IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

HYBRID APPS (React Native with OpenCV)

Prashant Srivastava
Student, Department of Information Technology
Dronacharya College of Engineering, Gurugram, India
prashantsrivastava5116@gmail.com **,7827677523**

Abstract : **Developing** mobile applications presents challenges due to device diversity and platform fragmentation. With various operating systems in the market, developers face hurdles in creating applications that seamlessly run across different platforms. This diversity complicates the development process, increases costs, and requires extensive effort in adapting applications to meet each platform's unique requirements.

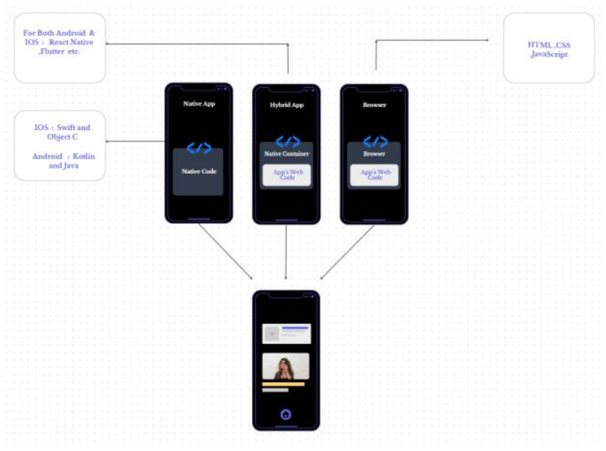
Despite these challenges, there are promising solutions to streamline the development process and reduce costs. Cross-platform development tools, although not fully matured, show great potential in enabling developers to create applications that can be deployed across multiple platforms with minimal modifications. These tools leverage frameworks and technologies that abstract away platform-specific complexities, allowing developers to write code once and deploy it across various platforms.

Hybrid mobile development is also gaining popularity as it combines elements of both native and web development. This approach allows developers to leverage web technologies like **HTML**, **CSS**, and **JavaScript** to create mobile applications that can be deployed across different platforms. By using frameworks like **Apache Cordova** or React Native, developers can access native device features while still benefiting from the flexibility and rapid development cycles of web technologies.

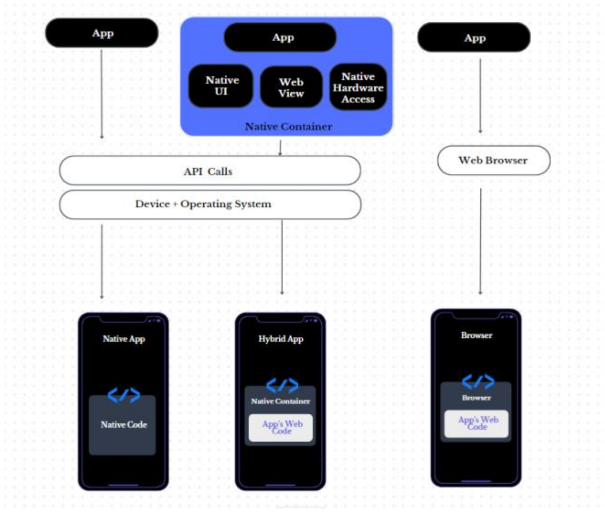
In conclusion, while native mobile development remains a viable option, cross-platform development is becoming increasingly attractive due to its potential to streamline the development process and reduce costs. With the ongoing evolution of technologies like **HTML5** and the continued improvement of cross-platform development tools, the future looks promising for developers seeking to reach a wider audience across different devices and platforms.

Index Terms : Native Mobile Application Development (**Kotlin , Java ,objective C , C++ ,Swift**) , Hybrid Mobile Application Development (**Flutter , React Native , Iconic**) && distinguisition Native and Hybrid Mobile Development (React Native)

Introduction



When people refer to native mobile applications, they typically mean apps developed using platform-specific languages and tools, such as **Swift** or Objective-C for **iOS** and **Kotlin** or **Java** for Android. These native apps offer optimal performance and full access to device capabilities, including sensors, address book, and the latest technologies. They also provide native user interface controls, enhancing user experience.



