Teknisk arkitektur for (mobile) apps/widgets

Formålet med dette dokument er at koordinere tekniske valg for open source (mobile) apps/widget implementation i biblioteksverdenen, så kode kan genbruges så let som muligt mellem projekter.

Dokumentet her er et udkast / oplæg til diskussion, - kom gerne med kommentarer til valgene og lad os prøve at finde en fælles platform.

Emnet her er i første omgang afgrænset til applikationer/widgets hvor logikken kører på klienten og som også kan bringes til delvis at fungere offline, i modsætning til (mobile) websites.

Kriterierne for de tekniske valg skal understøtte deling/genbrug af udvikling, og er derfor:

- Open Source
- Konsensus om valg
- Portabilitet (herunder også serverside)

Dette dokument er tænkt til senere at blive taget op på tekniske arkitekturforum hos DBC, og vil sandsynligvis blive interne guidelines her, så feedback nu er en måde at påvirke retningen her.

Opsummering/checkliste

Herunder en checkliste, primært skrevet til eget(rje@DBC) brug ved nye app/widget-projekter. Det opsummere også teknologivalgene diskuteret herunder:

- Packaging: `package.json`, `config.xml` (-> phonegap-build), `cache-manifest`, evt. Apache Callback m. plugins.
- Libraries: `requirejs`, `es5-shim`, `json2`, `jquery`, `underscore`, `backbone`, `jasmine`
- README.md: hvad/hvorfor, features, roadmap, api, filstruktur, credits
- Infrastruktur: docco, travis,
- Mobiltest: Android 2.1/4, iOS 4/5
- Browsertest: Webkit(Chrome), Gecko(Firefox), Trident(IE8), Presto(Opera Mini)

Ovenstående er konklusionen af diverse eksperimenter/analyse, og skal til at tages i brug - *in progress, not done yet.*

Platform

Widgets/apps implementeres med webteknologi så de både kan publiceres som mobile applikationer via market/appstore med adgang til mobilspecifikke features såsom stregkodeskanning, og også kan køre direkte i webbrowseren. Dette giver desuden mulighed for, udover mobile apps, også at køre dem som widgets på pc'er, tablets og storskærme.

Apache Callback / PhoneGap

Deciderede mobilapplikationer bygges via Apache Callback(aka. PhoneGap). Dette gør at applikationen i market/appstore kan få adgang til mobil-specifikke features der ikke er i browseren, såsom skanning af stregkoder, samtidigt med at kodebasen også kan bruges til webapps hvor disse features er disabled.

PhoneGap skiftede navn til Callback i forbindelse med at det blev overdraget til Apache Foundation. Dette kan ses som en sikkerhed for at den forbliver helt open source, også efter at Adobe har opkøbt Nitobi(udviklerne af PhoneGap).

Udover Apache Callback, er særligt titanium, rhomobile, mosync undersøgt unity, men de er alle mere lukkede og/eller understøtter ikke at applikationen samtidigt er en webapp. Bemærk at en del af dem har properitær bygning af mobilapps selvom de er open source.

Widgets og offline apps

Udover indpakningen som mobilapp, kan webapplikationerne også bygges som offline apps og widgets.

Offline applikationer implementeres via en manifest-fil, der er en del af html5, og som definerer hvad der skal downloades for at applikationen kan køre offline.

Widget indpakning er en anden måde at lave distribuerbare pakker. W3 har standard på området, som forøjeblikket er implementeret i PhoneGaps byggeservice, Opera browsers, samt Apache Wookie. Dette foregår som et zip-arkiv med applikationen, og en `config.xml` der indeholde metadata.

JavaScript applikationer og biblioteker har derudover ofte metadata i form af en `package.json` som defineret i commonjs

Programmeringssprog

JavaScript anvendes til udvikling af apps.

Udover JavaScript er særligt Java undersøgt, da dette sprog bruges en del på DBC og kan oversættes til JavaScript via GWT så det ligeledes kan bruges i browsere. Ulempen ved Java i forhold til JavaScript er at det ikke integrerer

helt så tæt med browseren, hvilket gør at sproget nogen gange kommer i vejen i applikationsudvikling. For applikationer der bruger en mere afgrænset del af browseren (ie. kun canvas eller lignende) vil Java oversat med GWT kunne give god mening, men hvis vi skal afgrænse os til et primært sprog må det blive JavaScript.

Øvrige alternativer: CoffeScript (Syntactic sugar med tæt integration af JavaScript), Dart(Googles foreslag til JavaScripts efterfølger) og HaXe(crossplatform JavaScript-lignende sprog der oversættes til JavaScript) er også potentielle sprog, men ikke vurderet som værende tilstrækkeligt mainstream.

Modulsystem

JavaScript mangler et decideret modulsystem.

`Requirejs` ser umiddelbart ud til at være det bedste bud. Det er designet så moduler kan genbruges både i browsere og på server. Det understøtter afhængigheder mellem moduler, og sikre også at modulerne ikke konflikter/forurener det globale scope. Det er også udviklet med et øje på commonjs moduler, som er ved at være standard serverside. Understøtter både at de enkelte scripts bliver resolved og loadet i selve browseren, og også at script-filer bliver kompileret/minimeret til en enkel fil til deployment.

`Commonjs` har en begyndende standardiseringen af JavaScript moduler, men er rettet mod servere og understøtter endnu ikke asynkrone moduler, hvilket er en nødvendighed hvis det skal kunne køre i browser. Requirejs er det client-side modulsystem der lægger sig tættest i retning af den kommende standard her.

