Webutvikling og API-design Dokumentasjon for innlevering

# Introduksjon

Eksamen går ut på å lage en webapplikasjon og mobilapplikasjon som inneholder en liste med noe av våre interesser. Jeg har valgt å lage en liste med filmer.

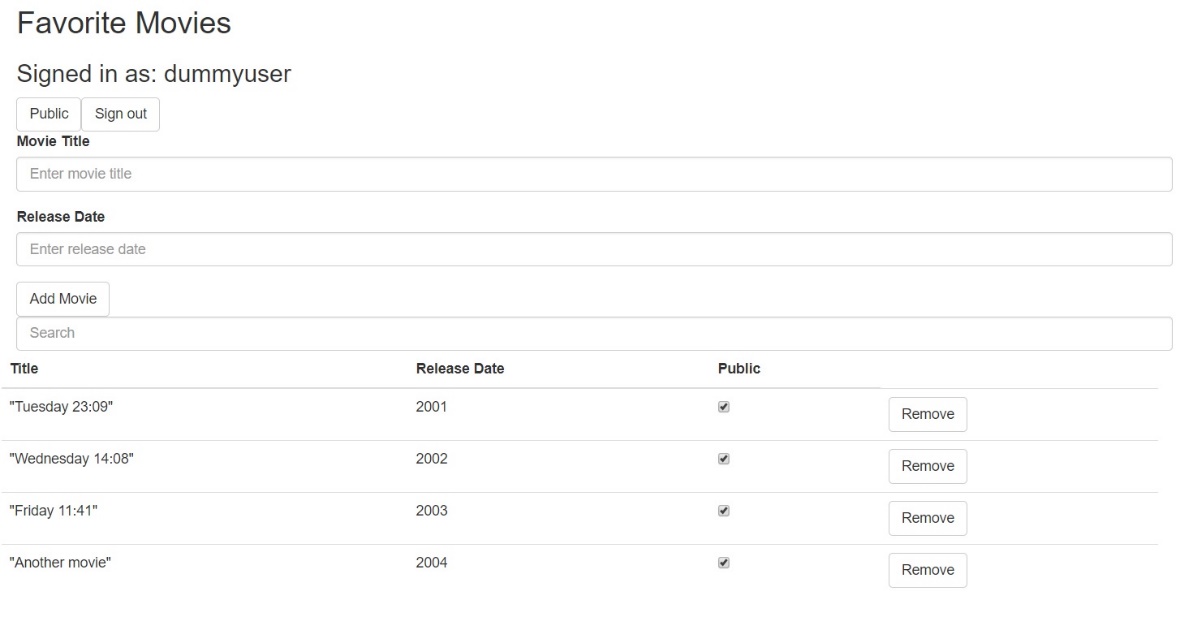
Eksamen skal utføres med bruk av MERN (MongdoDB, Express, React, Node.js) stacken.

# Skjermbilder av web løsning

Dette er startsiden (root) som inneholder mulighet for å registrere og logge inn bruker



Dette er siden til den innloggede bruker som vil inneholde filmer bruker selv har lagt til

