

A. B. ASSETBEKOV

*Kazakh-British Technical University,
Almaty, Kazakhstan
almar.asetbekov@gmail.com*

PROGRESSIVE WEB APPLICATIONS CAPABILITIES TO BECOME AN ALTERNATIVE TO MOBILE APPLICATIONS

These days, almost every successful company owns a web application for their business. Essentially, companies try to develop a website that is easy to navigate, so that a user could have a great experience regardless of a used device. But, a web app, even if developed to be used on any device, can be constraining for a user as it does not have features that have been reserved to native apps. Thus, companies are forced to develop a native application for their product and two applications are needed for two different OS - IOS and Android - to reach all potential users and match the modern criteria of a successful product. However, such a strategy of developing a web and two native applications are time and money consuming, which is far from ideal from a business point of view.

In 2015, Google introduced Progressive Web Apps (PWA), which aims to close the gap between web and native applications by combining the best features from web and native apps. In this article I am going to describe several APIs that can make PWA feel like a native app.

Key words: *progressive web applications, PWA, service worker, app manifest, API, mobile app, native application*

Introduction. Progressive web apps are web applications that behave and feel like a native one. That is to say, PWA should have all the functionalities that a platform-specific application has. It is done by implementing web-platform specific features such as service workers and manifests with progressive enhancement and certain APIs.

A service worker is a script that a browser runs in the background, separate from a web page acting as a network proxy. It intercepts any request and then decides whether it should serve the resource from the cache via the Cache Storage API, from the network as normally would happen without a service worker, or create it from a local algorithm. [1]

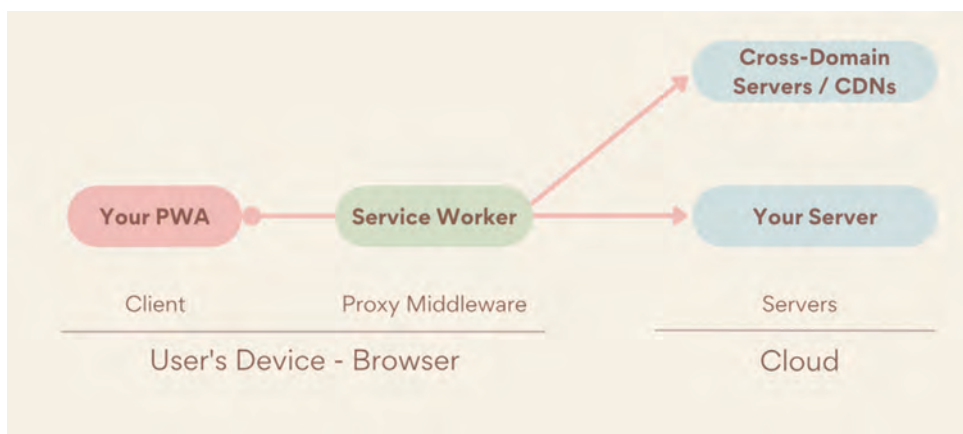


Figure 1 – Service Worker as a network proxy

The web app manifest is a JSON file that provides information about the application so that it displays properly as a native app when installed. The basic file must contain an app name, icon and URL that opens when the app launches.

These days technologies are evolving at a rapid pace, in particular web and mobile development. Big companies implement new technologies with better approaches into their applications/products. Thus, they set the bar higher for other companies to stay relevant and successful. As products on the market get better in general, users' demand grows as their expectations from other products' capabilities. That is why it is vital to have a reliable and up to date application.

In relation to the network connection, users are used to getting the information fast even with an unreliable and poor connection. They tend to leave an application if the wait is too long. For example, as page load times go from 1 second to three seconds, the probability of a user bouncing increases by 32% and the probability goes up to 123% by 10 second loading mark as illustrated in Figure 2. [2]

