HÖGSKOLAN DALARNA

Examensarbete

Kandidatnivå

Förbättrad offline-integration och användning på progressiva webbapplikationer med hjälp av Service Workers

En beskrivning av gränssnittet och dess cachnings-strategier

Improved offline-integration and usage of progressive web applications through Service Workers

Författare: Jonatan Brager Sidén
Handledare: Jörgen Wärngård, William Song
Examinator:
Ämne/huvudområde: Informatik
Kurskod: IK2017
Poäng: 15hp
Examinationsdatum:

Vid Högskolan Dalarna finns möjlighet att publicera examensarbetet i fulltext i DiVA. Publiceringen sker open access, vilket innebär att arbetet blir fritt tillgängligt att läsa och ladda ned på nätet. Därmed ökar spridningen och synligheten av examensarbetet.

Open access är på väg att bli norm för att sprida vetenskaplig information på nätet. Högskolan Dalarna rekommenderar såväl forskare som studenter att publicera sina arbeten open access.

Jag/vi medger publicering i fulltext (fritt tillgänglig på nätet, open access):

Ja ☒ Nej ☒

Sammanfattning

Service Workers är ett nytt applikationsprogrammeringsgränssnitt (API) och är en teknisk lösning som används vid bl.a. offline-integration av webbaserade applikationer. Gränssnittet används primärt för att möjliggöra optimalt stöd och erbjuder utvecklare full kontroll över hur den webbaserade miljön lagrar innehåll och data offline, något som tidigare inte varit möjligt.

En av Service Workers många fördelar är att gränssnittet hanterar push-notifikationer samt visar och meddelar användarna av applikationerna att något har inträffat. Utöver funktioner som förbättrar användargränssnittet och offline-integrationen, skapar gränssnittet även prestanda-, säkerhet- och flexibilitetsförbättringar.

Fördelarna med implementationen av denna nya tekniska lösningen är många. Trots fördelar finns det komplikationer som förhindrar att lösningen kan användas som en teknisk standard vid utvecklandet av webbaserade applikationer. Några av dessa komplikationer är b.la. uppdaterad utvecklarkunskap och kännedom samt kompabilitet i alla webbläsare.

I denna studie ges en ökad förståelse för vad gränssnittet Service Workers erbjuder och redogör hur gränssnittets angreppssätt genom tekniska lösningar kan förbättra en offline-integration av progressiva webbapplikationer samt dess användning.

Resultaten påvisar att Service Workers medför förbättringar som gör att webbaserade applikationer ökar prestandan, säkerheten, tillgängligheten och flexibiliteten för användare. Lösningen medför även många förbättringar för utvecklare, t.ex. full kontroll av offline-integration, ökat plattformsstöd och bakgrundssynkning av operationer.

Nyckelord: Service Worker, LocalStorage, AppCache, Progressive Web Applications, PWA, Caching strategy, Native Applications, Callback hell.

Abstract

Service Workers is a new application programming interface (API) and is a technical solution used in offline integration of web-based applications. The interface is primarily used to enable optimal support and offer developers full control over how the web-based environment stores content and data offline, something that previously was not possible.

One of Service Workers many advantages are that the interface handles push notifications and is a way to inform users of the application that something has occurred. In addition to features that enhance user interface and offline-integration, the solution also present performance, security and flexibility improvements.

There are many benefits of the implementation of this technical solution. Despite the advantages that comes with the implementation, there are a few complications that prevent the solution from being used as a technical standard in the development of web-based applications. Some of these complications are the developmental knowledge and compatibility in all browsers.

This study provides an increased understanding of what benefits that comes with the interface Service Workers, what it provides and explains how the interface's approach through technical solutions can improve the offline-integration of progressive web applications and its use.

The results show that the interface brings forward improvements that make web-based solutions increase application performance, security, accessibility, and user flexibility. The solution also comes with many improvements for developers. Some of these are full control of offline-integration, increased platform support by only developing one application instead of two and background synchronization of operations.

Key words: Service Worker, LocalStorage, AppCache, Progressive Web Applications, PWA, Caching strategy, Native Applications, Callback hell.

Innehållsförteckning

FIGURFÖRTECKNING

BEGREPPSLISTA

API – Applikationsprogrammeringsgränssnitt. En mall för hur kod som används vid utformning av en applikation.

Array – En behållare för objekt, kan presentera data i t.ex. en lista.

DOM – Document Object Model, används för att dynamiskt läsa och uppdatera ett dokuments innehåll, struktur och formatering.

Manifest – En fil som innehåller metadata. En manifest fil kan innehålla data om t.ex. applikationens namn, version samt vilka filer som ska användas vid exekveringen av applikationens kod.

Nästlad – När fler metoder är placerade inom varandra, måsvingar ”}” visar om metoderna är nästlade eller inte.

Native-applikation – En applikation som körs direkt på enheten som en installerad programvara. Nyttjar inte enhetens webbläsarfunktioner.

PWA – Progressive Web Applications (svenskans progressiva webbapplikationer)

1 Inledning

I det inledande kapitlet redogörs bakgrunden för gränssnittets utveckling i samband med förbättringar av progressiva webbapplikationer. Detta efterföljs av en problemformulering och mynnar ut i en frågeställning. I detta kapitel presenteras även studiens syfte och avgränsning.

1.1 Bakgrundsbeskrivning

Utvecklingen av webbapplikationer växer kraftigt. I takt med att nya utvecklingsverktyg framställs, blir teknologin mer avancerad.

Sedan år 1995 har webbutvecklingstekniken successivt bytt riktning från statisk till mer dynamisk (O'Reilly, T. 2009).
Detta innebär att designen tillåter layouten av en webbsida eller en webbaserad applikation att förändras dynamiskt, beroende på skärmupplösning och storlek (Russell, A. 2015).

Apple gjorde ett försök till att få applikationsutvecklingen att bli webbaserad på deras enheter tidigt år 2007 på den årliga Applekonferensen Worldwide Developers Conference (WWDC), i samband med att en ny iPhone och en förbättrad version av webbläsaren Safari släpptes.

Applikationer som var installerade direkt på enheten, s.k. native-applikationer, var en självklarhet att utveckla. Denna utvecklingsteknik medgav en stor förbättring av prestandan och möjliggjorde att applikationerna kunde köras direkt på enheten utan att använda sig av externa lager. Denna förbättring medförde att det inte krävdes någon internetuppkoppling samt att applikationerna gick snabbare att starta. I samband med att förbättringar i utvecklingsspråken HTML5, CCS3 och JavaScript gjordes, växte intresset för att förbättra webbaserade applikationer (Kaplan, J.A, 2010).

Samtidigt som förbättringar i utvecklingsspråk implementerades, växte teknologin för processorkraft och enheternas lagringskapacitet löpande. Detta gjorde att användare kunde starta processorkrävande applikationer snabbare. I samband med att processorkraften ökade, öppnades möjligheter för applikationsutvecklingen, främst för webbaserade applikationer. Förbättringarna i både processorkraft och utvecklingsspråken gjorde att nya utvecklade webbapplikationer blev säkrare, snabbare, pålitligare, flexiblare samt mer tillgängliga.

Traditionella webbapplikationer har lidit utav faktorer såsom opålitlighet, kompabilitet och prestanda och kräver att användare måste ha en fungerande internetuppkoppling för att uppdatera tjänsten i realtid (Russell, A. 2015). Detta är faktorer som länge påverkat användarnas upplevelser av webbapplikationer negativt och har varit anledningen till att man tidigare valt att endast nyttja native-applikationer.

Genom att använda applikationsprogrammeringsgränssnitt (API) ges utvecklarna nya möjligheter att cache-lagra innehåll och data. En ny tekniska lösning som kallas för Service Workers är ett gränssnitt som erbjuder mer än endast offline-integration (Firtman, M, 2012). Gränssnittet och dess cachnings-strategier är grunden för studien och kommer att belysas.

1.2 Problemformulering

Tidigare tekniska lösningar för offline-integration och användning har medfört fler brister än möjligheter. Detta har gjort att utvecklarna istället nyttjar native-applikationer istället för att utveckla webbaserade applikationer.

Fördelarna med att istället köra en webbaserad applikation är många, och utvecklare på företag som Triona AB önskar att de webbaserade applikationerna fungerar lika bra offline som online.
Detta leder till följande frågeställning:

- Hur kan gränssnittet Service Workers förbättra offline-integrationen av progressiva webbapplikationer samt dess användning?

1.3 Syfte

Syftet är att genom studien, ge en ökad förståelse för vad Service Workers erbjuder och redogöra hur gränssnittets angreppssätt genom tekniska lösningar kan förbättra offline-integrationen av progressiva webbapplikationer samt dess användning.

1.4 Mål

Målet är att genom ökad förståelse av Service Workers och dess tekniska lösningar påvisa hur gränssnittet löser tidigare uppkomna problem av offline-integration och användning på progressiva webbapplikationer. Detta är till nytta för alla utvecklare, speciellt de som har intresse att skapa en progressiv webbapplikation, alternativt migrera en native-applikation över till en webbaserad miljö.

1.5 Avgränsning

De tekniska lösningarna som finns för offline-integration och användning i dag på webbaserade applikationer har lämnat en efterfrågan på förbättrad offline-funktionalitet hos utvecklarna. Studien avgränsas till Service Workers och dess cachnings-strategier då gränssnittet betraktas som en ny teknisk lösning och erbjuder ny funktionalitet som tidigare inte funnits.

1.6 Samarbetspartner

Samarbetspartner är Triona AB. Företaget är en IT-leverantör och är etablerade i Finland, Norge och Sverige. De har ca 140 medarbetare totalt, varav 109 anställda är baserade i Sverige, ca 30 i Norge samt 1 anställd i Finland.

Triona har sitt huvudkontor i Borlänge och har teknisk erfarenhet inom branscher såsom skogsindustri, transportinfrastruktur, transport, industri samt entreprenörskap. Triona beskriver själva företaget på sin webbsida enligt följande:

”Triona är en ledande och pålitlig leverantör av innovativa IT-lösningar med anknytning till logistik- och infrastrukturrelaterad verksamhet. Vi kombinerar stort verksamhetskunnande inom framför allt transportinfrastruktur, kraft/energi, entreprenadverksamhet, transporter och skogsindustri med spetskompetens inom systemutveckling och systemförvaltning.”

2. Teoretisk referensram

I detta avsnitt definieras strategi som tillvägagångsätt. Här beskrivs även de beståndsdelar som gränssnittet Service Workers innefattar för att fungera på en progressiv webbapplikation och för att integrera offline-stöd på den webbaserade applikationen. Avsnittet behandlar begrepp som ligger till grund för att uppnå studiens syfte.

2.1 Definition av strategi

Ordet strategi kommer från grekiskans ”strategia” och är en kombination av de antika orden stratos och agein. Betydelsen är från början ”en truppledares konst”. En strategi var i grekiskans krig något som betraktades som en konst genom att välja strider för att vinna (General Carl Von Clausewitz, 1780).

Betydelsen av ordet har sedan dess utvecklats och använts i organisationer. Johnson et al definierade ordet år 2005 som ”en organisations riktning och syfte i långtidsförloppet som åstadkommer konkurrensfördelar i en föränderlig omvärld”. Henry Mintzberg har en liknande definition på ordet, denna är ”ett mönster i en ström av beslut”.

Definitionerna av en strategi skiljer sig åt, men antyder ungefär samma sak. En strategi betraktas generellt som ett vägval – en riktningsbestämning (PerConsus, 2014).
I denna studie är det gränssnittet Service Workers cachnings-strategier som utvärderas. Studiens definition på strategi är den samma som PerConsus, 2014 – ett vägval, en riktningsbestämning, då det är utvecklaren själv som väljer vilken cachnings-strategi som ska nyttjas för att uppnå det ändamål som krävs för den specifika situationen.

2.2 Progressiva webbapplikationer

År 2015 grundandes begreppet ”Progressive Web Apps” utav en av Google Chromes ingenjörer, Alex Russell och designern Frances Berriman.
”Progressive Web Apps” beskrivs som webbapplikationer som fungerar för alla användare, oavsett vilken webbläsare som används och, applikationer som tar ”fördel av nya funktioner med stöd av moderna webbläsare” (Russell, A. 2015). Detta möjliggjorde att utvecklare kunde uppdatera sina befintliga webbapplikationer till progressiva webbapplikationer.

Tanken bakom progressiva webbapplikationer är att nyttja de funktioner som fungerar optimalt hos en native-applikation, och inkludera dessa funktioner i en webbaserad miljö. Progressiva webbapplikationer erbjuder, enligt Google Developers ”en uppslukande helskärmsupplevelse med hjälp av en webapplikations manifestfil och tillåter användare att nyttja push notifikationer”, vilket tidigare endast varit möjligt på native-applikationer. Google Developers påstår att manifestet ger utvecklare möjlighet att fullt kontrollera hur applikationer startas och visas.

Utvecklingen går alltmer mot en progressiv applikationsutveckling som långsiktigt gynnar både utvecklare och användare på olika plattformar världen runt.

Möjligheterna med progressiva webbapplikationer är oändliga och möjliggör att applikationer blir universal på alla plattformar. Eftersom de progressiva webbapplikationerna är webbaserade och körs i en webbläsare, möjliggör de framför allt för användare att nyttja applikationerna på fler enheter, t.ex. en stationär dator. Detta är något som enligt Google Developers exponentiellt ökar det globala användandet av utvecklade applikationer. Utvecklingstekniken har idag en del komplikationer då tekniken grundas på ett web-baserat användargränssnitt.

En av dessa komplikationer är att applikationen inte fungerar fullt ut när användare inte har någon tillgång till internet.
För att applikationerna ska fungera optimalt för användare i alla förhållanden, oavsett plattform, plats och webbläsare, måste de vara tillgängliga och fungera som de ska, oavsett tillstånd. Tekniska lösningar på webbapplikationer som lagrar innehåll och data offline är något som efterfrågats av utvecklare sedan progressiva webbapplikationer anlände på IT-marknaden.