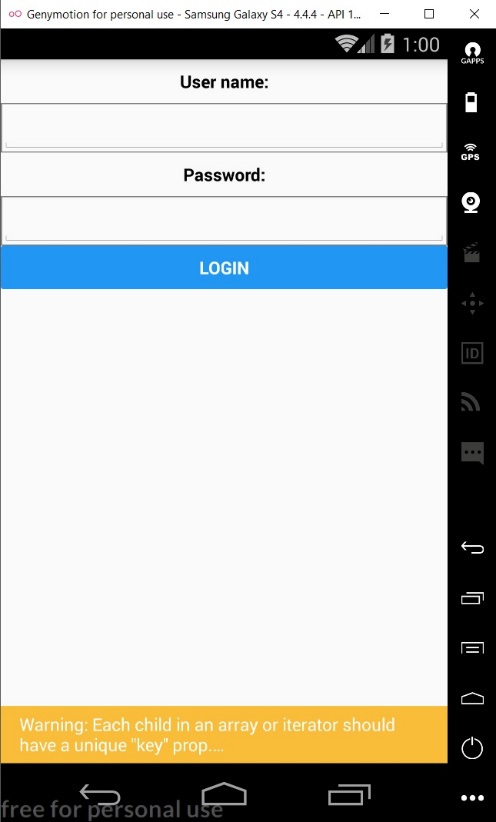
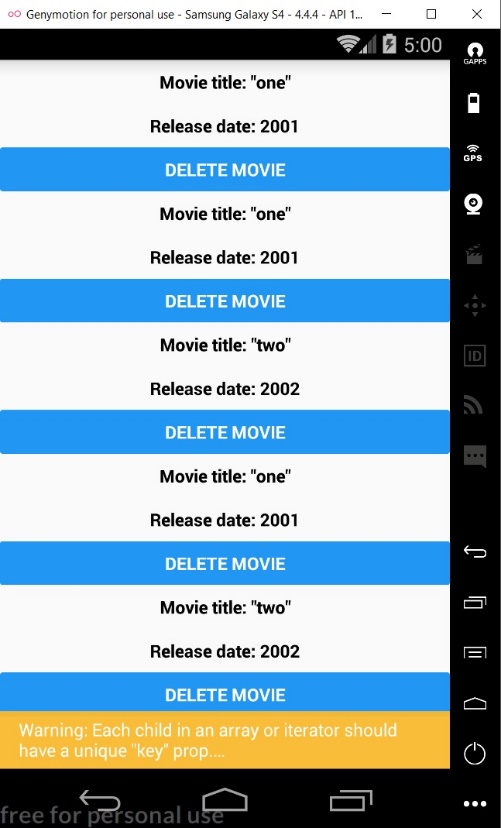


Dette er siden som inneholder en top 10 liste av de siste filmene som er lagt til og gjort «public»



# Skjermbilder av mobil løsning

Mobil applikasjonen inneholder kun 2 sider. Startsiden (root) vil inneholde mulighet for å logge inn eksisterende bruker som vil ta deg til siden som vil inneholde en liste med brukerens filmer

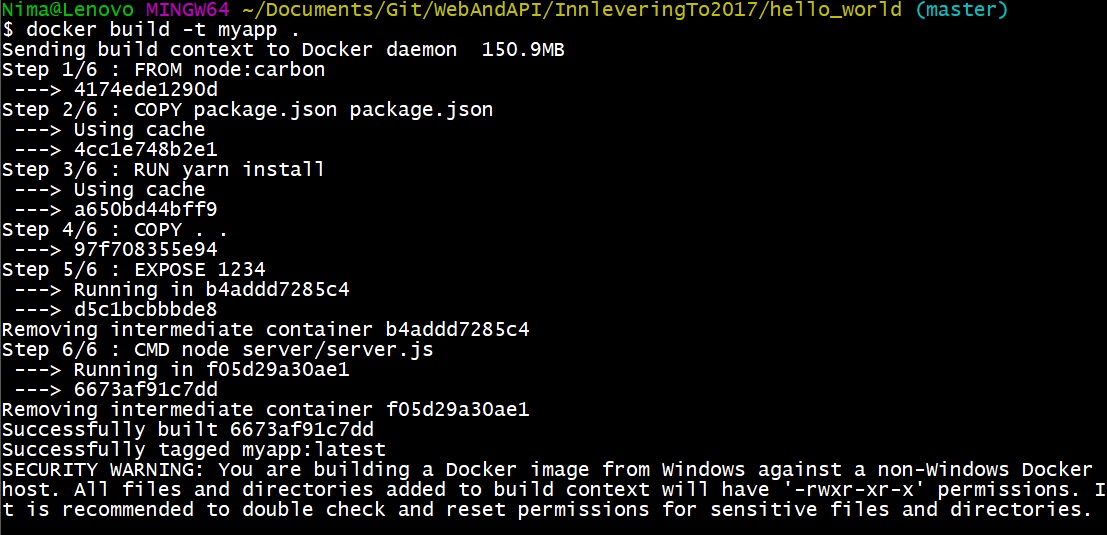
# Valg og forutsetninger

* For denne innleveringen har jeg fulgt en MERN tutorial som baserer seg på facebook sin egen tutorial. Valg og bruk av kode er gjort ut i fra eksempler funnet der og andre dokumentasjon sider i tillegg til eksempler fra forelesning. Kilder legges med som vedlegg.
* Jeg har flyttet encoding av password fra server til client for at passord ikke skal lagres eller sender i plain tekst.
* Måten jeg har satt opp public funksjonen er at det sendes inn et http kall til server hver gang en film krysser av og på for public. Dette kan kanskje være litt for mye når applikasjonen blir større. Tanken bak er at andre brukere skal få en så oppdatert top ten liste som mulig. Første måten jeg tenkte på var å gjøre et kall med alle forandringene kun en gang når bruker logger ut.

