CHALMERS

Offline First

Undersökning av lagringstekniker på klientsidan för webbapplikationer

Examensarbete inom Högskoleingenjörsprogrammet i Datateknik

JONAS HA
ERIK GIL FORSMAN

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2014

Offline First
Undersökning av lagringstekniker på klientsidan för webbapplikationer

Jonas Ha, Erik Gil Forsman

© JONAS HA, ERIK GIL FORSMAN, 2014

Institutionen för data- och informationsteknik
Chalmers tekniska högskola
412 96 Göteborg
Tel: 031-772 1000
Fax: 031-772 3663

Institutionen för data- och informationsteknik
Göteborg, 2014

Förord

Examensarbetet är utfört på Sigma IT & Management i samarbete med Institutionen Data- och Informationsteknik, Chalmers tekniska högskola, Göteborg. Examensarbetet omfattar 15 högskolepoäng och ingår i utbildningen Dataingenjör (180 högskolepoäng).

Examensarbetet har gett förståelse för de lagringstekniker som finns implementerade i dagens webbläsare, samt hur dessa tekniker kan kombineras för att skapa en interaktiv webbapplikation som är fungerande på samma vis både online som offline.

Vi vill tacka alla involverade från Sigma IT & Management, framförallt vår handledare Magnus Markling från Sigma IT & Management för stöd kring arbetet. Vi vill också tacka Linus Vidberg för det sociala stödet och för att vi har fått möjligheten att utföra examensarbete på Sigma IT & Management. Vi vill även tacka vår handledare på Chalmers, Uno Holmer för stöd, idéer och vägledning under rapportskrivningen.

Jonas Ha och Erik Gil Forsman, Göteborg den 10 juni 2014

Abstract

Web applications that are tailored for mobile platforms have become a popular solution among developers instead of native applications. It is primarily the cross-platform feature that is of interest to the developer when developing web applications. But in order to compete against native applications, it is important that the web applications are able to run offline. This means that the functionalities that do not require Internet access operate in the same manner as if the Internet connection existed. When HTML5 was released, a couple of storage technologies were also introduced. These technologies allow developers to store data on the users' devices. By combining these storage technologies with the right architecture, it is possible to create web applications that are not dependent on an Internet connection. This report describes storage technologies such as Application Cache, Web SQL, IndexedDB and Web Storage that are currently available in users' browsers, as well as an in depth analysis of each storage technology to determine which technology is best suited for certain types of data. This report also includes a case study to illustrate how these technologies along with AngularJS can be used in practice. The web application is developed using HTML's latest standard HTML5, which includes CSS3 and JavaScript. To achieve a more content rich theme and custom mobile application, the framework jQuery Mobile was also used. The application is an implementation of an amusement park's website and the data is downloaded from a public API. The conclusion in this report is that the different storage technologies are sufficiently mature to be used in production. It is recommended that larger amount of data are cached in either Web SQL or IndexedDB, simpler data sets in Web Storage and important files are recommended to be cached in the Application Cache.

Sammanfattning

Webbapplikationer som är anpassade för de mobila plattformarna har blivit en populär lösning bland utvecklare istället för native applikationer. Det är framförallt crossplatform egenskapen som gör webbapplikationer intressant för utvecklaren. Men för att kunna konkurrera mot native applikationer är det viktigt att webapplikationerna klarar av att köras offline. Det innebär att funktionaliteten som inte kräver Internet ska fungera på samma sätt som om Internetuppkoppling existerade. I samband med att HTML5 släpptes introducerades ett par lagringstekniker som gör det möjligt för utvecklaren att lagra data på användarnas enheter. Genom att kombinera dessa lagringstekniker med rätt arkitektur är det möjligt att skapa webbapplikationer som inte är beroende av Internetuppkoppling. Rapporten beskriver lagringsteknikerna Application Cache, Web SQL, IndexedDb och Web Storage som idag finns tillgängliga i användarnas webbläsare, samt en fördjupning i respektive lagringsteknik för att avgöra vilken teknik som lämpar sig för viss typ av data. Rapporten innehåller även en fallstudie som visar hur dessa tekniker tillsammans med AngularJS kan användas i praktiken. Webbapplikationen utvecklas med hjälp av HTMLs senaste standard HTML5, som inkluderar CSS3 och JavaScript. För att få mer innehållsrika teman och mobilanpassad applikation har även ramverk som jQuery Mobile använts. Applikationen är en egen implementation av en nöjesparks webbplats. Den data som används i applikationen hämtas från ett öppet API. Slutsatsen i denna rapport är att de olika lagringsteknikerna är tillräckligt mogna för att användas i produktion. Det rekommenderas att större datamängder antingen cachas i Web SQL eller IndexedDB, enklare datamängder i Web Storage och viktiga filer rekommenderas att cachas i Application Cache.

INNEHÅLLSFÖRTECKNING

Ordlista med definitioner

Förtydligande lista på förekommande ord samt nyckeltermer i bokstavsordning.

API: *"Application Programming Interface", är en regeluppsättning för hur en viss programvara kan kommunicera med annan programvara.*

Asynkron: *Motsatsen till synkron, det betyder att en process inte är bunden till en central klocka som delas mellan processer utan de går självständigt och börjar samt avslutar utan samordning med mottagaren.*

Back-end (server-sida): *Den del som i de flesta fallen sköter lagring av data, men kan även sköta den del där hämtning/lämning av data sker.*

Cache: *Lagringsutrymme som används i första hand för att förbättra exekveringstid genom att lagra data som används ofta.*

Callback: *JavaScript exekveras rad efter rad, men nästa kodrad kan köras även om föregående kodrad inte har exekverats klart, vilket kan orsaka problem. Detta kan lösas genom callbacks. En callback funktion exekveras efter att pågående kodstump har exekverat färdigt.*

Cookie: *En väldigt begränsad textfil för lagring av data i textformat, dels för att förbättra upplevelsen samt funktionaliteten på en webbplats.*

CORS: *"Cross-origin resource sharing", är en teknik som tillåter resurser på en webbsida att begäras från en annan domän utanför den domän där resursen ursprungligen kommer ifrån.*

DNS Spoofing: *En attack på en DNS-server som leder till att servern returnerar felaktig (spoofad) IP-adress vid en DNS-förfrågan.*

DOM: *"Document Object Model", är ett gränssnitt för åtkomst samt dynamisk ändring av ett dokuments innehåll och dess struktur.*

Front-end (klient-sida): *Det gränssnitt samt funktionalitet och information som presenteras när man besöker en webbplats i en webbläsare.*

HTML5:	*Senaste revidering av Hypertext Markup Language som beskriver strukturen samt uppbyggnad av webbsidor.*
JavaScript:	*Ett programmeringsspråk som används för utveckling av interaktiva samt dynamiska webbapplikationer. JavaScript ingår även i paraplybegreppet HTML5.*
Native applikation:	*En plattformsberoende applikation som kräver installation.*
Offline:	*När exempelvis en enhet inte har någon anslutning till en server eller webbplats.*
Ramverk:	*Ett utvecklat bibliotek med funktioner för att underlättar utveckling av webbapplikationer.*
Semantisk Mark-Up:	*Användning av element som är anpassade för det specifika ändamålet, som exempelvis en artikel skall ligga inom så kallade artikeltaggar; "<article>innehåll</article>". I HTML5 förenklas detta genom att det används taggar som är självförklarande.*
Synkron:	*Enkelt förklarat betyder synkron att processer som körs vid samma tidpunkt delar på samma klocka, vilket betyder att den ena inväntar på att den andra slutför sin uppgift innan den börjar med att exekvera.*
Webbplats:	*Webbplats som även kallas hemsida eller webbsajt är en samling av sammanhängande dokument, bilder, videor, etc. som är lagrade på en eller flera webbservrar. Dessa är vanligtvis nåbara genom Internet.*
Webbsida:	*Webbsida är en sida på en webbplats.*
W3C:	*"World Wide Web Consortium", är ett internationellt samfund där medlemsorganisationerna samt W3Cs anställda tillsammans med allmänheten utvecklar standarder.*
XSS:	*En attack där angriparen infekterar en webbplats med skadlig skriptkod så att den exekveras i besökarnas webbläsare.*

1. INLEDNING

1.1. Bakgrund

Sigma IT & Management är ett konsultbolag som tillhör Sigma koncernen. Webbapplikationer är en marknad som har expanderat markant under de senaste åren, vilket gör det till ett intressant område för Sigma. Den största fördelen med webbapplikationer är att man kan bygga, mer eller mindre, plattformsoberoende applikationer.

Webbapplikationer är oftast beroende av en uppkoppling till en server, även när det inte är nödvändigt. "Offline first" innebär att utvecklaren av en webbapplikation skall utgå från situationer då användarna saknar eller har dålig uppkoppling. Genom att cacha relevant data, finns det en möjlighet för användarna att fortsätta nyttja applikationen även när uppkopplingen försvinner.

1.2. Syfte

Sigma IT & Management vill fördjupa sig inom lagring av data för webbapplikationer för användning vid offline läge. Fördjupningen omfattar även jämförelser mellan olika tekniker och hur dessa beter sig i praktiken samt vad som kan fungera för nuvarande och för framtida potentiella kunder. Undersökningen av dessa tekniker sker i rent inlärningssyfte.

1.3. Frågeställningar

Målet är att inventera olika tekniker som är lämpade för att lagra data för webbapplikationer på ett effektivt sätt, få en översikt över hur de olika tekniker beter sig samt hur kostsamma de är. Egenskaper av intresse är:

- Mogenhet (hur etablerade är teknikerna?).
- Enkelhet/Kraftfullhet (enkel kod som utför mycket arbete).
- Inlärningströskel.
- Upplevd prestanda.