Förbättringar har gjorts av applikationsutvecklare genom satsningar på att utveckla gränssnitt och ramverk vars avsikt är att förbättra progressiva webbapplikationers säkerhet, pålitlighet, kompabilitet och prestanda. Framförallt handlar det om att öka användandet av de progressiva webbapplikationerna genom att integrera offlinestöd och därmed förbättra applikationens tillgänglighet. Dessa lösningar är baserade på applikationsprogrammeringsgränssnitt som har ett huvudmål - att lagra innehåll och data genom cachning direkt i webbläsaren (Archibald, J, 2012).

2.3 Tidigare tekniska lösningar

En tidig lösning av att lagra data för offline-användning utvecklades och lanserades sent år 2009 av Jan Lenhart och kom att kallas *LocalStorage*.

Denna tekniska lösning skapades för att undvika problem som uppstår när ~~en~~ användare tappar internetuppkoppling på sina enheter genom att spara data och innehåll som små bilder via lagring på ett lokalt cache-minne med begränsad lagringskapacitet.
LocalStorage är ett applikationsprogrammeringsgränssnitt som endast stödjer lagring av en färre mängda data, och är inte optimerad för att lagra större innehåll såsom dynamiska bilder, m.m. *LocalStorage* har endast en lagringskapacitet av 5MB och hämtar, konverterar samt spara data manuellt i applikationen. *LocalStorage* används oftast för en session i webbläsaren och rensas så fort att enheten startas om eller stängs av.

Intresset för vidare utvecklad offline-integration på webbapplikationer växer kraftigt. Detta medförde att nya och förbättrade gränssnitt utvecklades.
Application Cache är en senare teknisk lösning och ett gränssnitt som består av ett manifest som är grundat på HTML5. Gränssnittet möjliggjorde att webbaserade applikationer kunde köras offline genom att lagra en större mängd data och innehåll i en lokal cache-fil och löste problem som mindre gränssnitt vilket LocalStorage inte kunde lösa.

Denna cache-fil innehåller ett manifest samt en specifikation av vad för data som ska lagras och består utav en helt vanlig textfil som listar de filer som webbläsaren ska lagra för att en applikation ska fungera utan Internet och är en modifiering av de processer som normalt körs i applikationen.

Gränssnittet påstås ha obegränsad lagringskapacitet på Android-plattformar. Trots detta så har användare inte kunnat lagra mer än 150-300MB data innan ett felmeddelande skickats ut till användaren (Garth, 2014). Tillskillnad från *LocalStorage* har gränssnittet en större lagringskapacitet som möjliggör att större filer och mer data kan lagras. Gränssnittet var en lösning som i teorin fungerade bättre än i praktiken.
Application Cache innefattar funktionalitet som i teorin låter bättre än hur det fungerar i praktiken. Dess största nackdelar ligger i gränssnittets ogynnsamma design. Nackdelar såsom att uppdateringar aktivt inte fungerade samt att valt innehåll eller data som tidigare inte hade lagrats i *Application Cache*-manifestet gjorde att specifikt innehåll på webbsidan försvann, även nästa gång användaren besökte den.

Nackdelarna var för många och blev en besvikelse för både utvecklarna och användarna.

Detta gjorde att utvecklarna slutade använda lösningen då det skapade mer problem än nytta. Den tekniska lösningen Service Workers erbjuder en förbättrad offline-integration och offline-användning än något av de tidigare gränssnitten.

2.4 Service Workers

Service Workers är ett gränssnitt och en nylanserad teknisk lösning som kommer med många förbättringar som ger utvecklarna full kontroll över offline-integration. När denna tekniska lösning lanserades på marknaden blev det genast en häpnadsväckande framgång för progressiva webbapplikationer.

En Service Worker syftar till att förbättra progressiva webbapplikationers funktionalitet långsiktigt och lösa de tidigare gränssnittens funktionella brister och stöd.
Tillskillnad från tidigare tekniska lösningar, erbjuder Service Workers mer än offline-integration, vilket gör att dess användningsområde och flexibilitet har kommit att bli bredare än tidigare lösningar.

Gränssnittet har sedan år 2016 växt fram och betraktas som ett nytt tillskott på marknaden. Det var inte före mitten på år 2017 som webbläsare såsom Google Chrome, Mozilla Firefox, Opera och Safari uppdaterade kompabilitetsstödet för Service Workers, medans konkurrerande webbläsare såsom Microsoft Edge fortfarande kämpar med att hinna med och komma ikapp med utvecklingshärvan. Tidigare har försök gjorts för att utveckla snarlika gränssnitt som till viss del ämnar ge samma nytta som gränssnittet Service Workers täcker. Ett exempel på gränssnitt som riktar in sig på liknande offline integration som Service Workers täcker, är gränssnittet ”Application Cache” som introducerades i samband med HTML5. Detta gränssnitt medförde en rad problem i praktiken som service workers är tänkt att lösa genom att köra asynkrona metoder (metoder som ej är näst i tur i följd i koden), vilket gör att man kan kringgå ett felmeddelande såsom ”inget internet, processen avbryts”. Du kan alltså hoppa över funktioner som är i tidsföljd.

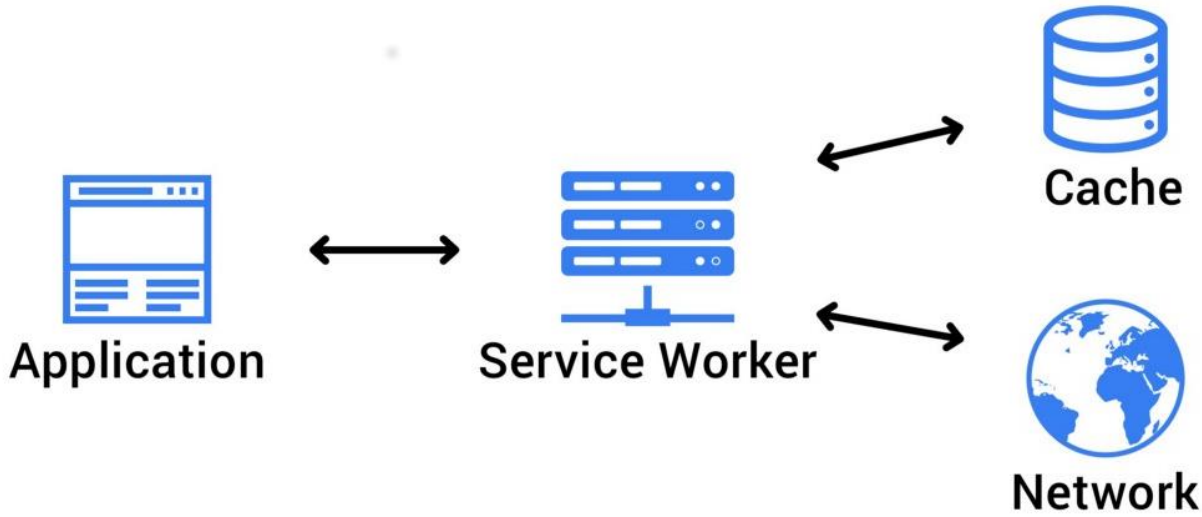
Gränssnittet Service Workers är unikt då det täcker funktionalitet som tidigare gränssnitt försökt lösa, men erbjuder också komplement till dessa lösningar, varav den största integrationen ligger i gränssnittets offlinestöd. Faktum är att Application Cache är ett gränssnitt som betraktas som en äldre version av Service Workers. Application Cache har kommit att avveklas i samband med att Service Workers framställts.

Detta eftersom Service Workers drar samma nytta som Application Cache, men erbjuder en lösning som fungerar bättre, samtidigt som att gränssnittet kommer med flertalet kompletterande funktioner. Utvecklare rekommenderar då istället att använda det nya gränssnittet Service Workers eftersom både uppdateringar och kompabilitetsstöd slutat utvecklats för gränssnittet Application Cache (Firtman, M, 2016).

I tabellen nedan redogörs övergripande fakta om Service Workers.

Övergripande fakta om Service Workers
En Service Worker behöver inte tillgång till applikationens DOM (Document Object Model), utan är ett script som körs i bakgrunden av en webbapplikation.
Körs i en separat tråd från användargränssnittet (UI) så att det inte stör eller fryser gränssnittet under processen för lagring av data, vilket ger snabb responstid vid inläsnings processen. Gränssnittet är en prestationshöjande implementation och avlastar processorn.
Är en mellanhand mellan applikationen och internet (klient och server).
Skickar tillbaka resultat till applikationen via fetch-händelser genom nätverksförfrågningar (requests) till servern.
Kräver en säker uppkoppling (HTTPS – Hypertext Transfer Protocol Secure). Detta förhindrar att fel data skickas i paketförfrågningar.
Beroende på vald cachnings-strategi kan utvecklaren bestämma vilken data och hur denna data ska lagras.
Använder *Promises* som kallar på asynkrona metoder (metoder som ej är näst i tur i följd i koden), vilket gör att man kan kringgå ett felmeddelande såsom ”inget internet, processen avbryts”. Du kan alltså hoppa över funktioner som är i tidsföljd.
Scope är ett objekt som reglerar hur mycket kontroll gränssnittet har i applikationen, och bestäms av vart en Service Worker placeras.

En Service Worker är ett kraftfullt verktyg som ovan nämnt fungerar som en mellanhand mellan applikationen och internet. Nedan illustreras processen i relation till en progressiv webbapplikation (Kinlan, P, 2018).



Figur 1 - Illustration av en Service Workers relation i webbapplikationer (Sessionstack, feb 8, 2018).

Eftersom gränssnittet innefattar ett script som körs i en separat tråd från användargränssnittet är möjligheterna med gränssnittet oerhörda. Det finns inget som begränsar gränssnittet till att endast vara ett verktyg för att lagra data i en lokal cache-fil eller att en Service Worker nyttjas endast för offline-användning av progressiva webbapplikationer. Gränssnittet har kommit en bit i utvecklingen och har ytterligare komma att användas för följande:

- Bakgrundssynkronisering av data.
- Ta emot uppdateringar för att beräkna data för geolokalisering eller gyroskop så att fler sidor kan använda samma uppsättning av data.
- Hantera och placera mallar genom att känna igen vissa URL mönster
- Push notifikationer som skickas till webbapplikationen för att meddela användaren om något specifikt, t.ex. när användaren har tappat internetuppkoppling.
- Geofencing där man kan sätta fördefinierade parametrar, s.k. Geofences som presenterar händelser för specifika områden för användarna av webbapplikationen.

(Mozilla Developers, 2018).
Dessa är ett fåtal användningsområden som Service Workers kan nyttjas inom. Gränssnittet har visat sig ha ett mycket bredare användningsområde och kapabilitet än tidigare gränssnitt någonsin erbjudit. En Service Worker tillåter användaren att exekvera JavaScript i bakgrunden utanför ett HTML dokuments kontext (Mozilla Developers, 2018).

Gränssnittet kapabilitet har använts för webbaserade push-notifikationer, bakgrunds-synkning och fortsättningsvis utvecklats för att, i framtiden komma att bli ett standardramverk. Som framöver kommer att vara ett funktionsstöd för lösningar såsom Geofencing och Beacon discovery. Huvudmålet med gränssnittet var att komplettera tidigare gränssnitt (Application Cache), men har kommit att fungera som en egen lösning och har kommit att bli mycket större än tänkt - nämligen en mellanhand mellan servern och klienten. Framtiden för webbaserade applikationer är runt hörnet och Service Workers är ett kraftfullt verktyg som bjuder in nya möjligheter som tidigare inte funnits.

Google Developers skriver att införandet av progressiva webbapplikationer kommer med otroliga fördelar, varav gränssnittet Service Workers har varit involverat. De presenterar resultat som redogör att gränssnittet möjliggör organisationen Konga att skicka 63% mindre data på första sidladdningarna och 84% mindre data för att slutföra den första transaktionen. Detta medför en betydligt snabbare responstid vid laddningen av webbsidan då mindre data behöver behandlas för att få igenom transaktioner. Fortsättningsvis beskrivs att en progressiv webbapplikation i kombination med gränssnittet hjälpt företag som AliExpress att nå ut till nya användare på plattformar som iOS med en ökning på 82% och nya användare över alla webbläsare med 104%. Service Workers är ett gränssnitt som hjälper progressiva webbapplikationer att bli mer pålitliga, snabba, flexibla och har kommit att bli en viktig beståndsdel i utvecklandet av progressiva webbapplikationer.

2.4.1 Kompabilitetsstöd

En av de största nackdelarna med ett nyutvecklat gränssnitt är kompabilitet. För att försäkra sig om att implementationen fungerar på alla plattformar och i alla webbläsare, krävs det att kompabilitetsstödet konstant kontrolleras genom att släppa kontinuerliga uppdateringar till både plattformar och webbläsare.

