Progressive Web Apps for Media Playback

Qutibah Hussein
Computer Science
Technische Universität Berlin
Berlin, Germany
q.hussein@campus.tu-berlin.de

Azmi Amiruddin
Information System Management
Technische Universität Berlin
Berlin, Germany
azmi.amiruddin@campus.tu-berlin.de

Abstract — **Progressive Web Applications (PWAs) has become disruptive web application framework which aim to refactor the legacy web and mobile application paradigm by bridging the intuitive web experience with native application functionality. PWA is now become preferred approach and standard de-facto for many global brands, few names including: Forbes [18], Netflix [15], and Twitter [36, 37] to ship a modern mobile web experience. PWAs can offer quick, compelling journeys similar to what a native application can do with discoverable and open to everyone via the mobile web. By complying with PWA guidelines, the web application offers improved user experience that includes faster load times, app-like navigation, and highly visual content. In addition, end user with smart devices will benefit from enhanced capabilities such as offline streaming and push notifications. Through this document, we will proof of works rich feature offer by PWAs framework, including online and/or offline, push notification capabilities, and cross-browser compatibility.**

Index Terms — **Application Shell, DASH, Media Streaming, Progressive Web Applications, Service Worker.**

I. INTRODUCTION

Progressive Web Applications (PWAs) was formally introduced in 2015 by Google, and the idea of standard web technologies "write once running everywhere" was offered by Steve Job [5, 30]. Main characteristic offer by PWAs are: (i) reliable in terms of network availability, (ii) fast respond for user action, and (iii) intuitive user interface and experience (UI/UX) [7, 11, 30, 31].

In mobile apps markets, developing a superior user experience is no longer a nice-to-have feature. All industrial sector and business services of all kinds are competing to adopt the latest mobile-friendly technologies, including PWAs [35]. The early-adopters companies (e.g.: Forbes, Financial Times, Lyft, Expedia, AliExpress) that made shift to PWAs before the peak adoption has gained a massive business advantage [35, 38]. The Netflix [15] website, for example, provides a marvellous experience for the user. Through Netflix web application users prompted to benefit from their streaming content by: pre-play, video playback and post play. Those streaming and media playback features effectively deliver on top of React [44], Redux [4], and Vanilla-JS framework [45]. Interestingly, during our proof of work we highlight those components implemented for Netflix website application are essential technology stack for PWAs building block.

PWAs aim to disrupt the mobile app paradigm by bridging the web experience with native applications functionality, by using the newest browser technologies to meld the accessibility of the web with the presence of the mobile applications. Mozilla, Microsoft, Google, Apple, Facebook start to provide the most advanced and innovative platform that enable web applications to behave like native apps. Open standards created using HTML5, CSS and JavaScript raised new pattern shift which label as "Service Worker". The service workers are embedded within browser to surface PWA functionality [7, 11]. This facilitates users to install the website as an apps icon to their home screen, as well as utilize app features such as push notifications and offline storage.

Media streaming providers responsible for mobile app strategies must determine when they need to factor in PWAs as part of their overall mobile development strategy. Application consumer is forcing media streaming provider to re-evaluate their approach to mobile web versus apps. PWAs forced media streaming providers to work with designers to reassess mobile websites and apply more app-oriented experience. Moreover, application developer will need to tighten use cases for home applications (desktop) and mobile applications with PWAs design functionality [34, 35].

Although PWAs relatively new disruptive web application framework, some remarkable PWAs are already on the web and being used by millions of users every day. Additionally, showcase from Google Developer shows Mynet website harnessing the power of PWAs and the possibility of a mobile and flexible transfer of content uses intelligent Service Worker. Mynet claimed they accomplished to refactor legacy web application into new PWAs framework and then lead to significant increase in user engagement, with the average time spent on their mobile site rising 43%, 25% higher revenue per article page view, 4X faster average page-load speed, 24% lower bounce rates [14]. Under the hood Mynet uses Service Workers and Application Shell to cache and provide a super-fast experience for the user and both channels provide intuitive user experience [19].

II. REVIEW OF CURRENT THINKING

PWAs emerged as a better alternative to traditional web and mobile application development (legacy application). Possessing the best of both native applications and web applications, PWAs also offers advantages like offline loading of the application as well as services like push notifications and background synchronization to enhance user engagement.

The paper provides an overview of using PWAs for media playback as modern application development cycle and detail review on how progressiveness in a web application offers advantage over native application. Furthermore, we also discusses about the architectural pattern followed to develop progressive web apps to improve loading time of the app. API provided by the service workers will also discuss in detail including push notifications and background synchronization to provide native app like experience. A project case on media streaming has been carried out to depict the working of progressive features (e.g.: offline, online and web push).

As we already start with introduction on Section I, current section will review the theoretical perspectives of PWAs and HTTP Media Streaming. Moving to Section III, we observed PWAs framework through our project case and on Section IV we cover the challenges during project execution. Eventually, on Section V and VI we closed the document with moving forward action and key takeaways to bring "PWAs for Media Playback" as main drivers that enable frictionless journeys for desktop and mobile application regardless their platform.

A. What are PWAs?

The world of web and mobile applications is highly competitive and increasingly crowded. In just the last few years, the number published application in AppStore or PlayStore has increased more than doubled. Numbers of statistics as of the fourth quarter of 2019, Android users were ready to choose from 2.57 million available apps in store, making Google PlayStore with biggest number of available apps [39]. Apple's AppStore was the second-largest app store with almost 1.84 million available apps for iOS [39].

