# Notater og kommentarer til implementering av MLAB-komponenter for Moodle

## 2.2.1

*getName*

Denne burde kanskje bli døpt om til ”getMode”. Så at det var gjort i eksempelfila, men den het fortsatt ”getName” i spesifikasjonen, så jeg beholdt den som ”getName”

*onMlabReady*

Her spesifiseres det at det skal kjøres ”initialiseApp” og ”initialiseComponent” dersom de finnes. Ettersom det kan finnes flere komponenter på en side, mener jeg at dette systemet ikke vil fungere. For hvis flere komponenter definerer funksjoner med disse navnene i det globale skopet, vil de bare overskrive hverandre. Så jeg lagde et system der det finnes to globale array ”initialiseApp” og ”initialiseComponent”. Komponentene kan så registrere funksjoner i disse arrayene når de lastes inn, og ”onMlabReady” kjører alle disse funksjonene.

*loadPlugin*

Ingen spesielle kommentarer, annet enn at jeg så meg nødt til å håndtere ”ferdig”-statusen med en event (”pluginloaded”).

## 2.2.2

Som vi diskuterte tidligere her, valgte jeg å bruke ”localStorage” i stedet for database-API-et, ettersom det ville blitt litt mye asynkron koding og callback-funksjoner.

Imidlertid er jo alle ajax-kall asynkrone. Og når en plugin skal lagre ting på en server, blir det nødvendigvis et slikt asynkront kall. Så vi klarte ikke å unngå asynkrone kall likevel. Jeg brukte riktignok events i stedet for callback-funksjoner, siden det blir litt mer ryddig kode, men skulle gjerne sett at vi fant en løsning som blir enklest mulig for utvikling av framtidige komponenter.

## 2.2.3

*loginRemotely*

Innlogging i Moodle er ikke helt rett fram. Man logger seg ikke inn i Moodle som sådan, men i en spesifikk webtjeneste i Moodle. Det er altså ikke helt enkelt å bruke xAPI-webtjenesten samtidig som filwebtjenesten, siden de er registrert som to separate tjenester i Moodle.

Når man logger inn, må man angi hvilken tjeneste man vi logge inn i (”local\_tcapi” for xAPI, ”core\_files” for fil-tjenesten, se pkt 2.4.2). Jeg løste dette ved å lagre innloggings-token i et objekt, der nøkkelen er tjenestenavnet.

Videre vil det kunne bli nødvendig å logge seg inn i flere Moodle-tjenester samtidig, ved åpning av en app som bruker for eksempel både en xAPI-basert komponent og en filkomponent. For å unngå at man må taste inn det samme passordet flere ganger, lagde jeg et system som ”slo sammen” flere slike innlogginger i det samme skjemaet.

*loginToken*

Systemet skissert ovenfor fører også til at man må angi hvilken tjeneste det gjelder når man skal ha tak i en token.

## 2.3

Jeg installerte Tin Can-pluginen i Moodle, og fant et javascript-bibliotek som implementerer klientsiden av Tin Can. Etter å ha henvendt meg til forfatteren av pluginen på Github, fikk jeg vite at pluginen ikke har blitt utviklet videre etter v.0.9 av Tin Can/xAPI (se <https://github.com/jgsmitty/Experience-API-for-Moodle/issues/1>).

Problemet med dette er at v.0.9 har veldig få verb, se liste her: http://tincanapi.com/wp-content/assets/spec/TinCanAPIversion09.pdf

TinCan-biblioteket er avhengig av et par av komponentene i biblioteket CryptoJS. Disse er lagt til nederst i tincan.js-fila.

En annen ting som jeg føler er litt uklart i spesifikasjonen er offline-funksjonaliteten. Det er skissert at mlab.js tar seg av den lokale lagringen, men overlater kontrollen og ansvaret for lagringen til en plugin (i dette tilfellet plugin\_moodle) hvis den er lastet inn. Videre kan man vel kanskje anta at pluginen gir kontrollen tilbake til mlab.js og lokal lagring dersom man ikke har nettforbindelse eller lagring eksternt feiler.

Men man kan miste og få tilbake nettverksforbindelsen når som helst, også mens man er midt i en app. Med systemet skissert over, vil man kunne risikere at man lagrer en del ting eksternt, mister nettverksforbindelsen, og så utfører en operasjon som skal hente ned data igjen. Da vil vi være prisgitt de dataene vi har lagret lokalt, og hvis alt så langt kun har blitt lagret eksternt, har vi ingenting.

Etter min mening ville det vært bedre med en slags synkroniseringsløsning. Det innebærer at alt alltid blir lagret lokalt, men at pluginen har ansvaret for å lese de lokale dataene, laste dem opp til en ekstern tjeneste, og eventuelt oppdatere de lokale dataene dersom det har kommet noe nytt på den eksterne tjenesten (hvis man for eksempel bruker appen fra to forskjellige enheter). Da må man imidlertid håndtere eventuelle kollisjoner og holde rede på hva som er synkronisert og ikke.

## 2.4.1

”getAllStates()”, definert i 2.2.2, har ingen motpart i Moodles TinCan-plugin eller TinCan-klient-biblioteket. Men ettersom det er ganske kontrollert hvilke states det er snakk om, er det like greit å ta dem enkeltvis.

## 2.4.2

La til en webservice i Moodle via ”Site administration” -> ”Plugins” -> ”Webservices” -> ”External services”, som eksponerte funksjon for å laste ned filer. Måtte manuelt sette inn et ”shortname” i databasetabellen ”mdl\_external\_services”. Valgte ”core\_files” som shortname, fordi det så ut som det passet inn, men det kunne i prinsippet vært hva som helst.

Videre er det litt uklart hvilke filer som skal laste ned. Filene i Moodle er organisert etter tilgangsnivå og kontekst, og kan dermed være både personlige og felles, og tilhøre et spesifikt kurs eller en person, eller Moodles forside eller systemet.