IN2000
Software Engineering med prosjektarbeid

Git, API og dataformater

Menti: **4630 2772**

Steffen Almås

Først…litt praktisk!

- Oblig 1 - frist onsdag 4. februar, kl. 23.59
- Teaminnmelding
  - Skjema - husk å levere før fristen til oblig 1
  - Hvordan settes teamene sammen?
  - Får vite teamet før kickoff

- Kickoff fredag 27. februar

Git og GitHub

Menti: **4630 2772**

Basert på slides laget av https://github.com/kudachambwe

Git og VCS

- Hva er git?
  - Git er et **versjonshåndteringssystem** (**VCS** - Version Control System)

- Hva er et versjonshåndteringssystem?
  - Verktøy som lar oss **samarbeide** med andre på ett eller flere prosjekter.
  - **Håndtere** og **overvåke** endringer. Med et VCS kan vi se hva som er blitt gjort og tilbakestille eventuelle feil.

Git vs. GitHub

git	GitHub
1. It is a software	1. It is a service
2. It is installed locally on the system	2. It is hosted on Web
3. It is a command line tool	3. It provides a graphical interface
4. It is a tool to manage different versions of edits, made to files in a git repository	4. It is a space to upload a copy of the **Git** repository
5. It provides functionalities like Version Control System Source Code Management	5. It provides functionalities of Git like VCS, Source Code Management as well as adding few of its own features

Figur hentet her

Git vs. GitHub og annen “hosting” av repo

- Du bruker alltid **git** lokalt med de samme kommandoene uansett hvor koden ligger remote.

Din maskin med

med git…

Bitbucket

GitLab

GitHub

…snakker samme “språk”

med alle disse

Azure Repos

Git - Hvorfor

- **Hastighet**
  - Henter informasjon om filer og prosjektet raskt og effektivt
- **Distribuert**
  - “Alle” har en kopi av “repoet”
- **Skalerbart**
  - Muliggjør at mange kan samarbeide på samme prosjekt og kode
- **Branches**
  - Skriv kode i et separat “miljø” fra prod. Enkelt å lage/endre/slette “features”
- **Sporbarhet**
  - Alle har oversikt over filene og historien til filene

Git - Språket

- **repository / repo**
  - Et “oppbevaringssted” eller “mappe” hvor alle filene som er i git-prosjektet befinner seg
- **commit**
  - Et objekt som holder på informasjonen om en spesifikk endring. Har som regel en melding (**commit message**).
- **branch**
  - En gren, eller en uavhengig serie med commits. Kan brukes som en “lekeplass” for å eksperimentere med features uten å påvirke andre grener.

Git - Språket (forts.)

- **merge**
  - Kombinerer to (eller flere) brancher til en samlet branch. Git vil automatisk prøve å integrere forandringer.
- **conflict**
  - Oppstår når flere gjør endringer i samme fil eller deler av en fil, og Git ikke klarer å merge dette automatisk.
- **head**
  - En “peker” til den mest nylige commiten i en branch.

Git - Kommandoer

```
git clone [url]
```

Kloner repositoriet du henter fra.

```
git pull
```

Henter nyeste versjon av repoet du befinner deg i.

```
git push
```

Laster opp nyeste comittede versjonen du har av repoet du befinner deg i.

```
git add [file]
```

Legge til, nye eller endrede filer som du vil laste opp til repoet.

Git - Kommandoer (forts.)

```
git commit -m [message]
```

Før filer kan lastes opp, må de committes. Dette må gjøres med en melding.

```
git branch [branch-name]
```

Lager en branch/gren.

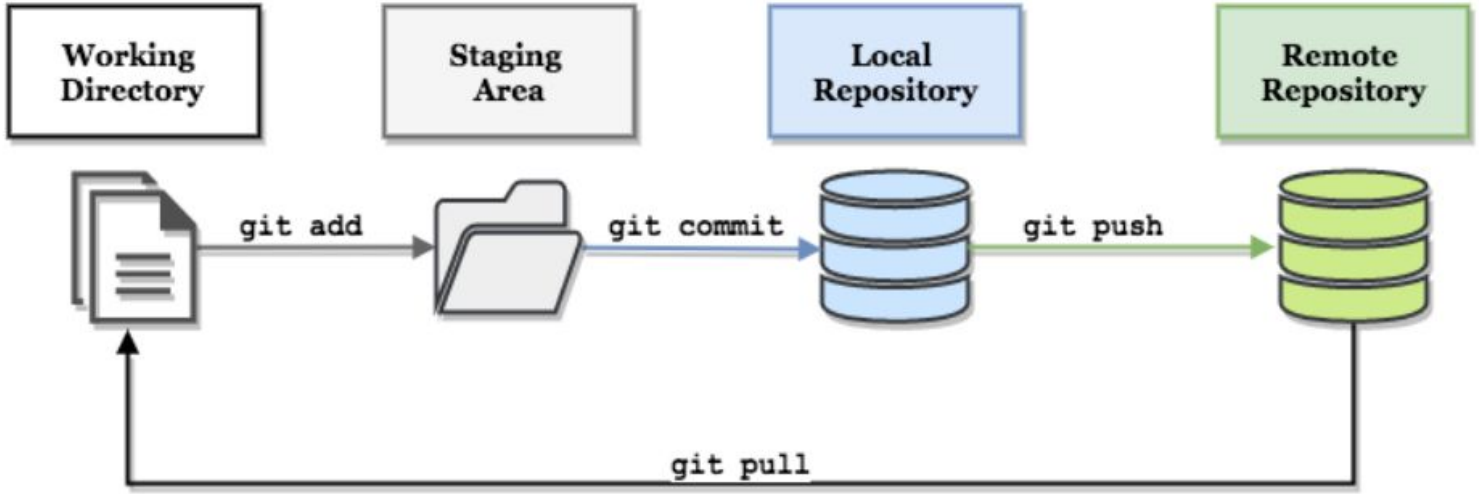
```
git merge [branch]
```

Laster opp alle endringer (i.e. **slå sammen**) fra en gren til den du står på (ved kall). NB: Her kan det oppstå konflikter(!).

```
git checkout [branch-name]
```

Skifter gren dun står på. Brukes for å navigere seg fra/til ulike grener.

Git - Arbeidsflyt, oppsummert



Git - filer å legge merke til

- **README.md**
  - Inneholder som regel en beskrivelse av prosjektet. Vises på “hovedsiden” i GitHub-repoet deres.
- **.gitignore**
  - Liste av filer / mapper i prosjektet som ikke skal spores av git. Denne filen gjør at du ikke “commiter” cache, libraries, API-nøkler, osv..
  - Tips: .gitignore generator og velg “Kotlin” og “AndroidStudio”
- **LICENCE.md/txt**
  - Lisensen prosjektet publiseres under. Ikke relevant for dere i IN2000, med mindre dere skal publisere appen på Google Playstore.

Tre viktige ting å huske på!

Menti: **4630 2772**

1. Respektér main

- IKKE gjør endringer direkte i main
  - Kan føre til at kode som ikke kjører ender der
  - Bruk branch og **pull request** i stedet

- **pull request**
  - Oversikt over endringer der andre kan komme med tilbakemeldinger før branchen kan merges inn i main
  - Kan legge spesifikke person / antall “required reviewers”

2. Beskrivende commit message

- Bør beskrive det du har gjort,

  men ikke skriv i preteritum
- Språk: Engelsk er beste praksiss

Eksempel på:

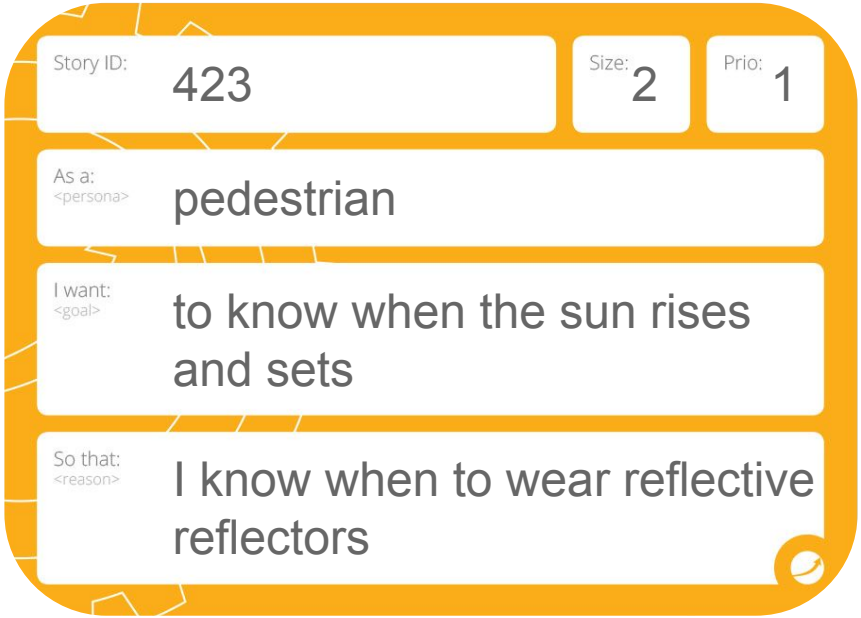
- God commit message
- Mindre god commit message

3. Branches - navngiving

- Navn skal være lett å kjenne igjen / sporbare
- Bygg opp navn på følgende måte:
  1. feat / docs / test / fix
  2. id på tasken / User Story
  3. det du løser i branchen

<u>Eksempel</u>

feat/423-display-sunrise-and-sunset



Branching strategier

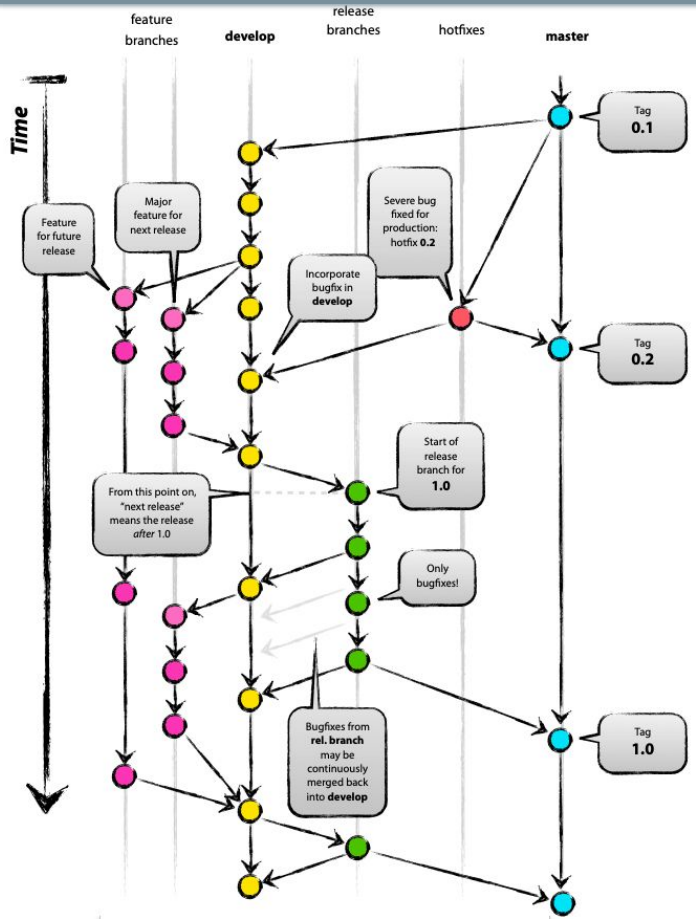
Menti: **4630 2772**

Branching strategier

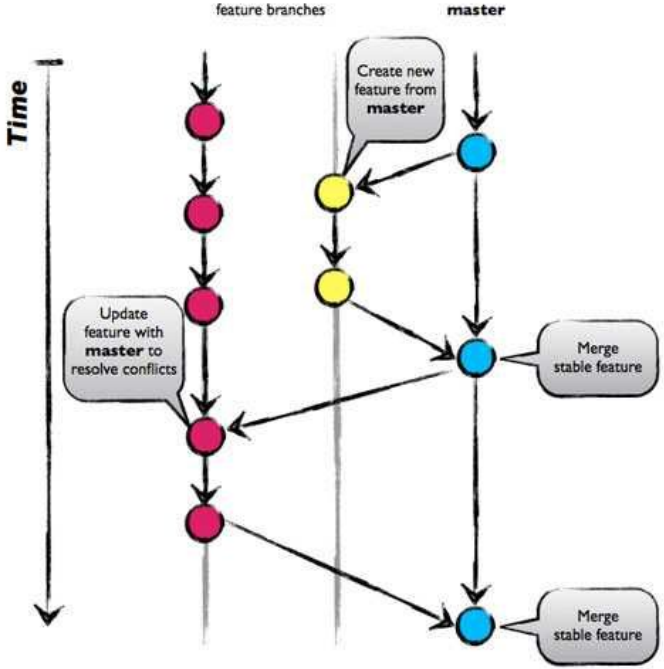
- **Hva** er en branching strategi?
  - En strategi for hvordan håndheve / bruke git i et prosjekt

- **Hvorfor?**
  - Ingen prosjekter er like
  - Felles forståelse over hvordan git skal brukes i prosjektet så alle “jobber likt”.

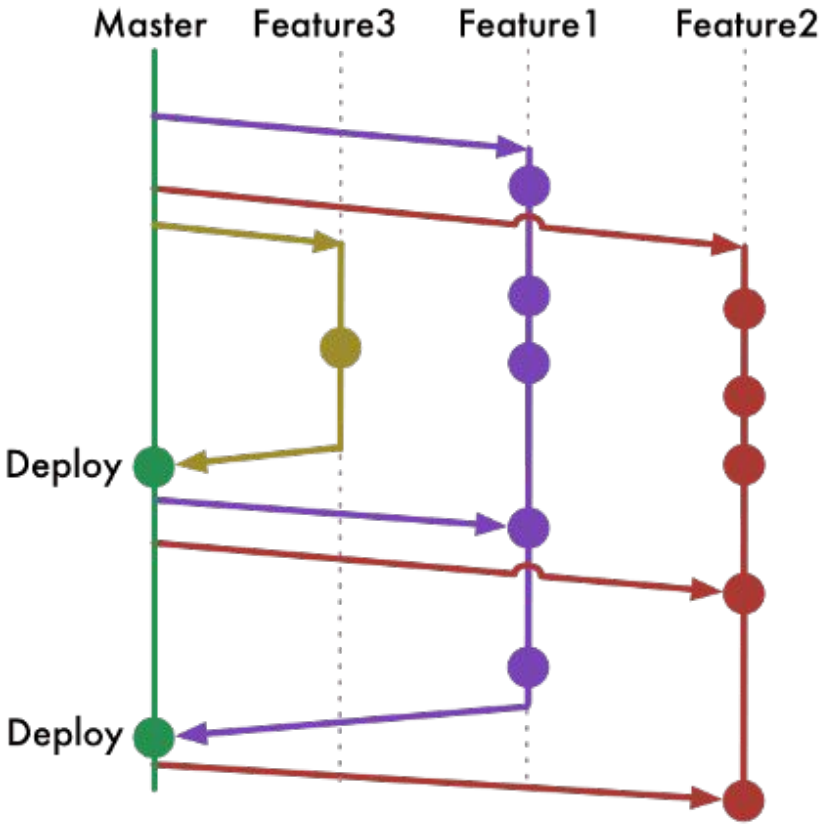
Branching strategier - GitFlow



Branching strategier - GitHub Flow



Branching strategier - Trunk-based Development



Branching strategier - Hvordan velge?

- Ingen “perfekt” strategi som en alltid bør velge
- Tenk på: ***kommunikasjon***, ***modenhet i teamet***, ***størrelsen på teamet***, ***hva som utvikles*** og mer!