Applications owner generate income revenue in a number of different ways, such as charging users a small amount of money for the use of an application (an average of 1.02 U.S. dollars per application in the Apple Store), or procure for access to premium features of an application otherwise free apps or simply selling advertisement space [39]. The impact of this is twofold: current application becomes legacy approach for users which lead to immense pressure for application developers and application provider need to find win-win solution to make sure their applications stand out from the crowd and provide better customer experience [20].

On the positive side, applications that deliver great content seamlessly and quickly, no matter what device a user has will benefit the financial impact on the organisation. This is why PWAs are quickly becoming the standard way to ship these great user experiences to the web. In fact, Gartner predicts that by 2020, 50% of general-purpose, consumer-facing mobile apps will be PWAs [6].

Moving forward from the economic impact of mobile and web application, the current state of web application hasn't been able to offer features such as offline capabilities, instant load times, and improved reliability. This is where PWAs are a game-changer. Technology giant (Google, Microsoft, Apple) have been working together to improve the way we build for the web and have created a new set of features that give web developers and web provider the ability to create fast, reliable, engaging websites with responsive design, connectivity-independent, interactive with a feel like a native applications [19, 30, 35].

Building and deliver solution using PWA allows organisation to increase their revenue stream and reach more potential users than we could ever achieve with native apps alone, because PWAs built for any platform capable of running in desktop and mobile device [38, 39]. Regardless of the stack organisation has implemented to develop website application, PWAs work hand-in-hand with existing web framework, this due to core framework behind PWAs build on top of HTML, CSS, and JavaScript [11, 12, 30].

B. PWAs Core Technology

While the web application is becoming more powerful every day, there are still a few things, like fingerprint authorization, that is not yet available [30]. Therefore, the foremost common case where a PWA won't be an excellent fit is for products that have a strict technical requirement that's only available on native platforms. Even in those cases, it's worth considering adding a PWA to enhance the customer experience using responsive UI/UX and additional "Add to Homescreen" feature for initial experiment for new visitors, then routing them to the app stores once they want to access more advanced functionality. PWAs building block are more capable to complete those requirements, this due to reused existing technology stack, which are: HTML, CSS, and JavaScript [11, 12, 30].

Service Workers [1, 11, 12, 19, 30, 35] are the "traffic control" [11], helping PWA reach those reliable, fast, and engaging UI/UX. Service Workers allow PWA to work offline (not possible in legacy web application) load reliably no matter the state of the user's network connection, and enable features like push notifications and background data syncing. Service Workers also give fine-grained control over caching through JavaScript APIs, enabled developer create a custom offline experience tailored for application. For example, HTML with JavaScript program allow service worker to always serve cached content first, keeping application nice and fast, but then to update the cache in the background so the next time the user opens PWA the content will be up to date. Finally, as "air traffic controllers" Service Workers give apps owner full ownership of each and every web request made from their applications [11].

Application shell [1, 11, 12, 19, 30, 35] allows PWAs to deliver the look and feel what people expect. They let users specify an icon, application name, and splash screen color, and enable users to install PWA to their device and have it appear right alongside their native applications.

C. HTTP Media Streaming

Delivery of streaming content over the Internet had started in the 1990s with sensible delivery and consumption of large amounts of data being the main challenge. The IETF's Real-time Transport Protocol (RTP) was designed to define packet formats for audio and video content along with stream session management that allowed delivery of those packets with very low overhead. RTP works well in managed IP networks [17, 31, 35]. However, video streaming industry in today's Internet

managed networks have been replaced by Content Delivery Networks (CDNs), many of which do not support RTP streaming. In addition, RTP packets are often not allowed through firewalls [5, 6, 8, 9, 10].

For the above reasons, HTTP streaming has become a popular approach in business deployments. For example, streaming platforms such as Apple's HTTP Live Streaming (HLS), Microsoft's Soft Streaming and Adobe's HTTP Dynamic Streaming all implemented on top of HTTP streaming technology as major distributor in the internet stack, as shown in Figure 1. However, each process uses different technique and segment formats, and therefore, to obtain content from each server, a device must support its associated proprietary client protocol. A standard for HTTP streaming of multimedia content allows a standard-based client to stream content from any standard-based server, enabling it to be seamless between servers and clients of different vendors.

Proprietary Adobe HDTB Dynamic Streaming (HDS), Apple HTTP Live Streaming (HLS), Microsoft Smooth Streaming (MSS) and all existing Adaptive HDDB streaming technologies and HDDB (MPEG-Dash) follow almost the same principle.

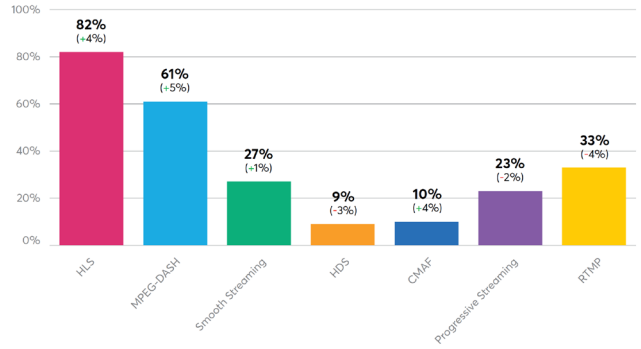

Figure 1 - Most Common Streaming Format [32]

In the context of HTTP Media Streaming [32], we also highlight most common streaming format which has been adopted by many entertainment industry (e.g.: HBO, Disney+, ARD, Netflix) [34]. The rationale of this section is to highlight important technical aspect of HTTP Media Streaming which we implemented in our project case. As a result, we highlight short technology description taken from: MPEG-DASH, Apple HLS, Microsoft Smooth Streaming and Adobe HDS as individual sections.