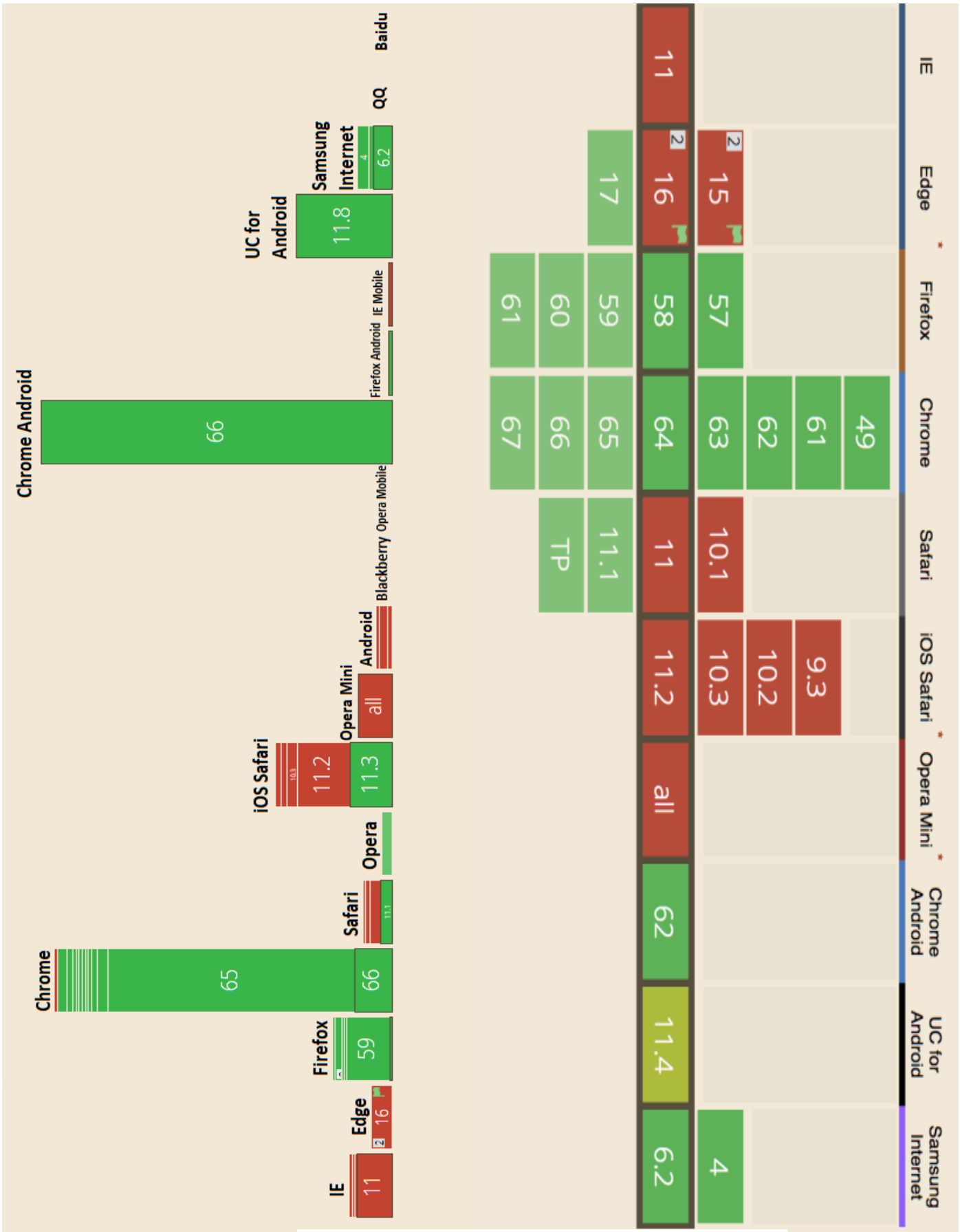
För att se om webbläsaren och dess version är kompatibel med gränssnittet Service Workers, letar man efter en egenskap kallad *serviceWorker*. Finns inte denna egenskap så stödjer inte webbläsaren gränssnittet Service Workers (Sheppard, D. 2017).
Denna egenskap implementerades i webbläsarna Microsoft Edge och Apple's Safari i en version i december, 2017 för tidig testning i en s.k. *early access* (tidig tillgång genom en beta-version) och möjliggjorde att utvecklare för plattformen iOS kunde använda sig av den tekniska lösningen i framtida applikationsutgåvor. Utvecklare för Apples iOS har sedan länge efterfrågat kompabilitetsstöd för gränssnittet. Det var inte först 30 april, 2018 som uppdateringar i dessa webbläsare implementerades på riktigt.

Detta gjordes i samband med att version iOS 11.3 och Safari 13 släpptes för iOS. Vid detta skede har de flesta största webbläsare stöd för progressiva webbapplikationer och Service Workers. Utvecklare har sedan länge tillbaka haft problem med kompabilitetsstöd i webbläsare. Tidigare gränssnitt som Application Cache stod sedan länge inför samma dilemma. Det var inte först Service Workers utvecklades som webbläsare kraftigt investerade i gränssnittets kompabilitetsstöd i hastig takt. Kanske var det för att de flesta utvecklare häpnats av gränssnittets potential och de kraftfulla möjligheter som verktyget erbjuder (Gaunt, M. 2018).

Eftersom att gränssnittet är utvecklat av Google är det ingen överraskning att deras webbläsare Chrome var den första att fullt stödja gränssnittet och att uppdateringar och implementationer fortsätter att utvecklas för webbläsaren. Men att integrera kompabilitetsstöd för Service Workers är inte endast en 1-stegs process, utan kräver att webbläsarna stödjer de olika applikationsprogrammeringsgränssnitt (API) som gränssnittet nyttjar för att överhuvudtaget fungera.

Gränssnittet kräver att webbläsare först och främst måste stödja ett applikationsprogrammeringsgränssnitt som kallas *Promises*. En förklaring på vad *Promises* innebär och erbjuder för gränssnittet beskrivs i kapitel 2.4 – Promises. *Fetch* är ett annat applikationsprogrammeringsgränssnitt som gränssnittet nyttjar för att kunna fungera. Detta för att gränssnittet använder sig utav fetch-händelser för att hämta och lagra data. En förklaring på vad *Fetch* innebär och erbjuder för gränssnittet beskrivs i kapitel 2.5 – Fetch.

Då det är svårt för utvecklare att veta vilka webbläsare som har fullt stöd för gränssnittet Service Workers, har Jake Archibald från Google Chrome's utvecklings-team skapat en hemsida för att sammanställa och lista de versioner av webbläsare som har kompabilitetsstöd för Service Workers. Detta ökar förståelsen över hur långt tillbaka kompabiliteten för gränssnittet stöds och är en säkerhetskälla för utvecklare att använda när den tekniska lösningen önskas att implementeras. Detta presenteras i en kompabilitetstabell och graf som visas nedan.



Figur 2 - Kompabilitetstabell, hämtad från caniuse.com, sökord: Service Workers

2.4.2 Scope

Scope är ett objekt i gränssnittet Service Workers som reglerar hur mycket kontroll det har i applikationen, och bestäms av vart du placerar din Service Worker. Detta är ett sätt för utvecklaren att avgränsa gränssnittets kontroll. T.ex. om Service Workern är placerad i roten av applikationen så har den tillgång till hela applikationen. Om den placeras i en subkatalog kommer den endast att kunna kontrollera den specifika subkatalogen.

Ett *scope* specificeras som en parameter i registreringsfasen av gränssnittet. I kapitlet nedan beskrivs gränssnittets livscykel, där presenteras även ett kodexempel på hur ett scope implementeras i registeringsfasen av Service Workers.

2.4.3 Promises

Framtiden för webbaserade applikationsprogrammeringsgränssnitt (API) består huvudsakligen av Promise-baserade implementationer (Mozilla Developers, 2018).
Promises är ofta vagt definierat som ”*en proxy för ett värde som så småningom blir tillgängligt*” och är ett objekt som representerar eventuella slutföranden eller misslyckande av en asynkron operation och dess resulterande värde. (Bevacqua, N, 2015).
Service Workers är ett gränssnitt som nyttjar *Promises* till stor del, det är viktigt att förstå vad objektet *Promises* gör och hur det fungerar.
Ett Promise är precis vad det låter som – ett löfte.

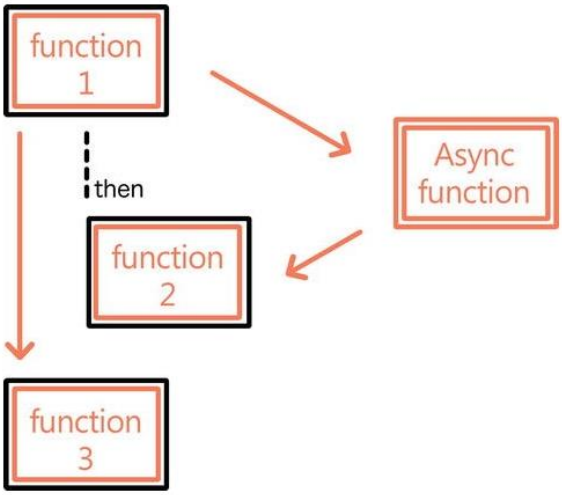
När en applikation gör en API förfrågan går den vidare till nästa kod rad utan att vänta på att förfrågan avslutats. Det finns då ingen mekanism som hanterar och hämtar resultatet av API förfrågan eftersom att nästa kod rad redan exekverats. Tidigare har man löst detta problem genom att använda s.k. *callback-funktioner* som gått tillbaka till förfrågan och hämtat resultatet. Dessa funktioner är svåra att tyda och är oftast nästlade ett flertal gånger, vilket leder till något som kallas för *callback hell*. I exemplet nedan demonstreras hur ett *callback hell* kan se ut.

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

Figur 3 - Exempel på callback hell, tagen från callbackhell.com

Genom att analysera kodexemplet ovan kan ett antal måsvingar ”}” identifieras. Detta gör att vi lättare kan identifiera att koden lider av *callback hell*. Det stora problemet med *callback hell* är att

om något går fel när programmet startas är det svårt att analysera vart problemet ligger eftersom alla funktioner är nästlade i varandra.
Promises löser *callback hell* genom att ge ett löfte till den asynkrona metoden att kalla på funktionen när den har slutförts och är ett sätt att förenkla koden utan att behöva nästla metoderna (Sheppard, D. 2017).

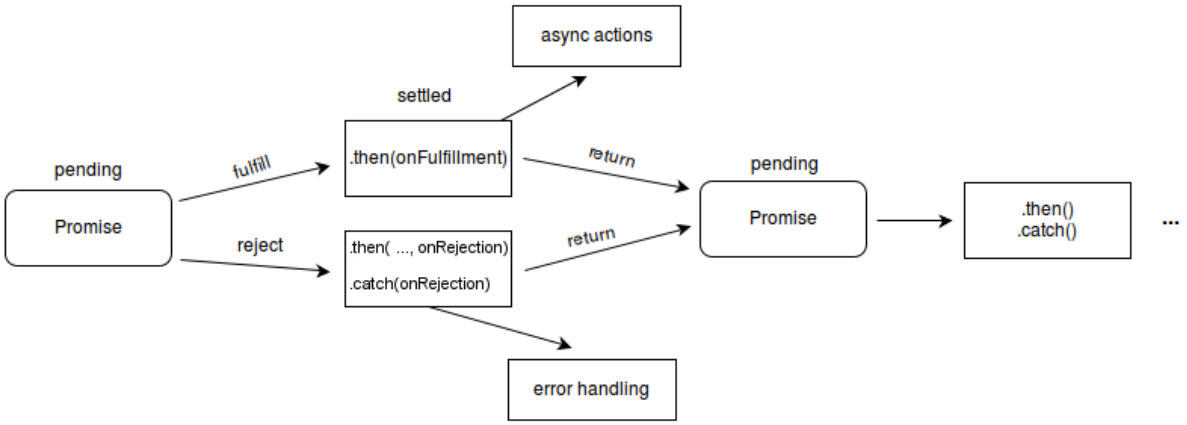


Figur 4 - Execution order when using promises (Sheppard D. s21)

En Promise har tre stycken tillstånd:

- Pending: Första tillståndet, är varken *fulfilled* eller *rejected.*
- Fulfilled: Menas att operationen har slutförts.
- Rejected: Menas att operationen misslyckades.

En väntande Promise kan antingen uppfyllas med ett värde eller avvisas med anledning *fel.*
När något av dessa alternativ sker, sätts de associerade händelserna i en kö som sedan kallas av *then*() metoden i en *Promise*.



Figur 5 - Promises tillstånd, hämtad från developer.mozilla.org

Metoden .then() som visas i exemplet ovan används för att hämta nätverksförfrågans resultat och har en central roll i gränssnittet Service Workers för att få webbapplikationen att fungera offline. Promises kan användas för att ladda bilder dynamiskt, dvs. att bilden laddas redan innan den visas på skärmen. Detta är en av huvudanledningarna till att Service Workers nyttjar Promises för att förbättra offline-användningen på progressiva webbapplikationer.

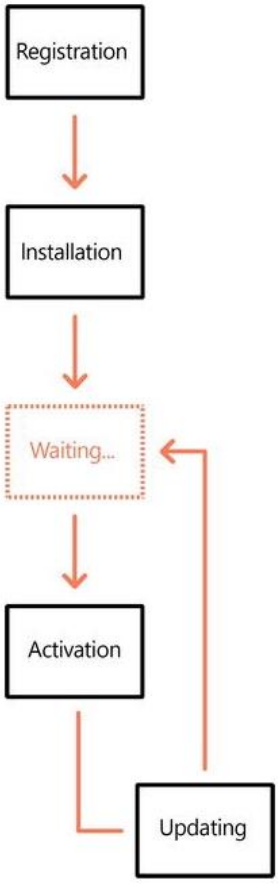
2.4.4 Fetch-händelser

I tidigare avsnitt behandlades *Promises* som är en essentiell komponent för att få gränssnittet Service Workers att fungera. För att dessa löften som utlovas att köras i koden genom att använda Promises ska fungera, krävs s.k. Fetch-händelser. Som tidigare nämnt i avsnitt 2.4.1 – kompabilitetsstöd är det viktigt att webbläsaren stödjer både *Promises* och *Fetch-händelser* då dessa applikationsprogrammeringsgränssnitt är beroende av varandra för att kunna fungera. *Fetch-händelser* är ett applikationsprogrammeringsgränssnitt som tillåter utvecklare att göra nätverksförfrågningar som returnerar löften – *Promises*.

2.4.5 Gränssnittets livscykel

För att förstå varför gränssnittet kräver kompabilitetsstöd för *Promises* och *Fetch*-händelser i webbläsare för att överhuvudtaget fungera, presenteras gränssnittets livscykel så kortfattat som möjligt.

