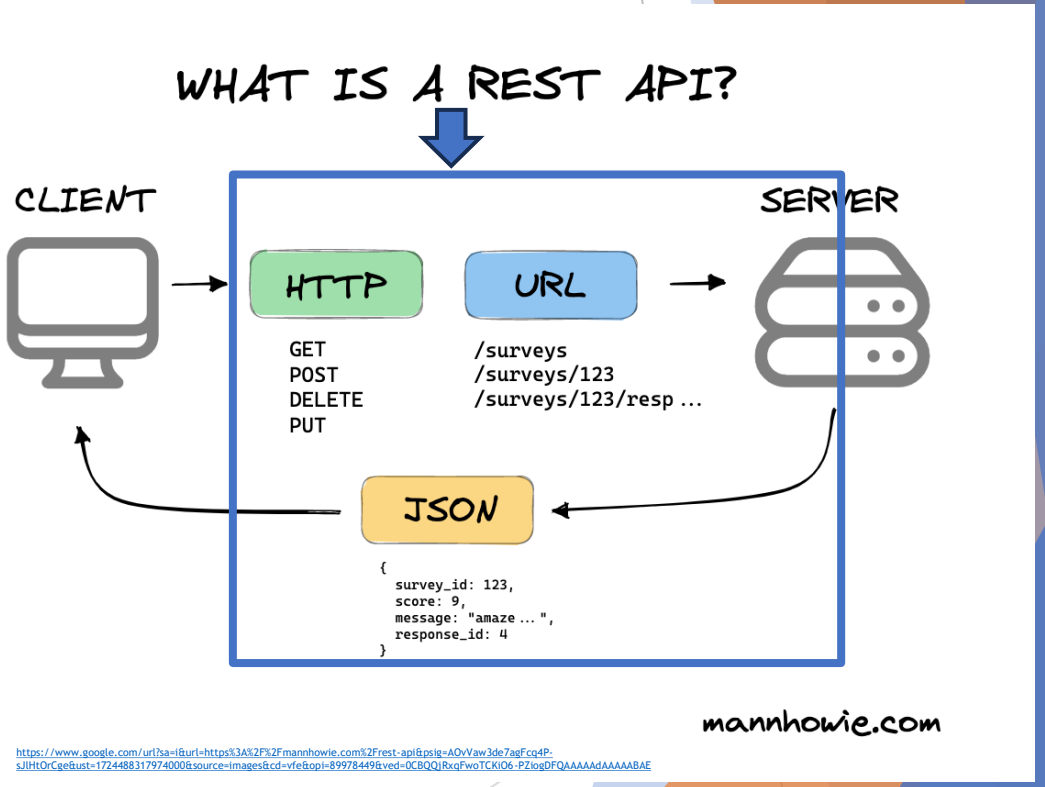
https://miro.medium.com/v2/resize:fit:940/1*y8Z4MgBS_s8d4o26arDJ4w.png



(Sommerville, 2021)

(Web-)API-er

- Application Programming Interface (API) – En type system-komponent som er ansvarlig for å håndtere kommunikasjon mellom andre komponenter
  - Kan tenkes på som en bro mellom komponenter (f.eks. Brukergrensesnitt og server/database)
- Kommunikasjonen foregår gjennom HTTP-forespørsler for abstrakte handlinger
  - Gi meg …, Legg til …, Oppdater …, Slett …, Utfør …
  - Kalles via URL-er
- Komponentene trenger altså ikke å ha direkte insikt i andre komponenters implementasjon
  - Bare hvilke handlinger som kan etterspørres gjennom API-et



https://www.google.com/url?sa=i&url=https%3A%2F%2Fmannhowie.com%2Frest-api&psig=AOvVaw3de7aqFcq4P-sJlHtOrCge&ust=1724488317974000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCKiO6-PZ1ogDFQAAAAAdAAAAABAE

Serialisering

- Vi kan ikke sende objekter (i tradisjonelt format) over nett
- HTTP-kommunikasjon er TEKST
  - ... altså må sendte objekter også være tekst

- Serialisering – Transformere et objekt til tekst
- Deserialisere – Transformere tekst tilbake til et objekt

- Typisk JSON-format
  - Alternativt XML, men JSON er mer lettvektig og ofte lettere å lese

- Det finnes mange biblioteker for serialisering/deserialisering
  - Jackson (kan anbefales)
  - Gson
  - Finnes typisk alternativer i alle språk

```
public class Person {
    private String name;
    private int age;
    private List<String> hobbies;

    // Getters and setters
}
```

```
{
    "name": "John Doe",
    "age": 30,
    "hobbies": [
        "Reading",
        "Hiking"
    ]
}
```

Data Transfer Object (DTO)

- Et Data Transfer Object (DTO) er en klasse/objekt som BARE benyttes i forbindelse med oversendelse av data (over nett)
  - Inneholder bare ren data (variabler med get-/set-metoder)
  - Kan samle data fra flere typer objekter i én struktur
  - Disse er typisk hva som blir serialisert/deserialisert
  - Typisk unike DTOs for spesifikke kontekster (views, typer komponenter, osv.)
  - Benyttes ofte ved API-kall (hente og endre/sette inn)
  - DTO-klassene hører til modellene og ligger på serveren(e)

- Fordeler med å DTOs
  - Komponenter blir uavhengige av den originale datastukturen (løsere kobling)
  - Vi kan samle data og begrense oss til de vi trenger (optimalisere hastighet og redusere kompleksitet)
  - Mer standardisert → Enklere å teste

```
public class ProductViewDTO {
    private Long id;
    private String name;
    private String description;
    private BigDecimal price;
    private String imageUrl;
    private boolean isAvailable;

    // Getters and setters
}
```

```
public class OrderSummaryDTO {
    private Order order;
    private Customer customer;
    private List<OrderItemDTO> items;

    // Getters and setters
}
```