1. MPEG-DASH

MPEG-DASH is an ISO standard for adaptive streaming of video content, which provides significant benefits for developers who want to provide high quality, adaptive video streaming output [5, 6, 8, 9, 10]. With MPEG-DASH, quality metrics on the video stream automatically adjust to low definition when the network becomes congested. Fragmented manifest reduces the viewer's chances of watching the "paused" video, while the player downloads the next few seconds to play (buffalo). As the network congestion increases, the video player will return to high-quality streams. This ability to optimize the required bandwidth results in faster start times for video [9]. This means that the first few seconds can be played in the fast-to-download lower quality segment and move up to higher quality once enough content is buffered [5].

Shaka Player and Dash.js is most common open-source MPEG-DASH video player written in JavaScript, with goal to provide a robust, cross-platform player that can be reused independently in applications that require video playback. It provides MPEG-DASH playback in any browser that supports W3C Media Source Extensions (MSE) [9, 13, 33].

2. Apple HLS

HTTP Live Streaming (HLS) provides a reliable, cost-effective means of providing continuous and prolonged video over the Internet. This allows a receiver to adapt the bit rate of the media to the current network conditions to maintain uninterrupted playback at the best possible quality. Apple HLS supports interstitial material boundaries. It provides a flexible framework for media encryption. It can efficiently present multiple renditions of the same material, such as audio translation. It provides compatibility with a large-scale HTTP caching infrastructure to support distribution for large audiences [8, 29].

Subsequent to first draft publication in 2009, Apple HLS has been implemented and deployed by a wide array of video content distributors, equipment vendors and manufacturers. In the subsequent eight years the Apple HLS has been improved by extensive review and discussion with various media streaming providers [29].

3. Microsoft Smooth Streaming

Almost one decade ago in October 2008, Microsoft announced that Internet Information Services (IIS) 7.0 would ship with new feature to support HTTP-based adaptive streaming extension. Microsoft branded that new technology as Microsoft Smooth Streaming (MSS) [2, 26]. Moving forward, for widespread technology adoption Microsoft announced an initiative with Akamai and launched a showcase website Akamai HD for Silverlight [49]. Support for smooth streaming will be announced in the future from other CDNs [2, 8, 26].

MSS dynamically detects local bandwidth and CPU status and native switches, in real-time, the video quality of a media file received by a player. Clients with a reliable network connection able to experience high-definition (HD) quality streaming, while others with low bandwidth speeds receive streams suitable for their connectivity, allowing clients across the board to enjoy a compelling, uninterrupted streaming experience and reduce the need for media companies to allow the lowest general within their audience base meet the quality level [8, 26].

Furthermore, MSS uses the MPEG-4 Part 14 (ISO / IEC 14496-12) file format as storage and wire (transport) format. Specifically, the Smooth Streaming Specification defines each component / GOP as an MPEG-4 movie fragment and stores it in a nearby MP4 file for easy random access. In other words, with Smooth Streaming, the file components can be created at the client's request, while the actual video is stored as a single full-length file for the bit rate encoded on the disk. It offers excellent file-management benefits [8, 26].

4. Adobe HDS

Adobe HDDB Dynamic Streaming (HDS) is a great way to deliver multi-bitrate content over HDTP to Adobe Flash Player or Adobe AIR. Adobe HDS feature offers the capabilities needed to deliver digital video and audio. The Adobe HDS technique includes intelligent multi-bitrate adaptive logic and robust content security using Adobe Prime Time DRM. Adobe HDS is based on the fragmented F4V file format (F4F) [3, 8].

Overall, the behaviour of adaptive media streaming combine with PWAs framework shows in Figure 2. The high level architecture demonstrates as schematic illustration of typical PWAs for media streaming. The downstream processing flow through different execution modes (JavaScript & HTML) and it is supplemented by an upstream CDNs as well as a network streaming services which has requested by client (browser & electron).

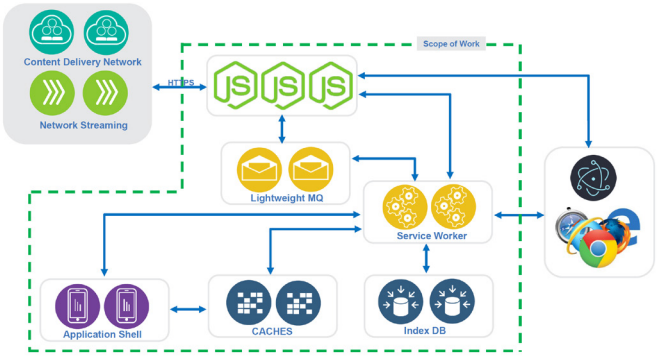

Figure 2 - PWAs High-level Architecture

In such setup illustrated in Figure 2, PWAs for media streaming will encompass with the following component [10]:

a) the streaming utilizing standard delivery infrastructure over a CDN and streaming-as-a-service deployed on standard cloud infrastructure (e.g.: Akamai, Amazon) taking the live source as input and providing multiple representations (e.g.: resolution, bitrate) according to the MPEG DASH standard as output.
b) PWAs component taking the role as the business logic within the customer web for the actual deployment.
c) DASH client [13] and/or Shaka Player [33] implementation integrated within major Internet browser and electron media player.

The above requirements are expressed through our project case and depicted by dotted line as scope in our project case.

III. Project Case: PWA for Media Playback

Developing web applications went a long way. PWAs take advantage of HTML5, CSS and JavaScript natural capabilities. *Progressive* term refers to how to use the capabilities available in the application area rather than having strict requirements [5]. Moving forward from the PWA definition, this section clarified the requirement and architecture approach for our PWAs project case.