Service Workers har en livscykel som är helt separat från webbsidan eller webbapplikationen. Denna livscykel består huvudsakligen utav tre faser. Dessa tre faser är registrering, installation och aktivering. Nedan finns en visualisering av livscykelns processer (hämtad från Beginning Progressive Web App Development, Creating a Native App Experience on the Web)



Figur 6 - Service worker lifecycle (Sheppard D, s.27)

Registrering är den första fasen Service Workers går in i för att fungera på webbapplikationen. Detta görs i webbapplikationens JavaScript dokument och görs genom att implementera koden nedan.

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
    navigator.serviceWorker.register('/sw.js').then(function(registration) {
      // Registration was successful
      console.log('ServiceWorker registration successful with scope: ', registration.scope);
    }, function(err) {
      // registration failed :(
      console.log('ServiceWorker registration failed: ', err);
    });
  });
}
```

Figur 7 - Kod för registrering av Service Workers, tagen från developers.google.com

Första koden som visas i exemplet ovan visar att gränssnittet kontrollerar om applikationsgränssnittet *serviceWorker* finns installerad i webbläsaren. Detta fungerar som en kompabilitetskontroll så att gränssnittet vet om det överhuvudtaget stödjs av webbläsaren. Om gränssnittet inte stödjs avbryts registreringen omedelbart genom att ett felmeddelande skrivs ut i webbläsarens öppna fönster enligt koden ovan.

Genom att kalla på metoden *register()* specificeras den katalog som gränssnittet installeras i. I exemplet ovan är gränssnittet installerat i huvudfilen */sw.js*. Gränssnittets *scope* sätts genom en vanlig console.log som inkluderar parametern *registration.scope*.
Efter att gränssnittet registrerats är det dags för nästa process – installation. Denna installation styrs utav en install-händelse och ser ut enligt följande:

```
self.addEventListener('install', function(event) {
  // Perform install steps
});
```

Figur 8 - Kod för installations initiering av Service Workers, tagen från developers.google.com

Vid installeringsprocessen kan utvecklare välja vilket innehåll som ska lagras på det lokala cacheminnet genom att specificera de sidor som ska läsas in. Detta görs genom att kalla på en metoder som heter *caches.open()* och *cache.addAll().* För att kunna kalla på metoden cache.addAll() krävs det att en array skapas, där de specifika sidor som ska lagras på det lokala cacheminnet läggs in. Detta görs via följande kod:

```
var CACHE_NAME = 'my-site-cache-v1';
var urlsToCache = [
  '/',
  '/styles/main.css',
  '/script/main.js'
];

self.addEventListener('install', function(event) {
  // Perform install steps
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(function(cache) {
        console.log('Opened cache');
        return cache.addAll(urlsToCache);
      })
  );
});
```

Figur 9 - Kod för fullständig installation av Service Workers, tagen från developers.google.com

I exemplet ovan skapas en *array* vid namn *urlsToCache*, det är i denna *array* som specifika sidor som ska läsas in vid installationen läggs in.
Som beskrivet i avsnitt 2.4.3 och 2.4.4 använder gränssnittet sig utav *Promises* och *Fetch-händelser* för att hämta resultatet utav en nätverksförfrågan istället för tidigare tekniska lösningar såsom gränssnittet Application Cache som istället använt sig utav *callback-funktioner*.

När installationen av gränssnittet färdigställts kan utvecklare välja att returnera resultatet om så önskas via en *Fetch-händelse*, i exemplet nedan beskrivs den kod som behöver implementeras för att få resultatet presenterat.

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request)
      .then(function(response) {
        // Cache hit - return response
        if (response) {
          return response;
        }
        return fetch(event.request);
      }
    )
  );
});
```

Figur 10 - Kod för hämtning av resultat från cachning, tagen från developers.google.com

I händelsen *respondWith()* läggs ett *Promise* in för att ta emot och läsa av nätverksförfrågan som skickats av en Service Worker. Detta görs via *caches.match()* metoden. Denna metod tar emot nätverksförfrågan och presenterar det lagrade innehållet som specificerats i en *array* som beskrivet i installationsfasen av Service Workers.

När gränssnittet registrerats, installerats och specifikt innehåll lagrats i det lokala cacheminnet, aktiveras gränssnittet automatiskt.

Den sista fasen i en Service Workers livscykel är uppdatering. Denna fas är något som utvecklare själv väljer om de vill nyttja sig av. Ett exempel på när en uppdatering av Service Workers kan behöva nyttjas är om utvecklare vill lagra mer innehåll än det som tidigare lagrats på det lokala cacheminnet. T.ex. om en ny html sida skapats och behöver fungera offline.
Uppdateringar görs genom att modifiera och uppdatera JavaScript filen till den senaste versionen. Webbläsaren laddar då ner *script-filen* i bakgrunden. Finns det någon som helst förändring i scriptet, klassar Service Workern filen som ny (Google Developers, 2018).

2.5 Service Workers cachnings-strategier

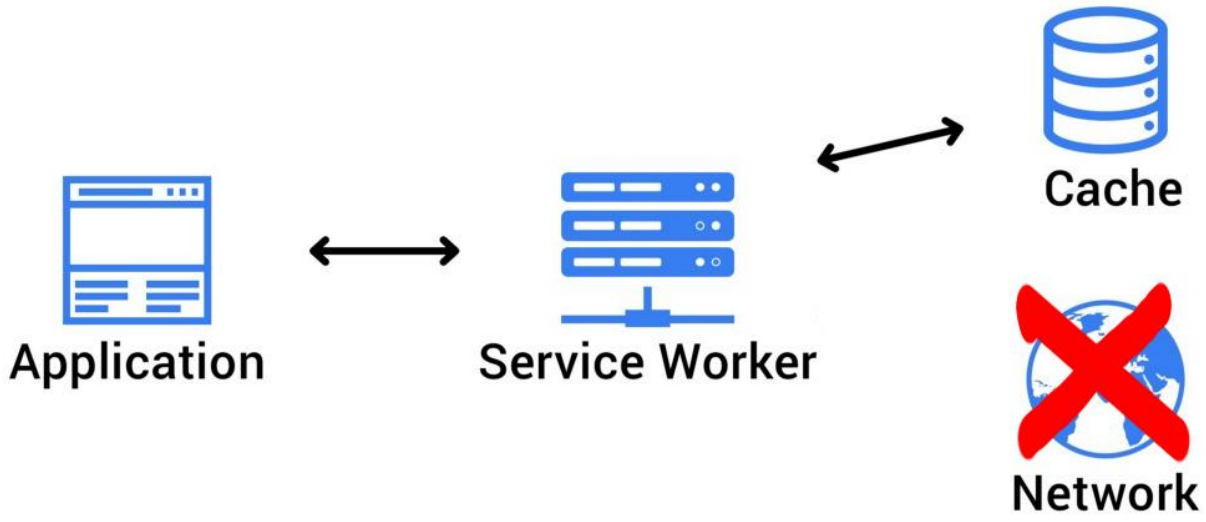
Hur och vad för innehåll som ska visas när webbapplikationerna används offline, bestäms av utvecklare genom att använda sig av olika strategier för hantering av webbapplikationens lagrade innehåll. Det är genom dessa strategier en Service Worker och dess tillvägagångsätt för datalagring sätts upp och kan kontrolleras genom en fördefinierad strategi som gränssnittet nyttjar när internet saknas.

Nedan redgörs cachnings-strategierna inom Service Workers för att förstå de olika strategierna och hur de hanterar datalagring offline på webbapplikationer. Detta bidrar till kunskap för utvecklare så att de vet vad och när de olika strategierna kan användas.

2.5.1 Cache only

Cache only är en strategi som huvudsakligen riktar sig in på statiskt innehåll på en webbsida. Det statiska innehållet kan t.ex. vara grundpelarna eller skalet på webbsidan. Detta betraktas som statiskt för att en webbsidans skal aldrig dynamiskt ändras, dvs. att innehållet alltid finns på samma plats, har samma form, färger, m.m. Strategin används främst för statiska bilder, HTML dokument, sprite sheets och CSS filer.

När strategin Cache only används, når aldrig förfrågningarna nätverket eller webbservern, utan skickas istället direkt till en Service Worker som presenterar och lagrar innehållet från en lokal cache-fil. Detta gör Service Workern genom att svara på anrop från den lokala cache filen genom *Fetch-händelser*, vilket tillhandahåller information om hur innehåll hämtas. *Fetch-händelserna* tar hand om nätverksförfrågningar och hur svaret på dessa förfrågningarna ska behandlas.



Figur 11 - Illustration av strategin Cache only

2.5.2 Network or cache

Network or cache strategin grundar sig på att Service Workers placerar sig mellan klienten och Internet. Det är en strategi som kan ses som en backup strategi för att se till att en webbapplikation alltid kommer att presenteras för användaren.

Genom att använda Network or cache strategin kan webbapplikationen alltid återgå till en tidigare version av webbapplikationen om nätverksförfrågningen tar för lång tid. Network och cache strategin erbjuder en s.k. *timeout* eller tidsbegränsning för nätverksförfrågan att gå igenom och nå Internet. Om förfrågan inte går hela vägen fram, visas istället en tidigare version av webbapplikationen från det lokala cacheminnet.

Strategin används främst när uppdateringar och förändringar av webbapplikationen görs för att se till att användaren alltid blir presenterad med en webbapplikation, även om uppkopplingen inte fungerar optimalt. Så snart att uppkopplingen fungerar igen kommer dessa uppdateringar och förändringar att presenteras för användaren (De la Puente, S, 2016).

2.5.3 Cache and update

Cache and update strategin presenterar tillgångar från det lokala cacheminnet, men sänder också nätverksförfrågningar för att uppdatera dess innehåll. Strategin erbjuder en snabb responstid och uppdateringar av ikoner, bilder och avatars och bör i de flesta fall användas för mindre beståndsdelar på en webbapplikation. Enligt (De la Puente, S, 2016), utvecklare på Mozilla Developers, bör denna strategi undvikas att användas för tillgångar som behövs för att få hemsidan att fungera, t.ex. en hemsidas skal eller användargränssnitt (UI), men bör användas för endast till för små oberoende tillgångar såsom bilder och ikoner. Dessa förändringar träder i kraft först nästa gång användaren besöker webbapplikationen.

2.5.4 Cache, update and refresh

Med Cache, update and refresh strategin meddelas klienten av en Service Worker att det finns innehåll som behöver uppdateras då ändrar har gjorts av applikationsutvecklaren. Detta ger användaren ett alternativ att kunna vänta med att uppdatera webbapplikationen om internet skulle fungera dåligt eller inte finnas på den befintliga platsen som användaren befinner sig på. Användaren kan istället välja att uppdatera webbapplikationen så fort att förhållandena är bättre och tillåter att en stabil internetuppkoppling finns. Istället visas då den tidigare versionen av webbapplikationen tills att användaren väljer att uppdatera.

Denna strategi kan användas för att uppdatera allt innehåll på en webbapplikation och är ett sätt att hämta nytt innehåll i bakgrunden av applikationen utan att presentera de nya uppdateringarna direkt. Applikationen kommer istället att vänta med att visa uppdateringarna tills att nätverksförfrågningen går igenom och en internetuppkoppling finns.

2.5.5 Embedded fallback

Denna strategi erbjuder en Service Worker att alltid visa en tidigare version av webbapplikationen om internet inte finns tillgängligt. Strategin är lik Network or cache, men fungerar lite annorlunda. Då Network or cache strategin endast visar tidigare version av webbapplikationen om en nätverksförfrågan tar för lång tid att nå Internet kan detta påverka applikationens responstid på det uppdaterade innehållet.

Embedded fallback erbjuder istället en snabb responstid genom att alltid presentera den tidigare versionen av användargränssnittet för användaren, utan att behöva vänta på att nätverksförfrågan nått fram till Internet. Embedded fallback strategin bortser då istället helt från nätverksförfrågningar

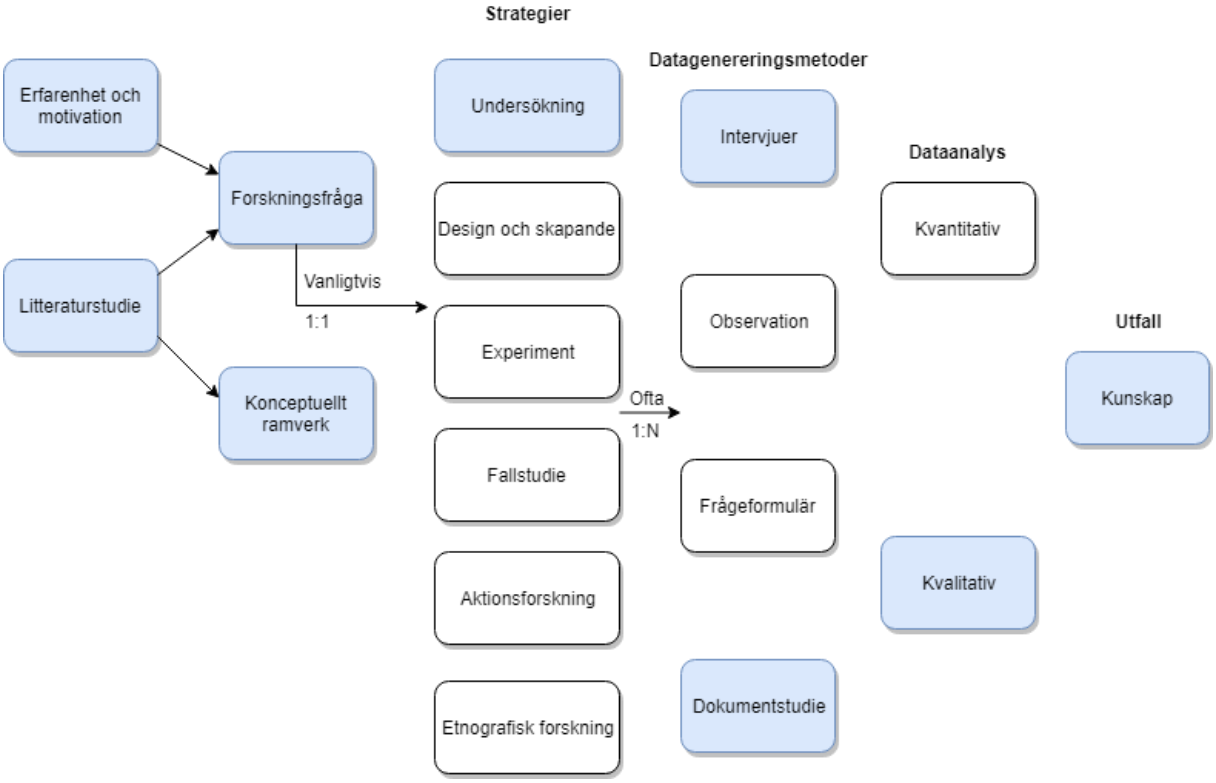
så länge att ingen uppkoppling finns. Genom att använda denna strategi säkerställs att en webbapplikation alltid visas, oavsett vilken förändring utvecklaren har implementerat. Network or cache strategin ämnar sig bättre för användare som återfår uppkopplingen efter en tid, t.ex. tågresenärer, medans Embedded fallback garanterar att en tidigare version alltid visas inom loppet av några millisekunder.

