ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

SCIENTIFIC Research and Community

Review Article Open Access

Comprehensive Analysis of Modern Application Rendering Strategies: Enhancing Web and Mobile User Experiences

Venkata Naga Sai Kiran Challa

USA

ABSTRACT
In today's fast-paced world of application development, how we render applications is crucial for delivering outstanding user experiences. This paper delves into the diverse rendering strategies used in modern web and mobile applications. For web applications, we explore Client-Side Rendering (CSR), Server-Side Rendering (SSR), Static Site Generation (SSG), Incremental Static Regeneration (ISR), Progressive Hydration, Streaming SSR, and Edge-Side Rendering (ESR). On the mobile front, we cover Native Rendering, Hybrid Rendering, Cross-Platform Rendering, Progressive Web Apps (PWAs), Server-Driven UI (SDUI), and Instant Apps. Drawing on real-world examples from Meta Platforms Inc., we showcase the practical benefits and challenges of each approach. Additionally, we discuss future improvements through AI/ML optimizations, highlighting the potential for personalized and high-performance applications. This paper aims to provide a clear and comprehensive guide to modern rendering techniques, helping developers create more interactive, scalable, and efficient applications.

***Corresponding author**
Venkata Naga Sai Kiran Challa, USA.

Received: November 05, 2022; **Accepted:** November 08, 2022; **Published:** November 18, 2022

Keywords: Application Rendering, Client-Side Rendering, Server-Side Rendering, Static Site Generation, Incremental Static Regeneration, Progressive Hydration, Streaming SSR, Edge-Side Rendering, Native Rendering, Hybrid Rendering, Cross-Platform Rendering, Progressive Web Apps, Server-Driven UI, Instant Apps, AI/ML Optimizations, Web Development, Mobile Development, User Experience, Meta Platforms Inc

Introduction
The development of mobile applications has gained prominence in the technological world, ceasing to be a trend and becoming a reality. Countless apps are launched and it's not uncommon to read a story about someone who has become a millionaire after developing a mobile app, usually with simple functionalities. As with any new technology, these cases have advantages as well as problems. One of the biggest problems involves supporting a quality characteristic, called portability by ISO/IEC 9126 (2000), to meet the diversity of mobile platforms available on the market.

Developing an application that serves the main platforms (Android, Windows Phone and iOS) can become a headache due to the high cost of programming hours for all of them, considering the time needed to get the application up and running without any problems. According to Nunes (2013), there is a common need for many companies to create applications or even mobile pages that serve a large part of the market and that work correctly on the most diverse existing platforms such as Android, IOS, Windows Phone, BlackBerry, among others.

The general objective of this work deals with a central issue regarding the diversity of platforms available for developing mobile applications on the market. To this end, we sought to elucidate the question: how can we develop a mobile application that works on several platforms using the same source code?

The methodological approach adopted in this work is bibliographical research, since a survey was carried out in books and scientific academic articles on strategies for developing applications that work on the main platforms from a single computer implementation.

As an answer to the central problem-question, there are tools available on the market, such as the open-source framework1 called Cordova, provided by Apache Community, which makes it possible to package an application that uses standard web technologies (HTML, CSS and Java Script) for Mobile applications, i.e. with a single source code it is possible to generate an application that works on the main platforms, being able to access native resources of each of them without the need to develop any line of native code. Example: Android platform, native Java code.

Thus, the relevance of this work lies in the importance of developing mobile applications that serve the main platforms on the market, Android, iOS and Windows Phone, in a more viable way. Developing the same application for each desired platform in its respective programming language, as native apps are built, generates excessive work and high development costs.

Theoretical Background
Before developing a mobile application, we must first analyze the target audience and the platforms on which it will operate. When thinking about developing mobile applications, it's important

to think about which platforms they should be made available on. Android is one of the most widely used platforms in the smartphone world, especially in Brazil. iOS is widely used and its main audience is the upper social classes, which includes users with greater purchasing power. Windows Phone is a good third option, which is growing rapidly. And there are still other platforms on the market, such as BlackBerry, Tizen and others.

Most mobile applications aim to serve a large mass of users, so it is recommended that the application works on at least the three main platforms on the market: Android, iOS and Windows Phone.

According to Lopes (2015 p. 01), "developing for different platforms has been a major problem because each platform supports a different programming language. For example: the Android platform has many Java features, the iOS platform uses native Objective C code, and the Windows Phone platform supports the .Net framework, generally the C# language. Each platform has its own combination of language and, above all, specific APIs".

