

## Intro HTTP

Tomas Holt, Institutt for informatikk og e-læring ved NTNU

Opphavsrett: Tomas Holt og Stiftelsen TISIP

Lærestoffet er utviklet for faget IFUD1042 Applikasjonsutvikling for mobile enheter

*Resymé: Denne leksjonen omhandler hvordan vi kan bruke nettverk til å kommunisere med omverden. Her er fokus på HTTP/Web, mens vi i en senere leksjon vil se på andre muligheter.*

## Innhold

6.1. OM DENNE LEKSJONEN.....	0
6.2. KOMMUNIKASJON MOT OMVERDEN.....	1
6.3. NØDVENDIGE RETTIGHETER.....	1
6.4. HTTP (HYPERTEXT TRANSFER PROTOCOL).....	1
6.4.1. Fra klient til tjeneren.....	1
6.4.2. Fra tjener til klient.....	3
6.4.3. GET- eller POST-forespørsler fra klienten?.....	3
6.4.4. Data fra tjener og strømmer.....	5
6.4.5. Mer om URL-omforming.....	5
6.4.6. Tilstander i HTTP.....	5
6.5. EKSEMPELKODE.....	6
6.6. VEDLEGG: HTTP-KODER.....	6
6.6.1. Statuskoder.....	6
6.7. VEDLEGG: MIME-TYPER.....	7
6.8. REFERANSER.....	7

### 6.1. Om denne leksjonen

Leksjonen omhandler hvordan man kan bruke nettverkskommunikasjon via HTTP. Leksjonen er frittstående, dvs. Det er ikke noen henvisning til læreboka.

Merk at det er flere muligheter for hvordan man lager HTTP-forbindelser. Boka viser hvordan det kan gjøres og eksempelkode vedlagt leksjonen viser flere alternative måter å gjøre dette på.

Eksempelkode vil være et godt utgangspunkt for selv å lage nettverksløsninger og å løse oppgaven til øvingen.

Merk at det tidligere (< android versjon 10) var mulig å lage I/O-kode (f.eks. nettverksoppkoblinger) uten bruk av tråder. Dette er ikke lenger mulig. Det er uansett en dårlig praksis. Leksjonen krever derfor at man vet hvordan man bruker tråder (noe som er tatt opp i en egen leksjon).

I tillegg til leksjonen så finner du mye god informasjon på [Android nett]. Der kan man blant annet finne informasjon om hvordan sjekke nettverksstatus (f.eks. om nettverk er tilgjengelig)

og ikke minst finner man informasjon om hvordan man kan hente og prosessere XML-data fra nett. Prosessering av XML-data er meget relevant, men man ser vel også at JSON-formatet blir mer og mer aktuelt.

## 6.2.Kommunikasjon mot omverden

Android tilbyr kommunikasjon mot omverden via blant annet HTTP (Hypertext Transfer Protocol). Du finner denne protokollen beskrevet i neste kapittel. Man er imidlertid ikke begrenset til å bruke denne protokollen, f.eks. er *sockets* en mulighet for kommunikasjon mot andre maskiner. Dette gir mindre begrensninger enn kommunikasjon over HTTP og kan i en del tilfeller være å foretrekke da det også gir mer kontroll. I tillegg finnes også støtte for HTTPS, noe som gir sikre overføringer over HTTP via at dataene krypteres. I denne leksjonen skal vi imidlertid nøye oss med å se på HTTP.

Det er mange forskjellige muligheter når man skal lage nettverksapplikasjoner. Vi vil i dette kurset fokusere på det som foregår på klienten (altså den mobile enheten). Andre fag som for eksempel "Web-applikasjoner med Java EE" og "Java EE og distribuerte systemer" er fag som kan være aktuelle i forbindelse med programmering av det som skal skje på tjeneren.