# Instruksjoner

* For å kjøre applikasjonen trengs det internett forbindelse for nedlastning av pakker og få bootsrap til å fungere.
* Når jeg skal kjøre applikasjonen lokalt kjører jeg kommandoene «node server/server.js» og «yarn start» i hver sin terminal i samme mappe som applikasjonen. For mondodb har jeg brukt ekstern database slik at eksempel brukere og filmer skal være tilgjengelig. Hvis dette av en eller annen grunn ikke skulle fungere har jeg satt opp i hello\_world/server/server.js to variabler med samme navn som bytter mellom lokal og ekstern database. Kommentere vekk linje 11 og fjerne kommentar fra linje 12 vil ordne dette.
* Forutsetningene for testene skal kjøres er at det er minst en bruker i databasen og at databasen kjører.
* For docker kjører jeg disse kommandoene for å starte node serveren, i git bash fra hello\_world mappen (docker må være startet for at det skal fungere)

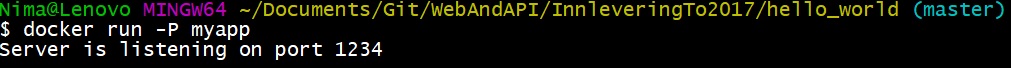
1. docker build -t myapp .



1. docker run -p 1234:1234 -d myapp



1. docker run -P myapp



Jeg måtte legge til steg 2 siden jeg ikke fikk til at applikasjonen skal bruker porten docker gir selv. Lagt ved kilde for dette oppsette i egen fil.

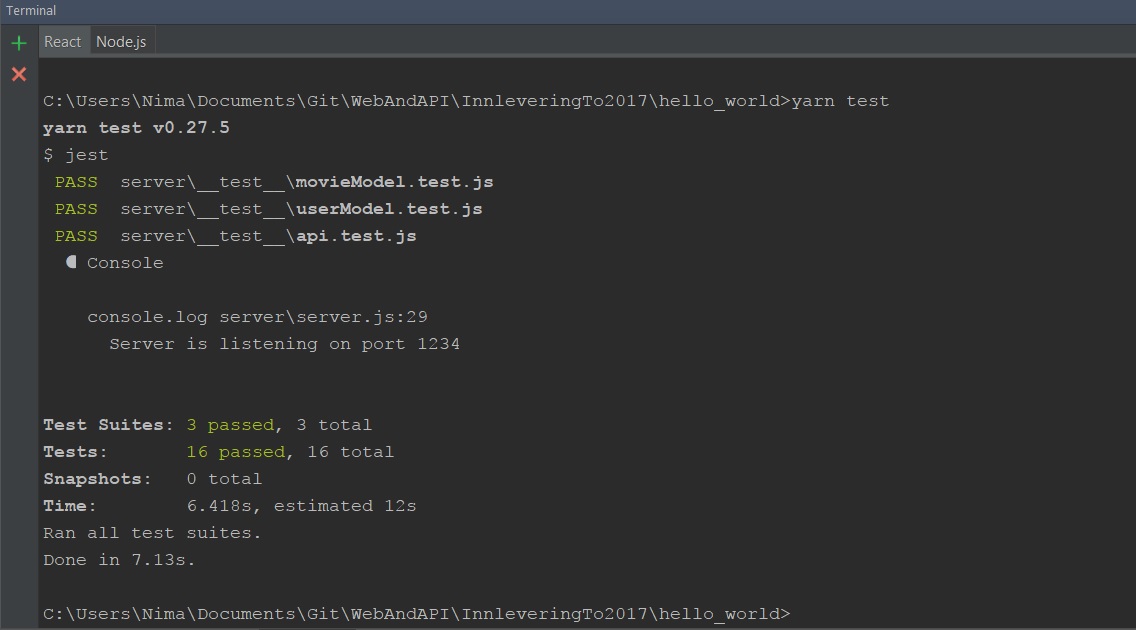
* I mobilapplikasjonen fungerer det heller ikke å bruke localhost, det må skrives inn IP adressen. Den må skiftes i hello\_world/moviePhone/config/config.js og redigere nativeDatabaseURL feltet
* Det er en package.json fil i hello\_world/package.json og en i hello\_world/moviePhone/package.json. Jeg hadde i utgangpunktet en egen package.json fil for server og react komponenter i tillegg til de førstnevnte for bedre struktur men det oppstod problemer som gjorde jeg måtte ta dem bort.
* Det er to dummy bruker du kan bruke hvis det skulle trenges.   
  Bruker 1 -> username: dummyuser password: password   
  bruker 2 -> username: userone password: password
* Applikasjonen er testet med Android versjon 4.4.4 API 19 (Samsung Galaxy S4)