Developing a mobile application that caters for the main platforms on the market can become a headache due to the high cost of specialized labor (programmers) with technical competence in each of the programming languages required. Another relevant aspect is the time needed to get the mobile application up and running without any problems (bugs).

This brings us straight to the question: how do you develop a mobile application that works on several platforms using the same source code?

This is where cross-platform applications come in. The company Avanti! Tecnologia & Marketing (2015), published and defended the idea that building a hybrid app is faster and cheaper than developing native apps. The reduction in time is due to the possibility of running the hybrid app on different platforms. Because of this feature, there is no need to develop the application several times to adapt it to different platforms, thus allowing for less impact on the budget. In situations that don't require high application performance, many companies also opt for hybrid application development or when the target audience is heterogeneous. In these cases, more generic solutions that can be used on multiple platforms, despite the high cost of development, have significant advantages.

The development of a multiplatform application can be done using certain market frameworks, such as Cordova (open-source), which makes it possible to package an application developed using HTML, CSS and Java Script web technologies for mobile applications.

Cordova is currently one of the most common solutions for developing multi-platform applications. To build and run applications, it uses one of the greatest advantages of the web, having standardized languages and the browser as the execution environment. According to Lopes (2016 p. 05), *"[...] They are installable Apps that you can publish in stores, and can use native features of the platform, but are written in HTML, CSS and JavaScript".*

This way, from the same source code, it's possible to generate a mobile application that works on the main platforms on the market and can access native features of each one. Without the need to develop any specific lines of code for the desired platform. Developing a mobile application using standard web technologies (HTML, CSS and JavaScript) is made easy by using a framework such as Cordova, which is responsible for encapsulating and transforming the source code for mobile application platforms. In this context, Lopes (2016, p. 5) states that: *Just writing HTML, CSS and JS is not enough to have an app in the end. So, what Cordova does is provide a native shell for our application that is responsible for bringing up a browser that will execute our code. Cordova's role is simply to create this browser window for us, and to communicate our code calls to native calls when necessary*.

But how does this framework work? Is it simple to develop a mobile application using just one programming language that serves the main platforms on the market? It's clear that these questions are on the minds of anyone looking to develop a multiplatform application. First of all, we need to understand a little more about hybrid applications. Hybrid apps are apps that use standard web technologies (HTML, CSS and JavaScript) and can access the native features of each mobile platform, such as: camera, GPS, accelerometer, etc. They are considered to be hybrid applications because, at the same time as they are developed for the web, they access the native resources of the devices on which they are running. As described in the IBM 2013 article, hybrid applications contain two elements: a web component, based on a web programming language, and a native container or bridge, which allows access to the native resources of the platform and device.

The Apache Cordova Open Source project is the most widely used container and consists of a set of support tools that allow the web application to access the native resources of the device. The hybrid application has its main code developed in HTML5 and is wrapped in a container, packaged as a native app and therefore residing in an app store (IBM, 2013).

The Cordova framework must be used by the developer to build a web application (WebApp). The WebApp can be accessed from a browser provided by the framework, called WebView, which allows access to the native resources of a Mobile device from a set of Plugins provided by Cordova. Both the WebView application and the set of plug-ins have cross-platform characteristics.

According to Silva (2016), hybrid mobile apps are built with a combination of web technologies such as HTML, CSS and JavaScript. The main difference, using Cordova, is that hybrid apps are accessed by the WebView app, which in turn has access to the native resources of a mobile platform. This approach allows access to device resources such as the accelerometer, camera, contacts and much more. These resources are generally restricted from access by mobile browsers. In addition, the Cordova framework allows other native interface elements (plugins) to be included when necessary (ANDRADE, 2016).

The WebView is the way in which a web application is viewed as an app on mobile devices, representing a relevant layer of the hybrid application. As we can see in figure 1, Cordova uses WebView to access both the source code (Web App) and the device's native resources (Cordova plugins) via specific APIs, thus achieving communication with the desired platform.