A. Project Requirement

Streaming services of all types are becoming some of the most popular and influential platforms in the world of video streaming industry. Streaming has quickly developed a magnitude 'network effect' in a variety of different media markets such as music, television, film and even podcasting. These days, anyone with a mobile device has access to seemingly endless hours of free and paid streaming content. Not only is the market widespread, but it is also becoming highly profitable [20]. Throughout this project, PWA technology is widely introduced to reconstruct the life of traditional web applications and deliver rich content for media streaming.

With the purpose of providing an online and/or offline streaming capabilities, PWAs with service worker and manifest files inherits the characteristics of a modern web app and upgrades to become a look and feel like native app. Moreover, users can interact and experience all main functions of a streaming application such as download content in different resolution by utilizing network capabilities. Furthermore, network-agnostic feature is recommended to indicate the client apps when the connection goes offline and device-agnostic feature whether they streaming the content via mobiles or desktops.

In addition, offline functionalities and background synchronization are attainable so users can send a new client request even when they are offline. Push notification will be prompted to all subscribed devices after a client synchronous the cache operation to the database (consider the role of Lightweight MQ in Figure 2). By tapping on the notification, users are redirected to the selected page (downloaded page, playlist page or main movie page).

In order to develop and deliver function as describe above, minimum requirement for PWAs needs at least:

- Service worker responsible for caching the shell of the web application and it is this cached shell on which app shell relies on. Service worker offer a threads to act as network proxies between server and client app in order to provide offline support, resource caching, push notifications, background data sync, and page load performance optimizations [1, 7, 11, 12, 31].
- Application manifest. Provide a JSON-based metadata file describing key information about web app (such as icons, language, and URL entry point). Application manifests also an approach that we can use to take advantage of Service Worker caching to provide your users with meaningful pixels instantly. Particularly, the application manifest enables a user to install web applications to the home screen of their device and allows you to customize the splash screen, theme colors, and even the URL that's opened. Another way round the client able to provide PWA users with an installable, native app-like experience [1, 7, 11, 12, 31].

On Figure 3 show skeleton of minimum requirement for our project case which run on Chrome browser and simple Service Worker.

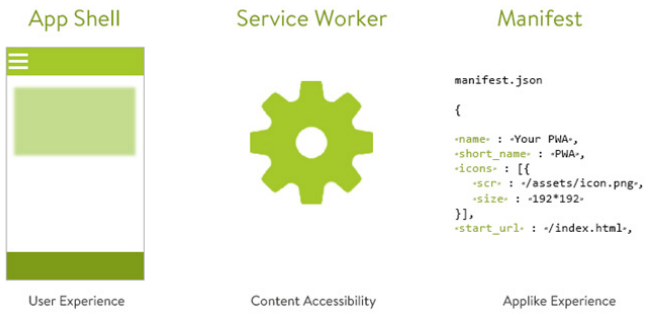

Figure 3 - Project MVP (minimum viable product)

Moving forward from minimum viable product to PWAs project capabilities we carry offer cross-browser compatibility, network-agnostic, and device-agnostic feature as additional work during our project execution.

B. Project Start Architecture

To deliver media streaming with PWA framework, our project case also offers the following additional capabilities:

- A lightweight UI/UX library to work in different device and browser.
- Work in Electron Player and major browser such as Mozilla Firefox, Microsoft Internet Explorer, Microsoft Internet Edge, Opera Browser.
- Device-agnostic, which means request from user will automatically detect either desktop or mobile device and the UI/UX will adjust automatically based the device metadata.

Furthermore, Table 1 depicted out high-level requirement which we carried during project execution. Addition to requirement engineering, the priority goes from scale 1 to 5. Requirement with priority 1 indicated high and/or must have requirement, and scale 5 indicate low priority and/or would not implemented [16]. Apart of the requirement priority, we also classified the type requirement into 3 main categories which are: (i) UI/UX, (ii) integration, and (iii) compatibility.

TABLE I. FUNCTIONAL REQUIREMENT

ID	Functional Requirement		
	Requirement Description	***Type***	***Priority***
FR1	Display homepage, detail movie page and playlist page in browser	UI/UX	1
FR2	Work in Chrome browser	Compatibility	1
FR3	Able to run smoothly in major browser including: Microsof IE, Microsoft Edge, Mozilla Firefox, Apple Safari.	Compatibility	3
FR4	Embeding Shaka Player	Integration	1
FR5	Media playback to enable user control the movie behaviour (e.g: speed, time, pip)	Integration	1
FR6	Movie / streaming control enable user control the movie including download feature, subtitle and network bandwith selection	UI/UX	1
FR7	Display push notification	UI/UX	2
FR8	Display homepage in mobile	UI/UX	2
FR9	Network-agnostic to detect bandwith consumption and show offline and online	Integration	1
FR10	Work in Electron desktop application	Compatibility	3

Non-functional requirements (NFR) also highlight in our proof of work. Our application describes two quality models for non-functional requirements: one for quality in use and product quality. These quality models can be used to collect and specify non-functional requirements of PWAs in a standardized way [23]. Furthermore, our project case NFR apart of performance efficiency and compatibility [23]. In terms of performance we refer to the characteristic of resource utilization and capacity. Resources utilization deal with network and memory consumption while we run PWAs and capacity will refer to the approach of data store in browser. Technique for data cache introduce in PWA namely: IndexDB [22], WebSQL [40], LocalStorage [25]. Table 2 highlights the NFR approach in terms of capacity consumptions and deal with persistence mechanism for PWAs.

TABLE II. NON FUNCIONAL REQUIREMENT

Technique	Storage comparison using 4 major browser			
	Chrome (40)	***Firefox (34)***	***Safari (7,8)***	***IE (10,11)***
IndexDB	up to quota	50MB, Unlimited	up to quota	100MB
WebSQL	up to quota	NA	5MB up to 700MB	NA
LocalStorage	up to quota	10MB	5MB	10MB

Storage comparison adopted from [41].