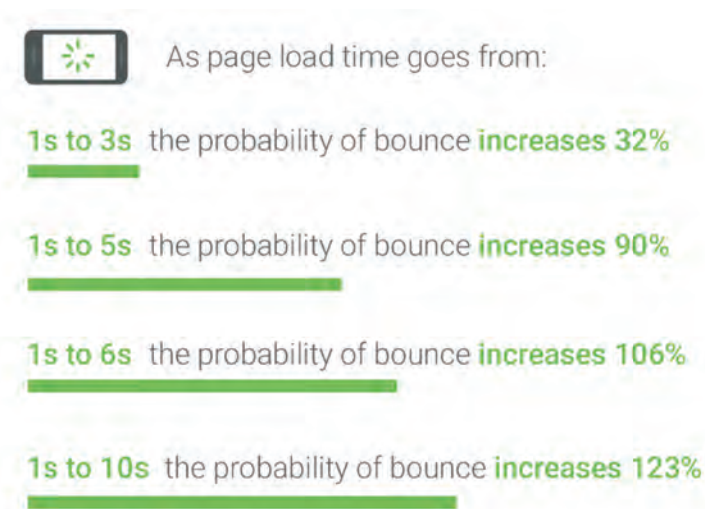


Figure 2 – The probability of bounce

Users' behavior and expectations vary when they use web and mobile applications. They expect certain features from both types of applications. So, the goal is to make an app that provides native-app experience in such a way that a user assumes he uses a mobile app when in fact it is PWA.

Until the appearance of PWA, platform-specific apps have been the only ones with features like push notifications, offline mode, idle detection, home screen installation and so on. Even though not every platform-specific feature can be implemented into PWA now, there are ongoing improvements in terms of new APIs that provide the capability to make PWAs with native-app experience. All of the APIs are built with the web's secure, user-centric permission model so that a user does not have to worry about his vulnerable data.

In the Main Part I am going to explain several platform-specific features that can be implemented into PWA with the help of certain APIs.

Main Part. There are five features that are explained in this section. They are presented in the following order:

1. A contact picker
2. Device Orientation and Motion
3. Idle detection
4. Content sharing
5. App shortcuts

A contact picker

A user expects to get access to his contact list whenever he uses a mobile app. This feature does not exist on web applications, but it is much needed in PWA. Therefore, Google has created the Contact Picker API, which allows one-off access to a user's contact information with full control over the shared data.

Thus, use cases of the Contact Picker API are:

1. discovering a contact who uses a social platform;
2. selecting a contact to send an email;
3. making voice calls with voice over IP, which could get a phone number from a contact list;

Security

A permission to use a contact list must be granted by a user every time when the request to the list is made. This differs from native applications where a user grants access to a contact list once. Additionally, there is no option to select all contacts to make sure that a user selects only needed contacts.

The API is only available from a secure top-level browsing context. It means that to run the code HTTPS is required. So, to develop locally a developer would need an SSH tunnel as localhost uses an HTTP connection.

The Chrome team took these measures using the core principles defined in Controlling Access to Powerful Web Platform Features for security reasons. [3]

Browser compatibility (Figure 3) depicts the main problem of the Contact Picker API - Safari IOS does not support it yet. As it can be seen, only Android browsers have full compatibility except Firefox for Android. Obviously, desktop browsers do not support this API as a contact list only exists on mobile devices.

Device Orientation and Motion







Most mobile applications can rotate the display to work in landscape mode. The most prominent example is YouTube where display rotation is the most common feature. There are other mobile applications which highly rely on this functionality as it provides a better UX.


DeviceOrientationEvent is an experimental Web API event which could be used to implement display rotation in PWA.


Additionally, this event could be implemented with geolocation for turn-by-turn navigation.


There is another experimental Web API - DeviceMotionEvent. This event is suitable for gaming and fitness applications, because it helps with character's/person's movement.

Coordinate system

	PC						Mobile					
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
ContactsManager  	No	No	No	No	57	No	80	80	No	57	No	13.0
getProperties  	No	No	No	No	No	No	80	80	No	57	No	13.0
select  	No	No	No	No	No	No	80	80	No	57	No	13.0

 Full support

 No support

 Experimental. Expect behavior to change in the future.


 Non-standard. Check cross-browser support before using.

Figure 3 – Browser compatibility of Contact Picker API

The Figure 4. represents a device coordinate frame and three values (alpha, beta, gamma) in the frame. X, y and z are in the plane of the screen and are positive towards the right, towards the top and out of hand side of the screen, respectively. Alpha, beta and gamma are properties of the DeviceOrientationEvent.