Product type and its release method	Team size	Collaboration maturity	Applicable mainstream branch mode
All	Small team	High	Trunk-Based Development (TBD)
Products that support continuous deployment and release, such as SaaS products	Middle	Moderate	GitHub-Flow and TBD
Products with a definite release window and a periodic version release cadence, such as iOS apps	Middle	Moderate	Git-Flow and GitLab-Flow with release branch
Products that are demanding for product quality and support continuous deployment and release, such as basic platform products	Middle	Moderate	GitLab-Flow
Products that are demanding for product quality and have a long maintenance cycle for released versions, such as 2B basic platform products	Large	Moderate	Git-Flow

Tabell hentet her: https://www.flagship.io/git-branching-strategies/

Koble Android Studio med UiO-GitHub

Menti: **4630 2772**

Koble Android Studio og UiO-GitHub

- Generering av PAT (Personal Access Token) for autentisering av GitHub-UiO bruker mot Android Studio
- Du finner genererte tokens under Settings → Developer Settings

- Forarbeid:
  - Sjekk at du kommer inn på github.uio.no med ditt UiO brukernavn- og passord

Koble Android Studio og UiO-GitHub

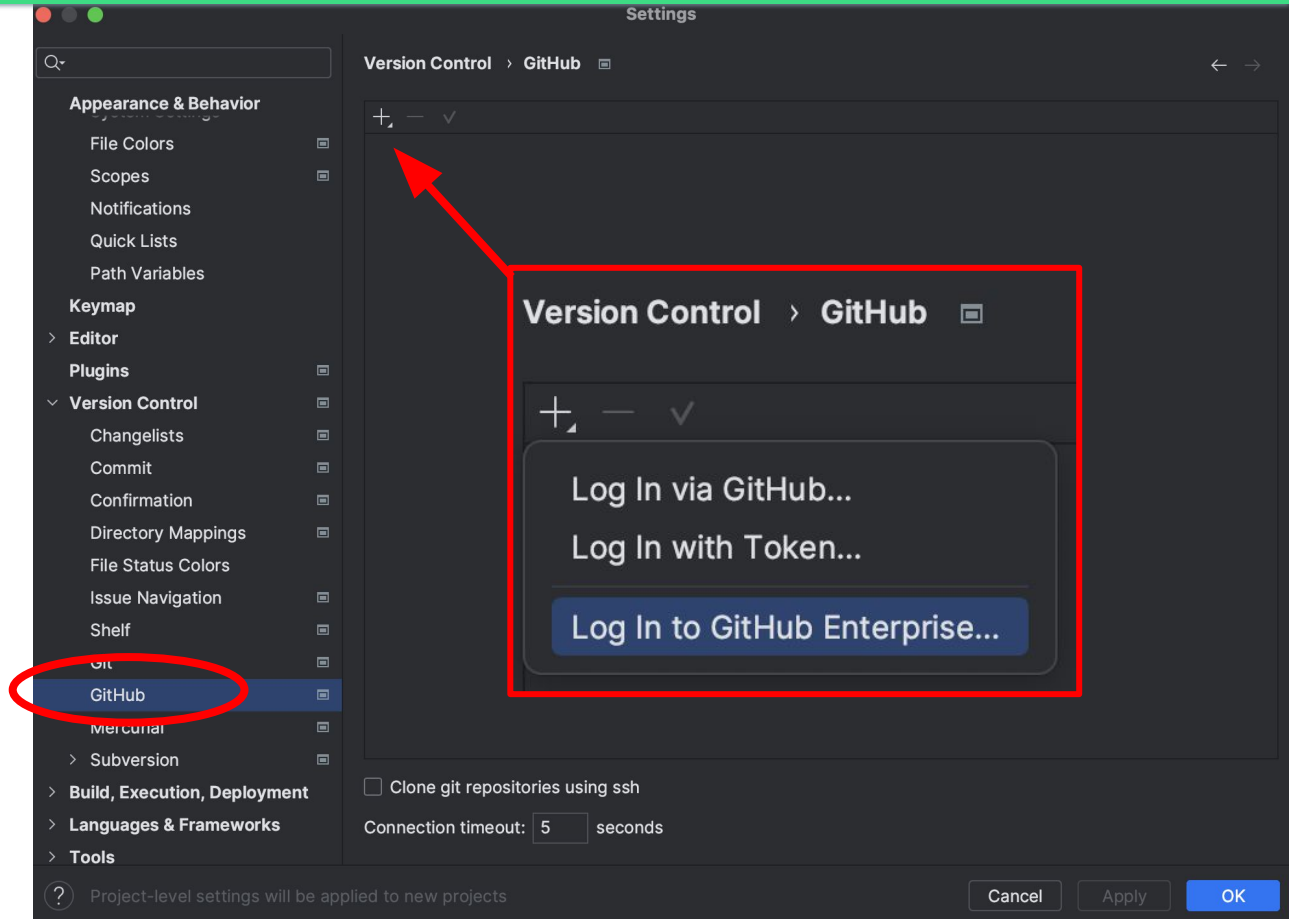
→ Settings

→ Version Control

→ GitHub

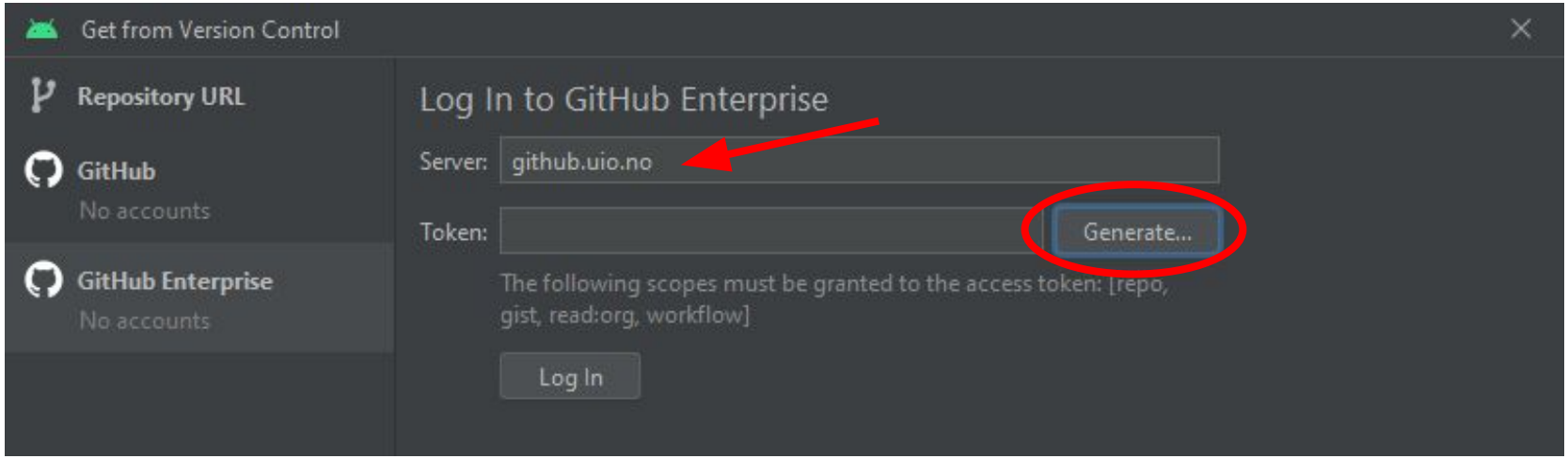
→ Trykk på “+”

→ Velg “GitHub Enterprise”



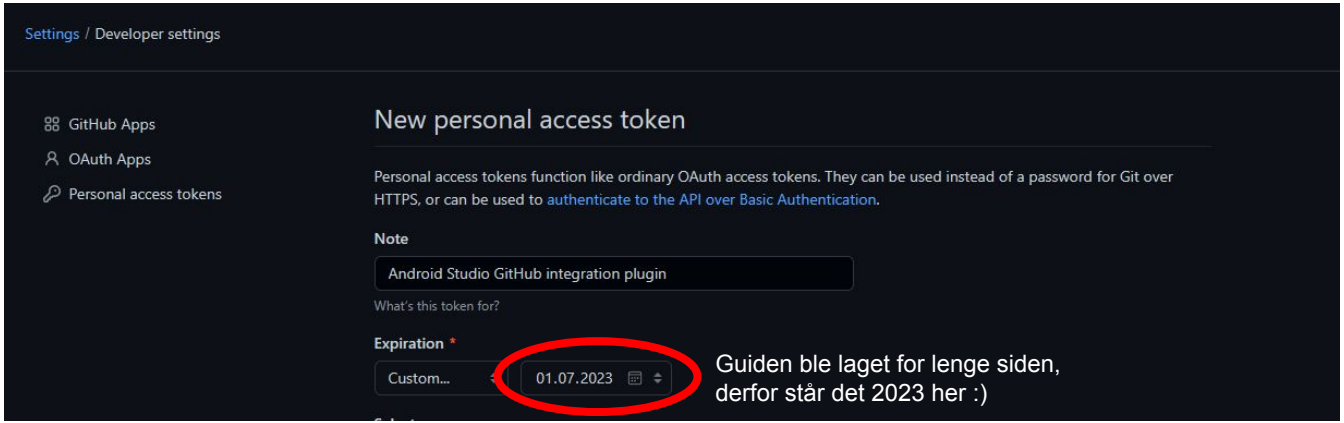
Koble Android Studio og UiO-GitHub

- Først, skriv inn “github.uio.no” i Server-feltet
- Trykk deretter på “Generate” bak Token-feltet
- Logg inn med UiO-brukernavn og passord i popup-vinduet