# Valg av URL-struktur og http-verb

Når det gjelder valg og struktur av URL og http verb har jeg prøvd å følge retningslinjene til REST. URL er satt opp slik at det gjenspeiler hva slags data de jobber med, URL til filmer er satt til movies og users for brukere. URL’en til top\_ten\_movies kan være litt misvisende siden henter alle filmene som er public, men tanken bak var det er den som brukes for å jobbe med top ten listen. Det er også lagt vekt på å bruke substantiv i flertalls form istedenfor verb. Når det gjelder http-verbene har jeg til denne eksamen brukt GET, POST, DELETE og PATCH.   
GET er brukt når jeg skal hente data fra databasen.   
POST er brukt for legge til brukere og filmer til databasen. Den er også brukt for innlogging siden det skal opprettes en token (ressurs) og sendes tilbake.  
DELETE er brukt for sletting av filmer og brukere.  
PATCH ble brukt istedenfor PUT siden jeg skulle kun oppdatere et felt og ikke erstatte hele objektet.

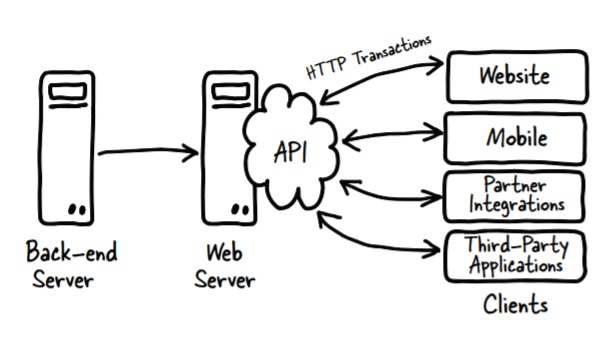
# Testing

Jeg har valgt å fokusere testingen mer mot node serveren og ikke React komponentene. Tanken bak det er siden bruker opplevelse ikke er satt i fokus på denne eksamen. Jeg har satt opp mock testing for skjemaene som er opprettet for filmer og brukere til MongoDB. Jeg startet med dette for jeg prøvde å sette opp all testing mot en mock database. Når det kom til testing av Node serveren fikk jeg ikke til mocking av databasen og måtte ha databasen kjørende mens jeg testet. Resten av testene er gjort med api kall til serveren. Det er ikke gjort tester på kallene som krever token autentisering siden jeg ikke fant ut hvordan man kommer seg rundt denne. Det er kun gjort et kall som skal feile siden token ikke er satt.



# Hva er et (web) API

API står for Application Programming Interface og er et bibliotek av funksjoner/metoder som hvis skrevet godt nok gjør utvikling av, i vårt tilfelle, web applikasjoner mindre komplisert. Dette er funksjoner som utviklere bruker ofte og vil være skrevet ferdig og optimalisert for oss. Web API vil også basere seg på http kommunikasjon. API’er kan gjøre det enklere og utvikle til mobil applikasjoner ved at det brukes samme API for utvikling til vanlig nettleser og mobil.