1.4. Avgränsningar

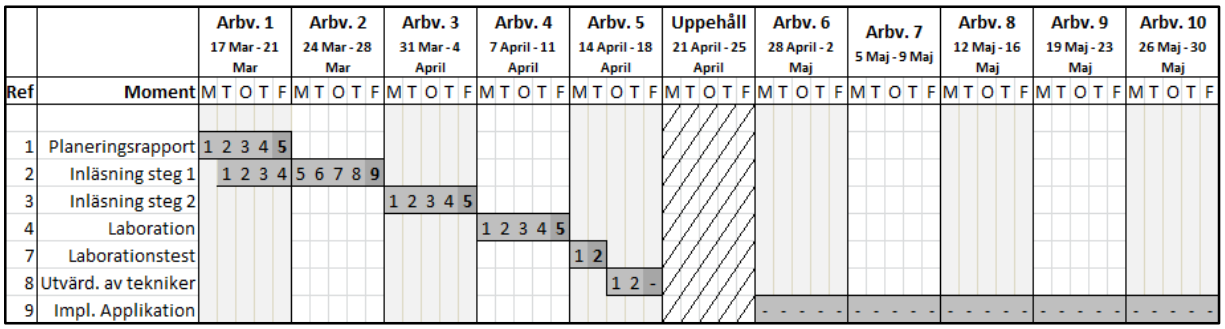
Arbetet kommer inte att omfatta en djupgående förståelse på serversidan. Fokus kommer att ligga på HTML5 och olika lagringstekniker för att kunna utvärdera vilka front-end-lösningar som är bäst i ett "Offline first"-projekt. Arbetet kommer att avgränsas till de mobila plattformarna.

1.5. Rapportöversikt

Rapporten innehåller en introduktion till webapplikationer för de som inte är bekanta med området. Introduktionen förklarar grundläggande begrepp såsom HTML och JavaScript. För den mer erfarna läsaren fungerar det utmärkt att utesluta introduktionen och direkt börja på avsnitt 3.2 Lagringstekniker.

2. METOD

Arbetet utförs iterativt enligt figur 2.1. Planeringsrapporten är det första momentet som utförs, där planering fastställs om vad som skall utföras under den bestämda tiden som arbetet pågår. Jämnlöpande påbörjas den första inläsningsfasen där fokus ligger på en djupgående förståelse av de grundläggande delarna om webbutveckling, såsom HTML5 och JavaScript tillsammans med jQuery, jQuery Mobile och AngularJS.
Moment tre som motsvarar andra inläsningsfasen, där mycket fokus ligger på de lagringstekniker som finns implementerade i de olika webbläsarna, såsom Application Cache, Web SQL, IndexedDB och Web Storage. Därefter påbörjas de praktiska momenten som består av laboration och tester, samt utvärdering av dessa olika tekniker. I slutmomentet kommer det även att fokuseras mycket på en egen implementering av ett befintligt projekt som går under namnet Blåa Lund.



Figur 2.1. *Tidsplan*

2.1. Litteraturstudie

Rapporten är uppdelad i två faser, inläsning och praktisk tillämpning. Den teoretiska fasen omfattas till största delen av sökningar genom sökmotorn Google. Eftersom rapporten innehåller en praktisk fas, är det enkelt att utvärdera information som man hittar.

Organisationerna bakom de stora webbläsarna har oftast webbplatser med s.k. Developer libraries, som innehåller teori som kan vara intressant för utvecklare som arbetar med respektive webbläsare. I detta fall kan dessa webbplatser anses vara tillförlitliga källor med nyttig information för läsaren.

2.2. Laboration och tester

För att kunna få en bättre förståelse av hur olika lagringstekniker beter sig, krävs det en implementation av teknikerna i testningssyfte, vilket har rubricerats som en laboration i planeringen. Laborationen består av att implementera olika fall som kan uppstå i produktionsmiljö, som exempelvis att uppkopplingen går förlorad när användaren skall ladda upp data till en server. Fallet skall då behandlas på ett sätt som kan fungera i produktion, där tanken är framförallt att användarupplevelsen inte skall påverkas.

2.3. Applikation: Blåa Lund

Blåa Lund är namnet på den egna implementationen av ett befintligt projekt. Ingen djupgående förståelse på seversidan kommer att hanteras, offlineläge är avsett att hanteras på klientsidan. Därmed används ett befintligt API från det befintliga projektet. Det som skall hanteras i Blåa Lund är de delarna där lagring av data för offlineläge kan vara möjlig, t.ex. är bokning inget tänkbart fall för implementationen.

3. TEKNISK BAKGRUND

3.1. Allmänna webbtekniker

Det här avsnittet behandlas de webbtekniker som är av intresse när man utvecklar webbapplikationer.

3.1.1. HTML5

Webbinteraktion har blivit ett populärt ämne det senaste decenniet. I följd med det har även en uppdatering av HTML ägt rum. HTML5 är nu den senaste versionen av märkspråket HTML.
Animering och effektfulla webbsidor kunde tidigare endast visualiseras med externa plugin som Flash. Dessa plugin är inte längre nödvändiga då HTML5 erbjuder samma typ av funktionalitet (Sackermark, 2014).
Själva begreppet HTML5 innefattar inte enbart HTML-språket utan är ett samlingsnamn där även CSS3 och JavaScript ingår (Axelsson, 2014).

Tillsammans med HTML5 kom många nya språkelement som skall underlätta utvecklingen av webbapplikationer ytterligare (Sackermark, 2014).
I detta sammanhang pratas det mycket om "semantisk markup". Semantik i detta sammanhang betyder att man använder sig av element som är anpassade för det specifika ändamålet, som exempelvis att en artikel skall ligga inom så kallade artikeltaggar; "<article>innehåll</article>". I HTML5 förenklas detta genom att det används taggar som är självförklarande som visas i Figur 3.1 (Amnell, 2014).

```
1  <header>HTML5</header>
2  <time>10:00</time>
3  <article>Offline First</article>
4  <footer>Copyright (C)</footer>
```

Figur 3.1. *HTML5, Semantisk Markup*

3.1.2. Cachning

Cachning innebär, i det här fallet, att webbläsaren tar en kopia av webbsidan och sparar den temporärt i ett snabbt minne för att enkelt kunna komma åt den i framtiden. Syftet med detta är att webbläsaren inte skall behöva ladda ner samma webbsida flera gånger om det inte är nödvändigt. Det resulterar i att webbsidan laddas snabbare, eftersom den finns lagrad på datorn, samt att webbsidan kan presenteras i webbläsaren även när datorn inte är uppkopplad till Internet (Rouse, 2012).

3.1.3. Base64

Base64-kodning är en metod som gör det möjligt att överföra binär data över medier som enbart tillåter tecken inom Base64-alfabetet. Base64 används exempelvis

frekvent inom E-post. I denna rapport används Base64 för att på ett effektivt, men ändå relativt enkelt sätt, konvertera bilder till sträng-format. Bilder som har konverterats till Base64-format kan presenteras i HTML. Det enda som krävs är att ”” (”TYP” är formatet på den faktiska bilden, t.ex. jpg) finns definierad innan Base64-värdet (Menon, 2008).

3.1.4. JavaScript

JavaScript är ett skriptspråk som används främst på klientsidan i webbsammanhang. JavaScript stöds av de flesta webbläsarna och ingår även i paraplybegreppet HTML5. Användningsområden kan vara servrar, tablets, smartphones med mera.
I HTML kan man använda JavaScript för att skapa interaktiva samt dynamiska webbsidor, eftersom JavaScript kan manipulera DOM:en (Refnes Data, 2014b).

3.1.4.1. jQuery

jQuery är ett JavaScript-bibliotek som innehåller en mängd olika funktioner vars syfte är att göra det enklare att skriva JavaScript-kod. Framförallt är det förenklingen av DOM manipulation, händelsehantering och AJAX som är de stora fördelarna med jQuery (jQuery, 2014).

3.1.4.2. jQuery Mobile

jQuery Mobile används för att skapa webbapplikationer som är anpassade för både smartphones och desktops. Ramverket är utvecklat av jQuery Foundation, som har mycket erfarenhet inom webbutveckling och har skapat flera kända ramverk som finns ute på marknaden (jQuery Mobile, 2014).

3.1.4.3. AngularJS

AngularJS är ett open-source ramverk skapat av Google och används framförallt för att bygga dynamiska webbapplikationer. AngularJS inför flera nya HTML-element och attribut som sedan används av AngularJS för att utföra arbete i JavaScript. Det går att kombinera AngularJS med andra ramverk, t.ex. jQuery, för att få det bästa ur ramverken (Google, 2014).

3.2. Lagringstekniker

I det här avsnittet behandlas den teori som är av intresse när man utvecklar webbapplikationer som skall lagra data på användarnas datorer.

3.2.1. Application Cache

Application Cache är den mest osofistikerade lagringstypen för offline applikationer. Utvecklare kan bestämma vad som skall lagras genom att använda sig av ett så kallat ”manifest”. I manifestet bestämmer utvecklare explicit vilka filer som skall cachas. Manifestet skall refereras genom en HTML-tagg så att manifestet kan laddas (Lubbers, Albers och Salim, 2010).