Koble Android Studio og UiO-GitHub

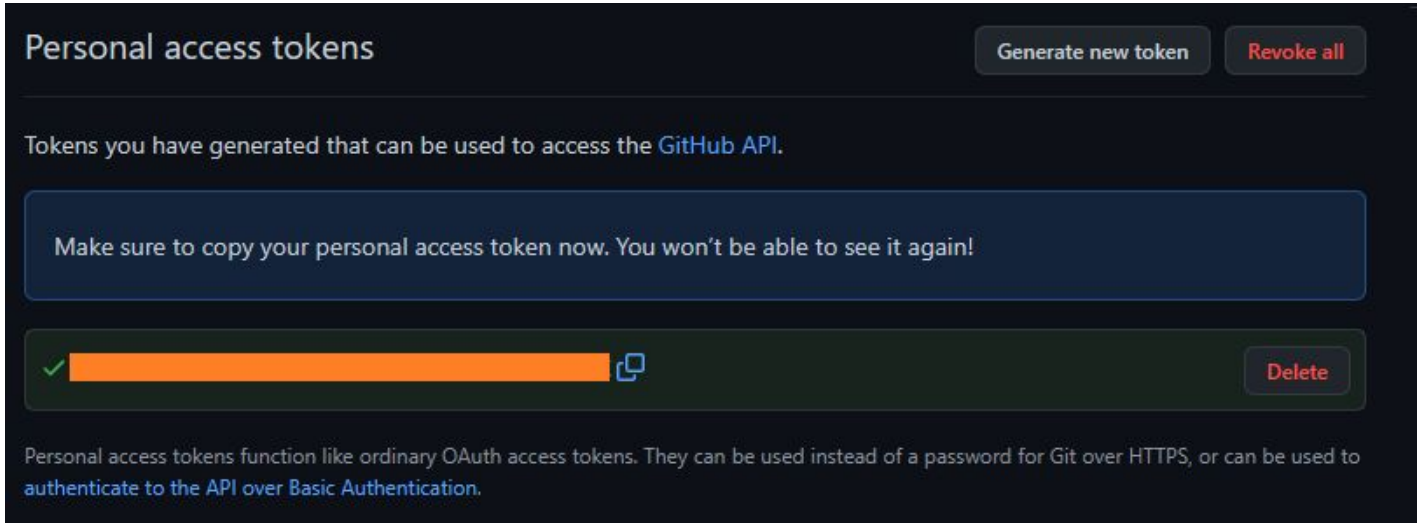
- Velg “Tokens (classic)”
- Sett “Expiration date” til 1. juli 2026 (eller en annen dato etter prosjektarbeidet)



- Trykk til slutt på “Generate Token”: **Generate token**

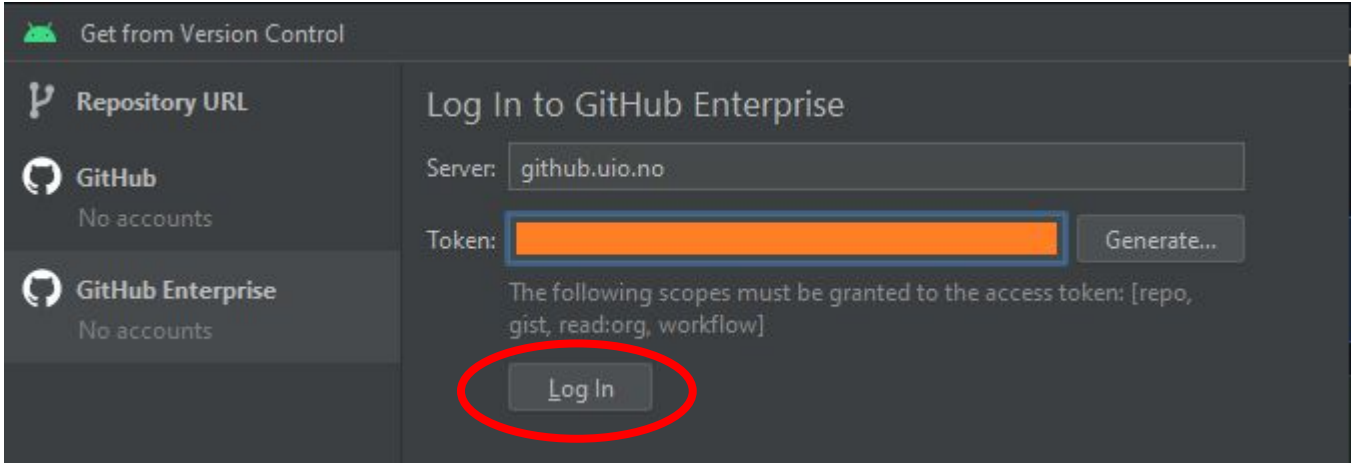
Koble Android Studio og UiO-GitHub

- Kopier så det genererte Tokenet…



Koble Android Studio og UiO-GitHub

- …legg det til i Token-feltet og trykk “Log in”



- Du har nå koblet UiO GitHub-bruker med Android Studio!

Git i terminalen

Menti: **4630 2772**

Git - GitHub UiO i terminalen

- Sjekk om git er installert: `git --version` → Hvis ikke last ned
- Kjør: `ssh-keygen -t ed25519` i terminalen
  - Skriv inn filsti til hvor nøkkelen skal legges
  - Hopp over passord
- Gå til github.uio.no
  - Gå til Settings → SSH / GPG keys
  - Lim inn nøkkelen du genererte i terminalen
- Du har nå tilgang til UiO-GitHub fra terminalen - sett i gang, klon et repo :D

Git - Demo

- Initialisere ett git repo
- Gjøre første commit og pushe til GitHub
- Lage en branch og utføre en endring
- Lage et pull request og merge med main-branch

Pause ☕

API og dataformater

Menti: **4630 2772**

HTTP

Menti: **4630 2772**

HTTP - Generelt

- **H**yper**T**ext **T**ransfer **P**rotocol
- Standard protokoll for å utveksle informasjon over internett
  - Benytter vanligvis port 80
- Har ulike HTTP-requests for ulike operasjoner som kan gjøres

- HTTPS → HyperText Transfor Protocol Secure

HTTP - Statuskoder

- Server responderer på forespørsler med statuskode(r)
- Tre siffer langt
- Indikerer status på HTTP-forespørselen

HTTP - Liste med statuskoder

- 1xx - Informasjon
- 2xx - Suksess
  - **200 OK**
- 3xx - Viderekobling
- 4xx - Klient-feil
  - **400 Bad Request**
  - **404 Not Found**
- 5xx - Server-feil

Se læreboka: s. 171

https://www.restapitutorial.com/httpstatuscodes.html

HTTP - Statuskodene 200 og 404

200 OK

Request URL: `https://www.uio.no/studier/emner/matnat/ifi/IN2000/`

Request Method: `GET`

Status Code: 🟢 `200`

404 Not Found

Request URL: `https://www.uio.no/studier/emner/matnat/ifi/IN2002/`

Request Method: `GET`

Status Code: 🔴 `404`

HTTP - Prøv selv i terminal!