Fordeler og ulemper ved bruk av et API

* Bra praksis med tanke på gjenbruk av kode
* Enklere og utvikle til mobil applikasjoner
* Abstraherer vekk logikk
* Krever mer arbeid å utvikle et eget API, bør sjekke om det lønner seg

Når det ikke lønner seg å lage et web API

Når det kommer til om man trenger et web API kommer det også an på størrelsen til applikasjonen. Er det snakk om et lite privat prosjekt vil det kanskje ikke være nødvendig med et eget web API. Man kan man spørre seg om man vil engasjere andre utviklere til å bruke løsningen eller om det skal nåes frem til mobil enheter, hvis dette ønskes kan det være greit å vurdere og utvikle et eget API.

# ESLint

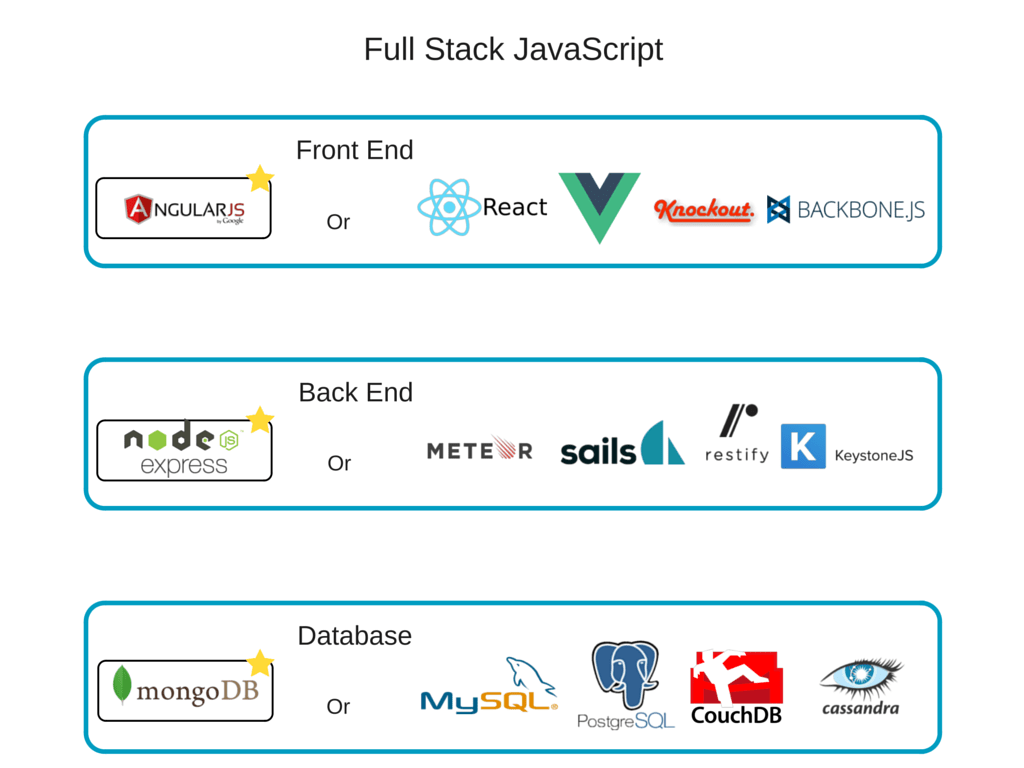
ESLint brukes for å kontrollere hvordan koden er skrevet i henhold til regler som blir satt. Reglene er mulig å tilpasse til eget ønske. Jeg brukte først et regelsett som er brukt av google også prøvde jeg å sette opp en egendefinert versjon. Varslene som kom fra ESLint gjorde i mitt tilfelle ikke at koden så bedre ut. Jeg bruker WebStorm som IDE og i tillegg til at den bruker egen lint verktøy har jeg satt opp egendefinerte regler for hvordan koden skal være så ESLint forandret for mye. ESLint kan være en bra løsning for noen som ikke har annet lint verktøy og ikke satt opp regler selv for hvordan koden skal se ut. Når det gjelder varsling for ubrukte variabler og importeringer kommer dette selv om ESLint ikke brukes. Etter dette ga jeg ESLint en sjanse til og bruker nå Airbnb sin kode standard. Det tok opptil en dag før fikk jeg satt opp alle reglene som trengtes. Jeg vil si hvis koden din skal brukes og jobbes med av andre vil ESLint være en god løsning for å gi samme utforming på kode, men er det et privat prosjekt føler jeg det blir mer bortkastet tid.