However, native development requires separate development efforts for each platform, leading to increased costs and time. Additionally, apps must go through platform-specific app store approval processes for every release.

On the other hand, hybrid apps bridge the gap between native and web apps. Built using web technologies like JavaScript, HTML5, and CSS, hybrid apps leverage a lightweight native app container to access certain device hardware and native features like the camera, GPS, and push notifications. This allows them to be distributed through app stores and installed on devices like native apps.

While hybrid apps may not match the performance of native apps, advancements in hybrid frameworks have significantly improved performance. They offer the advantage of cross-platform development, reducing development time and costs. Maintenance is also easier, as changes can be deployed across platforms simultaneously.

Overall, hybrid apps provide a balance between performance, access to native features, and cross-platform functionality, making them a viable option for many applications.

What Should I Choose Hybrid Developer?

Cost-efficiency: By creating a single hybrid app for multiple platforms, businesses can save significantly on development costs compared to building separate apps for each platform.

Easy Integration: Hybrid apps streamline integration processes by utilizing consistent SDKs and APIs across platforms, reducing the need for additional specialized tools.

Wider Audience: Hybrid apps enable businesses to target a broader audience across various operating systems, maximizing potential user reach.

Better UI/UX: Hybrid apps offer ample resources for enhancing user interface and experience, delivering faster performance and lightweight design.

Easy Maintenance: Managing one unified codebase for hybrid apps simplifies the maintenance process, facilitating quicker updates and version control.

Faster Development: Hybrid app frameworks expedite development timelines through reusable code components and efficient development practices.

Enhanced Speed: Hybrid apps boast faster performance dueto their streamlined codebase and reduced reliance on network communications.

Time-saving: Leveraging a single codebase for development saves time on hiring, research, and ongoing maintenance, optimizing resource allocation.

Improved Offline Support: Hybrid apps provide robust offline functionality, ensuring seamless user experiences even in low-connectivity environments.

Faster Time to Market: Hybrid app development accelerates the go-to-market process by enabling simultaneous deployment on multiple platforms, reducing launch timelines significantly.

Which are the Most Popular Hybrid Mobile App Development Frameworks?

Two Cross-Platform Mobile App Development Frameworks: A Balanced View

Choice A: A Feature-Rich Option with Established Backing(**React Native**)

- This framework, supported by a large community and extensive resources, leverages a familiar language (JavaScript) for development.
- Key strengths include cost-effectiveness due to code reusability and a potentially faster development process.
- However, performance might be slightly slower compared to native apps, and UI customization options may be limited due to reliance on native components.

Choice B: A High-Performance Up-and-Corner with Customization Power(**Flutter**)

- This framework offers excellent performance due to native code compilation and boasts rich tools for crafting unique user interfaces.
- Hot reload functionality rapid iteration and potentially .
- However, the framework itself is relatively new, leading to a smaller community and fewer resources compared to its competitor. Additionally, developers unfamiliar with the primary language (Dart) might need to invest in learning it.

Choosing the Right Fit: Consider These Key Factors

Your development team's expertise: Are they familiar with JavaScript or Dart?

Performance requirements: Is top-notch performance essential for your app?

UI customization needs: Does your vision demand a highly customized and unique interface?

Project timeline and budget: What are your constraints in terms of time and resources?

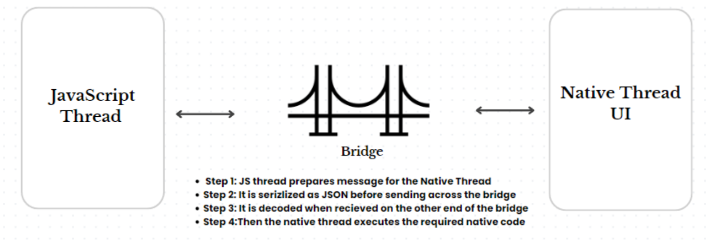
React Native

React Native stands out as a robust framework for crafting navigation apps, boasting numerous advantages over conventional native app development methods. It empowers developers to craft cross-platform applications with a solitary codebase, resulting in significant time and resource savings. Leveraging native APIs for rendering components ensures swift and seamless performance. Additionally, its vibrant developer community fosters continuous growth and knowledge sharing. With a plethora of pre-built navigation libraries at hand, React Native expedites and streamlines the development of navigation-centric applications. In summary, React Native emerges as a top-tier choice for navigation app development, equipped with a rich array of tools and frameworks to support developers' endeavors.