- **PowerShell (Windows)**
  - `Invoke-RestMethod -Uri www.uio.no -Method Get`
- **Unix-terminal (Mac, Linux)**
  - `curl -L www.uio.no`
- Se også i nettleseren under network-taben

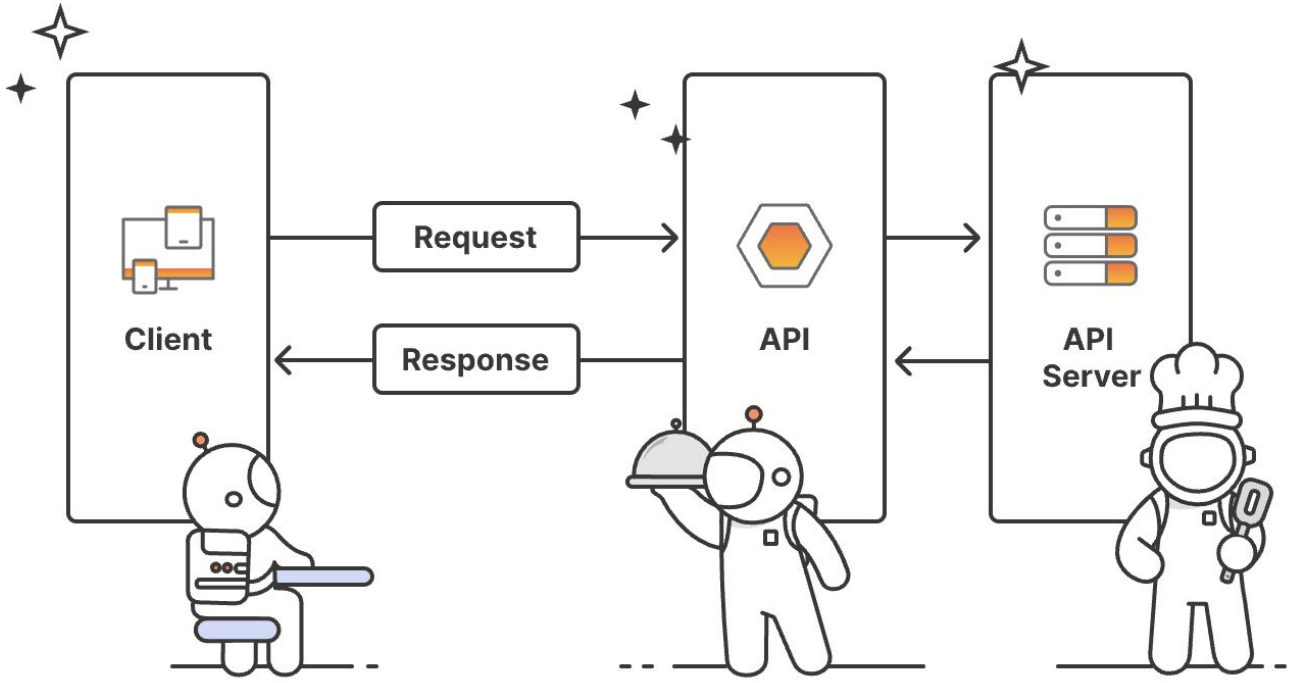
API

Menti: **4630 2772**

API - Generelt

- **A**pplication **P**rogramming **I**nterface
- Grenseressurser for en plattform / tjeneste
  - “Single Point of Entry”
  - Aksessere data
  - Trenger bare å forholde seg til API-et
- Dokumentasjon
  - OpenAPI

API - Analogi



https://www.postman.com/what-is-an-api/

RESTful API

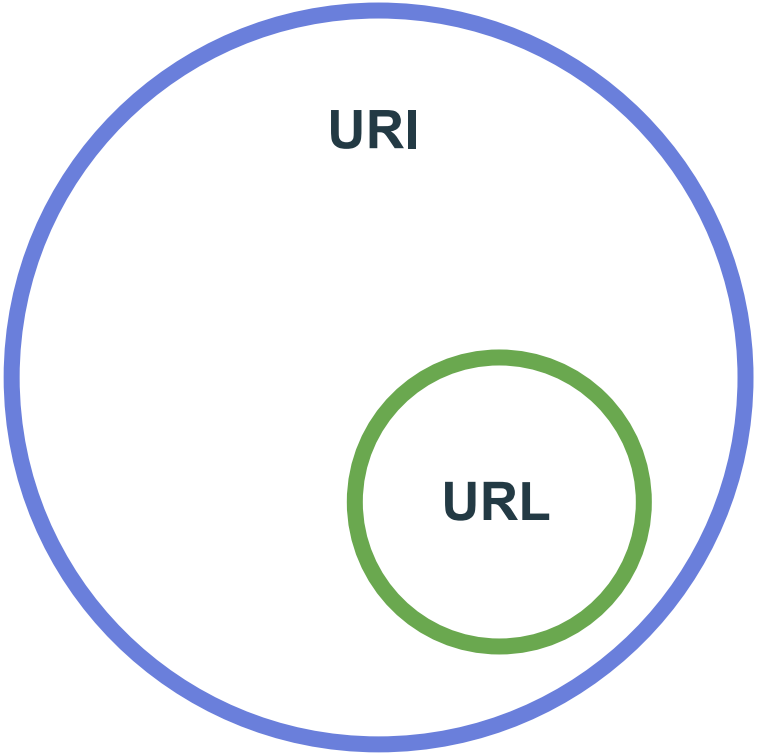
Menti: **4630 2772**

REST - Om

- **Re**presentional **S**tate **T**ransfer
  - REST og RESTful er det samme!
  - Arkitekturstil
- Ressurser adressert med en URI
- Ressurser er representert med et dataformat
- Bruker HTTP Internet Protocol for å få tak i en ressurs

REST - URI vs. URL

- URI = Uniform Resource Identifier
  - Eks. ditt navn
- URL = Uniform Resource Locator
  - Eks. din adresse
- URL ⊆ URI



REST - Prinsipper

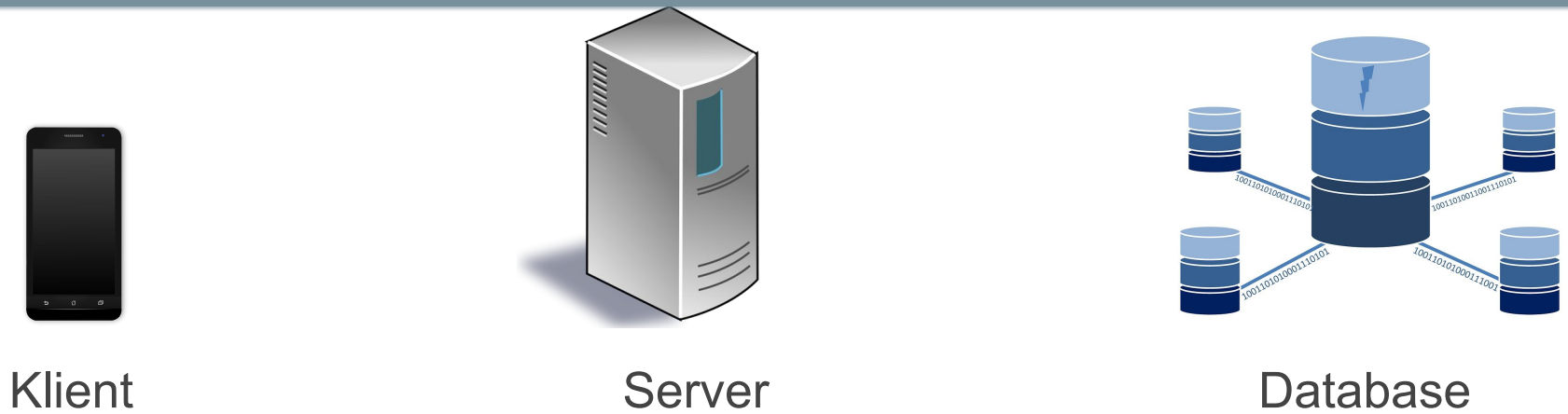
Prinsipp	Beskrivelse
Bruk HTTP verb	Metoder beskrevet av HTTP-protokollen skal benyttes for å aksessere ressurser.
Tilstandsløshet	Serveren skal ikke holde på ulike tilstander med mindre den må.
Adresserbar	Alle ressurser må ha en URI. En URI kan bare peke på én ressurs, mens en ressurs kan bli pekt på av flere URI-er.
Bruk XML eller JSON	Ressurser skal vanligvis bli representert som JSON eller XML (evt. en kombinasjon).