# JavaScript full stack development

JavaScript ble introdusert i 1995 og kom for å gi nettsider litt mer funksjonalitet. Det har nå blitt et veldig utbredt programmeringsspråk og kan brukes for lage komplekse web og desktopapplikasjoner. Siden Node.js ble introdusert i 2009 går det også å lage server siden med bruk av JavaScript som gjør det mulig for såkalt **‘full stack development’**. Full stack development i denne sammenheng menes at det skal være mulig å lage en komplett applikasjon med kun et programmeringsspråk for både front-end og back-end. Dette vil også gjøre det enklere å finne utvikler til prosjektet ditt siden samme person kan jobbe på front-end og back-end som også kan resultere i færre nødvendige ansatte. Det blir også god praksis for gjenbruk av kode med tanke på DRY når kode kan deles mellom hele gruppen. Siden JavaScript ble så enormt populært kom det også flere rammeverk som gjør JavaScript utvikling enklere, som for eksempel Facebook sin React løsning eller Github som kom med Electron. De to mest populære måtene å utvikle applikasjoner basert på kun JavaScript er med bruk av MEAN (**M**ongoDB, **E**xpress, **A**ngular, **N**ode.js) eller MERN stacken. Siden meste parten av verktøyene som lages til JavaScript er open source, er det nå mulig og laste ned over 450 000 pakker/moduler fra ‘Node Package Manager’ som gjør utvikling enda enklere.   
Men som med alle programmeringsspråk vil det være noen ulemper som må tas til betraktning. JavaScript vil ikke lønne seg hvis det vil være tunge kalkulasjoner som skal utføres i back-end. Utvikler må også lære seg flere områder ettersom det skal jobbes på alle elementer i hele stacken.

Fordeler og ulemper

* Samme kodespråk for front-end og back-end
* Kode gjenbruk
* Tilgjengelige rammeverk og moduler
* Begrenset back-end ressurser
* Fortsatt relativ ungt språk

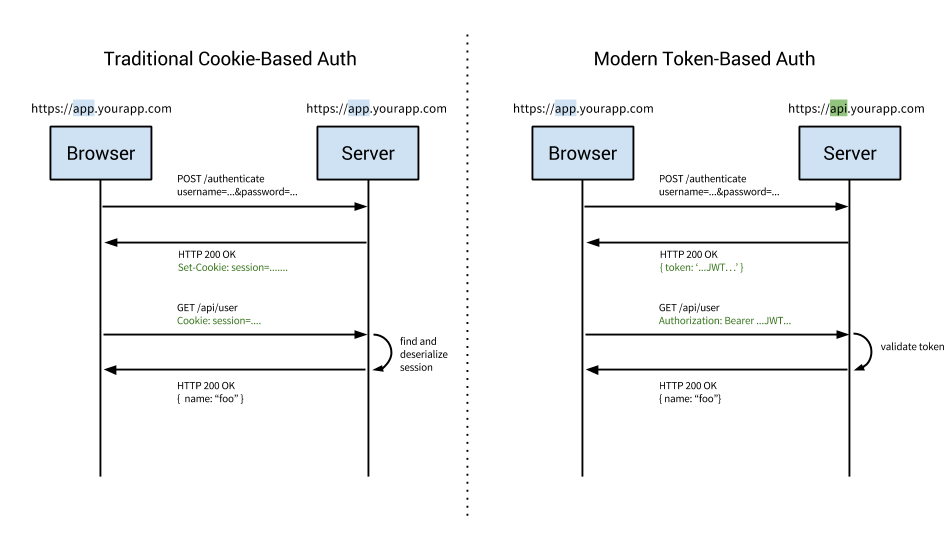


# JSON web-token

JWT er en alternativ til cookie brukt for autentisering, og har blitt veldig populært med introduksjonen av SPA (Single page app). JSON web token er blitt de facto standarden når det snakkes om token basert autentisering.