## 6.3. Nødvendige rettigheter

Når man skal lage nettverksapplikasjoner i Android så må man sørge for å gi nødvendige rettigheter. Dette gjøres ved å legge til

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

i AndroidManifest.xml. Toppen på fila kan da se slik ut:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="leksjon.http"
    android:versionCode="1"
    android:versionName="1.0">
```

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

.....

## 6.4.HTTP (Hypertext Transfer Protocol)

HTTP er mer eller mindre utelatt i læreboka, men bør være med her for forståelsens skyld.

### 6.4.1.Fra klient til tjeneren

HTTP-protokollen brukes blant annet i forbindelse med web. Når du sitter og surfer på Internett (web) bruker altså nettleseren HTTP for å kommunisere med tjeneren rundt omkring i verden. Når du enten trykker på en lenke eller skriver noe i adressefeltet så vil nettleseren sette opp en forbindelse mot tjeneren. Nettleseren vil så sende tekststrenger til tjeneren.

Hver gang vi skriver inn en URL i adressefeltet i nettleseren eller trykker på en hyperlenke så sendes en GET-forespørsel til tjeneren (dette er en måte å spørre etter en ressurs på). Det er slik at hver gang nettleseren kaller GET (eller POST) mot tjeneren så vil det sendes med informasjon om klienten. Dette gjøres i et HTTP-hode. La oss si at vi vil ha tak i en fil med følgende URL:

<http://aitel.hist.no/fag/index.html>

Første delen av denne URL'en spesifiserer at vi bruker HTTP. Neste del spesifiserer maskinen (aitel.hist.no) som har denne filen. Denne maskinen har en tjener (dette er altså et program) som venter på forespørsler. Denne tjeneren mottar nå flere linjer som resultat av vår GET-forespørsel. Første linje kan være som følger:

```
GET /fag/index.html HTTP/1.1
```

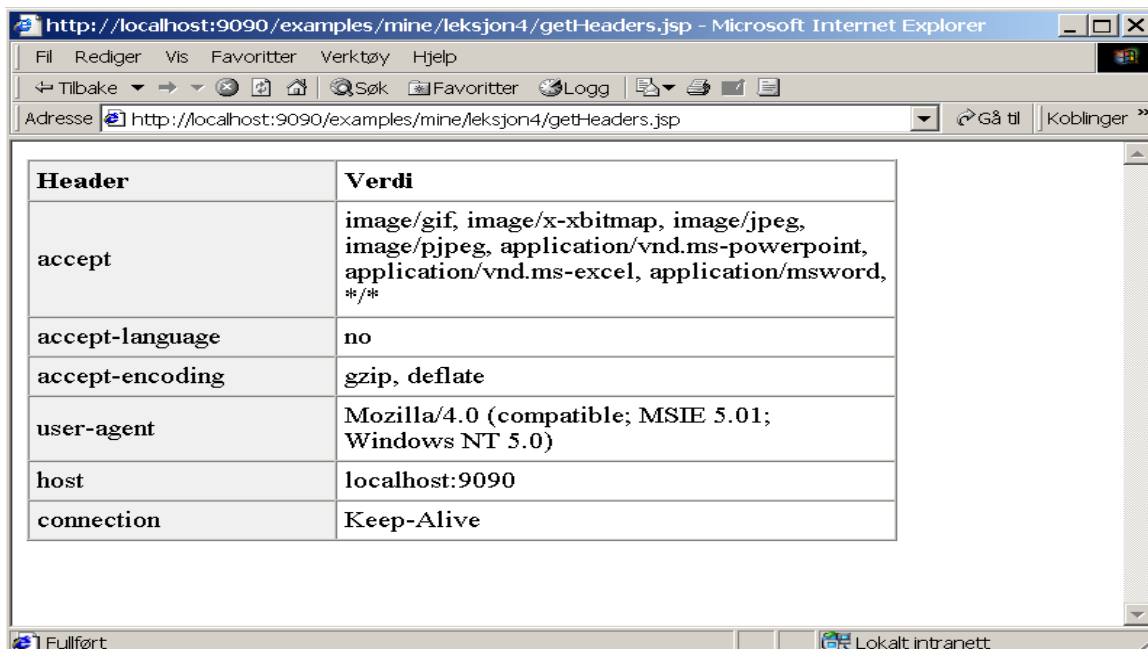
Vi ser av linjen over at tjeneren ikke mottar hele URL'en. Maskinnavnet kommer som en egen del av forsendelsen (under parameteren *host*). Tjeneren mottar hvilken type forespørsel det er (GET) og så hvilket dokument det er snakk om (*fag/index.html*). Til slutt på linjen kommer hvilken protokoll det er snakk om og hvilken versjon av protokollen. Tjeneren må sørge for ikke å bruke en mer avansert protokoll en det klienten kan bruke. Hvis f.eks. tjeneren nå sender et svar tilbake med HTTP/1.2 så vil klienten få problemer fordi dette ikke støttes.

HTTP-hodet inneholder imidlertid mer informasjon, og kan f.eks. være slik:

```
GET /fag/index.html HTTP/1.1
Host: aitel.hist.no
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Accept: text/html, image/gif, image/jpeg, audio/x
Accept-Encoding:gzip
```

*Host* beskriver her hvilken maskin forespørselen er sendt til (altså maskinen som kjører web-tjeneren). *User-Agent* beskriver hvilken klient vi gjør forespørselen fra (her er nettleseren Mozilla versjon 4). Til slutt ser vi at nettleseren sender med hvilke formater den kan akseptere tilbake (*Accept-feltet*).

*Accept-Encoding* refererer til hvilke komprimeringsteknikker klienten støtter. Under ser du en utskrift av HTTP-hodet når en nettleser kobler seg til en web-tjener (web-tjeneren skriver altså ut HTTP-hodet som kommer fra klienten).



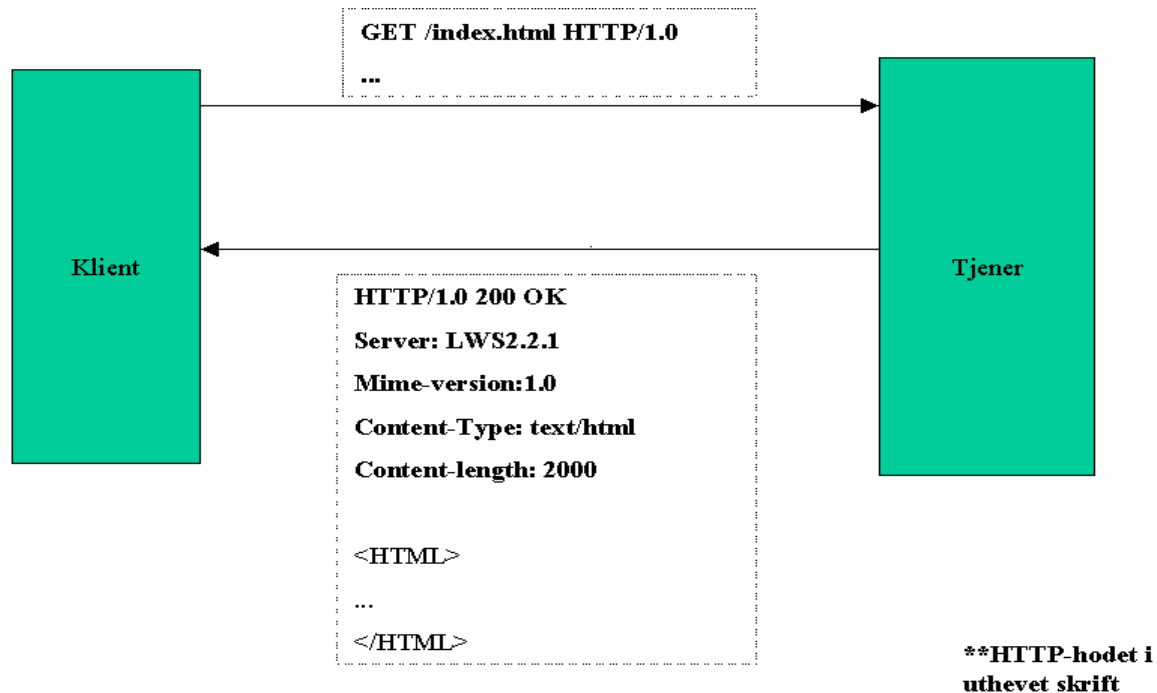
Header	Verdi
accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
accept-language	no
accept-encoding	gzip, deflate
user-agent	Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
host	localhost:9090
connection	Keep-Alive

Figur 1: HTTP-hodet i en forsendelse fra klienten til tjeneren

På adressen <http://tomcat.stud.iie.ntnu.no/studtomas/ekko.jsp> har du muligheten til selv å teste din egen nettleser og din mobilapplikasjon (som du skal lage etterhvert).

### 6.4.2. Fra tjener til klient

Forespørselen fra klienten vil inneholde informasjon som forklart i forrige delkapittel. På grunnlag av informasjonen som tjeneren mottar i denne HTTP-forespørselen (HTTP-hodet), sendes et svar tilbake til klienten. I de fleste tilfellene vil dette svaret inneholde et HTTP-hode, samt noe mer. Dette kan være HTML-kode som beskrevet i figuren under, eller det kan være andre dataformater som video/mpeg osv.



Figur 2: HTTP-forespørsel og svar

HTTP-hodet som sendes fra tjeneren kan være som vist i figuren over. Den første linjen beskriver hvilken protokoll (HTTP/1.0) som brukes, samt en kode (200), og at alt er ok. Koden som står etter protokollen er avhengig av utfallet av forespørselen. Tallet 200 indikerer at alt gikk fint. For mer informasjon om disse statuskodene se kap. 6.6.