Basert på læreboka: s. 174

REST - GET-kall

- Hente eller lese en representasjon av en ressurs
- Veldig relevant for dette kurset
- Eksempel på GET-requests:
    - www.google.com
    - https://api.entur.io/realtime/v1/rest/et?maxSize=10

REST - GET-kall (forts.)



REST - POST og PUT-kall

- **POST**
  - Brukes for å opprette en eller flere nye ressurser
  - Serveren bestemmer URI for nye ressurser

- **PUT**
  - Brukes for å oppdatere en eksisterende ressurs
  - Kan opprette, men bare èn av gangen

REST - Andre operasjoner

- HEAD
- DELETE
- CONNECT
- OPTIONS
- TRACE

Men disse er ikke relevante for prosjektet!

Dataformater

Menti: **4630 2772**

Dataformater - Om

- Data fra request vil komme på et format
- Som oftest kommer det enten som XML eller JSON
  - Kan også være HTML, CSV, PDF osv.
- Data må håndteres
  - Deserialisering

XML

Menti: **4630 2772**

XML - Om

- e**X**tensible **M**arkup **L**anguage
  - Utvidbart både som språk og dokument
  - Legge til metadata med tager
  - Kan definere andre språk (f.eks HTML)
- Versjon 1.0 mest brukt
  - Må deklareres!
- Struktur og kontekst på dataene
- Ble brukt til å definere layout i Android før Jetpack Compose

XML - Egenskaper

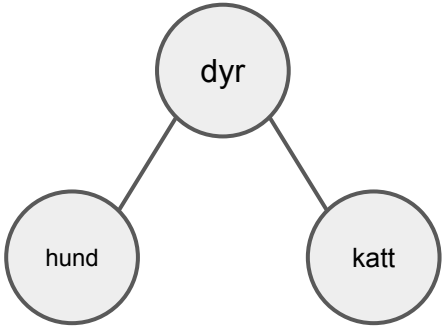
- Et rot-element
- Tag system
- Elementer og elementnavn
- Attributter
  - metadata inni start-tag
- Well-Formed
- Broken

```
<tag>   ← (Start-tag)

    verdi

</tag>← (Slutt-tag)
```

XML - Lite eksempel

```
<?xml version="1.0" encoding="utf-8"?>
<dyr>
  <hund>
  </hund>

  <katt>
  </katt>
</dyr>
```

Representert som et tre



XML - Lite eksempel (forts.) - Well formed

```xml
<?xml version="1.0" encoding="utf-8"?>
<dyr>
  <hund>
      <navn>Fido</navn>
      <alder>6</alder>
  </hund>
  <katt>
      <navn>Missy</navn>
      <alder>4</alder>
  </katt>
</dyr>
```

XML - Lite eksempel (forts.) - Broken

```xml
<?xml version="1.0" encoding="utf-8"?>
<dyr>
  <hund>
      <navn>Fido</navn>
      <alder>6</alder>
  </hund>
  <katt>
      <navn>Missy</Navn>
      <alder>4</alder>
  </katt>
</dyr>
```

XML - Større eksempel

```
<?xml version="1.0" encoding="utf-8"?>
<bilRegister>
    <bil>
        <prod>Norwegian Cars</prod>
        <regnr>IN2000</regnr>
        <eier>Yngve</eier>
    </bil>
    <bil>
        <prod>Norwegian Cars</prod>
        <regnr>IN2001</regnr>
        <eier>Steffen</eier>
    </bil>
    <bil>
        <prod>Norwegian Cars</prod>
        <regnr>IN2002</regnr>
        <eier>Stein Michael</eier>
    </bil>
</bilRegister>
```

```
<?xml version="1.0" encoding="utf-8"?>
<bilRegister>
    <bil
        prod="Norwegian Cars"
        regnr="IN2000"
        eier="Yngve">
    </bil>
    <bil
        prod="Norwegian Cars"
        regnr="IN2001"
        eier="Steffen">
    </bil>
    <bil
        prod="Norwegian Cars"
        regnr="IN2002"
        eier="Stein Michael">
    </bil>
</bilRegister>
```

XML - Namespace og schema

- XML Namespace
  - Unngå navnekonflikter
  - Gjenbruke elementnavn innenfor et gitt namespace
  - `xmlns`
  - xmlns : prefix = URI
- XML Schema
  - Regler om elementer, attributter, strukturer, verdier

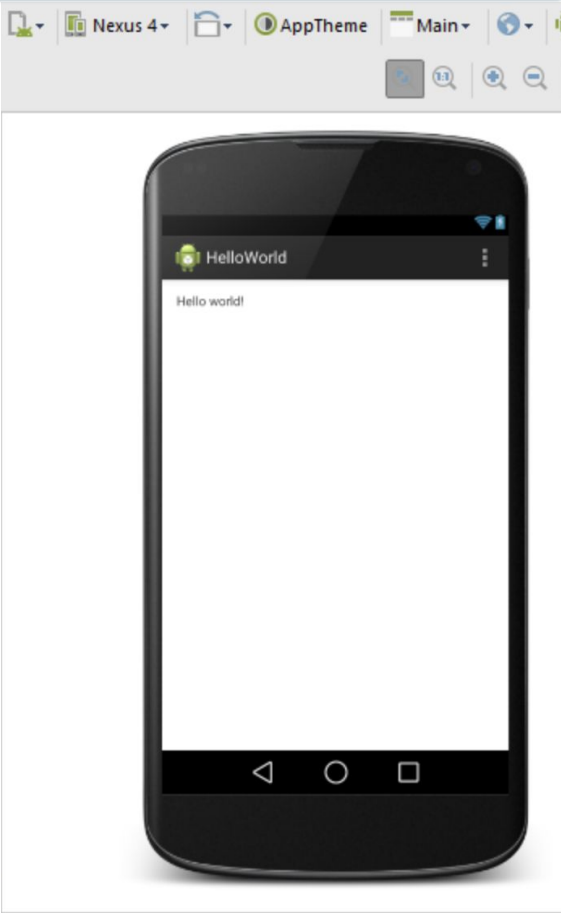
XML - Namespace og schema, eksempel

```
<?xml version="1.0" encoding="utf-8"?>
<bestilling
   xmlns="http://butikk.com/schemas/bestilling"
   xmlns:p="http://butikk.com/schemas/produkter">

   <bestillingId>1234</bestillingId>
   <p:produkt>Vannflaske</p:produkt>
   <antall>2</antall>
</bestilling>
```

Eksempel fra:
https://www.uio.no/studier/emner/matnat/its/TEK5120/

XML - Layout i Android (før Jetpack Compose)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/androi
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="64dp"
    android:paddingRight="64dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".Main">