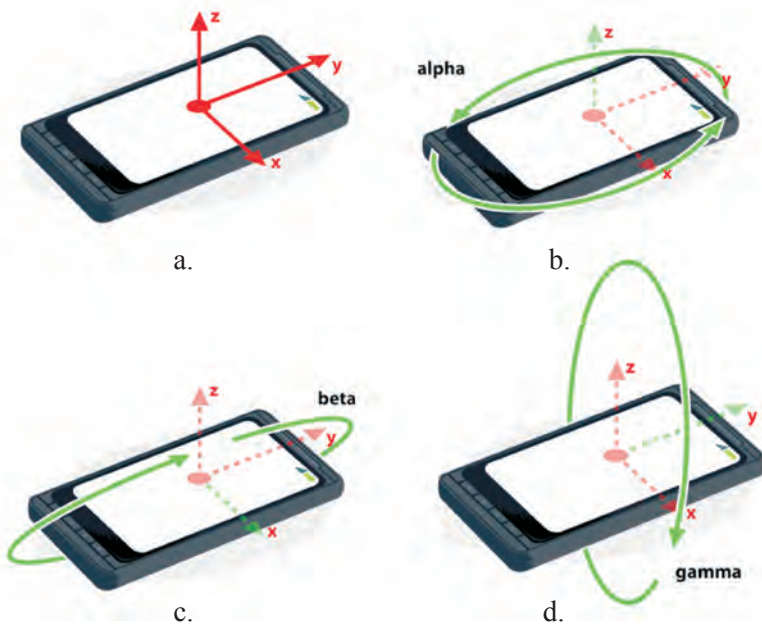



Figure 4 – Device coordinate frame.

The DeviceOrientationEvent returns rotation data, so that a developer knows whether a device is leaning front-to-back or side-to-side and to which degree. Therefore, the data allows a device to interactively respond to orientation changes. The DeviceMotionEvent provides the speed of device positioning and orientation changes, i.e. acceleration, and returns data about the rotation (in °/second).

The Device Motion Event includes four properties: acceleration, acceleration Including Gravity, rotation Rate and interval.

The acceleration and acceleration Including Gravity properties are objects providing data about acceleration on x, y and z axis, which represent the axis from West to East, from South to North and perpendicular to the ground, respectively.

Rotation Rate corresponds to the same alpha, beta and gamma values described earlier.

	🖥️						📱					
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
DeviceMotionEvent	31	12	6	11	18	No	37	31	6	18	4.2	2.0
DeviceMotionEvent() constructor	59	14	29	No	46	No	59	59	29	43	No	7.0
acceleration	31	12	6	11	18	No	37	31	6	18	4.2	2.0
accelerationIncludingGravity	31	12	6	11	18	No	37	31	6	18	4.2	2.0
interval	31	12	6	11	18	No	37	31	6	18	4.2	2.0
requestPermission 	No	No	No	No	No	No	No	No	No	No	14.5	No
rotationRate	31	12	6	11	18	No	37	31	6	18	4.2	2.0

Full support

No support


 Experimental. Expect behavior to change in the future.

Figure 5 – Browser compatibility of Device Motion Event.

Idle Detection

When a user does not use his device for more than a minute (if not changed manually), the device goes into an idle mode. This mode means that a device is on but the screen is off.

The Idle Detection API gives more flexibility to web development as it provides a user (active or idle) and a screen (locked or unlocked) idle state. That is to say, developers can

	🖥️						📱					
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
DeviceOrientation-Event	7 ★	12	6 ★	11	15	No	3 ★	18 ★	6 ★	14	4.2	1.0 ★
DeviceOrientation-Event() constructor	59	14	17	No	46	No	59	59	17	43	No	7.0
absolute	7	12	6	11	15	No	4.4	18	6	14	No	1.0
alpha	7	12	6	11	15	No	3	18	6	14	4.2	1.0
beta	7	12	6	11	15	No	3	18	6	14	4.2	1.0
gamma	7	12	6	11	15	No	3	18	6	14	4.2	1.0
requestPermission 🚧	No	No	No	No	No	No	No	No	No	No	14.5	No

Full support

No support

Experimental. Expect behavior to change in the future.

See implementation notes.

Figure 6 – Browser compatibility of Device Orientation Event.

start or stop the execution of the particular code by looking at the idle state and, consequently, provide a better UX.

The use cases of the API are the following:

1. Networking or messaging applications to see if someone is available at the moment (online/away/offline);
2. Publicly used applications return to the home page if there is no interaction with the application.
3. Updating a service worker when the app is not used.