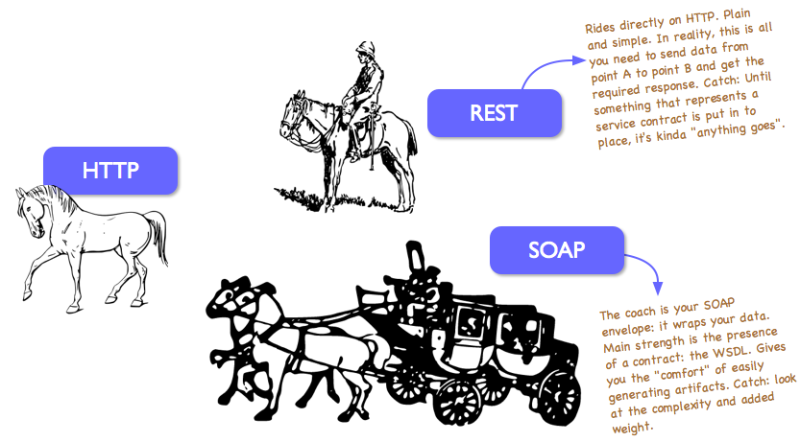
Fordeler og ulemper

* JWT i motsetning til cookie er stateless. Det vil si at det er kun klienten som trenger å kjenne til hvem har fått tilsendt token, noe som frigjør mye arbeid fra serveren. Siden server ikke trenger å kjenne til hvem som er autentisert sendes token med forespørslene til serveren, som da validerer token.
* JWT gjør det enklere å utvikle API på tvers av domener med hjelp fra CORS.
* I motsetning til cookie som inneholder en id, kan en JWT inneholde all data som er JSON godkjent.
* Siden server ikke trenger å gjøre spørringer for å sjekke om bruker er autentisert vil hastighet fordelene som kommer med tokens også være betydelig. I tillegg kan det lagres annen data i en token som brukerrettigheter som også gjør at server slepper enda en spørring.
* JWT gjør det også enklere og utvikle samme API til mobil.
* Siden JWT kan holde på all data du vil, kan størrelsen bli et problem hvis det utvikles for brukere med lite plass tilgjengelig.
* JWT blir signert og ikke kryptert som betyr at det ikke burde lagres sensitiv informasjon i en token. Hvis det måtte trenges har det kommet hjelpemidler for å kryptere tokens.
* Samme som med cookie vil det være fare for XSS og XSRF angrep, sistnevnte kan forebygges ved å lagre token i localstorage.



# REprensentational State Transfer

REST er en oppskrift (architectural style) for kommunikasjon, i vårt tilfelle, brukt til webutvikling og ikke en protokoll. REST bygger mer rundt http verbene og dens egentlige hensiktsmessige bruk. REST er ment for at avhengigheten mellom server og klient skal være så liten som mulig.

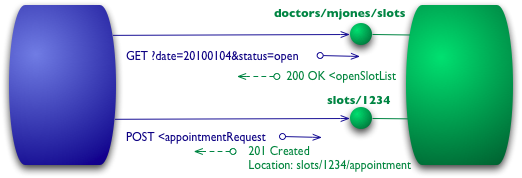


Nivå 2 og 3 av Richardson Maturity Model tar for seg introduksjon av http verb og Hypermedia Controls.



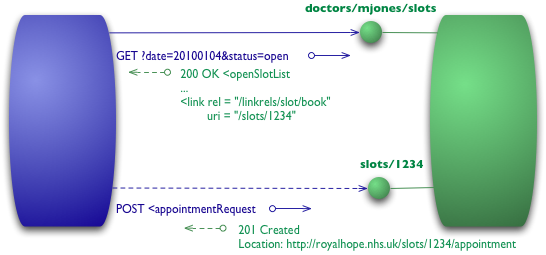
Nivå 2 fordeler og ulemper

* http verb blir brukt til det de er ment for
* mer detaljert bruk av http response code hvis noe går galt
* mer komplisert oppsett
* trenger mer forståelse av http (burde ikke være negativt)