    <TextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```



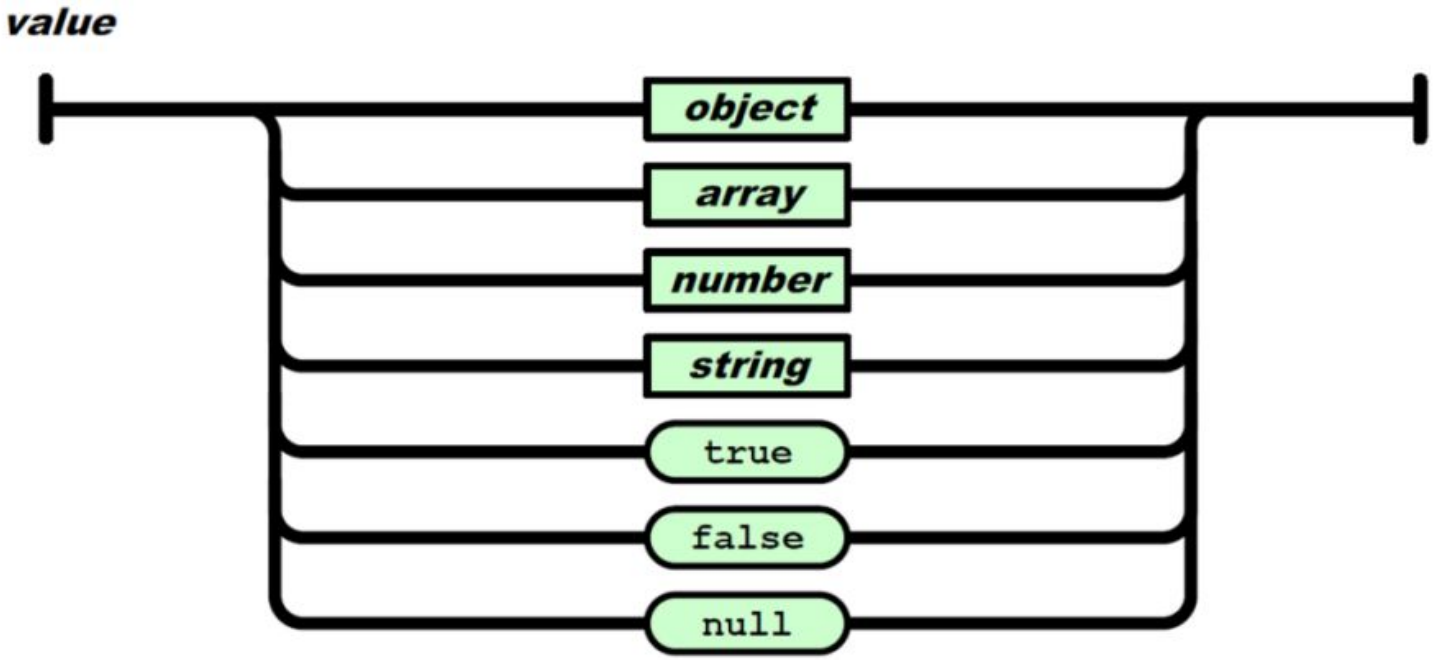
JSON

Menti: **4630 2772**

JSON - Generelt

- **J**ava**S**cript **O**bject **N**otation
- Definert struktur
- Ikke begrenset til JavaScript
- Vanlig representasjon i REST
- Likheter med XML
  - Tekstbasert
  - Lager struktur på data
  - Representert som trær

JSON - Gyldige verdier



JSON - Navn

- Navn som beskriver en verdi
- Må være en String definert med anførselstegn
- Skal være kolon mellom navn og verdi
- Må være unik
- Kan også sees på som nøkkel (key)

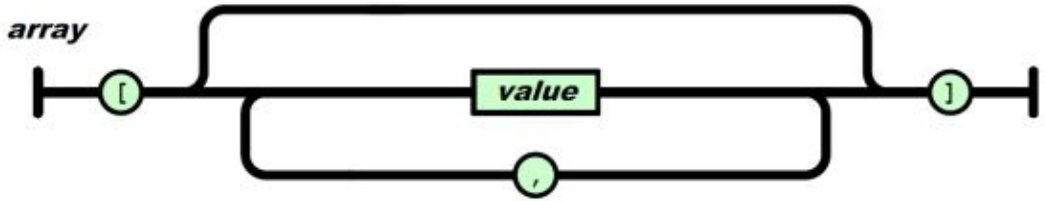
JSON - *string* og *number*

- **string**
  - En sekvens av Unicode tegn pakket inn i anførselstegn
  - Eksempel: `{ "name": "Lars" }`

- **number**
  - Må være Integers eller Float-verdier
  - Eksempel: `{ "age": 30 }`

JSON - *array*

- Defineres med hakeparantes (square brackets)
  - Verdi må være en av de sju gyldige verdiene
  - Elementer skilles med komma
  - Kan ha tomme arrayer [ ]




```
{
  "employees": [
    "Lars",
    "Hilde",
    "Sandra"
  ]
}
```

JSON - *array (forts.)*

JSON

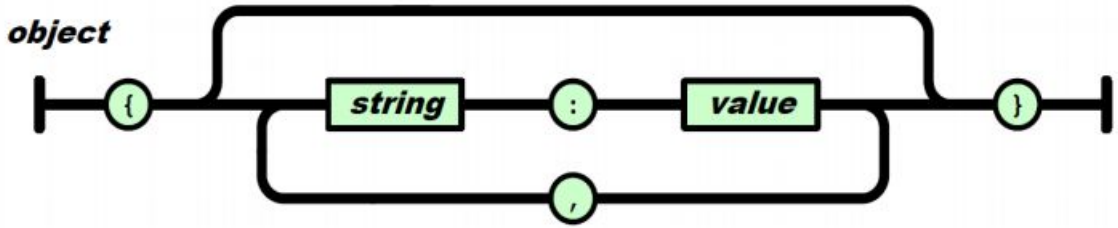
```
{
    "details": [
        false,
        true,
        "Hello World",
        34
    ]
}
```

Eksempel fra: https://www.uio.no/studier/emner/matnat/its/TEK5120/

JSON - *object*

- Et uordnet set av navn/verdi par
- Objekter kan være tomme { }
- Kan konverteres til Kotlin/Java-objekter

```
{
  "employee": {
    "name": "Lars",
    "age": 30,
    "city": "Oslo"
  }
}
```



JSON - Rot-elementet

Rot-elementet i JSON er enten et JSON-object eller JSON-array

```
{
  "hilsen1": "Hei",
  "hilsen2": "På",
  "hilsen3": "Deg"
}
```

```
[
  "Hei",
  "På",
  "Deg"
]
```

XML vs. JSON

Menti: **4630 2772**

XML vs. JSON - Konvertering

- Du kan konvertere fra XML → JSON

- Det er mulig å konvertere JSON → XML
  - Men du kan miste informasjon
  - Det er uansett JSON du som regel ønsker som respons

XML vs. JSON - Eksempel

XML

```xml
<?xml version="1.0"?>
<dyr>
  <hund>
      <navn>Fido</navn>
      <alder>6</alder>
  </hund>
  <katt>
      <navn>Missy</navn>
      <alder>4</alder>
  </katt>
</dyr>
```

JSON

```json
{
 "dyr": {
   "hund": {
     "navn": "Fido",
     "alder": "6"
   },
   "katt": {
     "navn": "Missy",
     "alder": "4"
   }
 }
}
```

XML vs. JSON - Oppsummert

XML

- Må alltid valideres
  - Evalueres av et XML Schema
- Kan konverteres til JSON
  - Ikke lett å konvertere fra JSON → XML
    - pga namespace conflict
- Namespaces → kompleksitet
- Har attributter
- Eksempel

JSON

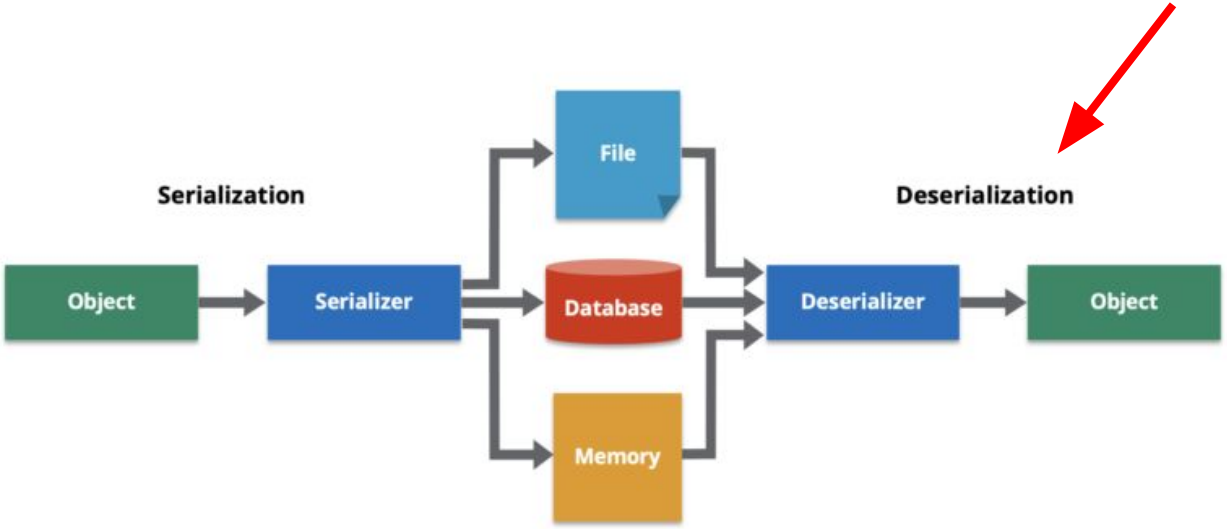
- Er mer verbose, mer kompakt
- Har ingen attributter
- Har ikke namespace
- Er primær dataformat for REST

Deserialisering

Menti: **4630 2772**

Deserialisering - Hva?