3.2.1.1. Cache Manifest

Cache Manifest är filen som utvecklaren använder för att specificera vilka filer som skall cachas. En manifestfil måste hanteras med den korrekta MIME-typen som är “text/cache-manifest”. Den rekommenderade filändelsen till manifestet är .appcache. Det finns tre olika nyckelord som används i manifestet. ‘CACHE’ används för att lista vilka filer som skall cachas i Application Cache. Under ’NETWORK’ listas filerna som kräver att användaren skall ha uppkoppling. ‘FALLBACK’ används för att skapa en sorts backup, utvecklaren kan specifikt bestämma vilken fil som skall laddas ifall en viss fil är otillgänglig (Refnes Data, 2014a). Figur 3.3 visar ett exempel på hur en Cache Manifest kan se ut.

```
1  CACHE MANIFEST
2  # v1
3
4  CACHE:
5  /styles/style.css
6  /Scripts/myScript.js
7  /home/index.html
8
9  NETWORK:
10 *
11
12 FALLBACK:
13 / /offline.html
```

Figur 3.3. *Strukturen i Cache Manifest*

3.2.1.2. Fördelar med Application Cache

Det finns ett antal fördelar med att använda sig av Application Cache. Skillnaden på laddningshastigheten för en sida är markant i jämförelse med att hämta data från en server. Detta är på grund av att ingen uppkoppling behövs för att hämta data som redan finns sparat i cachen. Cachen är det första platsen som kontrolleras innan en sida laddas. Om sidan samt dess relaterade data existerar i cachen, hämtas informationen direkt från cacheutrymmet.
Inlärningströskeln för Application Cache är relativt låg pga. att manifestfilen är enkel att hantera. Application Cache har även fördelen att det stöds i de allra flesta webbläsare, se Bilaga 1. (s. 34) (Refnes Data, 2014a).

3.2.1.3. Nackdelar med Application Cache

Application Cache har nackdelar som kan skapa stora problem för utvecklare. Om en fil som man explicit bestämt skall cachas inte existerar eller inte kan hämtas från servern avbryts hela processen med cachningen. Detta kan i sin tur leda till att webbapplikationen inte fungerar offline eftersom inga filer är cachade.
En annan nackdel är att om det utförs ändringar i någon av filerna som man explicit har bestämt skall lagras och att det på så sätt redan finns en äldre version lagrad i användarens webbläsare, kommer filen inte att uppdateras per automatik. För att filen skall hämtas eller uppdateras igen, måste manifestfilen ändras, vilket

leder till att alla filer som är listade i manifestet kommer att hämtas igen. (Refnes Data, 2014a)

3.2.2. Web Storage

Web Storage är en lagringstyp inom HTML5 där utvecklaren kan spara data direkt på klient-sidan i webbläsaren. Datan lagras i form av "key-value pair" och kan nås upprepande gånger efter att webbläsaren har stängts ner.

Web Storage är efterträdaren till Cookie (även känt som kaka i det svenska språket). Cookie fick namnet efter en äldre programmeringsteknik för överföring av små datavärden mellan program. Web Storage och dess enkla API underlättar för utvecklare att spara värden i webbläsaren i form av JavaScript-objekt, som är tillgängliga vid omladdning av webbsidan. Web Storage lagringsutrymme varierar mellan webbläsare, men det finns ett minimum på 2,5 MB (Lubbers, Albers och Salim, 2010).

3.2.2.1. sessionStorage & localStorage

I Web Storage finns det två olika lagringsmöjligheter, sessionStorage eller localStorage.
Den förstnämnda lagringsmöjligheten fungerar så att användaren kan välja att låta datavärdena överleva under sidladdningarna i ett specifikt fönster som sedan kasseras efter att fönstret har avslutats av användaren.
Den andra möjliggör beständig lagring av data (Lubbers, P., Albers, B and Salim F., 2010). Figur 3.4 visar exempelkod på hur localStorage kan se ut i praktiken.

```
1    localStorage.setItem("Key", "Value");
2    var getItem = localStorage.getItem("Key");
3    console.log(getItem); //Skriver ut: Value
```

Figur 3.4. *Exempel på Web Storage syntax för lagringsmöjligheten localStorage*

3.2.2.2. Styrkor med Web Storage

- Web Storage API är fullt kompatibelt med de modernaste webbläsarna, som även inkluderar webbläsarna på de mobila plattformarna. Opera Mini är dock ett undantag som inte stödjer Web Storage.
- Web Storage APIs dokumentation för implementering och användning är enkelt, vilket betyder att inlärningströskeln är relativt låg.
- Tekniken tillhör "synkron API".
- Stödjer "semantiska" händelser för att hålla andra webbläsarflikar synkroniserade (Mahemoff, 2010).

3.2.2.3. Svagheter med Web Storage

Det finns ett antal svagheter med Web Storage, även om lagringsmetoden verkar trivial.

- Dålig prestanda är det stora problemet när man använder sig av Web Storage. Med dålig prestanda i detta sammanhang menas att det leder till att data som är avsett för att laddas från Web Storage tar längre tid än vad som anses vara användarvänligt.
    - Bakomliggande faktorer som medför detta är bland annat hanteringen av stora eller komplexa datastrukturer med hänsyn till det "synkrona API:et".
    - En annan svaghet är att det inte finns någon riktig indexering för stora och komplexa datastrukturer, vilket medför att sökning av specifik data i lagringsutrymmet tar lång tid jämfört med andra lagringstekniker, på grund av att iterering över all data måste utföras.
    - Lagring och hämtning av stora och komplexa datastrukturer bidrar till dålig prestanda eftersom det är nödvändigt att manuellt serialisera samt deseralisera till och från sträng-format.
- Utöver bristfällig prestanda för Web Storage finns det även behov av att säkerställa överrensstämmelse mellan strukturen vid inläsning av data och strukturen som är fördefinierad i databasen (Mahemoff, 2010).

3.2.3. Web SQL

Web SQL är en strukturerad databas som baserar sig på SQLite som i sin tur är en minifierad version av en vanlig SQL-databas. Syntaxen som används i SQLite är till största delen identisk med ordinarie SQL men skiljer sig en aning (Mahemoff, 2010). Web SQL är dessvärre inte någon standard i W3 och är inte heller någon specifikation i HTML5, vilket betyder att ingen vidareutveckling av Web SQL kommer att ske (Hickson, 2010). Dock är tekniken fortfarande aktuell i en stor del av de ledande webbläsarna för smartphones.

Figur 3.5 visar ett exempel på syntax för initiering av en Web SQL-databas, där funktionen ”openDatabase” tar fyra inparametrar:

1. Namnet på databasen.
2. Databasversionen.
3. En kort beskrivning.
4. Den estimerade storleken på databasen.

Därefter skapas en transaktion efter att funktionen openDatabas har exekverats, vilket tar in en funktion som parameter. Funktionen i sin tur tar in ett objekt tx, som är ett

transaktionsobjekt. Med hjälp av transaktionsobjektet kan SQLite-kommandon exekveras av funktionen executeSql (Sharp, 2010).

```
var db = openDatabase('webSQL', '1.0', 'WebSQL DB', 2 * 1024 * 1024);
db.transaction(function (tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS WebSQLExample
                   (id  integer primary key autoincrement,
                        Firstname, Lastname)');
});
```

Figur 3.5. *Initiering av en Web SQL databas.*

I figur 3.6 visas ett exempel på hur syntaxen för att addera data till databasen kan se ut, samt hämtning av specifik data genom argument “Firstname = “Web””.

```
db.transaction(function (tx) {
    tx.executeSql('INSERT INTO DATA (Firstname, Lastname)
                   'VALUES (?, ?)', ["Web", "SQL]);
});

db.transaction(function (tx) {
    tx.executeSql('SELECT * FROM DATA where Firstname = "Web"'
                  , [], function (tx, results) {
          var rows = results.rows.length - 1;
          console.log(results.rows.item(rows).Lastname);
    });
});
```

Figur 3.6. *Addering i databasen samt hämtning genom argument, utskrift i konsol.*

3.2.3.1. Styrkor med Web SQL

- Tekniken har generellt bra prestanda, mycket på grund av att det är ett asynkront API.
- Även sökprestandan är optimerad, eftersom data kan indexeras genom söknycklar.
- Strukturen är rigorös, vilket gör underhåll relativt enkelt.
- Web SQL är en transaktionell databas. Det betyder bland annat att ifall det skulle uppstå problem under en transaktion, vid läsning eller skrivning till en databas, skall den återgå till tillståndet innan problemet uppstod (Mahemoff, 2010).

3.2.3.2. Svagheter med Web SQL

- Den stora svagheten är att Web SQL inte kommer att utvecklas vidare och kommer inte att vara inkluderat som en standard.
- Tekniken stöds i de flesta mobila plattformarna, men inte på de generella desktop-varianterna. Även om ingen vidareutveckling kommer att ske är Web SQL fortfarande det enda alternativet som exempelvis Safari stödjer.

- I och med att Web SQL är baserat på SQL krävs det förståelse för SQL, vilket gör att inlärningskurvan för att implementera tekniken i ett projekt kan vara ett hinder.
- Tekniken är strukturkänslig, vilket betyder att det som skall föras in i databasen måste ha en struktur som stämmer överens med det som är fördefinierat.
- Stöds inte av Internet Explorer och Mozilla Firefox (Mahemoff, 2010).

3.2.4. IndexedDB

IndexedDB (IDB) är ett avancerat API för datalagring på klientsidan. IDB använder sig av nyckel-värde principen, vilket innebär att datan som lagras går att identifiera med hjälp av en nyckel. En av de drivande parterna till idén bakom IDB är Oracle. Man ansåg att Web Storage var användbart men simpelt och ineffektivt och att man behövde ett kraftfullare alternativ (Metha, 2009).
IDB ses också som ett alternativ till Web SQL efter att W3C meddelade att det inte finns några planer på att fortsätta att underhålla Web SQL och att man rekommenderar andra datalagringsalternativ (Hickson, 2010).

IDB bygger på NoSQL principen och sparar all data i JavaScript-objekt.
Det finns ingen gräns för hur mycket data man får spara i själva IDB. Däremot finns det olika villkor beroende på vilken webbläsare som används.
Information om vilka webbläsare som stöds kan hittas i Bilaga 2 (s. 35).