3 Metod

I detta avsnitt redogörs hur studien har genomförts. Detta redogörs genom att beskriva vilka datainsamlingsmetoder, strategier, dataanalys och datagenereringsmetoder som använts för att forma studien.

3.1 Forskningsprocessen

En illustration av de processer som gåtts igenom under examensarbetets gång presenteras genom forskningsprocessen nedan (se figur 12). Forskningsprocessens första steg beskriver utgångspunkten för studien som är baserad på erfarenheter genom insamlad information från medarbetare på Triona AB. Motivationen till examensarbetet grundas på att få en djupare inblick om hur en teknisk lösning kan reda ut befintlig problematik inom datalagring offline. Ämnet har dessutom efterfrågats av uppdragsgivaren att undersöka då tekniken är ny och kan komma att gynna uppdragsgivarens utvecklare och kunder långsiktigt. En litteraturstudie genomfördes som gav underlag till ett konceptuellt ramverk. Detta konceptuella ramverk har varit som grund till litteraturen och formade i sin tur examensarbetets forskningsfråga.
Dokumentstudier och intervjuer har använts som datagenereringsmetod för att framställa data baserad på utvecklares erfarenheter och för att få en djupare inblick i ämnet. Genom kvalitativ dataanalys har data tagits fram för att uppnå syftet med arbetet och lättare kunna besvara frågeställningen. Genom att använda forskningsprocessen ledde detta till ett kunskapsbidrag.



Figur 12 - Forskningsprocessen, överblick, tagen från Oates (2006)

3.2 Litteraturstudie

”The litterature review provides the foundation for your research” är ett påstående skrivet av Oates (2006, s. 73). Som vägledning under examensarbetets gång har detta påstående legat som grund till arbetet. En djupare litteraturstudie har genomförts för att uppnå kunskapsbidraget om ökad förståelse inom studiens forskningsfråga. En litteraturstudie är enligt Oates (2006, s. 71) uppdelad i två delar, där den första delen beskrivs som en fas där studenten samlar relevant information för att få en djupare förståelse och forma en forskningsfråga.
När forskningsfrågan har framställts genom insamlad information granskas de utvalda källor och dess trovärdhet som ligger till grund för litteraturstudiens teoretiska referensram (Oates 2006, s. 83). Detta används för att främja forskningens kunskapsbidrag.

Litteraturstudien har mestadels använt sig av relevanta vetenskapliga artiklar som hittats på Internet. Dessa har hittats genom att använda sökmotorer såsom Google.com, Google Scholar samt Summon. Nyckelord såsom Service Worker, LocalStorage, AppCache, Progressive Web Applications, PWA, Caching strategy, Native Applications och Callback hell har använts.

För att få fler träffar och ett bredare resultat har engelska sökord använts istället för svenska. Tryckta källor har använts till stor del för att få inblick i gränssnittet Service Workers beståndsdelar och uppbyggnad. Fördelen med de tryckta källor som använts över de digitala är att de givit arbetet en bättre illustration på de aspekter som källan redovisar.

Redan i början av studiens utformning var det tydligt att det saknades vetenskapligt granskade källor för att säkerhetsställa att informationen är korrekt. Gränssnittet är en nyutvecklad teknisk lösning som inte verkar ha nått ut till alla utvecklare ännu. Anledningen till att fler elektroniska än tryckta källor har använts i studien är för att det finns väldigt begränsat antal tryckt litteratur som kan användas till grund för studien.

3.3 Forskningsstrategi

En forskningsstrategi är enligt Oates (2006, s. 35) ”den allmänna approachen för att svara på forskningsfrågan”. Hon beskriver de sex forskningsstrategierna som kan användas för att svara på forskningsfrågan i en studie. Dessa sex strategier är *undersökning, design and creation, experiment, fallstudie, aktionsforskning* och *etnografi*.
Utav dessa sex strategier har studien nyttjat strategin *undersökning*.

Denna strategi har använts för att utföra en undersökning kring gränssnittet Service Workers. Strategin används dessutom för att undersöka om gränssnittet förbättrar offline-integrationen och användningen på progressiva webbapplikationer.
Oates (2006, s. 35) beskriver en undersökning som något som ”fokuserar på att samla in snarlika data från en bred skara människor (eller events), på ett standardiserat och systematiserat sätt. I denna insamlade data hittas mönster genom att använda statistik så att en större population kan generaliseras istället för den grupp som arbetet riktar sig in på”.

I undersökningen har statistik för kompabilitetsstöd tagits fram. Mönster i generaliserbara data som är relevanta för forskningsfrågan har påvisat att det finns ett stort behov av ökad kunskap i nya tekniska lösningar för att förbättra offline-användningen på progressiva webbapplikationer. Studiens undersökning är speciellt riktad till utvecklare som utformar dessa progressiva webbapplikationer för att ge ökad förståelse för vad gränssnittet kan förbättra och möjliggöra.

3.4 Datainsamlingsmetod

I detta avsnitt beskrivs de datagenereringsmetoder som använts under studiens gång. Dessa metoder har använts för att säkerhetsställa att studiens syfte uppnås.
Gränssnittet är en nyutvecklad teknisk lösning som sedan endast några månader tillbaka har blivit lanserad på IT-marknaden. Det finns begränsat med information, vilket leder till att insamlingen av vetenskaplig information och data genast blir mer begränsad och svåråtkomlig.
Studien har använt sig av tre olika datagenereringsmetoder. Detta för att få en bredare kunskap inom forskningsfrågan och säkerhetsställa att tillräckligt med relevant information kunnat samlats in.

3.4.1 Genomförande

Arbetet har genomförts genom att skapa en förståelse kring gränssnittet Service Workers och dess nyutvecklade funktionalitet som löser tidigare lagringsproblem. Detta har gjorts genom att läsa tidigare dokumenterade studier, webbartiklar och genom intervjuer.
En del av det skrivna materialet som använts som utgångspunkt i studien har hittats både i tryckta och elektroniska böcker.

Boken “Beginning Progressive Web App Development: Creating a Native App Experience on the Web” av Dennis Sheppard (2017) har använts till stor del för att hitta information om gränssnittets utveckling och för att få skapa en förståelse om den tekniska lösningens användningsområde.
För att få en ökad förståelse av tidigare gränssnitt och hur de försökt lösa samma problem till gränssnittet Service Workers, har webbaserade artiklar använts som utgångspunkt då det inte finns någon tidigare vetenskaplig litteratur som behandlar gränssnittet, som tidigare nämnt i 3.4 (Datainsamlingsmetod).

Webbaserade artiklar har också använts för att granska gränssnittets kodstruktur och ge exempel på hur Service Workers och dess strategier skapas, vilket beskrivs inom den teoretiska referensramen (avsnitt 2). Utifrån webbaserade källor har figurer för att lättare visualisera gränssnittets struktur, livscykel, kompabilitet och beståndsdelar såsom *Promises* och *asynkrona metoder* (se avsnitt 2.4.3).
Till sist har intervjuobjekt tagits fram på Triona AB av företaget genom en mindre enkätundersökning. Intervjuerna har använts för att få en generell kunskap från respondenter baserad på tidigare erfarenheter.
Respondenterna har spetskompetens inom webbutveckling och deras kunskap är relevant för studien. Detta gör att studien kan förhållas till verkligheten. En mindre enkätundersökning gjordes och skickades ut av Triona AB på deras interna forum där anställda på företaget kan se och svara på information internt. Detta gjordes innan studien påbörjades för att finna intervjuobjekt som har erfarenhet inom forskningsämnet. Detta säkerhetsställde att relevant information kunde samlas in under intervjuernas gång och att personer med rätt kunskap intervjuades.

3.4.2 Dokumentstudier

Studien har till stor del använt sig av relevanta vetenskapliga artiklar på Internet via dokumentstudier. En dokumentstudie definieras som ”alla symboliska representationer som kan lagras och återhämtas för analys”. Detta kan användas som ett alternativ till intervjuer, observationer och frågeformulär och delas in i två stycken typer av dokument*, hittade dokument och forskningsgenererade dokument*.
Hittade dokument är dokument som redan existerar och finns i organisationer. Detta kan vara t.ex. produktionsbeskrivningar, redovisningar, manualer, rollbeskrivningar m.m.
Forskningsgenererade dokument är de dokument som är skapade för att uppfylla ett visst syfte för forskningsfrågan (Oates, 2006, s. 233).
Dokumentstudier har framförallt använts för att generera insamlade data för gränssnittet Service Workers och dess cachnings-strategier.

3.4.3 Intervjuer

Intervjuer har använts som sekundär strategi. Detta eftersom att material om gränssnittet är väldigt begränsat. Detta resulterar i att det inte finns tillräckligt med kunskap att hämta från dokumentstudier. Att använda intervjuer som sekundär datagenereringsmetod för datainsamling är ett effektivt sätt att få trovärdig och relevant information baserad på erfarenhet. Det är också ett sätt att samla in detaljerad information och fånga erfarenhet och känslor som annars kanske inte kan bli observerade och nerskrivna (Oates, 2006, s. 187). Oates (2006) beskriver även att det finns tre typer av intervjuer, strukturerade intervjuer, semi-strukturerade intervjuer samt ostrukturerade intervjuer. *Strukturerade intervjuer* är intervjuer som har en tydlig struktur och ej ändras för varje respondent. *Semi-strukturerade intervjuer* är strukturerade intervjuer som ändras beroende på flödet av konversationen i intervjuerna.

Ostrukturerade intervjuer är intervjuer som inte har någon struktur alls. Respondenten är istället den som styr konversationen och utformningen av frågor genom att prata om forskningsämnet.
I denna studie har både strukturerade och semi-strukturerade intervjuer använts. Strukturerade intervjuer användes för intervjuer över Skype, medans de semi-strukturerade intervjuerna användes när det fanns möjlighet att ställa frågor vid personlig kontakt med respondenterna.
Det genomfördes totalt sju stycken intervjuer med utvecklare som hade erfarenhet inom webbapplikationsutveckling med offline-integration och där intresse för nya tekniska lösningar inom forskningsämnet fanns. Dessa intervjuer hade ett tidsspann mellan 45 och 60 minuter per intervju.

Majoriteten av respondenterna är personer som arbetar som utvecklare på företaget Triona AB. Denna strategi valdes för att ha möjlighet att ställa följdfrågor och få intryck av utvecklares reaktioner på frågorna under intervjuernas gång.

3.5 Dataanalys

En dataanalys kan antingen vara kvalitativ eller kvantitativ enligt Oates (2006), där ”kvantitativa data är baserad på bevismaterial eller nummer”, medans kvalitativa data ”inkluderar ord, bilder, ljud, osv”.

En kvantitativ dataanalys innebär att man analyserar statistiska och numeriska data, tillskillnad från en kvalitativ dataanalys som behandlar en studies syfte genom t.ex. ord, bilder och ljud. Det är också data som genereras via intervjuer, forskningsdokument, företagsdokument, webbsidor och utvecklingsmodeller (Oates, 2006, s. 266).

Denna studie är en kvalitativ studie då studiens data till stor del kommer från tidigare dokumentstudier. Den data som framtagits via dessa dokumentstudier är data som inte innehåller statistiska och numeriska beräkningar, utan beskriver istället beståndsdelar, uppbyggnad, användningsområde och metoder som används som tillvägagångssätt i gränssnittet Service Workers och dess cachnings-strategier.

4 Resultat och analys

I detta kapitel redogörs det empiriska resultat som tagits fram utifrån litteraturstudier och intervjuer för att uppnå studiens syfte. Dessa resultat analyseras genom tolkning och förklaring.

4.1 Push-notifikationer

I litteraturen framgick det att tidigare utvecklade tekniska lösningar som t.ex. *Application Cache* skapade frustration hos både utvecklarna och användarna. Ett frustrationsmoment var att användarna aldrig meddelades när internetuppkopplingen tappades. Istället visades en blank sida med texten ”Unable to connect to the Internet”.
Respondenterna i de personliga intervjuerna relaterade till denna frustration och bekräftade att push-notifikationer är något som generellt förbättrar användarupplevelsen på progressiva webbapplikationer.

Studien påvisar att Service Workers kan lösa detta genom att använda push-notifikationer som presenteras i användargränssnittet och meddelar användarna att uppkopplingen bröts. Push-notifikationer är något som överlag förbättrar användarupplevelsen av progressiva webbapplikationer genom att meddela användaren när någonting gått fel. Implementationen av push-notifikationer har i tidigare tekniska lösningar för progressiva webbapplikationer inte funnits och är ett nytt angreppssätt för att säkerhetsställa att användarna blir presenterade med information om händelser som inträffar på webbapplikationen.