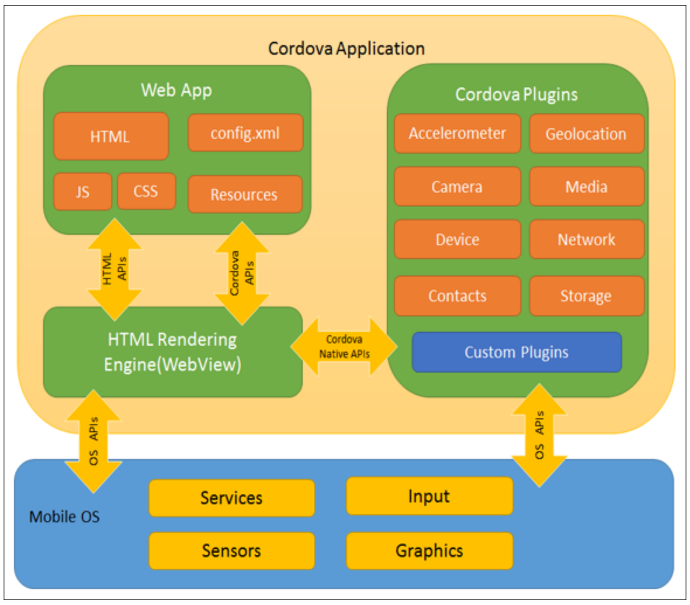

Figure 1: Architecture of the Hybrid Application Using Cordova
Source: Cordova [1]

You can also see that all access to the resources of the different platforms on the market is supported by the set of APIs that the Cordova Framework makes available to developers. There are several native code plug-ins, each of which has a function for accessing resources from different platforms. It should be remembered that the use of these APIs always follows the same implementation pattern, i.e. there is no need to change the implementation of the APIs to access the device's native resources, regardless of the platform used to run the developed application.

Lopes (2016) reports that, when using Cordova, the web source code is packaged to become a normal application. These apps are closer to native apps than web apps. In this sense, Lopes (2016, p.7) indicates that:

[...] *They have the same advantages and shortcomings of normal Apps: they need to be generated for each platform, they need to be made available in each manufacturer's store, and they are subject to the rules of each platform. They're not browsable, they're not on the internet, and they don't have URLs. However, they are fully integrated into the device. They can be installed and used offline. They can use platform APIs and use advanced hardware resources. They can be advertised in stores and sold easily to users*.

Lopes makes it clear that although the applications are developed in a web programming language, they can be run normally without the need for an internet connection. This means that a hybrid application developed using the Cordova framework is much closer to a native application than to a web application. The above quote also describes that in order for the hybrid application to be compatible with the desired platforms, even though they are the same source code, it is necessary to package it, with the help of Cordova, for each of them.

Building a hybrid application is easier than it looks. This is due to the use of a well-known application development format, web development, which uses standard web technologies (HTML, CSS and JavaScript). Even to access the device's native resources. Apache Cordova makes its framework available to various development tools. This makes it even easier to use and increases the production speed of the application. One of the most widely used tools is Visual Studio from version 2013 onwards. An example of a development tool using Cordova is Visual Studio 2013. The Cordova tools are released as a preview version. They will be embedded as part of Visual Studio 2015. Microsoft (2016) recommends and makes available for download Visual Studio 2015 RTM for developing applications using the Visual Studio Tools for Apache Cordova plug-in. Now that we know some of the characteristics associated with the functioning and difficulties of developing a multiplatform application, we can analyze some of the main advantages and disadvantages of building hybrid and native applications.

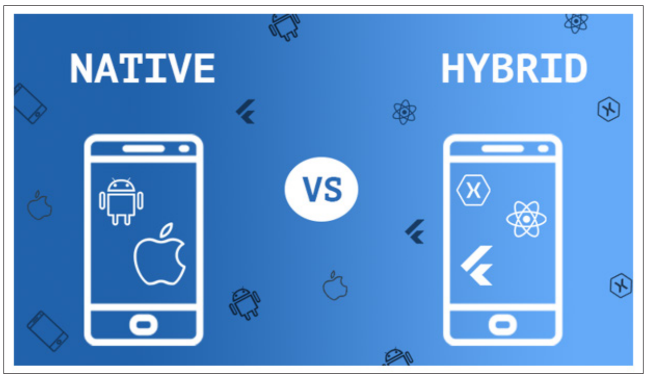

Figure 2: Native vs. Hybrid Applications
Source: Ádames [1]

As highlighted in figure 2, the hybrid application requires less technical knowledge from the programmer for its development, since it always uses web programming regardless of the platform in question.