I HTTP-hodet vil det være et felt som heter *Content-Type*. Dette feltet beskriver MIME-typen (*Multipurpose Internet Mail Extension*) på etterfølgende dataformat. MIME er altså bare en standard måte å beskrive hvilket format innholdet i forsendelsen er på, se kapittel 6.7 for mer informasjon. I figuren over er denne *text/html*, og klienten vet dermed at det som kommer etter HTTP-hodet er HTML-kode. HTTP-hodet er alltid atskilt fra resten med en blank linje. Feltet *Content-Type* gir altså klienten mulighet til å behandle dataformatet som kommer på riktig måte, mens *Content-length* angir hvor mange byte innholdet er på.

### 6.4.3. GET- eller POST-forespørsler fra klienten?

Når en klient gjør en GET forespørsel til tjeneren kan for eksempel URL'en bli slik:

<http://tomcat.stud.iie.ntnu.no/studtomas/ekko.jsp?fornavn=Tomas&etternavn=Holt>

Du finner et spørsmålstegn inne i denne strengen. Etter dette kommer navn og verdi på parametere som vi sender til tjeneren som en del av forespørselen (vi sender altså en forespørsel om at tjeneren skal returnere ekko.jsp, men i tillegg så sender vi med fornavn og etternavn). Disse er i par, atskilt med = tegn. Hvert slikt par skilles fra de andre parene med

tegnet &. Du kan for eksempel se at en parameter heter fornavn og har verdi Tomas. Tjeneren kan så bruke disse parametrene til for eksempel å sjekke hvem brukeren er.

URL'en over inneholder altså parametrene fra klienten. Dette er kun tilfelle når vi bruker **GET** som overføringsmetode. Det finnes også en annen måte til å sende parametere i HTTP. I dette tilfellet bruker vi en POST-forespørsel (dette er veldig likt GET). I dette tilfellet sendes ikke parametrene som en del av URL'en, men i en egen innpakning (og vil ikke vises i URL'en).

Å lage HTTP forespørsler i Android er forholdsvis enkelt, se vedlagt eksempelkode.

**Merk!** Når man sender parametere i en GET forespørsel så er det ikke lov å sende blant annet særnorske tegn og blanke tegn (f.eks. mellomrom). Det gjør at om vi hadde prøvd