4.2 Asynkrona metoder och bakgrundssynkning

Asynkrona metoder genom *Promises* som gränssnittet Service Workers behöver för att fungera, visas i studien motverka att webbapplikationerna stannar. Kompilatorn som exekverar webbapplikationernas kod visade sig att förstå att en metod som kräver uppkoppling inte kan genomföras när uppkoppling saknas. Ett löfte om att exekvera koden när uppkoppling finns, som skapas upp via *Promises* medför att användarna fortfarande kan använda applikationen, som att den vore native. Dessa löften som består av asynkrona metoder beskrivs som ett effektivt sätt att undgå att applikationen havererar.

Tidigare gränssnitt har påvisats medföra opålitlighet genom att applikationen slutar fungera när kompilatorn inte kan exekvera koden. Bakgrundssynkning beskrivs genom litteraturstudier och personliga intervjuer som en funktionell förbättring inom gränssnittet som hör ihop med asynkrona metoder genom att köra operationer i bakgrunden för att sedan utföra dessa när möjlighet finns (t.ex. när uppkoppling återfås). Genom att använda asynkrona metoder löser Service Workers denna opålitlighet som tidigare gränssnitt påvisats medföra.

Resultatet är en förbättrad användarupplevelse offline och en fullt funktionell applikation trots att kompilatorn upptäckt problem vid exekvering av koden.

4.3 Säkerhetsförbättring

Service Workers användning av nätverksprotokollet HTTPS beskrivs av respondenterna i de personliga intervjuerna ge en ökad säkerhet av webbapplikationer, genom kryptering. Ett säkrare nätverksprotokoll medför att applikationer som tidigare endast kunnat vara native pga. säkerhetsrisker, kan migreras över till en helt webbaserad miljö. Applikationer som kan nyttja denna tekniska lösning är bl.a. finansiella applikationer som t.ex. Handelsbanken, Swedbank m.fl., men också applikationer som innehar betaltjänster, t.ex. iTunes, AliExpress, m.fl. De säkerhetsförbättringar som kommer med Service Workers gör att applikationers tillit ökar hos användare.

4.4 Ökad kontroll av offline-integration

De olika cachnings-strategier som gränssnittet Service Workers erbjuder för utvecklare ligger som grund till forskningsfrågans delfråga i studien.
Service Workers och dess cachnings-strategier löser tidigare tekniska lösningars problem med lagring offline genom att ge utvecklarna flexibilitet och mångsidighet när det kommer till lagring offline. Detta gör att webbapplikationernas offline-integration kan formas utifrån webbapplikationens användningsområde och målgrupp.
”Fördelen med Service Workers är att du kan styra nätverksförfrågningarna genom att säga hur den ska cacha, när den ska cacha” säger Erik Olofsson (system och webbutvecklare på Triona AB).
Som ovan nämnt av respondenten, kan nätverksförfrågningarna styras individuellt. Detta görs genom att använda Service Workers cachnings-strategier vid integrationen av offlinestöd på progressiva webbapplikationer och ger möjlighet till att anpassa hur webbapplikationen fungerar offline hos användare under specifika situationer. En cachnings-strategi ger utvecklare valmöjligheten att berätta för gränssnittet hur, vad och när innehåll eller data ska lagras. Nyttan med dessa cachnings-strategier är att lagringen av innehåll eller data är fullt kontrollerad. Detta innebär att valmöjligheter alltid ges för att välja om det ska lagras i det lokala cache-minnet, om webbapplikationen ska återgå till senast laddade version och om push-notiser ska meddela användare att en ny uppdatering finns.

Utvecklarna har med gränssnittet Service Workers och dess cachnings-strategier möjlighet att på en helt ny nivå kontrollera lagringen av webbapplikationens data och innehåll. Detta innebär i sin tur att offline-användningen kan kontrolleras.

Ett exempel är att om webbapplikationen specifikt är riktad mot tågresenärer, kan utvecklaren välja att nyttja cachnings-strategin *Cache, update and refresh*. Användaren kan då vänta med att uppdatera webbapplikationen om internet skulle fungera dåligt eller inte finnas på den befintliga platsen. Genom strategin kan användaren ges möjlighet att använda en tidigare version av webbapplikationen då enheten redan lagrat och läst in hela webbsidan.

Detta ger både utvecklare och användare kontroll. Om användare istället inte har tillgång till Internet på flertalet timmar då de arbetar ute i skog där internetuppkoppling är obefintlig, kan t.ex. strategin *Embedded fallback* användas. Detta innebär att webbapplikationen alltid återgår till en tidigare version utan att ta hänsyn till hur lång tid en nätverksförfrågan tar på sig att nå servern.
Alla dessa strategier går att passa in på olika funktioner i en webbapplikation samt användares situationer och krav vid användning. Service Workers är också ”den enda lösningen för att styra nätverksåtkomst på låg nivå från webbläsaren”. Utvecklarna kan även via Service Workers välja att nyttja flertalet olika strategier för att kontrollera olika delar av webbapplikationen.

”Du kan ha network or cache för ett nyhetsflöde samtidigt som du har cache and update för sidans skal. Det är också det som är styrkan i service workers”, säger Johan Nyström som jobbar med webbutveckling på företaget Tretton37 i Borlänge.

5 Diskussion

Efter att ha gjort sju stycken personliga intervjuer kan ett generaliserbart mönster hittas.
Alla respondenter är utvecklare med flera års erfarenhet inom IT-branschen. De har alla olika erfarenheter när det kommer till offline-integrering på webbapplikationer. Något som de flesta har erfarenhet inom är gränssnitt som heter *IndexDB* och *WebSQL*. Dessa tekniska lösningar har inte redogjorts i studien, utan framkom sent under intervjuerna. Gränssnitten verkar vara en väl använd teknisk lösning som specifikt har använts inom företaget Triona AB. Det finns en mängd olika tekniska lösningar för offline-integration, och att behandla och redogöra alla är omöjligt och skulle kräva mer tid och arbete.

Application Cache valdes istället som jämförande gränssnitt för att det är en tidigare utvecklad version av Service Workers. Service Workers byggs dessutom utifrån Application Cache och är en ersättare av gränssnittet.
Då gränssnittet är så pass nytt på IT-marknaden var det svårt att få tag på respondenter som har teknisk erfarenhet inom forskningsämnet. Resultatet från intervjuer och dokumentstudier visar att nyfikenhet hos utvecklare finns, och att förbättringar krävs för att de ska välja progressiva webbapplikationer över native.
Respondenterna i intervjuerna beskriver problematiken med att utveckla en progressiv webbapplikation, men ser också en hel del fördelar med att göra applikationen webbaserad.

En av de respondenter som intervjuades berättar att det finns både för- och nackdelar med en webbaserad applikation, en av de fördelar som kommer med webbaserade applikationer som som nämns i intervjun är att native-applikationer gör att ”gränssnittet blir plottrigt och är svårt att få utseendet att se likadant ut på alla plattformar”. Han menar att eftersom native-applikationer är utformade efter specifika plattformar, måste fler versioner av applikationen utvecklas. Kort sagt, en för varje plattform. Det blir då svårt att få gränssnittet att se likadant ut på alla plattformar.

Fortsättningsvis berättar han att det finns många aspekter som spelar stor roll vid utvecklingen av en applikation och avgör om den ska vara webbaserad eller ej. Han nämner detta när han talar om de för- och nackdelar som kommer med progressiva webbapplikationer. Respondenten säger ”Ur ett utvecklarperspektiv: Det är enklare att utveckla, kräver inte att man är specialist på ett specifikt språk – gör att det blir lättare att hitta utvecklare för företag. Fungerar på alla telefoner och enheter”.
En aspekt att besinna är t.ex. sekretess på progressiva webbapplikationer, det nämner även respondenten under intervjun.
Han säger ”Ur ett användarperspektiv: Handelsbanken appen t.ex. skulle inte tyckas vara bra att köra PWA på, utan föredrar att den är native och installerad på enheten. Beroende på vad det är för typ av app. Man blir dessutom begränsad att använda webbläsare, och sekretess kan spela stor roll”.

Han nämner också att om det i framtiden inte skulle märkas någon skillnad i utvecklingsteknikerna så är progressiva webbapplikationer att föredra pga. ”utökade funktioner såsom animationer, samma utseende oavsett vilken plattform, att det responsivt och snabbt. Offlinestöd, snabb lagring av data till webben, komma åt flera funktioner såsom sensorer såsom kamera m.m. Då vet man att det finns ett gemensamt gränssnitt för GPS, kamera, osv”.

Teknikens utveckling går fort fram och det är inte lätt att ständigt vara uppdaterad och påläst kring nya tekniska lösningar. Eftersom de flesta respondenter inte hade någon erfarenhet av gränssnittet Service Workers och vad det erbjuder, utan endast hade hört talas om dess existens, finns en begränsad kunskap om att många av de problem och nackdelar som kommer med progressiva webbapplikationer som nämns i intervjuerna redan är lösta genom implementationen av gränssnittet Service Workers.

6 Slutsats

Hur kan gränssnittet Service Workers förbättra offline-integrationen av progressiva webbapplikationer samt dess användning?

Studien har visat att offline-integrationen och användandet av webbapplikationer tidigare har medfört fler nackdelar än nytta. Genom att implementera Service Workers, kan utvecklandet och användandet av progressiva webbapplikationer exponentiellt öka. Gränssnittet Service Workers och dess funktion som en mellanhand mellan server och klient ökar flexibilitet och mångsidighet när det kommer till offline-integration på en webbaserad applikation.

Men för att denna tekniska lösning ska bli standardiserad att använda och ~~vara~~ en självklarhet vid utvecklingen av webbaserade applikationer, måste kompabilitetsstödet öka i webbläsare på olika plattformar.

Slutsatsen från studien är att även om gränssnittet Service Workers förbättrar progressiva webbapplikationer genom tekniska förbättringar såsom push-notifikationer, offline-användande, sekretess (genom att använda HTTPS), är det fortfarande en nyutvecklad teknisk lösning som få har kännedom om. Det krävs att utvecklare är uppdaterade på nya tekniska lösningar, vilket är svårt med dagens snabba teknikutveckling.

Framtidens applikationer är runt hörnet, och det är endast en tidsfråga innan gränssnittet Service Workers möjliggör att webbaserade applikationer tar över native-applikationer genom att förbättra kompabilitetsstöd, offline-integration, offline-användning, applikationens säkerhet, prestanda och pålitlighet.

6.1 Metodreflektion

Oates (2006) skriver att dokumentstudier inte kan åberopas för att ge en objektiv bild av verkligheten.

Detta kändes till i utformningen av studien och är anledningen till att intervjuer användes som komplement för att påvisa erfarenhet ur en verklig kontext och för att bekräfta att information genom dokumentstudier stämde.

Då det fanns begränsat material om Service Workers fanns inga andra användbara metodalternative för att samla in data för att besvara studiens syfte och problemformulering.

6.2 Förslag på fortsatt studie

Något som kan användas till motiv för fortsatta studier i framtiden är övergången från native-applikationer till progressiva webbapplikationer för utökat plattformsstöd med hjälp av Service Workers.

Det saknas idag kunskap och kännedom hos utvecklare om både progressiva webbapplikationer samt Service Workers. Detta då det tekniskt sätt inte varit möjligt att tidigare garantera samma användbarhet av applikationer genom att kräva en internetuppkoppling.

Litteraturförteckning

Archibald, J. (8 maj 2012). *ApplicationCache is a douchebag*. Hämtad från http://alistapart.com/article/application-cache-is-a-douchebag den 20 april 2018.

Archibald, J. (25 mars 2014). *Is Service Worker ready?* Hämtad från https://jakearchibald.github.io/isserviceworkerready/ den 21 april 2018.

Bevacqua, N. (28 september 2015). *ES6 Promises in Depth*. Hämtad från https://ponyfoo.com/articles/es6-promises-in-depth. Hämtad 5 maj 2018.

Bevacqua, N. (10 november 2018). *Making a Simple Site Work Offline with Service Worker*. Hämtad från https://css-tricks.com/serviceworker-for-offline/ den 12 april 2018.

Callback hell. (4 februari 2016). *What is callback hell?* Hämtad 22 april 2018, från http://callbackhell.com/

De la Puente, S. (19 oktober 2016). *Offline strategies come to the Service Worker Cookbook.* Hämtad från https://hacks.mozilla.org/2016/10/offline-strategies-come-to-the-service-worker-cookbook/. den 5 maj 2018.

Firtman, M. (11 april 2016). *Service Workers replacing AppCache: a sledgehammer to crack a nut*. Hämtad från https://medium.com/@firt/service-workers-replacing-appcache-a-sledgehammer-to-crack-a-nut-5db6f473cc9b. Hämtad 16 april 2018.

Gaunt, M. (29 mars 2018). *Service Workers: An Introduction*. Hämtad från https://developers.google.com/web/fundamentals/primers/service-workers/ den10 april 2018.

Google Developer. (2018). *Progressive Web Apps*. Hämtad 16 april 2018, från https://developers.google.com/web/progressive-web-apps/

Garth. (2014, 15 april). *Developing a Cross-Platform HTML5 Offline app – Part 1*. Hämtad från http://grinninggecko.com/2011/02/24/developing-cross-platform-html5-offline-app-1/ den 20 april 2018.

Kaka, F. (4 maj 2018). *Service Worker API*. Hämtad från https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API den 7 maj 2018

Kinlan, P. (9 april 2018). *Adding a Service Worker and Offline into your Webb App*. Hämtad från https://developers.google.com/web/fundamentals/codelabs/offline den 6 april 2018.

Kaplan, J.A. (29 april 2010). *No Flash on the iPhone? Apple's Steve Jobs Finally Explains Why.* Hämtad från http://www.foxnews.com/tech/2010/04/29/flash-iphone-apples-steve-jobs-finally-explains.html den 17 april 2018.

Mills, C.D. (26 mars 2018). *Using Service Workers*. Hämtad från https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers den 6 april 2018.