There is one remark when using the API - a user has to grant the permission for the API to be used. Otherwise, the idle detection will not work. The Chrome team has designed and implemented the API this way for security reasons using the core principles defined in Controlling Access to Powerful Web Platform Features[3].

Initially, idle detection was gated behind the notifications permission, but the Idle Detection spec editors have decided to gate it behind a dedicated idle detection permission. Referring to native applications, there is no need for a user gesture to grant the permission as such applications have access to locked/idle states. So, the Idle Detection API implemented

in PWA is more transparent as it gives a choice to a user whether to allow the idle detection or not.

	Desktop						Mobile					
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
<code>IdleDetector</code>	94	94	No	No	80	No	94	94	No	No	No	No
<code>IdleDetector()</code> constructor	94	94	No	No	80	No	94	94	No	No	No	No
<code>onchange</code>	94	94	No	No	80	No	94	94	No	No	No	No
<code>requestPermission</code>	94	94	No	No	80	No	94	94	No	No	No	No
<code>screenState</code>	94	94	No	No	80	No	94	94	No	No	No	No
<code>start</code>	94	94	No	No	80	No	94	94	No	No	No	No
<code>userState</code>	94	94	No	No	80	No	94	94	No	No	No	No

Full support No support

Experimental. Expect behavior to change in the future.

Figure 7 – Browser compatibility of Idle Detection API.

Content sharing

Content sharing is one of the most used functionalities in native applications. It helps users to save time by doing the minimal effort of clicking the share button and picking the receiver.

Web Share Target API enables a web site to receive shared data from other sites or apps. The goal is to allow PWA to appear in the UI for picking an app to share to. This way content could be shared from native to web apps and vice versa.

In order to use Web Share Target API web app manifest file must be updated with `share_target` entry so PWA is registered as a share target. A developer explicitly sets what kind of data PWA will accept. The most common scenario is accepting data, links, and text. Additionally, files could be accepted and some application changes as well.

Web Share API makes it possible for web apps to share links, text, and files to other apps installed on the device to use the same system-provided share capabilities as platform-specific apps.

For a progressive web application to appear in the sharing UI among native applications the PWA must be installed on a device. After installing a user can open any app that supports share functionality and by clicking the share button a user will be prompted with a target picker where the installed PWA will be displayed (Figure 8). Finally, the shared data will be available in the PWA with the implemented Share Target API.

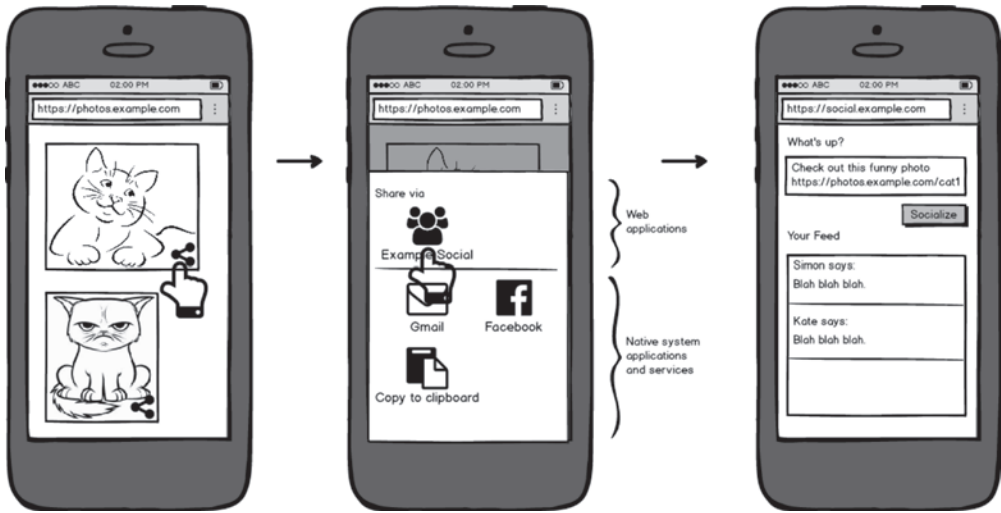


Figure 8 – Content sharing in PWA.

The Web Share API has two methods - `navigator.share()` and `navigator.canShare()`.

`Navigator.canShare()` is used before `navigator.share()` to identify whether the data is sharable or not.