```
String url = " http://tomcat.stud.iie.ntnu.no/studtomas/ekko.jsp?fornavn=Kåre  
Andre";
```

så får vi problemer. I eksempelkoden kan du se bruken av metoden `encodeParameters()` for å håndtere dette problemet.

Eksempelkoden viser også hvordan POST-forespørsler lages. Hvorfor vil man så bruke POST i stedet for GET? Saken er at GET forespørsler ofte har begrensning på hvor mange bytes som kan overføres. Dette gjelder imidlertid ikke for POST-forespørsler. Skal man sende større datamengder må man derfor velge POST.

#### 6.4.4.Data fra tjener og strømmer

Du vet nå hvordan du kan sende GET- og POST-forespørsler til tjeneren. Hvordan tar vi så i mot data fra tjeneren? Hvis du ser på koden fra forrige delkapittel vil du se denne linjen:

```
String responseBody = client.execute(request,responseHandler);
```

Denne kodelinje sørger for å fylle `responseBody` med alt – untatt HTTP-hodet - som kommer fra tjeneren. Dette er altså en veldig enkel måte å få tilgang til innholdet på. Vanligvis er det tekst som returneres og da vil denne måten å gjøre det på være grei. Det kan imidlertid være situasjoner hvor man ønsker mer kontroll. Det kan f.eks. hende at man vil lese linje for linje (slik at man kan fylle skjermen etterhvert som data kommer fra tjeneren) eller at det ikke er tekst som kommer.

Jeg har laget klassen `HttpWrapper` som hjelp for nettverkskommunikasjon via HTTP. Denne klassen har tre nettverksmetoder (`httpGet()`, `httpPost()` og `httpGetWithHeader()`). De to første bruker standard GET og POST som forklart over. Den siste bruker også GET, men den sørger både for å lese ut HTTP-hodet, samt at den leser det som kommer tilbake linje for linje. Sistnevnte gjøres gjennom å åpne en innkommende strøm mot tjeneren (`BufferedReader in`). Mao. data fra tjeneren kommer via denne strømmen.

#### 6.4.5. Mer om URL-omforming

Som beskrevet tidligere har man begrensninger på hvilke tegn som kan sendes ukodet via HTTP. Standarden for URL'er (URI'er) har nemlig bare et sub-sett av ASCII-tabellen som lovlig tegn. De lovligte tegnene er det engelske alfabetet (store og små bokstaver), tall (0-9), samt punktum (.), stjerne (\*), bindestrek (-) og underscore (\_). Hvis man skal sende andre tegn enn dette (f.eks. mellomrom, eller særnorske tegn) så må tegnet omkodes (eng; url encoding)! Mer informasjon kan du finne på [URL Encoding].

#### 6.4.6. Tilstander i HTTP

HTTP har en begrensning som virker merkelig for de fleste. I utgangspunktet har nemlig ikke protokollen noen løsning for å huske tilstander. Det vil si at hver gang det kommer en forespørsel fra en klient til tjeneren så vil ikke tjeneren ha noen formening om tidligere forespørsler. Her er et eksempel på hvorfor det er problematisk: Du skal lage en kalkulatorløsning. Den skal fungere slik at du først sender et tall til tjeneren (via HTTP GET eller POST) så sender du det andre tallet i en ny forsendelse (ny HTTP GET eller POST). Tjeneren skal så legge sammen tallene og returnere svaret. Problemet er at når du sender det andre tallet så er allerede det første tallet glemt av tjeneren! Akkurat samme problem oppstår f.eks. ved bruk av handlekurv på et nettsted. Tjeneren er i dette tilfellet nødt til å huske innholdet i handlekurven. Så hva gjør vi?

Løsningen man har laget på problemet er kalt cookies og sesjoner. Sistnevnte er ( gjerne ) et påbygg på den første. Cookies fungerer på den måten at man utvider HTTP-hodet med et ekstra felt. Dette feltet kan inneholde en tekststreng. Når tjeneren ønsker å huske en tilstand vil den bruke dette feltet. Typisk vil tjeneren lagre en id i dette feltet som er unik for hver klient. Hvis klienten nå sender med denne id'en i etterfølgende forespørsler så har tjeneren mulighet til å kjenne igjen klienten. Det gjør det mulig for tjeneren å huske hva som har skjedd i tilknytning til klienten tidligere (f.eks. kan den huske et tall fra forrige forespørsel). Du kan selv se din sesjonsid om du bruker nettleseren din mot <http://tomcat.stud.aitel.hist.no/studtomas/ekko.jsp>.