Mozilla, Serviceworke. (13 april 2018). *Service Workers. Caching strategies.* Hämtad från https://serviceworke.rs/ den 4 maj 2018.

Mozilla Developers. (18 april 2018). *Promise – Javascript.* Hämtad från https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. den 5 maj 2018.

Oates, J. B. (2006*). Researching Information Systems and Computing*. London: SAGE Pulications Inc.

O'Reilly, T (2009) *What is Web 2.0? Design Patterns and Business Models for the Next Generation of Software*. O'Reilly Media, Inc.

Progressive Web Apps. (2018, 10 maj). *I Wikipedia*. Hämtad 15 april 2018 https://en.wikipedia.org/wiki/Progressive_Web_Apps

PerConsus. (29 mars 2014). *Vad är strategi?* Hämtad från http://www.perconsus.se/strategi/vad-ar-strategi/ den 6 maj 2018.

Russel, A. (15 juni 2015). *Progressive Web Apps: Escaping Tabs without Losing Our soul*. Hämtad från https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/ den 15 april 2018.

Sheppard, D. (2017). *Beginning Progressive Web App Development: Creating a Native App Experience on the Web.* Apress.

BILAGOR

Intervju #1 (System och webbutvecklare på Triona AB Stockholm)

1. **Vad heter du och hur gammal är du?**
   Respondent 1, 41 år gammal

2. **Kan du berätta lite om vad du jobbar med på Triona?**
   Lite olika projekt, har jobbat mycket med Ragn Sells med deras mobila ordersystem, mobila klienter som de kan nå ut i deras olika typer av biler såsom sopbilar och transporter. De får ut ordrar och kan se vad det är som ska göras, och säga vad som är gjort, godkänna ordern, osv. Samla in information om vad de har gjort.
   Håller även på med system för fastighetsvärdering. Kommer med från ett uppdrag på ett företag som han jobbade tidigare. Utvecklingar en WPF (Windows Presentation Foundation) webbapplikation som ska hålla koll på fastigheters värderingar, m.m. Har jobbat för Arla lite med ett system som samlar in positioner från deras leveranser och transporter. GPS positioner och matchar deras positioner mot deras planerade rutter och kör ordrar. Hur bra i tid de har varit och leveranskvalitet.

3. **Vilket område jobbar du inom? Skogsindustri? Transportinfrastruktur? Transporter? Industri? Entreprenörskap? Annat? Isåfall vad?**
   Transport och transportinfrastruktur.

4. **Finns det något du tycker är extra roligt inom området du jobbar med?**
   Webbutveckling främst då tekniker går fram väldigt mycket och snabbt fram, javascript är kul och molntjänster går fort fram i utvecklingen. Webbutveckling är kul för att det känns som att det är värt att investera tid i att lära sig alla lösningar/tekniker.

5. **Vad för erfarenhet har du av offlinestöd på applikationer?**
   I ragnsells synkade han data lokalt i intern databas via ett webb interface. Tidigare körde de … och nu migrerar de mer över till Service Workers. Fördelen med Service Workers är att du kan styra nätverksförfrågningarna genom att säga hur den ska cacha, när den ska cacha. En service worker synkar i bakgrunden vad som ska visas eller ej och du kan bestämma själv vad som ska cachas. Man kan lägga in en synkroniseringsfunktion som ska synka data mot en databas i indexDB. Man behöver inte tänka på att service workern slöar ner gränssnittet m.m. Det är lättare att skapa applikationer som är snabba och responsiva. Service Workers gör detta möjligt. Det är dessutom en fördel att du kan köra operationer i bakgrunden och synkar. En worker kan i bakgrunden kolla av om det är någon

uppdaterad data, t.ex. i en arbetsorder, och detta fungerar helt offline utan någon som helst internetuppkoppling. Datat uppdateras sedan när internetuppkoppling återupptas.

6. **Vad betyder andvändbarhet för dig?**

   Så lite hinder ivägen för användaren som möjligt. Tydlighet, snabbhet, responsivitet, naturligt flöde.

7. **Hur förbättrar detta offlinestöd användbarheten av applikationen?**

   Den är brukbar offline. Även när du har dålig uppkoppling (att det kommer och går). Som användare idag förväntar man sig inte att en applikation stannar upp helt, om det inte finns någon anledning till det. Sidan ”Det finns inget internet” ska inte visas, det är ett dåligt sätt att presentera att applikationen är offline.

8. **Vissa applikationsprogrammeringsgränssnitt (API) eller ramverk har fler funktioner än endast funktionsstöd för offline användning. Har den lösning du arbetat med någon annan funktion som förbättrar andvändbarheten av applikationen för användaren? Isåfall, vad?**

   Ja, att man inte behöver hämta data från en server hela tiden. Cachar man grunddata så förbättrar det användbarheten för användare för att man inte behöver hämta data från en databas, m.m. Sen finns det lite säkerhetsaspekter såsom att man inte vill använda en webbapplikation till t.ex. bankärenden. En tanke är att en webbapplikation är mer sårbar.

9. **Hur skulle denna eller liknande lösningar av offlinestöd förbättra användbarheten hos Trionas kunder?**

   Lösningarna skulle gynna kunder som tidigare kört native-applikationer. Dessutom är det enklare och roligare för utvecklare att arbeta med applikationer som kan skrivas i en och samma kodbas, som kanske inte stödjer allt i alla plattformar, men som har så pass mycket stöd så att man kan framställa en färdig affärsapplikation. Native kommer alltid att finnas som stöd för mer krävande applikationer, när man kräver mer utav design och har applikationer som kräver mer prestanda. Han tror att webbfunktioner underskattas då det finns mycket mer än vad man tror som man kan göra. Han tror att det är en stor fördel i sig att bygga applikationer med offlinestöd, det blir billigare för triona och dess kunder.

10. **Inom vilket område eller kundkrets skulle detta vara användbart? Någon speciell marknad?**

    Särskilt för Triona skulle detta vara användbart för alla typer av marknader egentligen. Inget specifikt område, men Triona har mycket kunder som de måste få ut grunddata och de är ute i skogen t.ex. och ska registrera något avvikande, t.ex. uppdatering av kartor, trafik, osv.

11. **Har du någon erfarenhet kring ramverket Service Workers?**
    Ja, se fråga

12. **Tidigare applikationsgränssnitt har inte haft stöd för push notiser för användaren, hur tror du att detta påverkar användarens upplevelse av applikationen?**
    Dels så förstår användaren att de är offline – det är viktigt för användaren att förstå vad som händer.

13. **Progressiva webbapplikationer sägs vara att föredra över native-applikationer, nämn minst tre förbättringar som kommer med progressiva webbapplikationer ut ett utvecklarperspektiv och användarperspektiv.**

    Det finns massa fördelar med båda. Men native-applikationer är bättre för att få tillgänglighet till interna delar såsom kamera.
    Fördelen med PWA är att det går fortare att få ut uppdateringar till kunder utan att behöva godkänna att allt installeras på enheten. Det är lättare att ta betalt för tjänster via native-applikationer som är kopplade mot ”stores”.

14. **Ser du några nackdelar med att köra progressiva webbapplikationer över native idag? Isåfall, vilka?**
    En nackdel är att det är så pass nytt, så det stödjs inte i vissa webbläsare och versioner, m.m.

15. **Allmän erfarenhet om Service Workers:**
    Finns det inte stöd för en service worker i en viss webbläsare så kan sidan fortfarande köras. Några nackdelar idag är att du inte kan komma åt addressbok i telefonen m.m. Du är frånskild från enhetens funktioner mer, och det är riktat mer mot webb. Man är lite mer begränsad från enheterns interna funktioner.

Intervju #2 (System och webbutvecklare på Triona AB Stockholm)

1. **Vad heter du och hur gammal är du?**

   Anonym respondent 2, 29 år

2. **Kan du berätta lite om vad du jobbar med på Triona?**

   Apputvecklare med huvudfokus på webbappar, lite xamarin appar, fick jag bestämma skulle de vara native för det gillar jag.

3. **Vilket område jobbar du inom? Skogsindustri? Transportinfrastruktur? Transporter? Industri? Entreprenörskap? Annat? Isåfall vad?**

   Transportinfrastruktur framförallt, men jobbar även inom flera olika marknader.

4. **Finns det något du tycker är extra roligt inom området du jobbar med?**

   Det är kul att göra appar, just för att man får utveckla ny teknik som är användbart för åkare (lastbilar m.m.) de har rätt dålig datavana, så det är lite utmaningen och är kul att se att appar blir användbara. Exempel är: Pocket PC vs appen Flow, fördelarna är att det är bättre anpassat gränssnitt då den är mer modern och lättanvändligt, det är roligt som utvecklare att vara med i teknikens utveckling.

5. **Vad för erfarenhet har du av offlinestöd på applikationer?**

   LocalStorage, sessionstorage, SQLite och WebSQL, IndexDB. Jag föredrar IndexDB för att tekniken fortfarande stödjs och är mest användarvänlig. IndexDB har en nackdel att det krashar pga kompabilitetsproblem, otroligt kraftfullt och smidigt sätt att lagra data på, och fungerar på allla plattformar. Persistant IS via webSQL (för flow appen, en webbprodukt de har).

6. **Vad betyder andvändbarhet för dig?**

   Användbarhet = det är enkelt att komma igång med, man behöver inte läsa massa dokumentation för att få allt att fungera. Ett exempel är WebSQL som Respondenten har använt utan att ens läsa igenom dokumentation.

7. **Hur förbättrar detta offlinestöd användbarheten av applikationen?**

   Svarades på i förra frågan

8. **Vissa applikationsprogrammeringsgränssnitt (API) eller ramverk har fler funktioner än endast funktionsstöd för offline användning. Har den lösning du arbetat med någon annan funktion som förbättrar andvändbarheten av applikationen för användaren? Isåfall, vad?**

   Inget annat, det är ren lagring i cacheminnet.

9. **Hur skulle denna eller liknande lösningar av offlinestöd förbättra användbarheten hos Trionas kunder?**

   Jag tycker att det kan vara bra för användaren att reagera på vad som händer via event som känner igen att uppkopplingen tappats. Synkroniseringen med verksamhetsobjekt kan förbättras. Push notifikationer är något som saknas för användaren också.

10. **Inom vilket område eller kundkrets skulle detta vara användbart? Någon speciell marknad?**

    Lösningen kan gynna Lasset framförallt, som är inriktad mest mot skogsindustri och andra marknader.

11. **Har du någon erfarenhet kring ramverket Service Workers?**

    Ingen erfarenhet av Service Workers, men erfarenhet av LocalStorage, sessionstorage, SQLite och WebSQL, IndexDB.

12. **Tidigare applikationsgränssnitt har inte haft stöd för push notiser för användaren, hur tror du att detta påverkar användarens upplevelse av applikationen?**

    Att användaren inte vet vad som händer när de tappar uppkoppling.

13 **Progressiva webbapplikationer sägs vara att föredra över native-applikationer, nämn minst tre förbättringar som kommer med progressiva webbapplikationer ut ett utvecklarperspektiv och användarperspektiv.**

Utvecklarperspektiv: Enklare att utveckla, kräver inte att man är specialist på ett specifikt språk – gör att det blir lättare att hitta utvecklare för företag. Fungerar på alla telefoner och enheter.
Användarperspektiv: Handelsbanken appen t.ex. skulle han inte tycka är bra att köra PWA, utan föredrar att den är native och installerad på enheten. Beroende på vad det är för typ av app. Man blir dessutom begränsad att använda webbläsare, och sekretess kan spela stor roll in. Skulle det inte märkas någon skillnad så är det att föredra pga. Utökade funktioner såsom animationer, samma utseende oavsett vilken plattform, att det responsivt och snabbt. Offlinestöd, snabb lagring av data till webben, komma åt flera funktioner såsom sensorer såsom kamera m.m. Då vet man att det finns ett gemensamt gränssnitt för GPS, kamera, m.m.

14 **Progressiva webbapplikationer sägs vara att föredra över native-applikationer, nämn minst tre förbättringar som kommer med progressiva webbapplikationer ut ett utvecklarperspektiv och användarperspektiv.**

Fördelar med native: bättre enhetlighet, allt fungerar smidigt.
Nackdelen: gränssnittet blir plottrigt och är svårt att få utseendet att se likadant ut på alla plattformar.

15 **Hur tror du att framtiden kommer se ut för progressiva webbapplikationer? Kommer de ta över native så småningom?**

Vi kommer att få en bibehållen native marknad, men PWA kommer med en hel del fördelar. Kan ha med att det kostar att utveckla PWA mer än native. Respondenten tror att utvecklare kommer att envisas med att fortsätta köra native-applikationer då det finns mycket pengar att hämta via betalningar/subscriptions och liknande som dras varje månad. PWA är mer fritt och gör att företag kan förlora pengar. Allt beror på hur bra applikationen som är progressiv är utvecklad.

16 **Någon specifik lösning som Triona har utvecklat som skulle gynnas av att köras progressivt istället för native? Hur kan användbarheten förbättras genom att köra lösningen progressivt?**
Inte specifikt progressivt, men som en hybrid app skulle Corodova ha stora fördelar att köra en hybrid app (både PWA och native). Just för att få ner kostnaden och bibehålla plattformsstöd. För då skulle man kunna köra Android och IOS och köra de saker som fattas i progressiv och native-applikationer.

Intervju #3 (System och webbutvecklare på Triona AB Stockholm)

1. Har du någon erfarenhet av ramverket" Service Workers"? Om inte, har du någon erfarenhet av snarlika ramverk eller applikationsprogrammeringsgränssnitt (API)?

Har inte själv jobbat med "Service Workers". Har dock designat en snarlik lösning som baserades på en HTML5-app (Cordova) som använde WebSQL för lokal lagring samt ett Rest-baserat CQRSbaserat protokoll för synkning tillsammans med Push-meddelanden (GCM / Apple Push Notification Service)

2. Ser du att detta ramverk skulle kunna förbättra användarvänligheten på progressiva webbapplikationer?

Ja, är ju alltid bra att kunna luta sig mot en standardlösning som dessutom inte kräver paketerade webbapplikationer. Detta är ju en vidareutveckling av lösningen jag beskrev i fråga ett som kräver mindre egen utveckling.