3.2.4.1. Transactions, Object Stores, Requests

Transactions, Object Stores (OS) och Request är tre begrepp som är viktiga när man arbetar med IDB. Figur 3.7 visar hur dessa kombineras i en funktion som lägger in data i databasen (Microsoft, 2014).

```
function addTextToIDB(value) {
    var transaction = db.transaction([TextOS], "readwrite");
    var store = transaction.objectStore(TextOS);
    var request = store.add(value);
```

Figur 3.7. *Exempel på hur Transaction, Object Store och Requests flöde fungerar.*

Object Store (OS) fungerar som en tabell med två kolumner, Key och Value, där all data lagras i kolumnen Value. Key används till att identifiera datan man har lagrat. All data som lagras i OS sparas som JavaScript-objekt (Mozilla Developer Network and individual contributors, 2014).

Transaction är ett begrepp som används i databassammanhang. Syftet är bl.a. att isolera databasoperationer. Det innebär att operationerna som utförs i en transaktion sker i, mer eller mindre, en isolerad miljö, vilket gör att man kan

isolera felen som sker i transaktioner från resten av databasen. En transaktion utförs endast om alla operationer i transaktionen är lyckade. Om minst ett fel skulle uppstå i transaktionen avbryts den och alla operationer som hann köras i transaktionen kasseras (Microsoft, 2014).
Alla skriv- och läsoperationer i IDB måste ske i transaktioner. För att skapa en transaktion måste man ange två parametrar. Den första är ett fält med namnen på alla Object Stores som man vill arbeta med. Den andra parametern bestämmer typen på transaktionen. Antingen har den läs- och skrivrättigheter (”readwrite”) eller så kan den endast läsa (”readonly”). Det finns även ett tredje alternativ, ”version_change”, men den är inte aktuell i rapporten och beskrivs därmed inte djupgående (Mozilla Developer Network and individual contributors, 2014).

IDB är ett asynkront API, och den asynkrona egenskapen i IDB bygger på att användaren använder requests. Alla asynkrona metoder returnerar ett request-objekt. Objektet befinner sig till en början i ett “pending”-tillstånd, vilket innebär att transaktionen inte är färdig och alla försök att nå resultaten kommer att resultera i ett InvalidStateError-exception. När transaktionen har exekverats eller misslyckats ändras objektets tillstånd till “done”. När detta sker triggas ett event, antingen success eller error, för att informera om att transaktionen är färdig (Millis, 2013).

I de fall där man inte vill ange fördefinierade nycklar när man lägger till data i databasen, erbjuder IDB två metoder för att dynamiskt skapa nycklar, key generator och key path. Den förstnämnda genererar nycklar i en ordnad sekvens med hjälp av en enkel räknare som ökar för varje objekt som läggs till i databasen. Det andra alternativet key path, använder data från själva objektet för att skapa en nyckel. Exempelvis, när ett objekt med tre variabler (id, namn, ålder) ska sparas, är det möjligt att använda id som key path. Det innebär att alla objekt som sparas kommer att ha värdet som finns i id som nyckel. Vilken sorts nyckelhantering man vill använda anges när man skapar Object Store, det är möjligt att lämna denna parameter tom och istället själv ange nyckel för varje objekt som lagras (Millis, 2013). Figur 3.8 visar hur dessa två metoder för nyckelhantering initieras.

```
var objectStore1 = db.createObjectStore("objStore1",
                                        { keyPath: "id" });

var objectStore2 = thisDb.createObjectStore("objStore2",
                                            { autoIncrement:true });
```

Figur 3.8. *Exempel på två olika Object Stores som använder key path respektive key generator*

3.2.4.2. Styrkor med IndexedDB

- En av styrkorna med IDB är att det är ett asynkront API, vilket innebär att det inte stoppar annat JavaScript från att exekveras i väntan på att databasanropen utförs. IDB använder DOM-events för att meddela applikationen när anropen har genomförts (Millis, 2014). IDB har även ett synkront API, men i nuläget är det inte implementerat i någon webbläsare (Mozilla Developer Network and individual contributors, 2014).
- IDB är en transaktionell databas, vilket innebär att alla operationer måste ske i så kallade transaktioner. Syftet med detta är bland annat att isolera eventuella fel från resten av databasen samt att göra det möjligt för parallella processer att kommunicera med samma databas på ett tillförlitligt sätt (Microsoft, 2014).
- IDB kräver relativt lite kunskap om traditionell databasteknik eftersom den följer en enkel key-value struktur. Det finns avancerade verktyg att tillgå som t.ex. Indexering och Cursors, men det fungerar att bygga en väl fungerande databas utan att implementera dessa verktyg.
- Eftersom IDB är en NoSQL-databas behöver inte utvecklaren ta hänsyn till SQL injections (Millis, 2014).

3.2.4.3. Svagheter med IndexedDB

- IDB har ett komplext API som kräver att utvecklaren är van vid asynkron programmering i JavaScript. Det resulterar i stora funktioner och nästlade callbacks.
- Stöds inte av Safari (Mozilla Developer Network and individual contributors, 2014).

4. FALLSTUDIE

Något man måste ta hänsyn till när man utvecklar mobila webbapplikationer är användarens Internetuppkoppling. Utan Internetuppkoppling kan användaren inte använda applikationen överhuvudtaget, det ger native applikationer en fördel. Ett vanligt fall är när man befinner sig utomlands. Även om det mobila nätet är tillgängligt är det vanligt att turister inte kopplar upp sig på grund av ekonomiska skäl.

Applikationen i det här fallet är tänkt att vara en webbplats för en turistattraktion som lockar besökare från utlandet. Turisterna skall kunna samla så mycket statisk information som möjligt, t.ex. öppettider, så att den finns tillgänglig när Internetuppkoppling saknas.

Informationen som är listad nedan skall användaren kunna komma åt både med och utan Internetuppkoppling.

- Öppettider
- Priser
- Dagens program
- Attraktioner och spel
- Restauranger
- Kontaktinformation

Applikationen kommer gå under namnet Blåa Lund.

4.1. Implementation

I det här avsnittet behandlas den faktiska implementationen av fallstudiet.

4.1.1. Plattformar

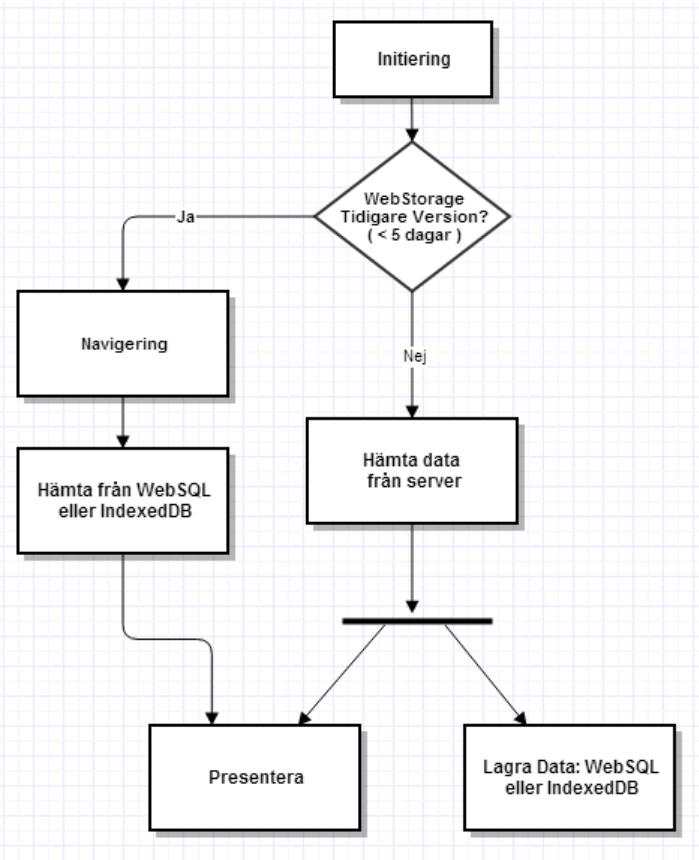
Mycket fokus har lagts på de mobila plattformarna vid implementationen, där typen av operativsystem inte väsentligt påverkar webbplatsens funktionalitet eller utseende. Däremot är det avgörande hur webbplatsen påverkas genom vilken typ av webbläsare som används i störst utsträckning i respektive operativsystem. Därmed har tanken med hela implementationen varit att funktionalitet samt utseende skall vara helt kompatibelt oberoende av vilken plattform som används, vilket även kallas “cross-platform”.

4.1.2. Arkitektur

Arkitekturen i applikationen har inte följt några strikta rekommendationer, eftersom AngularJS har varit det primära ramverket i utförandet har det lett till en stor frihet. AnguarJS baserar sig på “MVW”-arkitekturen. ”MVW” står för “Model, View, Whatever”. ”Model” är den del som notifierar tillhörande ”Views” när det skett en förändring i dess tillstånd. ”View” är den del som begär information från ”Model”, för att kunna generera sin presentation för användaren. ”Whatever” är något som just

utmärker AngularJS, vilket gör att det är väldigt fritt att använda vad som passar utvecklaren, för “Whatever” står för “Whatever works for you” (Google, 2014).

AngularJS har underlättat utvecklingen av applikationen genom de dynamiska webbsidorna som är samlade i den enda statiska sidan index.html. Figur 4.2 visar flödet för hur lagring och presentation av data sker i applikationen.



Figur 4.2. *Flödet av lagring och presentation i applikationen*

Angular: Templates/Views:
Dynamiska webbsidor definieras som så kallade “mallar”, inom skript-taggar; <script type=”template/text”></script>.
Denna typ av mallar är en del av AngularJS. Dessa dynamiska webbsidor är de som visas när man navigerar sig genom webbplatsen. De laddas vid initiering av webbplatsen, vilket gör att navigeringen sker betydligt snabbare.
Teoretiskt sett är webbapplikationen en så kallad “single-page”-applikation, eftersom den endast består av en sida och allt annat sköts genom JavaScript.