C. Architecture Decision

With the purpose of providing an online and/or offline video streaming capabilities, PWAs with Service Worker and Manifest Files inherits the characteristics of a modern web application and upgrades to become a look and feel like native application. Overall component employed in our project case show in Figure 4 below.

During project execution stage, we employ UI/UX framework using ReactJS [46] and Styled Components [47]. Furthermore, we wrapped UI/UX using HTML5 [48] and supported with the JavaScript libraries from [24, 42]: (i) plugin.js, (ii) bootstrap.js, (iii) custom.js, (iv) jquery.js. Additionally, we implemented latest libraries from: modernizr.js [43], webpack.js [50], and owl.carousel.js [28]. All in all, library dependency in our project manages by NPM [27].

With the decision UI/UX framework that implemented in our project case, our approach break away from the confines of the old web, our UI/UX adapt as well. This starts with our app's icon on the homescreen, continues through adapting our design for the limitations of each medium (e.g.: missing address bar and back button in full-screen or changing screen

orientation in mobile websites) and culminate in instilling assurance in the application regardless of mobile network conditions. Within our UI/UX we also reflect changing network conditions verbally and this also be communicated visually in our project case, refer to wireframe on Figure 5.

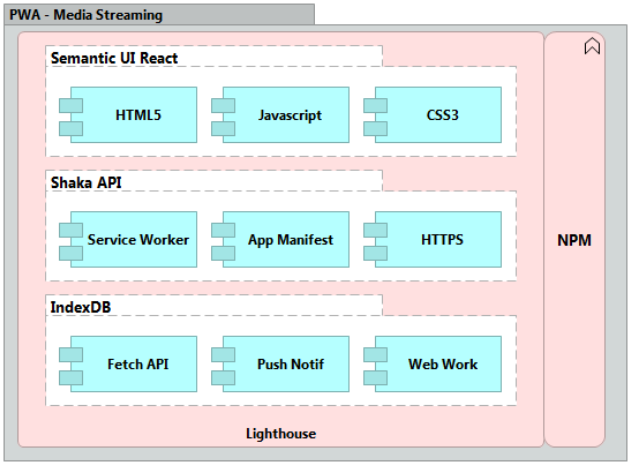

Figure 4 - Component for PWAs

Finally our decision on UI/UX marking the design philosophy which adapt and fits with requested environment (e.g.: mobile, desktop, low latency network). Consider our application shell with a large amount of dynamic content that is cached on demand as client requested our application.

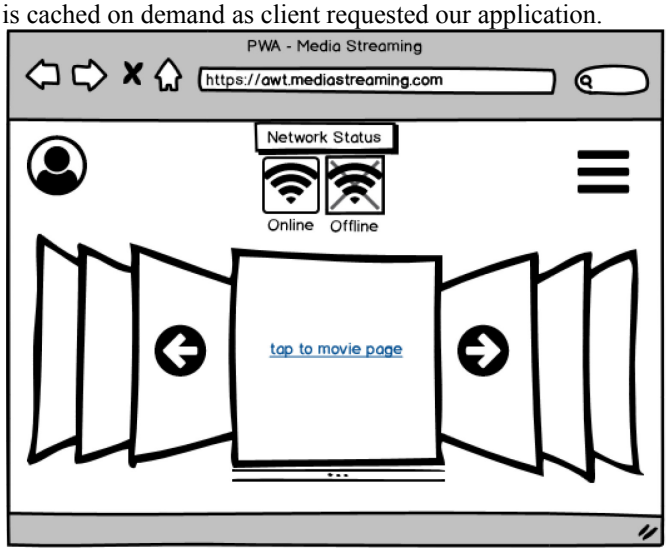

Figure 5 – Homepage (modified project MVP)

To meet the objective of delivering media playback, we decide to go with Shaka API [33] which offers adaptive stream using modern browser technologies and size of core library relatively small, simple, and free from third-party dependencies. Additionally, our project also offer stand alone media player using Electron[1]. Approach on the integration with UI/UX shown in Figure 6.

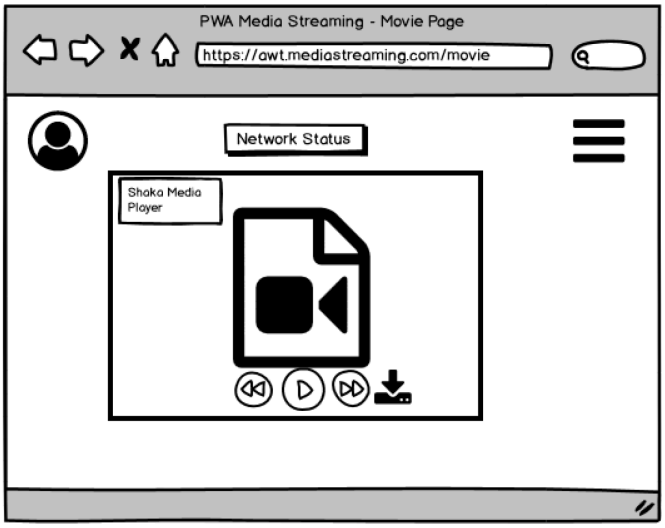

Figure 6 – Movie page using Shaka Player

After Shaka Player successfully integrated with UI/UX, we work toward persistence strategy. In case of background data synchronization (BackgroundSync API), transaction can be queues as a safe wrapper around an operation or group of PeriodicSync [11] hence we decide implement IndexDB and LocalStorage implementation as cache operation for our application and simple data sync operation for a predetermined time [11, 22, 25].

D. Application Process

In our project case we have made simple operation which shown on Figure 7.