- Vi ønsker å gjøre om dataformat fra JSON / XML til **data class** i **Kotlin**
  - Deserialisering er denne prosessen



https://hazelcast.com/glossary/serialization/

Deserialisering - JSON til Kotlin

```json
{
    "persons": [
        {
            "name": "Lars",
            "age": 28
        },
        {
            "name": "Hege",
            "age": 25
        },
        {
            "name": "Zandra",
            "age": 29
        }
    ]
}
```

```kotlin
data class Person(val name: String, val age: Int)

data class Persons(val persons: List<Person>)
```

api.met.no

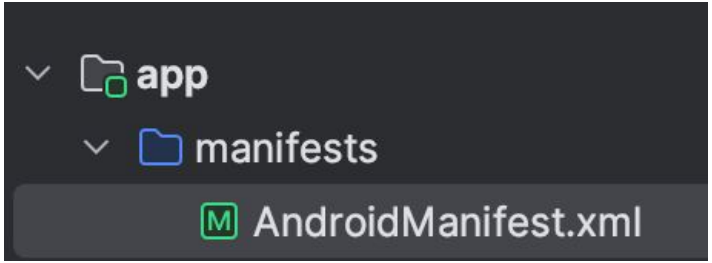
Menti: **4630 2772**

Hente fra API i Android Studio

Menti: **4630 2772**

Android Studio - Internet Permission

- Må gi prosjektet tilgang til internett i AndroidManifest.xml
- Gå til



- Og legg til:

```
<uses-permission android:name="android.permission.INTERNET" />
```

utenfor <application>-tagen

Android Studio - Biblioteker for HTTP-requests

- Må bruke en HTTP-klient (bibliotek)
  - Ktor, Retrofit, Volley, Fuel, okhttp
- Hva er viktig når vi ser etter et bibliotek?
  - Finn ut om biblioteket støtter det du ønsker å gjøre
    - Sjekk ut dokumentasjonen til biblioteket
  - Er det godt vedlikeholdt?
    - Sjekk GitHub - hvor mange brukere er det? Hvor ofte oppdateres det? Se under “issues” tabben.

Ktor client

Skilt ut i mange forskjellige “plugins” som gjør at vi kan ta den funksjonaliteten vi vil ha, f.eks.:

- Client
- Engine (motor som driver selve http-kallene)
- Content Negotiation
- Serialization

API Engine/motor

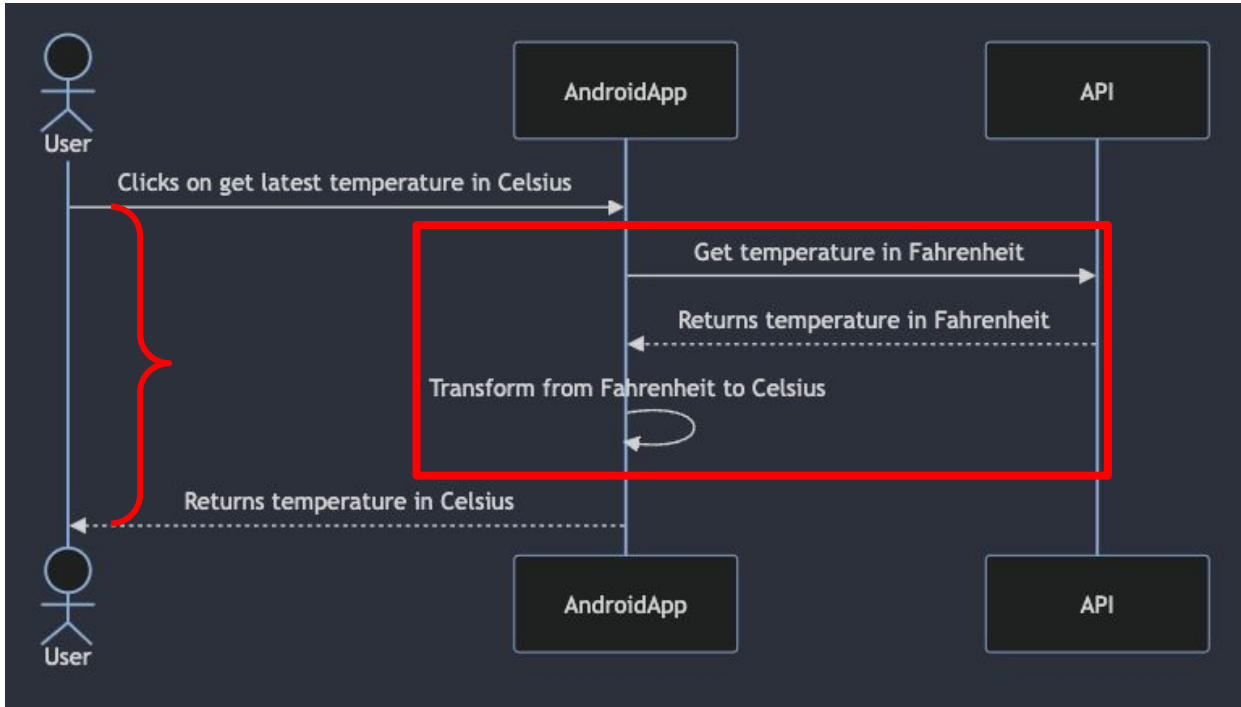
- Android: https://ktor.io/docs/http-client-engines.html#android
- OkHttp: https://ktor.io/docs/http-client-engines.html#okhttp

Hvorfor samtidighet og parallellitet? - Recap

Se for deg:

En bruker interagerer med AndroidApp som henter data fra et API og gjør en beregning.

Vi ønsker IKKE at appen skal "henge" når appen gjør oppgaver i bakgrunnen



Coroutines - Recap

- Avbrytbar funksjon (suspendable)
- Funksjoner som tillater asynkron eksekvering av kode
- Coroutine builders
  - **launch** starter uten å forvente en returverdi
  - **async** starter, men gir tilbake en lovnad om at det til slutt skal bli en verdi

https://kotlinlang.org/docs/coroutines-overview.html

Kotlin Coroutines

Coroutines - Dispatchers

- Dispatchers bestemmer hvilke(n) tråd(er) Couroutinen skal bruke

Dispatcher	**Description**	**Uses**
Dispatchers.Main	Main thread on Android	- Calling suspend functions - Call UI functions - Update LiveData
Dispatchers.IO	Disk and network IO*	- Database - File IO - Networking
Dispatchers.Default	CPU intensive work	- Sorting a list / other algorithms - Parsing JSON - DiffUtils

Coroutines - Dispatchers (forts.)

```
class LoginViewModel(
    private val loginRepository: LoginRepository
): ViewModel() {

    fun login(username: String, token: String) {
        val jsonBody = "{ username: \"$username\", token: \"$token\"}"
        loginRepository.makeLoginRequest(jsonBody)
    }
}
```

https://developer.android.com/kotlin/coroutines/

Coroutines - Dispatchers (forts.)

```
class LoginViewModel(
    private val loginRepository: LoginRepository
): ViewModel() {

    fun login(username: String, token: String) {
        // Create a new coroutine to move the execution off the UI thread
        viewModelScope.launch(Dispatchers.IO) {
            val jsonBody = "{ username: \"$username\", token: \"$token\"}"
            loginRepository.makeLoginRequest(jsonBody)
        }
    }
}
```

https://developer.android.com/kotlin/coroutines/

Coroutines - Dispatchers (forts.)

```
suspend fun fetchDocs() {                                // Dispatchers.Main
    val result = get("https://developer.android.com") // Dispatchers.IO for `get`
    show(result)                                         // Dispatchers.Main
}

suspend fun get(url: String) = withContext(Dispatchers.IO) { /* ... */ }
```

https://developer.android.com/kotlin/coroutines/coroutines-adv

viewModelScope.launch

```
class MyViewModel: ViewModel() {
    init {
        viewModelScope.launch {
            // Coroutine that will be canceled when the ViewModel is cleared.
        }
    }
}
```

- Namage
- WizApp

Avslutningsvis…

“Ikke benytt dere av API-er der en må legge inn bankkort” 💀