Angular: Dependencies

AngularJS är ett relativt stort samt effektivt ramverk, men man behöver inte använda sig av allt som följs med i AngularJS. AngularJS är uppdelat i olika JavaScript-filer. Genom att inkludera en specifik fil samt använda sig av ”dependencies” kan man explicit bestämma vad för extrafunktioner som skall inkluderas från ramverket utöver standard funktionerna. De “dependencies” som var användes i implementationen är ngRoute samt ngSanitize:

ngRoute: För att konfigurera de länkarna som användes för dynamisk sidvisning
ngSanitize: För att försäkra att HTML-strängar som skickas från “controller” till “view” accepteras för rendering.

Angular: $routeProvider

$routeProvider kommer från *ngRoute* för att konfigurera länkningen för de dynamiska webbsidorna. Detta behövs för att kunna dirigera besökaren till rätt “view”, som i detta fall är en dynamisk sida som renderas när man navigerar genom en specifik länk. Om det skulle hända att man försöker besöka exempelvis ”http://blaalund.se/#/oppettider” utan att det finns någon konfigurering för det, kommer besökaren att få upp ett felmeddelande om att sidan inte existerar. Detta betyder att man länkar en specifik “view” till en specifik URL- ändelse.

Angular: Controller:

Informationen i applikationen laddas från servern oberoende från vilken sida man besöker vid initiering. Det har blivit möjligt genom att man definierar en så kallad “Controller” i body taggarna. En “Controller” kan man döpa fritt, i detta fall döptes den specifikt till “initController”. Det gör det möjligt att specificera vad som skall ske i just den specifika “view”. Eftersom body är en central del i applikationen kommer “initController” att exekveras oavsett vilken sida man initieras från. Alltså kommer all data som finns att hämta att ske varje gång man besöker webbplatsen när man har uppkoppling. Datan som hämtas sparas sedan ner i den lagringstypen som är tillgänglig. I detta fall är det antingen Web SQL eller IndexedDB.

Anglar: $scope

I Angular finns det något som kallas $scope och är en definition i AngularJS API. I den här implementeringen användes $scope framförallt för att kunna presentera information för respektive webbsida. Det sker genom att “view” känner till den specifika $scope variablen som exempelvis $scope.webbsida1. Eftersom $scope kan fungera som en global variabel används den i “initController” där alla de olika variablerna initieras på en och samma gång, t.ex. $scope.oppettider, $scope.attraktioner etc. Detta sker vid varje besök av webbplatsen.

När ingen kommunikation med servern finns, hämtas sparad data från det lokala utrymmet.

Anglar: Directives
”Directives” i applikationen gör det möjligt för jQuery Mobile att utföra tema och CSS-ändringar efter att dynamisk data har laddats in i applikationen. Detta för att jQuery Mobile inte är medveten om att DOM:en har ändrats. Därmed används ”Directives” för att notifiera att det har skett en ändring i DOM:en.

jQuery Mobile:
För implementeringen användes även jQuery Mobile för att få mer innehållsrika teman och en mer mobil webbapplikation. Därmed användes en “custom theme” med hjälp av jQuery Mobiles egna temaskapare.

Data: JavaScript Object Notation (JSON):
Varje webbsida har en egen “Controller” som gör det möjligt att koncentrera händelser för den specifika sidan. Respektive “Controller” används när applikationen saknar Internetuppkoppling. Varje webbsida har en egen samling av data som sparas som ett JSON-objekt. Den JSON-data som är relaterad till den specifika sidan hämtas genom ”Controllern” ifall någon tidigare version av sidan finns lagrad.

Bilder: Base64:
En del bilder som visas, finns integrerat i datan i form av bildlänkar som hämtas från severn. På grund av att det är så omständigt att ersätta bildlänkarna med Base64-värden, uteslöts detta moment. Däremot sparas enkla bilder som inte är integrerade i JSON-objekten, som exempelvis logga (Scarfone, 2013).

Webbtjänst
Under utvecklingens gång uppdagades att det befintliga API:et som var avsett för hämtning av data för presentation på webbplatsen inte tillät CORS-anrop. CORS står för “Cross-Origin Resource Sharing” och är en teknik som tillåter resurser på en webbsida att begäras från en annan domän utanför den domän där resursen ursprungligen kommer ifrån. Detta medförde att ingen data kunde hämtas från det befintlig API:et, vilket ledde till utvecklingen av en egen webbtjänst som fungerar som en mellanhand som tillåter CORS-anrop (Shepherd, 2014).

4.1.3. Offline

Det finns tre typer av data som man behöver spara i applikationen Blåa Lund.

- HTML
- JavaScript
- Information om Blåa Lund

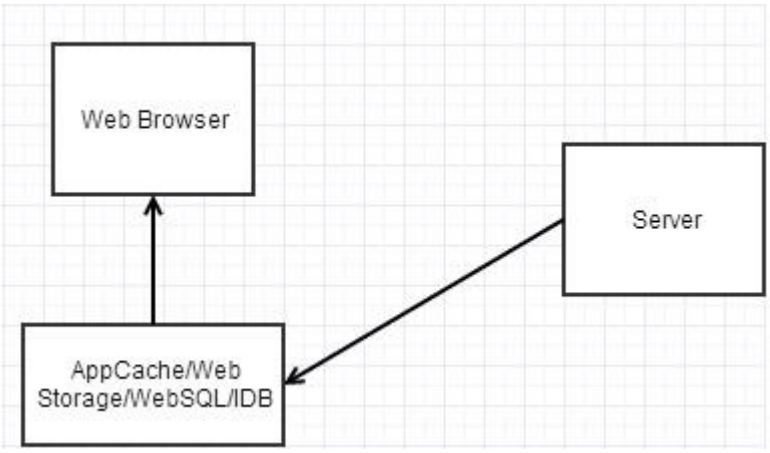
HTML och JavaScript har vi valt att spara i Application Cache för enkelhets skull. Informationen om Blåa Lund som hämtas från en extern webbtjänst sparas i de lokala databaserna IDB och Web SQL. LocalStorage används för att lagra kort information som skall vara lättåtkomlig, t.ex. tid och datum på senaste uppdatering.

Att implementera både IDB och Web SQL är viktigt för att kunna nå ut till så många användare som möjligt. Däremot behöver man inte använda båda databaserna parallellt eftersom de uppfyller samma funktion. I applikationen finns det tre JavaScript-filer som används för att implementera de lokala databaserna:

1. app.js
2. db_indexeddb.js
3. db_websql.js

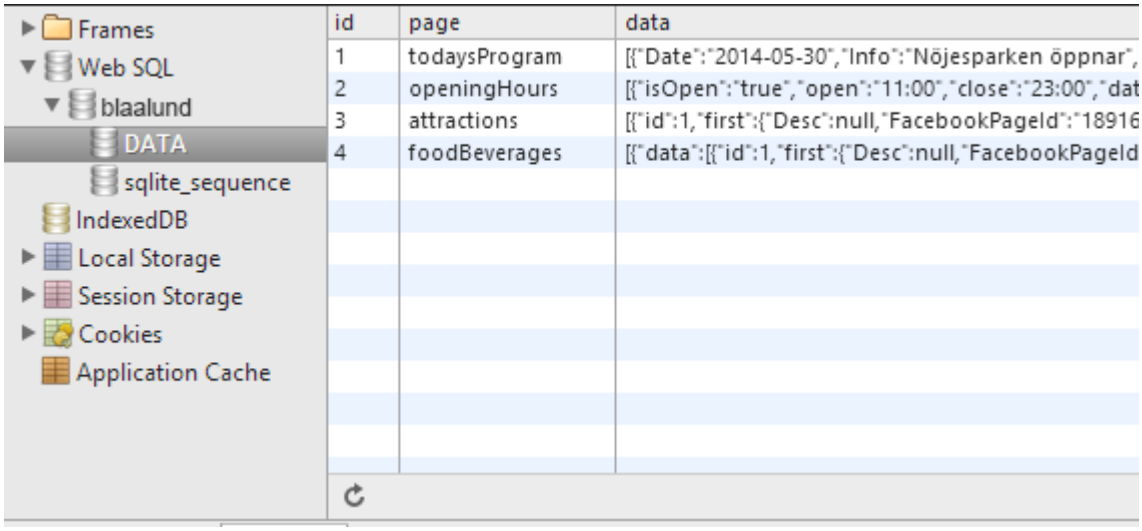
Filerna idb.js och websql.js innehåller JavaScript för respektive databas. Filen app.js sköter kommunikationen med båda databaserna, dock kan den endast kommunicera med en i taget. Därmed måste det bestämmas vilken databas som skall användas vid initieringen av webbplatsen. Vilken databas som skall användas beror på vilken webbläsare som används.

Blåa Lund är byggd så att den hämtar all data till alla sidor i webbplatsen när man laddar sidan första gången. Datan sparas sedan i den databasen som stöds i form av JSON-objekt, som visas i figur 4.3.



Figur 4.3. *Flödet av data från server till webbläsare*

När användaren navigerar bland olika sidor exekveras inga http-anrop eftersom all data redan är hämtad. När Internetuppkoppling saknas i applikationen används istället datan som finns sparad i databasen. I figur 4.4 visas ett exempel på hur datan är lagrad i Web SQL.



Figur 4.4. *Exempel på lagrad data som JSON-objekt i Web SQL*

För att motverka att data blir föråldrad rensas databaserna och uppdaterad data lagras varje gång man besöker webbplatsen med Internetuppkoppling. I detta fall räknas data som är mer än fem dagar gammal som föråldrad. För att avgöra datans ålder används Web Storage genom att spara tidpukten på senaste uppdateringen. Informationen kan användas för att varna användaren om att datan är föråldrad.