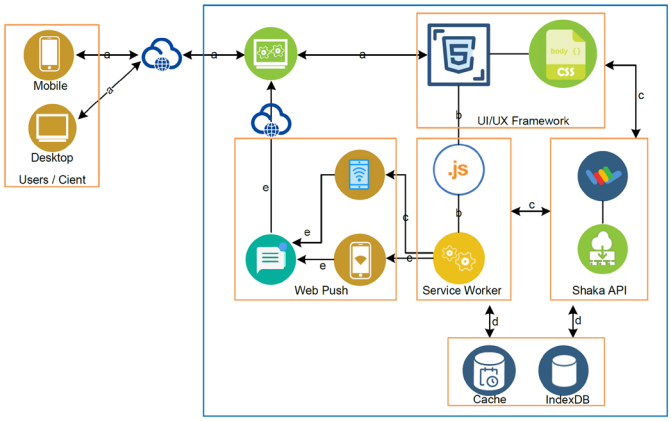

Figure 7 – Simplified process PWAs Media Playback

As depicted in Figure 7, below process carried out during the user (clients) interactions with our PWAs.

a) user navigate to our URL (*https://awt.pwa-mediastreaming.com*[2]) and the content request will be fine-tune based on user device (e.g.: mobile or desktop browser). While navigating in application, we offer "hamburger menu" which show on the Figure 8.

[1] Electron — Build Cross Platform Desktop Apps with JavaScript, HTML, and CSS. Retrieved from: https://electronjs.org/

[2] Public domain is not acquired & opens to public.

Figure 8 - Navigation menu

b) once the user landing in our homepage, we register Service Worker according to user preference and during the registration process, the browser downloads, parses, and executes the Service Worker. Finally the thread also assign when user request for other page action (e.g.: Movie Page or Download Page).
c) immediately after the Service Worker executes, the install event is activated. When client request for content download, Shaka API and Service Worker responsible to manage the capabilities includes the associating Shaka API into existing thread and enable dynamic content control on the requested movie. During "on the fly compilation", Service Worker also contributes to manage the persistent strategy defines by client behaviour (browser or mobile).
d) after successful initial life-cycle, the Service Worker is now able to control clients and handle functional events. If client interaction request media to be downloaded, subsequently Service Worker initiate the Quota Management API which implemented using IndexDB and LocalStorage.
e) in case of NFR (e.g.: network availability) our PWAs able to detect when client lost connection to the internet. Service Worker will work to send web push (notification) to client devices and Manifest File will have responsibility to work with minimum UI/UX capabilities.

Our project case presents a pattern shift in the way of legacy web vs. modern web application using PWAs framework. Proof of works provides an experience that is beyond users expectations of the legacy web. Users do not expect web applications to continue working when they are offline. Yet our project case over-deliver on those expectations. Clients side do not expect web apps to send them updates when new information relevant to them is available. Yet we offer push notification feature over-deliver on those expectations too. Finally, we offer intuitive navigation menus on desktop and mobile and it almost synonymous with the hamburger icon (even in offline we have its own hamburger).

IV. Project Challenges

Surprisingly, PWAs combines all the optimal features of traditional websites and native apps to create a significant generation of web application. Specifically, PWA is the most suitable approach for media streaming provider that focuses on providing movie and live streaming capabilities. With its superior features and distribution capabilities, PWA is evaluated to replace all traditional web applications in the near future. Not only does PWA compete robustly with classic web applications, it is predicted as a challenger to native applications. Furthermore, a PWA improves user experience by supporting a system of identical app-like features that users are acquainted with. Therefore, users meet no difficulties in altering between devices or platforms.

Throughout the project, the most challenging part is getting familiar with different feature UI/UX framework since every predefined front end uses different layout strategy. Furthermore, notifications event for web push, inform clients that they will receive updates as they become available (we think about requesting permission to send push notifications, consider two requirements: timing and content message). At the time of integration development, we faced the integration with Shaka Player frequently corrupted if the UI/UX implementation absent during runtime compilation by NPM which result the application will not work as usual. Considering all the technological aspects (browser platform and API for media streaming), the profound roadblock is how well PWAs framework helps our development that will support cross-browser functionality. Significant challenge that we have face is cross browser functionality which always brings halt to our proof of work during the test cycle.

V. Conclusions

We have presented a study and project case to success factors PWAs development for media playback. Our anchored research goals include an in-depth assessment of PWAs development frameworks within video streaming industry. Evaluations performed during project execution stage encompassed three main areas, and that highlights the role of UI/UX, Service Worker and Application Manifest in the context of PWAs framework.

Concerning user experience, PWAs have been able to offer a much better experience compared to traditional websites. We offer 'mobile-first' [31] feature in our project, this attribute giving the user the best possible experience for his device when connected online and 'offline-first' [31] which reflected the behavior of our application used the best possible experience to adapt with current network condition (limited bandwidth or offline). 'Mobile-first' and 'Offline-first' has been proof in our project which means our media playback application able to cache its own interface and logic locally, as well as the content of the recent user state. The web application framework is evolving and technology giant (namely: Apple, Google, Microsoft, Mozilla, and Facebook) contribute to support the development of HTML5, CSS and JavaScript as open standards. A PWA is built on top of those open standards and this one rationale why PWAs framework more capable to release hidden features which previously only available in legacy web, native and cross-platform applications.

Service Workers are the key to unlock the power within browser application. Think of them as 'air traffic controllers' that are capable of intercepting HTTP requests. The web has always offered something awesome, and there's no reason

why we can't improve the application and get ahead greater features to our users [11]. Finally, PWAs help to contribute toward richer development, harmonized web and mobile development through uniform adoption, and increasing end users participation which will influence significant monetary impact for the organization.