According to Total Cross (2016), the great advantage is that it only requires knowledge of web development and therefore costs less to develop. It also stands out in terms of production time, as it is built much faster and only once, unlike native apps. It offers greater flexibility, in the sense that it can serve the main market platforms, and this is certainly its greatest attraction. Another great advantage of the hybrid application is the ease with which future updates can be made available. The hybrid app is more suitable because part of its code can be online and updated by the web system within the app, without having to update the entire app or send new versions to the stores (GOUVÊIA, 2015).

When it comes to a test app, the use of a hybrid mobile app also comes first, because, as it is a test, it becomes more feasible to invest less, both in time and costs, to analyze the public's reaction and, depending on it, invest more in the app. In this context, Gouveia (2015) states that "don't spend all your coins on an app to see what happens". This statement defends the idea that it's not worth developing a test app in native language when you don't know the level of user acceptance for a new app. The best option in this case is to develop it as a hybrid application, and if it receives market acceptance, it can even be built as a native application (when the focus is also on performance).

This set of advantages presents an interesting attraction associated with developing a hybrid application in today's competitive market: low development costs. This is undoubtedly the greatest advantage of this type of application, since what companies are looking for most today is to reduce costs.

Compared to other types of application, native applications have a much higher development cost, since they require developers with specific knowledge for each platform.

Even with all these advantages, the world of hybrid applications is perfect. As shown in figure 2, one of the biggest disadvantages of hybrid apps is their performance. When an app requires a lot of a mobile device's processing power, native apps come out ahead.

Due to the use of APIs to access the device's native resources, it becomes slower. But this difference is only noticeable to the user when a high processing rate is used.

Another disadvantage is that it doesn't have access to all the device's native resources. These include background execution, operating system notifications, additional information from the accelerometer (in addition to detecting the coordinate axes in the vertical and horizontal directions) and complex gestures. This also applies to visual components, i.e. the screen components responsible for the layout (graphical part) of the app. Because of this, hybrid apps do not follow the standard screens known to users of native apps, i.e. there is a variation in terms of user experience and usability.

Figure 3: Hybrid vs Native Apps Comparison Table

	Hybrid	Native(android/ iOS)	Best
Graphics	HTML, Canvas, SVG	Native APIs	Native
Performance	Slow	Fast	Native
Natural appearance	Emulator	Real Appearance	Native
Equipment features	Others	Total	Native
Publication in stores	Almost Normal	Normal	Native
Code reuse	Total	None	Hybrid
Development cost	Medium	High	Hybrid
Development time	Low	High	Hybrid
Ease of updating	Easy	Medium	Hybrid
Required knowledge	HTML5, CSS and Javascript	Java, ObjectiveC and Swift	
Learning curve	Half	Slow	Hybrid

After presenting the concepts of multiplatform application development and their main advantages and disadvantages, it is interesting to analyze and compare a hybrid application with a native application, so that you can choose the best option depending on your needs.

Figure 3 shows a comparison of the main items taken into consideration when developing a mobile application. Looking at the items above, if a hybrid application meets your needs, then it's a great option for your development, but if it's something very specific and it doesn't meet your needs, then the option would be to develop a native application.

Web Applications

Different rendering strategies play a critical role in shaping user experiences. One such strategy is Client-Side Rendering (CSR), where the server sends a minimal HTML file to the client. The client then loads JavaScript to dynamically render the rest of the page. CSR relies heavily on JavaScript for fetching data and building the DOM dynamically. This approach is particularly effective for Single Page Applications (SPAs), which allow users to interact with the app without experiencing full-page reloads. CSR is ideal when the initial load time is less critical than overall interactivity, especially when client devices are sufficiently powerful and have reliable internet connections. It efficiently manages client-state and enhances interactivity. Popular frameworks for CSR include React, Vue.js, and Angular. At Meta, for instance, the desktop version of Facebook (facebook.com) is built using React. I was involved in migrating the settings pages from basic HTML, CSS, and JavaScript to React, enhancing the styling and utilizing Hack (a typesafe version of PHP) for the backend and GraphQL for API calls. To manage the immense traffic, which can amount to millions of requests per second, we employed Relay, a GraphQL client that scales effectively with the number of requests. This implementation employs CSR for rendering.