Att avgöra om Internetuppkoppling existerar är en viktig del i applikationen. I denna applikation används statusen på *navigator.onLine*, som tillhör HTML5, för att kontrollera om det finns uppkoppling. Tyvärr är detta en funktion som inte riktigt uppfyller de krav som egentligen ställs i produktion, t.ex. ser funktionen inte skillnad på lokal uppkoppling och Internetuppkoppling (Scarfone, 2013). Blåa Lund använder *navigator.Online* eftersom det sker i en kontrollerad miljö och man är medveten om begränsningarna.

4.2. Resultat

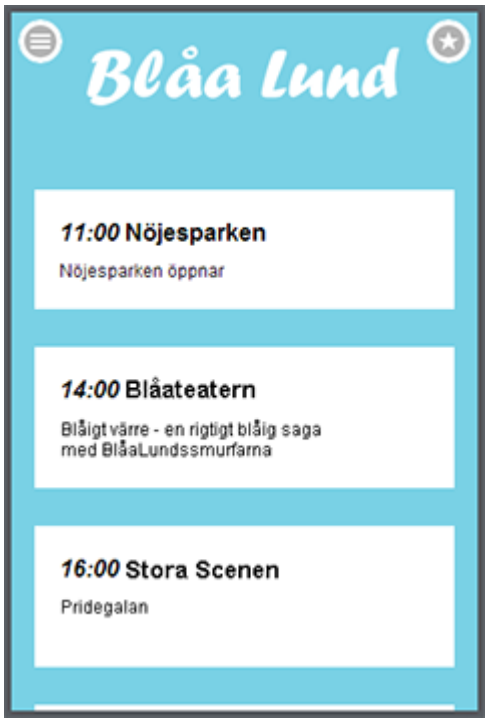
Det som åstadkommits genom implementationen av Blåa Lund är främst att det finns möjlighet att kunna utveckla en webbapplikation som är fullt fungerande såväl offline som online. De aspekter som har tagits i akt under planeringen innan implementationen påbörjades är framförallt; “Vad kan sparas offline?”, “Vad borde man spara offline?”. Dessa frågor har varit avgörande för applikationens utseende och beteende. I figur 4.5–4.13 visas resultatet av applikationen.

Figur 4.5. *Startsidan*

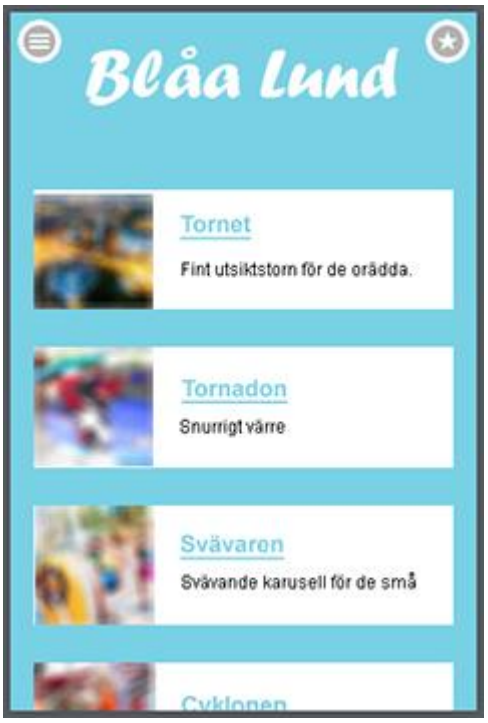
Figur 4.6. *Meny*

Figur 4.7. *Öppettider*

Figur 4.8. *Priser*

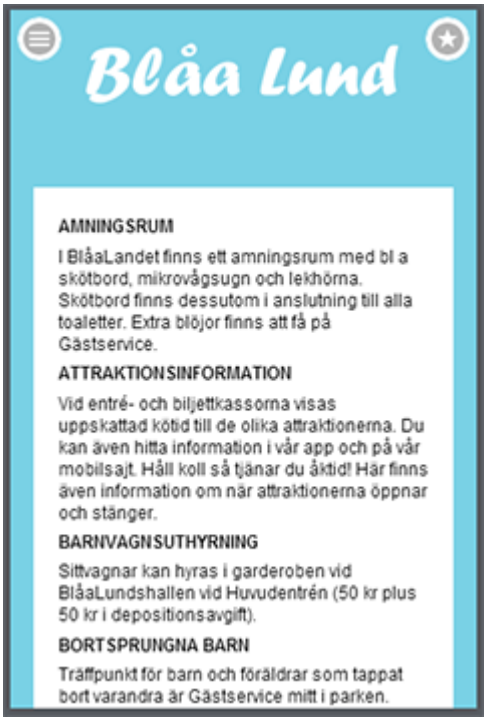


Figur 4.9. *Dagens Program*



Figur 4.10. *Attraktioner*

Figur 4.11. *Mat & Dryck*



Figur 4.12. *Information & Service*

Figur 4.13. *Kontakta Oss*

Att lagra kan vara relativt enkelt, men att spara sådan data som hanteras som färskvara kan vara avgörande för applikationens helhet. Scenarion som exempelvis bokningar är ett av de fall som kan vara opålitliga, eftersom det kan ske kollisioner i bokningar. Ett exempel kan vara när det sker bokningar på två olika enheter som är offline och sedan skickas till servern när en uppkoppling återupptas. Detta kan leda till att dessa bokningar kolliderar med varandra.

5. DISKUSSION

Följande avsnitt avser kritisk diskussion och tankar som uppstått under projektets gång.

5.1. Val av lagringstekniker

Vi valde Application Cache för att lagra filer som är nödvändiga för att webbplatsen skall kunna presenteras överhuvudtaget. De filer som utgör webbplatsens grund är ”index.html”, JavaScript-filer och CSS-filer. Eftersom att risken för att fel uppstår vid cachening beror på antal filer som ska cachas, har vi försökt att minimera antalet filer som skall cachas i Application Cache.

Av de andra lagringstekniker som finns tillgängliga, är Web SQL och IndexedDB de två primära teknikerna som vi har valt att fokusera på. I dessa tekniker har vi valt att spara större datamängder. Vi har även valt att använda oss av Web Storage för att lagra enkel information, t.ex. när datan uppdaterades senaste.

Anledningen till att vi inte valde att avgränsa oss till en av dessa tekniker är att vi försöker nå ut till så många användare och webbläsare som möjligt. Den enda tekniken av dessa tre som är kompatibel med de flesta webbläsarna är Web Storage, men på grund av att dess API är synkron, har det medfört att det inte är något alternativ för att ensamt skapa en interaktiv och responsiv webbapplikation.
Genom ett synkront API skulle laddningen för hela webbapplikationen ta avseendevärt längre tid än vad som anses vara användarvänligt, eftersom varje hämtning kommer att ske iterativt där kön kan bli hur lång som helst beroende på hur många hämtningar som finns.

En annan anledning till att Web Storage inte används till att lagra större datamängder är för att det finns storleksbegränsningar i de olika webbläsarna.
IndexedDB samt Web SQL däremot ger större möjligheter för lagring av större datamängder, till skillnad från Web Storages lagringsutrymme som har en gräns på 2.5 MB på mobila plattformar och 5 MB i desktop.

5.2. Varför valdes ramverket AngularJS?

Anledningen till att vi valde AngularJS är för att det framförallt är ett omtalat ramverk just nu, samt att det är så pass moget i sin utveckling. Det som utmärker AngularJS till skillnad från andra ramverk är att AngularJS ger mycket frihet till utvecklaren. AngularJS ger även möjligheten att skapa en dynamisk webbplats, vilket är något som vi specifikt sökte efter.

5.3. Kontroll av Online/Offline

För att kunna avgöra om en enhet har någon uppkoppling krävs det kontroller. I HTML5 kan man relativt enkelt kontrollera om en enhet är online eller offline med hjälp av JavaScript. Men det finns dock ett antal fallgropar när man implementerar en sådan kontroll, vilket vi stötte på vid implementering av både Laborationsdelen samt Blåa Lund. I skrivbordsmiljö, som är ett av de vanligaste sätten att använda sig av en webbläsare och vara uppkopplad till Internet, kan det uppstå problem med att webbläsaren inte reagerar lika snabbt som på en smartphone när uppkopplingen går förlorad. Detta uppstod under två fall, det första var när uppkopplingen var genom en nätverkskabel, det andra fallet var genom Wi-Fi. Det första fallet visar att responsen när kabeln rycktes ut och anslutningen bröts var relativt långsam samt att det även förekom att den inte reagerade alls. Det andra fallet när Wi-Fi användes, uppstod samma problem när det trådlösa nätverkskortet avaktiverades.

En annan fallgrop vi stötte på vid kontroll av uppkopplingsstatus var att det inte fanns direkt implementation om typ av uppkoppling eller kvalitet på uppkopplingen. Det kan medföra flera nackdelar vid användning av en webbapplikation som har en

nedladdningsbuffert som är så pass stor att nedladdningstiden påverkar användarupplevelsen. I praktiken är det inte så enkelt att man antingen har uppkoppling eller inte. Beroende på sammanhanget kan låga uppkopplingshastigheter vara ekvivalenta med ingen uppkoppling alls.

5.4. Säkerhet & Etik

Att utnyttja användarnas enheter för att lagra data är ett känsligt ämne och vi anser att utvecklare har ett ansvar för sina användare. Innan man börjar dra nytta av möjligheterna som dessa lagringstekniker erbjuder bör man vara medveten om riskerna som medföljer. Att spara känslig data som t.ex. lösenord i de lokala databaserna är inget som man skall göra. En enkel regel som man kan följa är "Data som man i vanliga fall inte hade sparat okrypterat på server-sidan bör aldrig sparas lokalt". Som utvecklare bör man alltid utgå från att det finns någon med onda avsikter som är ute efter att utnyttja sårbarheter i programmet. Fysiska attacker som t.ex. att läsa minnet på en allmän dator i ett Internetcafé, är enkla att utföra och kräver ingen avancerad IT-kunskap.