VI. Future Works

There are still many challenges to tackle, such as the human side particularly in the form of end-user experience. Future work will include a longitudinal research study to investigate framework maturity, as well as an in-depth perspective on customer experience, user interface design and interaction.

Feature we also consider "design that should fits its environment". We consider when client navigate to *https://awt.pwa-mediastreaming.com* using mobile device, the client will be redirected to *https://m.awt.pwa-mediastreaming.com* and shown a different version (lite version). This requirement includes using a range of devices instead of focusing on reduced categories as a mobile-first approach. Passing through a different set of locations should include taking advantage of the available options, such as a larger screen and adapting to smaller screen limits. Future work will also consider the screen as a modality point of choice given the availability of voice, touch, and other modes (multimodal interaction).

While we encourage the 'lite version' approach, we assured it will serve our NFR, because it takes up less than a megabyte of storage, and claims it can save up to 70% on data while loading 30% faster [36, 37]. However, additional interesting feature is to consider the impact of "Add to Homescreen" vs. installation of applications through the application marketplace. We profound the idea will have significant impact for AppStore or PlayStore, since the users/client able to 'experiment first' on browser prior to downloading the applications from marketplace.

PWA for media playback has been shown in the majority of project objectives by demonstrate how PWA works. Moreover, media playback application still could be more intuitive and approach larger groups of users. Other native application features such as tweaking Electron to achieve in future work. In this application, pure JavaScript is used. However, modern ECMAScript (ES2019) could replace old JavaScript for structural code. Nevertheless, all JavaScript files should be condensed to minified version as offer with latest update of bootstrap.js, custom.js, and jquery.js library files in order to reduce the file sizes and accelerate execution time [24, 42].

Acknowledgment

This document is based on the (not yet published) proof of work 'PWAs for Media Streaming' in Advanced Web Technology Project, written under chair of Open Distribution Systems at *Technische Universität Berlin*.

Findings have first been condensed before we extended and revised the work. We would like to acknowledge our project supervisor *Alexander Futasz* and *Stefan Fam*. During project execution stage we have been provided with ample feedback that helped us to provide applied, relevant research and in general improve the project delivery and the paper.

We acknowledge the website *https://awt.pwa-mediastreaming.com* which has been highlight on this paper is not yet publically published, hence our proof of work available online at GitHub *https://github.com/azmiruddin/awt-pwa-mediaplayback* and GitLab repository https://gitlab.tubit.tu-berlin.de/azmiruddin/awt-pwa-mediaplayback. Finally, with our email we declare that the present project case and associated document was composed by ourselves and that the work contained herein is our own. We also confirm that we have only used the specified resources respectively in references section.

References

[1] A. Biørn-Hansen, T. A. Majchrzak, and T. M. Grønli, "Progressive web apps: The possible web-native unifier for mobile development," *WEBIST 2017 - Proc. 13th Int. Conf. Web Inf. Syst. Technol.*, no. Webist, pp. 344–351, 2017.

[2] A. Zambelli and M. T. Evangelist, "IIS Smooth Streaming Technical Overview," *Whitepaper, Microsoft Corp.*, no. March, 2009.

[3] Adobe, "Adobe HTTP Dynamic Streaming," http://www.adobe.com/products/httpdynamicstreaming/. [Accessed: Jan. 25, 2020].

[4] Getting started with Redux framework. Retrieved from: https://redux.js.org/introduction/getting-started. [Accessed: Nov. 22, 2019].

[5] B. Frankston, "Progressive Web Apps [Bits Versus Electrons]," *IEEE Consum. Electron. Mag.*, vol. 7, no. 2, pp. 106–117, 2018.

[6] Bitmovin, "BITMOVIN bitdash MPEG-DASH Clients," 2016.

[7] C. Love, *Progressive Web Application Development by Example Develop*. Birmingham: Packt Publishing, 2018.

[8] C. Mueller, "MPEG-DASH vs. Apple HLS vs. Microsoft Smooth Streaming vs. Adobe HDS," *Bitmovin*, 2015. [Online]. Available: https://bitmovin.com/mpeg-dash-vs-apple-hls-vs-microsoft-smooth-streaming-vs-adobe-hds/. [Accessed: Feb. 05, 2020].

[9] C. Müller, S. Lederer, and C. Timmerer, "An evaluation of dynamic adaptive streaming over HTTP in vehicular environments," *MoVid'12 - Proc. 4th Work. Mob. Video*, pp. 37–42, 2012.

[10] C. Timmerer, D. Weinberger, M. Smole, R. Grandl, C. Muller, and S. Lederer, "Live transcoding and streaming-as-a-service with MPEG-DASH," *2015 IEEE Int. Conf. Multimed. Expo Work. ICMEW 2015*, vol. 610370, 2015.

[11] D. A. Hume, *Progressive Web Apps*, 5th ed. New York: Manning Publications, 2018.

[12] D. Sheppard, *Beginning progressive web app development creating a native app experience on the web*. 2017.

[13] DASH an overview, retrieved from: https://cdn.dashjs.org/latest/jsdoc/index.html. [Accessed: Nov. 22, 2019].

[14] Developer Google Showcase. Retrieved from: https://developers.google.com/web/showcase/2017/mynet. [Accessed: Feb. 15, 2020].

[15] Digest the Netflix UI. Modernizing the Web Playback UI, showcase from Netflix. Retrieved from: https://netflixtechblog.com/modernizing-the-web-playback-ui-1ad2f184a5a0. [Accessed: Feb. 15, 2020].

[16] E. Miranda, "Time boxing planning: Buffered MOSCOW rules," *ACM SIGSOFT Softw. Eng. Notes*, vol. 36, no. 6, p. 1, 2011.