Another important strategy is Server-Side Rendering (SSR), which involves rendering web pages on the server before sending them to the client. This approach delivers fully generated HTML, improving initial load time and enhancing SEO. SSR is particularly suitable for content-heavy websites and pages with infrequent content changes. Popular frameworks for SSR include Next.js for React, Nuxt.js for Vue.js, and Angular Universal. At Meta, while most of Facebook's desktop app uses CSR, some pages, such as certain Help Center articles, use SSR for the content portions that do not change frequently. This allows these pages to benefit from better SEO and faster initial loads.

Static Site Generation (SSG) is another strategy that generates HTML at build time rather than on each request. The generated static files can be cached and served by a CDN, making this method highly scalable and ideal for blogs, documentation sites, and similar content. Popular frameworks for SSG include Next.js and Gatsby. At Meta, many internal documentation sites use SSG for improved performance, with React as the base framework.

Incremental Static Regeneration (ISR) is a hybrid approach that allows static pages to be regenerated incrementally after the site is built. This combines the benefits of SSG with the ability to update static content without a full site rebuild. ISR is particularly useful for large sites where frequent content updates are needed. Next.js is a popular framework that supports ISR.

Progressive Hydration is another strategy, where server-rendered HTML is sent to the client, and JavaScript progressively enhances parts of the page with interactivity. This method is useful when quick initial load times are needed, but interactivity is added gradually. Frameworks like React (with React.lazy and Suspense) and Vue.js (with async components) support Progressive Hydration.

Streaming SSR is an approach where parts of the HTML are sent to the client as soon as they are ready, allowing the browser to start rendering parts of the page while other parts are still being generated. This is beneficial for complex applications where minimizing initial load time is critical. React Suspense for Data Fetching and Node.js streams are popular technologies for implementing Streaming SSR. For example, I worked on the Facebook like button, a widely used component on the Facebook website. The page is rendered as an SPA using CSR, and components like the like button and comments use React Suspense to display a loading state until the server responds with user-specific data.

Edge-Side Rendering (ESR) involves rendering parts of the web page at the edge, closer to the user, using edge computing resources. This approach reduces latency and improves performance for dynamic and personalized content. Technologies such as AWS Lambda and Vercel Edge Functions are commonly used for ESR. In modern web applications, many components can now be implemented serverless, including databases, backend servers, frontends, and caching, enhancing scalability and performance.

Mobile Applications

In the context of mobile applications, different rendering strategies cater to various development needs and user experiences. Native Rendering utilizes platform-specific libraries and frameworks, offering the full benefits and earliest access to new features. This approach is exemplified by technologies like Swift for iOS and Kotlin/Java for Android. Hybrid Rendering takes a different approach by allowing developers to write code once and run it on multiple platforms. This method employs web technologies such as HTML, CSS, and JavaScript, which are then wrapped in a native container. Hybrid Rendering is particularly useful for simple UIs and for situations where there is a time crunch or a large codebase that can be shared across platforms. Popular frameworks for this strategy include the Ionic Framework and Apache Cordova. Cross-Platform Rendering offers another versatile solution, compiling a single codebase into native code for multiple platforms. This approach achieves near-native performance while enabling code sharing across different platforms. Prominent frameworks in this category are React Native, Flutter, and Xamarin. During my tenure at Facebook, I contributed to redesigning parts of the mobile app, particularly modifying legacy React Native code to make privacy settings more accessible to users. Progressive Web Apps (PWAs) represent a fusion of web and mobile technologies, delivering native-like experiences on mobile devices. Built using standard web technologies, PWAs offer features like offline support, push notifications, and access to device hardware. This strategy is beneficial for applications that need to work across various devices with minimal development overhead and where reducing installation friction is important. Technologies supporting PWAs include the Web App Manifest, Angular, React, and Vue.js. Server-Driven UI (SDUI) provides a dynamic and flexible approach by allowing the server to control the structure and content of the UI. In this model, the client renders the UI based on data and instructions received from the server. SDUI is particularly advantageous for applications that require frequent UI updates without redeploying the client, and for scenarios where the server needs tight control over presentation logic.

Comparison with Facebook

Although Facebook created and open-sourced React Native, much of the code and features I implemented on the mobile side are part of a new, confidential framework that leverages Server-Driven UI (SDUI). The Facebook application now incorporates a core rendering engine, capable of utilizing other engines such as Safari from Apple and Chromium from Google. This core engine's primary function is to interpret instructions received from the server. When the app loads, it initializes this basic rendering engine, which then fetches data and renders information on the device according to the server's instructions. This approach facilitates cross-platform development and enables rapid release of new features, as the instructions for rendering are sent to the pre-existing engine.