Webbläsare erbjuder en form av säkerhet genom att inte tillåta kommunikation mellan databaser och webbplatser som inte har samma ursprung. Ursprunget bestäms genom en kombination av protokoll/domän/port, vilket gör att webbplatser inte kommer åt andra webbplatsers databaser. Detta kallas även för same-origin policy (Shepherd, 2014). Även om detta skyddar till en viss nivå, är det fortfarande möjligt att komma åt databaserna genom att utnyttja t.ex. DNS spoofing eller XSS (Hickson, 2010) (Mehta et al., 2013).

Man skall alltid utgå från att datan i databaserna på klientsidan inte är att lita på. Exempelvis, om man väljer att presentera data från databaserna direkt i webbläsaren finns det alltid en risk att man råkar ut för en XSS-attack. Man bör alltid validera datan som finns sparad på klientsidan innan den används i applikationen.

Att veta vad man skall spara i databaserna kan bli en svår etisk fråga, gränsen för vad som är acceptabelt kan variera mycket beroende på användaren. Är det t.ex. acceptabelt att lagra användarens shoppinglista så att den finns sparad inför nästa besök, även om det betyder att andra shoppingwebbplatser har möjlighet att läsa denna? Som utvecklare vill man inte att användarna skall tappa förtroendet för en. Det är få användare som tar reda på vad webbplatser väljer att lagra, och skulle det vara så att de som inte är intresserade av sådan information får höra det i efterhand från en tredje part, då är skadan redan skedd och förtroendet borta.

5.5. Miljöaspekter

Offline First applikationer bidrar inte endast till användarnas upplevelse. Om man tar Blåa Lund som exempel kan man se att det även bidrar till miljön. Genom att ha den statiska informationen i mobila enheter, oavsett om det finns Internetuppkoppling eller

inte, undviker nöjesparken att använda lika mycket papper till att skapa pappersversioner av informationen.

5.6. Rekommendationer för framtida arbete

Det finns oerhört många sätt att kombinera lagringen av data i webbläsare genom de tekniker som finns tillgängliga idag. Under arbetets gång märkte vi att olika utvecklare väljer att kombinera lagringsteknikerna på olika sätt. Det finns ingen standard för vad som är bästa lösningen. En av de alternativa lösningarna som vi kom i kontakt med använde sig av ett så kallat "script loader"-bibliotek i JavaScript som heter Basket.js. Med hjälp av Basket.js kan utvecklaren lagrar JavaScript som strängformat i Web Storage. Biblioteket skapades som ett "proof-of-concept" (Osmari, 2014). Det betyder att det är en realisering av att det går att lagra JavaScript i Web Storage och sedan injicera det i DOM:en. Detta är ett alternativ för att spara JavaScript-filer i Web Storage istället för Application Cache och ett område som man kan undersöka mer genom att testa egenskaper som t.ex. responstider.

De lokala databaserna erbjuder i dagsläget ingen säkerhet utöver standardsäkerheten som finns i webbläsarna. I takt med att utvecklare hittar möjligheterna med de lokala databaserna kommer även angripare att hitta nya möjligheter att attackera dessa. Därför anser vi att säkerhet är ett område som bör utforskas i framtiden.

Att implementera både Web SQL och IDB i en webbapplikation kan kännas redundant eftersom de egentligen fyller samma syfte. Tyvärr är det nödvändigt för att kunna nå ut till alla stora webbläsare. Det finns olika Open Source projekt som erbjuder verktyg för att lösa problemet genom att tillhandahålla ett API som kan skapa Web SQL- eller IDB-databaser beroende på användarens webbläsare. Detta är ett område som vi inte utforskade i rapporten men vi anser att man definitivt borde undersöka om dessa lösningar är kompatibla i produktion.

5.7. Metodkritik

Den metod som användes under projektets gång har varit relativt bra för att kunna slutföra arbetet. Men det finns ett fåtal saker som skulle kunna förändras för att få en förbättrad och effektivare metod. Eftersom det krävdes mycket efterforskning om vad som var relevant för området lades mycket tid på förståelse för vad som var väsentligt och vad som kunde ha uteslutits. Laborationen utfördes i rent inlärningssyfte och det går att diskutera om man inte lika gärna kunde ha börjat på Blåa Lund med en gång.

6. SLUTSATS

Marknaden för webbapplikationer tillsammans med smartphones har växt avsevärt under de senaste åren, vilket har lett till att allt fler företag försöker nå ut till sina kunder genom de mobila plattformarna. Det har medfört att offline har blivit ett omtalat ämne bland de aktörer som har resurser till att gräva djupare inom detta område. Det finns många tillvägagångsätt genom olika webblagringstekniker för att uppnå en fungerande webbapplikation som fungerar både offline och online. De olika teknikerna är kända samt dokumenterade till en viss grad. Men för att kunna utvärdera dessa tekniker behövs mer förståelse om hur de beter sig, vilket kräver en realistisk implementation, därav skrivs denna rapport.

För att kunna börja utvärdera dessa olika lagringstekniker valde teamet att använda sig av ramverket AngularJS som går under en MIT-licens, vilket innebär att den är kostnadsfri. AngularJS ger möjligheten att utveckla en dynamisk webbapplikation för att minimera återladdning av information, vilket ger en bättre prestandaupplevelse.
För lagring av den information som var väsentlig för webbplatsen valdes det att använda en kombination av Web SQL eller IndexedDB, beroende på vilken webbläsare som används, tillsammans med Web Storage.

Inlärningströskeln för de olika teknikerna varierade. Från en skala från hög till låg ligger IndexedDB högst, medel för Web SQL samt Web Storage som lägst. Graderingen beror även på enkelheten av de olika teknikerna.

En avgörande faktor till valet av lagringstekniker var den upplevda prestandan. Eftersom de olika teknikerna beteer sig på olika sätt är det väsentligt att kunna använda sig av rätt teknik för att minimera en dålig upplevelse. De två olika beteendeaspekterna var synkron och asynkron. Dessa två typer av beteende påverkar besökarens upplevelse om datan är stor och komplex.

Teamet tror att framtida utveckling av webbapplikationer kommer att ha ett stort fokus på “Offline First”, eftersom teknikerna som existerar är tillräckligt mogna och kraftfulla för att användas i produktion. Under projektets gång stötte teamet på att par exempel där offline har varit ett scenario som tagits i beaktande. Ett av de exempel är “Financial Times”. Deras webbapplikation är byggd med “Offline First” som grund, vilket ger en inblick i vad som är möjligt samt effektivt. Det gav även en start till teamets egna slutsatser.

Teamet ser även fördelar med webbapplikationer som bygger på “Offline First”, eftersom webbapplikationer som är anpassade för de mobila plattformarna konkurrerar med de applikationer som är av typen native.

Den tid som teamet har lagt på att studera dessa olika tekniker och möjligheter, har gett en bredare förståelse om vad som kan förbättras samt vilka fallgropar man kan hamna i när man arbetar med detta område. Det fanns svårigheter med de problem som uppstod men utifrån teamets perspektiv har det även varit drivkraften till projektet.

Utvärderingstabell av de olika lagrinsteknikerna:

Inlärningströskel LÅG – HÖG	**Upplevd Prestanda** BRA – SÄMRE	**Enhelhet** ENKEL - KOMPLEX	**Mogenhet** -
1. Web Storage 2. Application Cache 3. Web SQL 4. IndexedDB	1. IndexedDB & Web SQL 2. Application Cache 3. Web Storage	1. Web Storage 2. Application Cache 3. Web SQL 4. IndexedDB	De olika lagrinsteknikerna anses mogna för att användas i produktion.

REFERENSER

Referenser i bokstavsordning

Amnell, M. (2014) Semantik - Mathias Amnell.
http://blog.proliit.se/semantik/ (2014-04-28)

Axelsson, M. (2014) Vad är HTML5? | Happiness Webbyrå.
http://www.happiness.se/artiklar/vad-ar-html5 (2014-04-28)

Google. (2014) AngularJS -- Superheroic JavaScript MVW Framework.
http://angularjs.org/ (2014-04-28)

Hickson, I. (2010) Web SQL Database.
http://www.w3.org/TR/webdatabase/ (2014-04-28)

Mehta, N., Sicking, J., Graff, E., Popescu, A,. Orlow, K,. Bell, J. (2013) Indexed Database API.
http://www.w3.org/TR/IndexedDB/ (2014-05-30)

jQuery, The jQuery Foundation. (2014) jQuery Foundation.
https://jquery.org/ (2014-04-28)

jQuery Mobile, The jQuery Foundation. (2014) jQuery Mobile.
http://jquery.com/ (2014-04-28)

Lubbers, P., Albers, B and Salim F. (2010) Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. New York. Apress.