`Navigator.share()` is used for sending the data to a share target. The method must be called on a transient activation, for example a click on the PWA icon in the sharing UI, which means that the method cannot be called programmatically without a user gesture. `Navigator.share()` is a promise-based method with a required properties object, which must contain at least one of the following properties: title, text, url or files

App shortcuts

App shortcuts are a handy way to get access to frequently used actions. So, a user can boost his productivity and facilitate re-engagement with an application. Shortcuts are invoked by right-clicking the app icon on desktop or long-pressing the icon on a mobile device.

This functionality used to be reserved to native applications, but now it is available for progressive web applications as well.

Shortcuts are defined in the app manifest json file in `shortcuts` entry, an array of objects. Each object must contain a name displayed in the context menu and a url within the application. There are other optional fields:

- 1) `Short_name` is used when there is insufficient space for a full name to be displayed;
- 2) `Description` is not used at the moment, but is going to be exposed to assistive technology;

	Desktop						Mobile					
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
share / canShare	89★	81★	71	No	75	12.1	No	61	79	48	12.2	8.0
data.files parameter	89★	89★	No	No	75★	15	No	76	No	54	15	11.0
data.text parameter	89★	89★	71	No	75★	15	No	76	No	54	15	11.0

Full support

Partial support

No support

User must explicitly enable this feature.

See implementation notes.

Figure 9 – Browser compatibility of Web Share API.

3) Icons are used for shortcut representation in the context menu;
 Best practices:

1) Order app shortcuts by priority. The available number of shortcuts varies depending on the platform from 3 in Chrome 92 for Android 7 to 10 in Chrome and Edge on Windows. So, the crucial shortcuts should be first in the array.

	Desktop						Mobile					
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
shortcuts	85★	85★	No★	No	71★	No★	84	84	No★	60	No★	14.0

Full support

Partial support

No support

Experimental. Expect behavior to change in the future.

Figure 10 - Browser compatibility of shortcuts API.

2) Use distinct names. As icons are optional there could be a situation when they are absent. Therefore, it is unsafe to rely on icons to describe what kind of app they represent.

3) Analyze shortcuts usage. To provide a better UX it is better to track what shortcuts are used more frequently than others.. This way the first point will be satisfied as well.

Conclusion. It is not necessary for a progressive web application to have all the native functionalities to work properly. That is why transforming a web application into a progressive web application is a gradual process, which allows developers to integrate native-like features one by one without breaking the application. The APIs described in the article have shown that features which have been reserved to native applications are now available in web applications due to certain APIs. All of the APIs are designed securely using the core principles defined in Controlling Access to Powerful Web Platform Features. [3] In most situations a user gesture is required to use the API for PWA, which differs from native apps where it is not needed due to the implementation differences. In other respects PWA feels like a native app, even though not every browser provides support for interfaces and methods of the APIs or they are in an experimental stage. So, a user experience can vary from browser to browser, which is the most apparent PWA issue. PWA is still an evolving technology and it will take time for browsers to catch up and be in sync with one another.

REFERENCES

- 1 Service Workers: an Introduction (2021). <https://developers.google.com/web/fundamentals/primers/service-workers>
- 2 Find out how you stack up to new industry benchmarks for mobile page speed (2018). <https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/>
- 3 Controlling Access to Powerful Web Platform Features <https://chromium.googlesource.com/chromium/src/+lkgr/docs/security/permissions-for-powerful-web-platform-features.md>
- 4 A contact picker for the web (2021). <https://web.dev/contact-picker/>
- 5 Contact Picker API (2021). https://developer.mozilla.org/en-US/docs/Web/API/Contact_Picker_API
- 6 Detecting device orientation (2022). https://developer.mozilla.org/en-US/docs/Web/Events/Detecting_device_orientation
- 7 Device Orientation & Motion (2019). <https://developers.google.com/web/fundamentals/native-hardware/device-orientation>
- 8 Detect inactive users with the Idle Detection API (2021). <https://web.dev/idle-detection/>
- 9 Is Chrome's Idle Detection really a threat to privacy? (2021). <https://scottiestech.info/2021/10/05/is-chromes-idle-detection-really-a-threat-to-privacy/>
- 10 Idle Detection API (2022). https://developer.mozilla.org/en-US/docs/Web/API/Idle_Detection_API
- 11 Integrate with the OS sharing UI with the Web Share API (2021). <https://web.dev/web-share/>
- 12 Receiving shared data with the Web Share Target API (2021). <https://web.dev/web-share-target/>
- 13 Web Share API (2022). https://developer.mozilla.org/en-US/docs/Web/API/Web_Share_API
- 14 Web Share Target API Explained (2020). <https://github.com/w3c/web-share-target/blob/main/docs/explainer.md>
- 15 Get things done quickly with app shortcuts (2021). <https://web.dev/app-shortcuts/>
- 16 Shortcuts (2021). <https://developer.mozilla.org/en-US/docs/Web/Manifest/shortcuts>