3. Förbättrar ramverket något speciellt för användaren förutom offlinestöd för progressiva webbapplikationer? Isåfall, vad?

Time to market. Mindre risk för buggar och synkproblem.

4. Är implementation av ramverket eller liknande ramverk för offlinestöd något som efterfrågas av kunder på Triona?

Misstänker att det kan finnas tillämpningar som förenklar mycket jämfört med lösningen beskriven i fråga 1. Är toppen om man kan slippa "krånglet" med paketering, framförallt under IOS, som kräver kundspecifika utvecklingslicenser per kundtillämpning.

5. Finns det några nackdelar med att implementera lösningar för offlinestöd tror du (baserat på dina tidigare erfarenheter)?

Blir ju i regel en krångligare lösning som kräver mer test och leder till flera användningsfall.

6. Hur kan man lösa/kringgå kompabilitetsproblem såsom browser support med ramverket/liknande lösningar? Har du något förslag?

I regel handlar det om hur lösningen ska användas. Är det en publik lösning är detta i regel ett stort problem. Handlar det om kundunika lösningar kan man ju lättare ställa krav på klientmiljöer.

7. Vad tror du krävs för att ett offlinestöd ska fungera optimalt?

Branschstandard bland de stora webbläsarna är nog ett väldigt bra steg.

8. Är ramverket eller liknande lösningar (såsom API:et AppCache) relativt enkla att implementera, sett ur ett utvecklarperspektiv?

Har ingen erfarenhet.

9. Service workers gör stor användning av" promises", vilket representerar eventuella slutföranden (eller misslyckanden) av en asynkron operation och dess resulterande värde. Varför tror du att" promises" och service workers löser s.k." callback hell" som AppCache hade stora problem med?

Är ju ett enkelt mönster att implementera som fungerar i praktiken, även i skiktade lösningar.

10. Vad tror du att framtiden har att erbjuda när det kommer till offlinestöd för progressiva webb applikationer? Är det något du skulle vilja se implementerat?

Jag håller tummarna att det blir en riktig kioskvältarteknik.

11. Föredrar du progressiva webb applikationer över native? Isåfall, varför? Nämn gärna fördelar respektive nackdelar.

Ur ett användarperspektiv är det såklart bra. Ur ett utvecklarperspektiv == kostnadsperspektiv är det troligen en dyrare mer komplex lösning.

Intervju #4 (System och webbutvecklare på Triona AB Stockholm)

1. **Har du någon erfarenhet av ramverket" Service Workers"? Om inte, har du någon erfarenhet av snarlika ramverk eller applikationsprogrammeringsgränssnitt (API)?**

   Nej.

2. **Ser du att detta ramverk skulle kunna förbättra användarvänligheten på progressiva webbapplikationer?**

   Kan inte svara för detta ramverk då jag inte använt det, men utifrån vad det verkar ha för egenskaper utifrån det jag läst om det så tror jag att det kan förbättra och effektivisera kommunikationen mellan data och presentation för att ge en bättre användarupplevelse.

   Exempelvis spara data på klienten så att vid uppstart kommer användaren alltid presenteras med sparat data som sedan i bakgrunden kan uppdateras till det mest aktuella vid det givna tillfället. Det kan även bidra till mer effektiv kommunikation mellan datalager och presentationslager genom att inte alltid hämta om alla data.

3. **Förbättrar ramverket något speciellt för användaren förutom offlinestöd för progressiva**

webbapplikationer? Isåfall, vad?

   Samma svar som fråga 2.

3. **Är implementation av ramverket eller liknande ramverk för offlinestöd något som efterfrågas av kunder på Triona?**

   Ja, det anser jag.

5. **Finns det några nackdelar med att implementera lösningar för offlinestöd tror du (baserat på dina tidigare erfarenheter)?**

   Inte tillräckligt tillförlitlig teknik, eller att implementationen inte möter upp mot kraven.

   Exempelvis att offlinestödet inte fungerar som förväntat i en given situation eller att offlinefunktionen inte stödjer alla krav och hela användningen. Offlinestöd i en applikation där man arbetar med data tillsammans med en karta exempelvis och kartan inte har fullt offlinestöd men övriga delar av applikationen har det.

6. **Hur kan man lösa/kringgå kompabilitetsproblem såsom browser support med ramverket/liknande lösningar? Har du något förslag?**

   Man behöver ha kontroll på vilka enheter som det skall användas på eller att vara defensiv i vilken funktionalitet/teknisk lösning man använder för att kunna garantera kompabilitet.

7. **Vad tror du krävs för att ett offlinestöd ska fungera optimalt?**

   Bra tekniska ramverk är lika viktigt som att ha en tydlig och väl genomarbetad kravspecifikation. Viktigt att klargöra förväntningar på vad en offlinelösning klarar av i verkligheten.

8. **Är ramverket eller liknande lösningar (såsom API:et AppCache) relativt enkla att implementera, sett ur ett utvecklarperspektiv?**

   Ingen uppfattning.

9. **Service workers gör stor användning av” promises”, vilket representerar eventuella slutföranden (eller misslyckanden) av en asynkron operation och dess resulterande värde. Varför tror du att” promises” och service workers löser s.k.” callback hell” som AppCache hade stora problem med?**

   Det är förhoppningsvis en lösning/teknik som lärt sig av de tidigare problem som vart förknippade med denna hantering.

10. **Vad tror du att framtiden har att erbjuda när det kommer till offlinestöd för progressiva webbapplikationer? Är det något du skulle vilja se implementerat?**

    Ingen uppfattning.

11. **Föredrar du progressiva webbapplikationer över native? Isåfall, varför? Nämn gärna fördelar respektive nackdelar.**

    Det finns så klart fördelar och nackdelar med att välja det ena eller det andra. Jag tror att det beror väldigt mycket på vilken typ av tillämpning man skall utveckla och i vilket sammanhang den skall existera.

    Personligen så tycker jag att det finns ett ökat antal fördelar med att använda progressiva webbapplikationer före applikationer anpassade till ett visst operativsystem.

Intervju #6 (System och webbutvecklare på Triona AB Stockholm)

1. **Har du någon erfarenhet av ramverket" Service Workers"? Om inte, har du någon erfarenhet av snarlika ramverk eller applikationsprogrammeringsgränssnitt (API)?**

   Har precis satt ihop en PWA-applikation åt en kund för att testa om det fungerar i deras fall. Som ramverk har jag använt Vue.js, som har en inbyggd template för PWA. Denna använder Googles WorkBox i bakgrunden vilket döljer mycket av den kod man själv annars måste skriva för att bara komma på banan. Enkelt beskrivet får man en färdigkonfigurerad service worker som man kan lägga till egen kod i. Och i mitt fall anropar jag serviceworkern från huvudapplikationen för att synkronisera data mellan lokala data och servern.

2. **Ser du att detta ramverk skulle kunna förbättra användarvänligheten på progressiva webbapplikationer?**

   Ja, service workers är en grundbult för PWA:er. I alla fall enligt den spec. som Google har satt upp.

3. **Förbättrar ramverket något speciellt för användaren förutom offlinestöd för progressiva webbapplikationer? Isåfall, vad?**

   Service workers gör det möjligt att köra kod på en annan tråd i bakgrunden, vilket gör att prestandan i appen förbättras. En annan detalj är att man tvingas att använda HTTPS, vilket gör appliktioner säkrare i grunden.

4. **Är implementation av ramverket eller liknande ramverk för offlinestöd något som efterfrågas av kunder på Triona?**

   Det är ingen som frågar specifikt efter service workers. Kunden efterfrågar i så fall offlinestöd.

5. **Finns det några nackdelar med att implementera lösningar för offlinestöd tror du (baserat på dina tidigare erfarenheter)?**

   Inte direkt. Men det innebär ofta en ytterligare komplexitet i projektet. Man får ofta skria synkroniseringskod själv och det är inte alltid så lätt. Ibland kanske inte vissa delar av applikationen behöver offlinestöd, men andra måste har det…

6. **Hur kan man lösa/kringgå kompabilitetsproblem såsom browser support med ramverket/liknande lösningar? Har du något förslag?**

   I javascript går det att "känna av" vilka funktioner som det finns stöd för, så det är hyfsat enkelt att villkorsstyra funktioner eller tex. informera användaren om att en funktion inte stödjs pga. typ av browser eller plattform.

7. **Vad tror du krävs för att ett offlinestöd ska fungera optimalt?**

   Att de stora spelarna drar ungefär åt samma håll och enas om specifikationer. Google driver den utvecklingen idag men Apple har faktiskt (sent) hakat på tåget och slängt med PWA-stöd i senaste versionen av iOS (11.3).

8. **Är ramverket eller liknande lösningar (såsom API:et AppCache) relativt enkla att implementera, sett ur ett utvecklarperspektiv?**

   Jag har inte använt AppCache och som sagt inte implementerat en service worker från grunden. Men tex. IndexedDb är hyfsat enkelt att jobba med och det finns även väl utbyggda open source API:er som underlättar (tex. db.js eller dixie).

9. **Service workers gör stor användning av” promises”, vilket representerar eventuella slutföranden (eller misslyckanden) av en asynkron operation och dess resulterande värde. Varför tror du att” promises” och service workers löser s.k.” callback hell” som AppCache hade stora problem med?**

   Det blir mer strukturerad kod och felhanteringen blir mycket lättare. Numera kan man ju även använda async / await (som i C#) vilket underlättar ännu mer.

10. **Vad tror du att framtiden har att erbjuda när det kommer till offlinestöd för progressiva webb applikationer? Är det något du skulle vilja se implementerat?**

    Jag tror att PWA:er kommer ta över mer och mer av app-marknaden. För oss utvecklare är detta positivt eftersom vi kan jobba i samma kodbas till 100% men publicera till i princip vilken enhet som helst.

    Men native appar kommer så klart finnas långt framöver.

11. **Föredrar du progressiva webb applikationer över native? Isåfall, varför? Nämn gärna fördelar respektive nackdelar.**

    Jag har inte jobbat tillräckligt med native för att ge ett bra svar.

Intervju #7 Johan Nyström (Tretton37 AB Borlänge)

1. Kan du berätta lite om vad du arbetar med och på vilket företag?

Senior fullstackutvecklare på Tretton37. Arbetar främst med modern webbutveckling och .NET

2. Något speciellt du tycker är kul med det du arbetar med?

Att arbeta med ny teknik och lösa svåra tekniska problem samt att ständigt utmanas och utvecklas.

3. Vilka fördelar kommer gränssnittet Service Workers med?

Service workers är ett finkornigt API som ger möjlighet att helt styra nätverksanrop och cache. Fördelen är att man som utvecklare kan bygga egna lösningar och abstraktioner ovanpå ett lågnivå-API istället för att behöva förlita sig på mer kompletta lösningar, så som Application Cache.

4. Vilka nackdelar finns inom gränssnittet? Vad skulle du vilja se förbättrat?

Nackdelarna är att det är en något hög tröskel för att sätta sig in i, just därför att man har kontroll på låg nivå.

5. Varför Service Workers? Några personliga åsikter kring ramverket som gör att du tycker denna tekniska lösning är bättre än andra på marknaden?

Service workers är en webbstandard. Det är också den enda lösningen för att styra nätverksåtkomst på låg nivå från webbläsaren. Det finns inget annat att jämföra med.

6. Vad gör gränssnittet intressant att forska om tror du? Varför vill företag som Triona veta mer om forskningsämnet, tror du?

Det är intresserat därför att det ännu inte är beforskat. Service workers i sig är bara en teknisk standard. Det intressanta är att den möjliggör att skapa nya lösningar på helt nya sätt, vilka bör utforskas både praktiskt och akademiskt.

7. Varför är Promises och Fetch-händelser så viktigt för att gränssnittet ska fungera?

Därför att service workers är ett asynkront API och promises är en abstraktion över eventuella framtida värden. Utan promises skulle webbläsaren vara låst under varje interaktion med en service workers.

8. Till skillnad från Application Cache kommer Service Workers med flertalet funktioner. Finns det något som är en avsevärd förbättring eller komplement från tidigare tekniska lösningar såsom AppCache?

Ja, att det det är ett lågnivå-API som möjliggör långt fler lösningar för cachning. Rent subjektivt är service workers också ett bättre designat API.

9. Finns det något värt att nämna som få vet om att man kan göra med Service Workers?

Det viktigaste du kan göra är att få din applikation att fungera helt eller delvis offline.

10. I vilken situation skulle du kunna tänka dig använda de olika cachnings-strategierna? (Cache only, Network or cache, Cache and update, Cache, update and refresh, embedded fallback) Om möjligt, ge gärna exempel på en situation där specifika strategier kan användas.

Mycket handlar hur man hanterar volatil eller statisk kod, där volatil kod är kod som förändras ofta. För statisk kod, som till exempel tredjepartskod eller gemensamma komponenter kan man använda cache only för prestanda. Network or cache är bra om man gärna hämtar från nätverket, men kan nöja sig med cache om nätverket är nere eller svarar långsamt. Ofta är det bättre att visa gammal data än ingen alls. Cache and update ger bra prestanda samtidigt som cachen uppdateras, vilket dock inte syns förrän nästa sidladdning. Cache and update fungerar inte om det är viktigt att alltid visa aktuell data men är annars en bra kompromiss. Update and refresh är samma sak, men man visar ny data dynamiskt efter laddning. Har ingen erfarenhet av embedded fallback.

11. Är någon strategi att föredra eller beror det på vilken situation alt. kund applikationen är ämnad för?

Man kan ha flera strategier för samma applikation, men för olika delar av den. Du kan ha network or cache för ett nyhetsflöde samtidigt som du har cache and update för sidans skal. Det är också det som är styrkan i service workers.