[17] Embedding an MPEG-DASH Adaptive Streaming Video in an HTML5 retrieved from https://docs.microsoft.com/en-us/azure/media-services/previous/media-services-embed-mpeg-dash-in-html5. [Accessed: Feb. 15, 2020].

[18] Forbes https://www.forbes.com/sites/forbespr/2017/03/07/forbes-launches-all-new-mobile-web-experience-for-forbes-com/#4f4194925168. [Accessed: Nov. 22, 2019].

[19] Google Developer, Introduction to PWA Retrieved from: https://developers.google.com/web/progressive-web-apps/. [Accessed: Nov. 22, 2019].

[20] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, “Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation,” *Proc. - 2nd ACM Int. Conf. Mob. Softw. Eng. Syst. MOBILESoft 2015*, no. March 2017, pp. 56–59, 2015.
[21] I. Sodagar, "The MPEG-DASH Standard for Multimedia Streaming Over the Internet," in *IEEE MultiMedia*, vol. 18, no. 4, pp. 62-67, April 2011. doi: 10.1109/MMUL.2011.71
[22] IndexDB API, retrieved from: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API. [Accessed: Nov. 22, 2019].
[23] ISO: ISO/IEC 25010: 2011, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.
[24] jQuery API, Library. Retrieved from: https://api.jquery.com/. [Accessed: Nov. 22, 2019].
[25] Local storage with Window.localStorage retrieved from: https://developer.mozilla.org/en-US/docs/Web/API/Storage/LocalStorage. [Accessed: Nov. 22, 2019].
[26] Microsoft, “IIS Smooth Streaming Transport Protocol,” http://www.iis.net/community/files/media/smoothspecs/[MS-SMTH].pdf, September 2009. [Accessed: Jan. 25, 2020].
[27] NPM Documentation. Retrieved from: https://docs.npmjs.com/about-npm/. [Accessed: Nov. 22, 2019].
[28] Owl Carousel 2, Library. Retrieved from https:// https://owlcarousel2.github.io/OwlCarousel2/. [Accessed: Nov. 22, 2019].
[29] Pantos, R. and May, E. W., “HTTP Live Streaming”, Informational Internet Draft, http://tools.ietf.org/html/draft-pantos-http-live-streaming-06, March 2011. [Accessed: Nov. 22, 2019].
[30] PWA Book, Open Guide to Progressive Web Apps. Retrieved from: https://divante.com/pwabook. [Accessed: Feb. 05, 2020].
[31] R. Baxter, N. Hastings, A. Law, and E. J. . Glass, *Building Progressive Web Apps*, vol. 39, no. 5. O’Reilly Media, 2008.
[32] S. Lederer, “Video Developer Report 2018,” *Bitmovin*, p. 25, 2018.
[33] Shaka Player, An adaptive video player library for the web. Retrieved from https://opensource.google/projects/shaka-player. [Accessed: Nov. 22, 2019].
[34] Streaming - Statistics & Facts retrieved from: https://www.statista.com/topics/1594/streaming/. [Accessed: Feb. 05, 2020].
[35] T. A. Majchrzak, A. Biørn-Hansen, and T.-M. Grønli, “Progressive Web Apps: the Definite Approach to Cross-Platform Development?,” *Proc. 51st Hawaii Int. Conf. Syst. Sci.*, pp. 5735–5744, 2018.
[36] Twitter, PWA with support for push notifications. Retrieved from: https://www.windowscentral.com/twitter-updates-its-windows-10-uwp-app-pwa-support-push-notifications. [Accessed: Feb. 05, 2020].
[37] Twitter Lite and High Performance React Progressive Web Apps at Scale. Retrieved from: https://medium.com/@paularmstrong/twitter-lite-and-high-performance-react-progressive-web-apps-at-scale-d28a00e780a3. [Accessed: Feb. 05, 2020].
[38] W. Jason, "Impact of PWA" retrieved from https://blogs.gartner.com/jason-wong/pwas-will-impact-your-mobile-app-strategy/. [Accessed: Feb. 05, 2020].
[39] Number of apps available in leading app stores 2019, retrieved from https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. [Accessed: Jan. 25, 2020].
[40] Web SQL Database, retrieved from: https://www.w3.org/TR/webdatabase/
[41] Working with quota on mobile browsers retrieved from: https://www.html5rocks.com/en/tutorials/offline/quota-research/. [Accessed: Jan. 25, 2020].
[42] Working with mobile-first sites, with Bootstrap CDN https://getbootstrap.com/docs/4.4/getting-started/introduction/. [Accessed: Nov. 22, 2019].
[43] What is Modernizr. Retrieved from: https://modernizr.com/docs. [Accessed: Nov. 22, 2019].
[44] What is React, Getting Started with ReactJS. Retrieved from: https://reactjs.org/docs/getting-started.html. [Accessed: Nov. 22, 2019].
[45] What is Vanilla-JS, retrieved from: http://vanilla-js.com/. [Accessed: Nov. 10, 2019].
[46] ReactJS adopted from https://reactjs.org/docs/getting-started.html. [Accessed: Nov. 10, 2019]
[47] Styled-Component adopted from https://styled-components.com/docs. [Accessed: Nov. 22, 2019].
[48] HTML5 latest documentation adopted from https://html.spec.whatwg.org/multipage/. [Accessed: Nov. 22, 2019].
[49] Silverlight demo website. Akamai HD for Sliverlight http://wwwns.akamai.com/hdnetwork/demo/silverlight/default.html. [Accessed: Nov. 22, 2019].
[50] WebPack Concepts. Retrieved from: https://webpack.js.org/concepts/. [Accessed: Nov. 10, 2019].