Mahemoff, M. (2010) Client-side Storage - HTML5 Rocks.
http://www.html5rocks.com/en/tutorials/offline/storage/ (2014-04-28)

Menon, R. (2008) Base64 Explained (Ramkumar Menon's Blog).
https://blogs.oracle.com/rammenon/entry/base64_explained (2014-04-28)

Metha, N. R. (2009) WebSimpleDB API.
http://www.w3.org/TR/2009/WD-WebSimpleDB-20090929/ (2014-04-28)

Microsoft. (2014) What is transaction (Windows).
http://msdn.microsoft.com/en-us/library/aa366402(VS.85).aspx (2014-04-28)

Millis, C. (2014) Basic concepts - Web API Interface | MDN.
https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB (2014-04-28)

Mozilla Developer Network and individual contributors. (2014) IndexedDB - Web API Interfaces | MDN.
https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API (2014-04-28)

Osmari, A. (2014) basket.js - a simple script loader that caches scripts with localStorage.
http://addyosmani.github.io/basket.js/ (2014-04-28)

Refnes Data. (2014a) HTML5 Application Cache.
http://www.w3schools.com/html/html5_app_cache.asp (2014-04-28)

Refnes Data. (2014b) JavaScript Introduction.
http://www.w3schools.com/js/js_intro.asp (2014-04-28)

Rouse, M. (2012) What is caching? - Definition from WhatIs.com.
http://whatis.techtarget.com/definition/caching/ (2014-04-28)

Sackemark, M. (2014) HTML5 - Vad är det och varför är det sjukt bra?
http://iveo.se/vad-ar-html5-och-varfor-ar-det-sjukt-bra/ (2014-04-28)

Scarfone, K. (2013) NavigatorOnline.onLine - Web API Interfaces | MDN.
https://developer.mozilla.org/en-US/docs/Web/API/NavigatorOnLine.onLine (2014-05-26)

Sharp, R. (2010) Introducing Web SQL Databases | HTML5 Doctor.
http://html5doctor.com/introducing-web-sql-databases/ (2014-04-28)

Shepherd, E. (2014) HTTP access control (CORS) - HTTP | MDN.
https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS (2014-04-28)

BILAGOR

Bilaga 1: Webbläsarstöd för Application Cache
Bilaga 2: Webbläsarstöd för IndexedDB
Bilaga 3: Webbläsarstöd för Web SQL
Bilaga 4: Webbläsarstöd för Web Storage

BILAGA 1: Webbläsarstöd för Application Cache

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
30 versions back			4.0										
29 versions back		2.0	5.0										
28 versions back		3.0	6.0										
27 versions back		3.5	7.0										
26 versions back		3.6	8.0										
25 versions back		4.0	9.0										
24 versions back		5.0	10.0										
23 versions back		6.0	11.0										
22 versions back		7.0	12.0										
21 versions back		8.0	13.0										
20 versions back		9.0	14.0										
19 versions back		10.0	15.0										
18 versions back		11.0	16.0										
17 versions back		12.0	17.0										
16 versions back		13.0	18.0		9.0								
15 versions back		14.0	19.0		9.5-9.6								
14 versions back		15.0	20.0		10.0-10.1								
13 versions back		16.0	21.0		10.5								
12 versions back		17.0	22.0		10.6								
11 versions back		18.0	23.0		11.0								
10 versions back		19.0	24.0		11.1								
9 versions back		20.0	25.0		11.5								
8 versions back		21.0	26.0		11.6								
7 versions back		22.0	27.0	3.1	12.0			2.1					
6 versions back	5.5	23.0	28.0	3.2	12.1			2.2		10.0			
5 versions back	6.0	24.0	29.0	4.0	15.0	3.2		2.3		11.0			
4 versions back	7.0	25.0	30.0	5.0	16.0	4.0-4.1		3.0		11.1			
3 versions back	8.0	26.0	31.0	5.1	17.0	4.2-4.3		4.0		11.5			
2 versions back	9.0	27.0	32.0	6.0	18.0	5.0-5.1		4.1		12.0			
Previous version	10.0	28.0	33.0	6.1	19.0	6.0-6.1		4.2-4.3	7.0	12.1			
Current	11.0	29.0	34.0	7.0	20.0	7.0	5.0-7.0	4.4	10.0	21.0	33.0	26.0	10.0
Near future		30.0	35.0		21.0								
Farther future		31.0	36.0		22.0								
3 versions ahead		32.0	37.0										

Källa: http://caniuse.com/offline-apps

BILAGA 2: Webbläsarstöd för IndexedDB

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
30 versions back			4.0										
29 versions back		2.0	5.0										
28 versions back		3.0	6.0										
27 versions back		3.5	7.0										
26 versions back		3.6	8.0										
25 versions back		4.0 moz	9.0										
24 versions back		5.0 moz	10.0										
23 versions back		6.0 moz	11.0 webkit										
22 versions back		7.0 moz	12.0 webkit										
21 versions back		8.0 moz	13.0 webkit										
20 versions back		9.0 moz	14.0 webkit										
19 versions back		10.0 moz	15.0 webkit										
18 versions back		11.0 moz	16.0 webkit										
17 versions back		12.0 moz	17.0 webkit										
16 versions back		13.0 moz	18.0 webkit		9.0								
15 versions back		14.0 moz	19.0 webkit		9.5-9.6								
14 versions back		15.0 moz	20.0 webkit		10.0-10.1								
13 versions back		16.0	21.0 webkit		10.5								
12 versions back		17.0	22.0 webkit		10.6								
11 versions back		18.0	23.0 webkit		11.0								
10 versions back		19.0	24.0		11.1								
9 versions back		20.0	25.0		11.5								
8 versions back		21.0	26.0		11.6								
7 versions back		22.0	27.0	3.1	12.0			2.1					
6 versions back	5.5	23.0	28.0	3.2	12.1			2.2		10.0			
5 versions back	6.0	24.0	29.0	4.0	15.0	3.2		2.3		11.0			
4 versions back	7.0	25.0	30.0	5.0	16.0	4.0-4.1		3.0		11.1			
3 versions back	8.0	26.0	31.0	5.1	17.0	4.2-4.3		4.0		11.5			
2 versions back	9.0	27.0	32.0	6.0	18.0	5.0-5.1		4.1		12.0			
Previous version	10.0	28.0	33.0	6.1	19.0	6.0-6.1		4.2-4.3	7.0	12.1			
Current	11.0	29.0	34.0	7.0	20.0	7.0	5.0-7.0	4.4	10.0 webkit	21.0	33.0	26.0	10.0
Near future		30.0	35.0		21.0								
Farther future		31.0	36.0		22.0								
3 versions ahead		32.0	37.0										

Källa: http://caniuse.com/indexeddb

BILAGA 3: Webbläsarstöd för Web SQL

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
31 versions back			4.0										
30 versions back		2.0	5.0										
29 versions back		3.0	6.0										
28 versions back		3.5	7.0										
27 versions back		3.6	8.0										
26 versions back		4.0	9.0										
25 versions back		5.0	10.0										
24 versions back		6.0	11.0										
23 versions back		7.0	12.0										
22 versions back		8.0	13.0										
21 versions back		9.0	14.0										
20 versions back		10.0	15.0										
19 versions back		11.0	16.0										
18 versions back		12.0	17.0		9.0								
17 versions back		13.0	18.0		9.5-9.6								
16 versions back		14.0	19.0		10.0-10.1								
15 versions back		15.0	20.0		10.5								
14 versions back		16.0	21.0		10.6								
13 versions back		17.0	22.0		11.0								
12 versions back		18.0	23.0		11.1								
11 versions back		19.0	24.0		11.5								
10 versions back		20.0	25.0		11.6								
9 versions back		21.0	26.0		12.0								
8 versions back		22.0	27.0		12.1								
7 versions back		23.0	28.0	3.1	15.0			2.1					
6 versions back	5.5	24.0	29.0	3.2	16.0			2.2		10.0			
5 versions back	6.0	25.0	30.0	4.0	17.0	3.2		2.3		11.0			
4 versions back	7.0	26.0	31.0	5.0	18.0	4.0-4.1		3.0		11.1			
3 versions back	8.0	27.0	32.0	5.1	19.0	4.2-4.3		4.0		11.5			
2 versions back	9.0	28.0	33.0	6.0	20.0	5.0-5.1		4.1		12.0			
Previous version	10.0	29.0	34.0	6.1	21.0	6.0-6.1		4.2-4.3	7.0	12.1			
Current	11.0	30.0	35.0	7.0	22.0	7.0	5.0-7.0	4.4	10.0	22.0	35.0	29.0	10.0
Near future		31.0	36.0	8.0	23.0	8.0		4.4.3					
Farther future		32.0	37.0		24.0								
3 versions ahead		33.0	38.0										

Källa: http://caniuse.com/sql-storage

BILAGA 4: Webbläsarstöd för Web Storage

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
31 versions back			4.0										
30 versions back		2.0	5.0										
29 versions back		3.0	6.0										
28 versions back		3.5	7.0										
27 versions back		3.6	8.0										
26 versions back		4.0	9.0										
25 versions back		5.0	10.0										
24 versions back		6.0	11.0										
23 versions back		7.0	12.0										
22 versions back		8.0	13.0										
21 versions back		9.0	14.0										
20 versions back		10.0	15.0										
19 versions back		11.0	16.0										
18 versions back		12.0	17.0		9.0								
17 versions back		13.0	18.0		9.5-9.6								
16 versions back		14.0	19.0		10.0-10.1								
15 versions back		15.0	20.0		10.5								
14 versions back		16.0	21.0		10.6								
13 versions back		17.0	22.0		11.0								
12 versions back		18.0	23.0		11.1								
11 versions back		19.0	24.0		11.5								
10 versions back		20.0	25.0		11.6								
9 versions back		21.0	26.0		12.0								
8 versions back		22.0	27.0		12.1								
7 versions back		23.0	28.0	3.1	15.0			2.1					
6 versions back	5.5	24.0	29.0	3.2	16.0			2.2		10.0			
5 versions back	6.0	25.0	30.0	4.0	17.0	3.2		2.3		11.0			
4 versions back	7.0	26.0	31.0	5.0	18.0	4.0-4.1		3.0		11.1			
3 versions back	8.0	27.0	32.0	5.1	19.0	4.2-4.3		4.0		11.5			
2 versions back	9.0	28.0	33.0	6.0	20.0	5.0-5.1		4.1		12.0			
Previous version	10.0	29.0	34.0	6.1	21.0	6.0-6.1		4.2-4.3	7.0	12.1			
Current	11.0	30.0	35.0	7.0	22.0	7.0	5.0-7.0	4.4	10.0	22.0	35.0	29.0	10.0
Near future		31.0	36.0	8.0	23.0	8.0		4.4.3					
Farther future		32.0	37.0		24.0								
3 versions ahead		33.0	38.0										

Källa: http://caniuse.com/namevalue-storage