DBCs `jscommon-use`-system er ligesom Commonjs server-side, og kan ikke direkte anvendes på browsere. Det vil desuden give mening at rette `use`-systemet sålede at det også understøtter requirejs-moduler (eller hvad vi vælger til client-side applikationer), så kode også kan deles/genbruges mellem DBC server side og client-side applikationer.

`enderjs` er en browser-JavaScript pakkemanager der kompilere moduler til en enkel fil. Fordelen her er at modulerne laves i commonjs-format, og det gør styring af afhængigheder langt lettere. Ulempen er at modulerne skal kompileres hvilket tilføjer et trin i forhold til udvikling.

`yepnope` er en mere letvægts script-loader. De enkelte script har her selv ansvaret for eventuelt at oprette variable de kan blive tilgået via i det globale scope. Til gengæld har det god understøttelse af betinget load af scripts, så til små projekter med forskellig kode afhængig af platform giver dette god mening.

`labjs` ligger i samme kategori som `yepnope`, dog ikke med samme conditional support.

`jquery` plugins var også nævnt som mulighed ved udviklermødet. Fordelen ved dette er at det (måske?) har mindshare hos udviklere i biblitekerne? Forbeholdet her er at det indføre en afhængighed på jquery, også i moduler der er ren JavaScript og ellers ville kunne genbruges andre steder.

JavaScript og DOM-abstraktion

For at normalisere JavaScript-miljøet anvendes `es5-shim.js` og `json2.js` på de platforme hvor der er behov for det. Disse moduler backporter de features fra EcmaScript 5 som kan implementeres oven på EcmaScript 3 (EcmaScript er JavaScript-standardiseringen).

Til DOM-manipulation anvendes JQuery, da det er det mest udbredte DOM-abstraktion, har betydelig mindshare og anvendes allerede en del i bibliotekssammehæng.

Funktionsbiblioteker

`underscore.js` samt `backbone.js` anvendes for at undgå at genopfinde den dybe tallerken. Disse indeholde diverse utilities, samt framework og konventioner for MVC-design i JavaScript. Indenfor den type biblioteker virker det som om at disse er dem der har størst mindshare, kildekoden ser meget fornuftig ud, og det integrerer let med jquery/zepto.

Best practices

Koden er open source. For at støtte en åben process lægges koden ud løbende. Github anvendes, da dette er det primære kodedelingsværktøj for TING-bibliotekerne i mellem.

Koden skal gennem mindst to sæt øjne før den er *done*. DBC bruger reviewboard internt til code review. Derudover er der en form for code review når patches accepteres genne pull-requests i github.

Dokumentation

Kode forventes at have tre typer dokumentation: intern dokumentation, api dokumentation og projektdokumentation.

Til intern dokumentation anvendes `docco`, og til API- og projekt-dokumentation anvendes markdown.

Motivation for valg af markdown: 1) samme dokumentationsformat for al dokumentation(docco bruger også markdown internt), 2) tekstbaseret format bedre til versionshistorik, 3) integration med github, 4) da JavaScript er dynamisk

har api-dokumentationsværktøj ikke samme fordele som med statiske sprog, så almindelig tekst kan være et fornuftigt valg 5) det er let at konvertere dokumentation til andre formater (html, pdf, ...)

Projektdokumentationen ligger i `README.md` forventes at indeholde:

- Beskrivelse af projektet: hvad og hvorfor
- Overblik over filstruktur / moduler
- Features, hvad er implementeret
- Roadmap, hvad er det planen at implementere
- Credits, hvem har bidraget til projektet, og hvilke andre projekter bygger det på

Intern test

Der skal være intern test af koden.

Til intern test ser jasmine ud til at være det mest mature testframework, og virker på tværs af platforme, så dette er umiddelbart det bedste bud.

Et muligt alternativ kunne være mocha, som ser ud til at have samme syntaks til definition af tests som jasmine, men er mere asynkront og overlade testsammenligninger til andre biblioteker elelr manuelle throws.

Automatisk grænsefladetest

Automatiske grænseflade test er velsete. Selenium anvendes her.

Platforme til (manual) test af mobilapplikationer

Ved applikationer til smartphones testes de at virke på:

- Android 2.1 low dpi device
- Android 4 high dpi device
- iOS 4 low dpi device
- iOS 5 high dpi device

Bemærk: Android 2.1 har en del bugs på `canvas` samt lav market share, så hvis det er en canvas-tung applikation kan det afgrænses til 2.2+. iOS 4 understøtter ikke `position: fixed` ved css, så der kan være quirks her.

Afgrænsningen til Android og Safari skyldes at dette dækker 95+% af mobilbrugerne. Derudover skal koden testes på de forskellige browser-engines:

- Webkit (Chrome)
- Gecko (Firefox)
- Trident (Internet Explorer (8+))
- Presto (Opera Mini (tester også for featurephones))

Dette sikre både at applikationen virker på desktopplatformen indlejret i en webside, og giver også en fremtidssikring når/hvis nye platforme begynder at spille en større rolle (Windows Phone, mobilbrowser fra Mozilla eller Opera, etc.).

Integrationsserver

Umiddelbart er det to integrationsservere: Jenkins og travis.

Jenkins anvendes på DBC, og der er vist også planer om at eksponere den udadtil. Vi bruger den allerede til det meste byg og test, så det er oplagt at anvende denne internt til apps/widgets også.

Da travis er triviel at sætte op anvender vi begge integrationsservere. Testproceduren er i forvejen beskrevet i `package.json`, så travis kræver blot tilføjelse af en 4-linjers `.travis.yml` konfigurationsfil til repositoriet for at fungere. Fordelen ved at have travis som supplement til jenkins er, at integrationen med github gør det trivielt for eksterne udviklere at have integrationsserver for deres kloner af repositoriet.