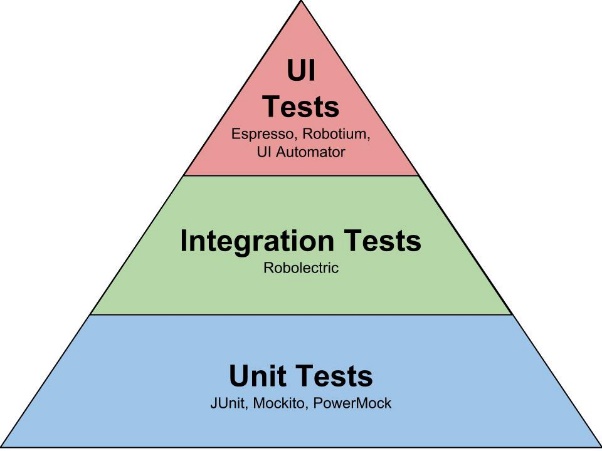
Nivå 3 fordeler og ulemper

* mer detaljert respons, gir bruker mer info om hva som kan gjøres
* kompleksitet



# Testing

Hensikten med å sette opp tester for sin applikasjon er for å gi en ekstra sikkerhet over funksjonalitet som er blitt implementert fungerer som tiltenkt. Det vil føre til mindre bugs som igjen fører til mindre bruk av ressurser til feilretting. Unit testing tar for seg testing av enkelte metoder som du har laget, unit tester testes som regel uten avhengigheter. Integrasjons testing går mer videre på å teste metoder som bruker andre avhengigheter som for eksempel teste http kall mot en server og database eller I/O fra bruker. End to end testing går mer ut på helhetlig testing av din applikasjon. Det vil si tester som utfører handlinger en vanlig bruker har mulighet til. Dette innebærer testing av alt fra front end UI til back end server. Når det kommer til hvilken tester som burde brukes er svaret alle. Alle testene har sine egne områder de ment for å teste og da for å være mer sikker på at applikasjonen fungerer som tiltenkt burde det legges minst like mye tid og ressurser på testing som på selve produksjons kode.

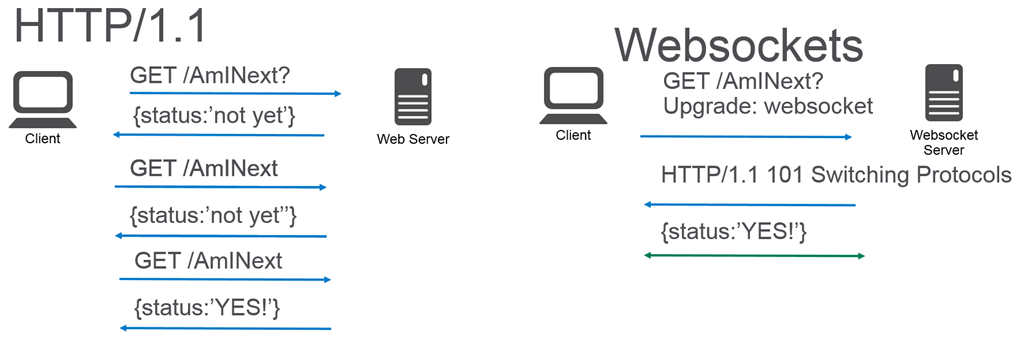


# Websockets

Med websockets fikk problemene som oppstod pga http er stateless endelig en vel fungerende løsning. Websockets gjør det mulig med toveis kommunikasjon mellom klient og server. Det betyr at server kan opprette forbindelse til klienten hvis det skulle trenges uten at klienten først sender en forespørsel. Dette gjorde at funksjoner som f.eks. chat, online spill og push meldinger kunne bli satt opp på en bedre måte med mindre overhead og tregheter. Websockets trenger kun å være støttet av nettleser uten at det trengs å installere andre programmer. Når det kommer til bruk av websockets kommer det mer an på om du som utvikler har funksjoner som kan bruke websockets. Websockets er blitt veldig utbredt og brukere ser på websocket funksjoner som standard. I de tilfellende du ønsker server kunne gitt beskjed om endringer uten at det skal gjøres en spørring kan websockets være løsning.

Fordeler og ulemper med websockets

* toveis kommunikasjon
* bruker TCP istedenfor http
* fortsatt problemer med støtte for bruk av websockets



# Sluttord

Jeg startet med å implementere redux i løsningen min men ble ikke ferdig med den så den er ikke tatt i bruk men koden er heller ikke fjernet siden jeg tenker å fortsette implementeringen av redux