А. Б. ӘСЕТБЕКОВ

Қазақстан-Британ техникалық университеті, Алматы, Қазақстан
almar.asetbekov@gmail.com

**МОБИЛЬДІ ҚОЛДАНБАЛАРҒА БАЛАМА БОЛУ ҮШІН
ПРОГРЕССИВТІ ВЕБ- ҚОЛДАНБАЛАРЫНЫҢ МҮМКІНДІКТЕРІ**

Бұл күндері әрбір дерлік табысты компанияда өз бизнесіне арналған веб-қосымшасы бар. Негізінде, компаниялар пайдаланылған құрылғыға қарамастан пайдаланушы тамаша тәжірибеге ие болуы үшін шарлау оңай веб-сайтты жасауға тырысады. Бірақ веб-бағдарлама, тіпті кез келген құрылғыда пайдалануға арналған болса да, пайдаланушыны шектей алады, себебі оның мобильді қолданбалар үшін сақталған мүмкіндіктері жоқ. Осылайша, компаниялар барлық әлеуетті пайдаланушыларға қол жеткізу және табысты өнімнің заманауи критерийлеріне сәйкес келу үшін екі түрлі ОЖ, IOS және Android, үшін өз өнімі үшін жергілікті қосымшаны әзірлеуге мәжбүр. Дегенмен, веб пен екі мобильді қосымшаны әзірлеудің бұл стратегиясы уақыт пен ақшаны талап етеді, бұл бизнес тұрғысынан идеалдан алыс.

2015 жылы Google екеуінің де ең жақсы мүмкіндіктерін біріктіру арқылы веб және мобильді қолданбалар арасындағы ашақтықты жоюға бағытталған Progressive Web Apps (PWA) ұсынды. Бұл мақалада мен PWA-ны мобильді қолданбаға ұқсата алатын бірнеше API сипаттайтын боламын.

Түйін сөздер: прогрессивті веб-қосымшалар, PWA, сервис қызметкері, қолданба манифесті, API, мобильді қосымша, жергілікті қолданба.

А.Б. АСЕТБЕКОВ

Казахстанско-Британский технический университет, Алматы, Казахстан
almar.asetbekov@gmail.com

**ВОЗМОЖНОСТИ ПРОГРЕССИВНЫХ ВЕБ-ПРИЛОЖЕНИЙ
СТАТЬ АЛЬТЕРНАТИВОЙ МОБИЛЬНЫМ ПРИЛОЖЕНИЯМ**

В наши дни почти каждая успешная компания владеет веб-приложением для своего бизнеса. По сути, компании пытаются разработать веб-сайт, на котором легко ориентироваться, чтобы пользователь мог получить отличный опыт, независимо от используемого устройства. Но веб-приложение, даже если оно разработано для использования на любом устройстве, может ограничивать пользователя, поскольку оно не имеет функций, которые были зарезервированы для нативных приложений. Таким образом, компании вынуждены разрабатывать нативное приложение для своего продукта и необходимо два приложения для двух разных ОС, IOS и Android, чтобы охватить всех потенциальных пользователей и соответствовать современным критериям успешного продукта. Однако такая стратегия разработки веба и двух нативных приложений требует времени и денег, что далеко не идеально с точки зрения бизнеса.

В 2015 году Google представил прогрессивные веб-приложения (PWA), целью которых является сокращение разрыва между веб-приложениями и нативными приложениями за счет объединения лучших функций из обоих видов. В статье описано несколько API-интерфейсов, которые могут сделать PWA похожим на нативное приложение.

Ключевые слова: прогрессивные веб-приложения, PWA, сервис-воркер, манифест-приложения, API, мобильное приложение, нативное приложение.