Instant Apps represent another innovation, allowing users to access specific functionalities of an app without installing it entirely. This is particularly useful for users who want to try out an app before committing to installation or need quick access to a particular feature. App Bundles are commonly used for this purpose.

Significant improvements can be made through AI and ML optimizations, tailoring different views for different users on the same page. By running various AI/ML models based on user data, we can offer personalized versions of a website or application, similar to how Netflix recommendations vary for each user. At Facebook, we utilize data from multiple Facebook apps, such as Facebook and Facebook Lite, to tailor the settings page to users' preferences, highlighting the options they are most likely to adjust.

In the context of SDUI, the rendering engine is shipped as part of the app, with rendering instructions provided by the server. As mobile and desktop devices become more powerful, custom models can be shipped alongside these rendering engines within application bundles. These models can run locally on the device for tasks that require privacy or less computational power, while more demanding problems can be handled in the cloud. Model weights can be updated by releasing newer versions of the application.

Furthermore, AI and app rendering models have vast potential applications beyond mobile and desktop devices. In the automotive industry, where technological growth has lagged despite its long-established presence, OEMs can ship their own AI models, leveraging the power of modern cars to run these models effectively. SDUI can be used to deliver new features rapidly, providing users with a consistent experience of regular updates. This approach can also be extended to AR/VR applications, IoT devices, wearable technology, and more, demonstrating the versatility and potential of advanced rendering strategies in various domains.

Final Considerations

This work aimed to conceptualize the strategies for developing multi platform applications and identify the main advantages and disadvantages of this type of development. A literature search and analysis of scientific articles on the subject was carried out in order to gain a better understanding of the aspects involved in developing mobile applications that work on different platforms without the need for additional coding.

The results of the research in this article show that Cordova is one of the main frameworks used as a basic tool for developing multi platform mobile applications. Using this framework allows mobile applications to work on the main platforms, such as Android, iOS and Windows Phone.

The main advantages of using hybrid applications can be summarized as follows: it requires less technical knowledge, so less time is needed to learn the programming language used in development (web development); it is flexible; it serves the main platforms on the market using the same source code; it is easy to provide future updates and it is ideal for building test applications (prototypes). These advantages result in low development costs.

The main disadvantages are: low performance, requiring greater computational effort; not having access to all the device's native resources, such as running in the background and operating system notifications. One aspect that is also desirable is access to a greater number of native components for building screens, since the resources provided by the Cordova framework do not follow a pattern of screens familiar to users of native applications, which ends up compromising usability to some extent.

Despite this, it is possible to say that the development of multiplatform applications has a great ally, Cordova, which has facilitated the development of Mobile applications for various platforms. In this way, it is possible to provide mobile applications for a much larger number of users, since this approach allows the use of multiple platforms.

Finally, it can be concluded that an analysis of all the needs associated with the application must first be carried out before deciding on the best option for development: a hybrid application or a native application [2-6].

References

1. Covington P, Adams JK, Sargin E (2016) Deep Neural Networks for YouTube Recommendations. Proc. 10th ACM Conf. Recommender Systems 191-198.
2. Portugal I, Alencar P, Cowan D (2018) The use of machine learning algorithms in recommender systems: A systematic review. Expert Syst. Appl 97: 205-227.
3. De Paula DFO, Menezes BHXM, Araujo CC (2014) Building a Quality Mobile Application: A User-Centered Study Focusing on Design Thinking, User Experience and Usability. Design, User Experience, and Usability. User Experience Design for Diverse Interaction Platforms and Environments 8518.
4. Carvalhido A, Novo R, Faria PM, Curralo A (2018) A User Experience Design Process in Mobile Applications Prototypes: A Case Study. SpringerLink https://www.springerprofessional.de/En/a-user-experience-design-process-in-mobile-applications-prototyp/19782304.
5. Li X, Heng Q (2021) Design of mobile learning resources based on new blended learning: a case study of superstar learning app. Proc. 2021 IEEE 3rd International Conference on Computer Science and Educational Informatization (CSEI) 333-338.
6. Gong J, Tarasewich P (2004) Guidelines for handheld mobile device interface design. Proc. Annual Meeting. Northeastern University, Boston https://personal.cis.strath.ac.uk/sotirios.terzis/classes/52.504/2010/GuidelinesGongTarase.pdf.