For å støtte sesjoner i Android-applikasjoner har vi to valg. Vi kan enten sørge for å lese sesjonsid'en som tjeneren tildeler oss (og som vi finner i HTTP-hodet) og sørge for å sette denne i HTTP-hodet når vi gjør en ny forespørsel til samme tjener. Det er den tungvinte måten å gjøre det på. Den enkle er å bruke CookieHandler. Du kan selv se hvordan dette er gjort i HttpWrapper-klassen.

### 6.5. Eksempelkode

Du bør nå se på eksempelkoden i HttpWrapper og HttpActivity. Koden er satt opp til å kommunisere med ekko.jsp som igjen sørger for å returnere alt som kommer fra klienten tilbake igjen. Dette gir gode muligheter til å teste hvordan dette fungerer selv. Du kan selv velge via menyen hvilken av nettverksmetodene du vil bruke.

Legg også merke til at det ved hver utveksling opprettes en ny tråd i HttpWrapper – via den egenlagde interne klassen RequestThread.

### 6.6. Vedlegg: HTTP-koder

I dag er det mest vanlig å bruke HTTP-versjon 1.1. Du kan finne all informasjon om HTTP-kodene [HTTP koder].

Under finner du en grov oversikt over de ulike statuskodene i HTTP.

#### 6.6.1. Statuskoder

Kodene er delt opp i 5 grupper:

Kode	Kommentar
100-199	Informasjon, indikerer at klient bør gjøre ett eller annet.

200-299	Indikerer at forespørsel har gått bra
300-399	Brukt for filer som har skiftet plass, ofte med ny adresse
400-499	Feil av klient
500-599	Feil av tjener

Det finnes mange spesifikke koder innen disse gruppene. Kode 200 indikerer f.eks. at forespørsel gikk fint. Kode 201 derimot indikerer at tjeneren har opprettet et nytt dokument i respons til en forespørsel, og at dette gikk bra. Det er for mange koder til å gå inn på her, men det er noen du nok kommer til å se (og muligens allerede har sett).

Kode	Kommentar
400	Dårlig forespørsel. Indikerer syntaksproblemer i forespørsel fra klient.
401	Ikke autorisert. Klient prøver å få tak i side uten rettigheter til det.
403	Ikke lov. Tjener nekter å returnere ressurs uansett rettighet på klient. Indikerer ofte problemer med fil/katalog adgang på tjeneren.
404	Ikke funnet. Ressursen kan ikke finnes.
500	Intern tjenerfeil. Kommer ofte fra servlet'er og andre tjener-programmer som krasjer.
503	Tjeneste ikke tilgjengelig. Kan være pga. overbelastning på tjeneren.

## 6.7.Vedlegg: Mime-typer

MIME (Multipurpose Internet Mail Extension) er en måte å beskrive innholdet i et dokument, dvs. på hvilket format det er. Tenk det hvis du skal lage en applikasjon som kan ta i mot HTML-kode og Word-dokumenter. Applikasjonen gjør en forespørsel til en tjener som hvis den har et word-dokument tilgjengelig vil returnere det, hvis ikke returneres et HTML-dokument (vær overbærende med dårlig eksempel ;-) For at klienten nå skal kunne vise dokumentet på riktig form så må man vite om det er Word- eller HTML-format som kommer i responsen. Mime-feltet i HTTP-hodet forteller klienten dette, men for at dette skal fungere må vi ha en standard. Det hjelper ingenting om en tjener beskriver word-dokumentet som *word*, hvis en annen tjener beskriver det med *microsoft-word*, mens igjen en tredje sier *application/word*. Som programmerer av klienten vil dette være helt uholdbart. Man har derfor standardisert dette. Et word-dokument beskrives med strengen *application/msword*, mens HTML-kode beskrives med *text/html*.

En oversikt over MIME-typer kan du finne på [MIME] (merk at det hele tiden kommer nye formater):

## 6.8.Referanser

[Android nett] - <http://developer.android.com/training/basics/network-ops/index.html>  
 [HTTP Koder] - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>  
 [MIME] - <http://www.utoronto.ca/webdocs/HTMLdocs/Book/Book-3ed/appb/mimetype.html>

[URL Encoding] – <http://www.blooberry.com/indexdot/html/topics/urlencoding.htm>