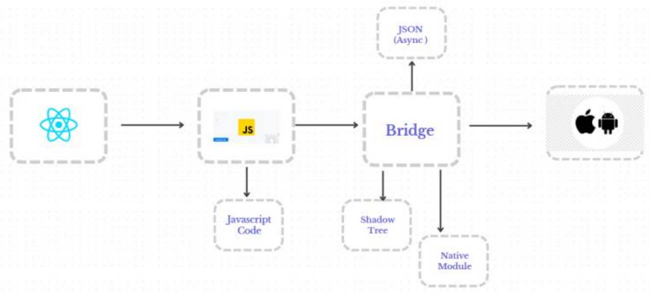
No we are going to integrate native module in react native with this we can create native module in native language like kotlin and java , c++ , swift etc,then we can use this with the help of bridge of react native,

React Native Architecture

Old React Native Architecture: The communication bridge facilitates interaction between the JavaScript thread and the native UI thread. Initially, it compiles the JavaScript code into a bundle for execution, utilizing the JavaScript thread managed by the JS engine. Within the bridge, both the JavaScript thread and the native thread are unaware of each other. The native thread is responsible for executing native modules and handling UI operations such as UI rendering and gesture events. Additionally, a shadow thread is employed to compute the layout of elements prior to their rendering on the host screen, leveraging tools like Yoga for layout calculations.



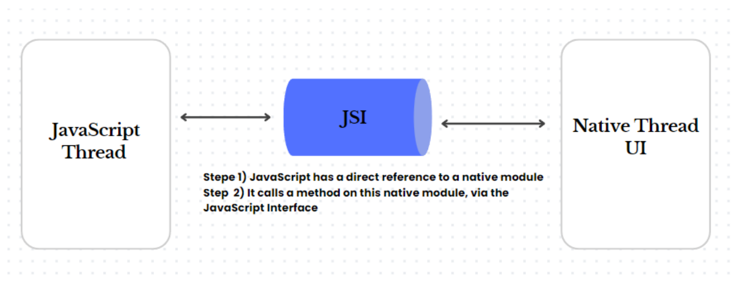
- Step 1: JS thread prepares message for the Native Thread
- Step 2: It is serizlized as JSON before sending across the bridge
- Step 3: It is decoded when recieved on the other end of the bridge
- Step 4:Then the native thread executes the required native code

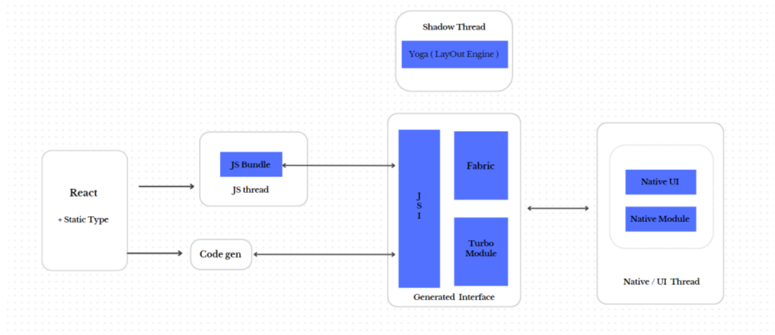


New Architecture of React Native (JSI)

In the above scenario, the JavaScript and native worlds operate independently of each other. Before data is sent to the bridge, it must be batched (optimized) and serialized as JSON, which is then decoded on the native end. While sending messages over the bridge asynchronously is beneficial in many scenarios, it can occasionally overload the bridge, leading to congestion.

This problem is removed by JSI (JavaScript Interface) in the new architecture. Instead of relying on a bridge for communication between the JavaScript and native threads, JSI establishes a direct connection between the JavaScript and native worlds. Operating on reference-based principles, it eliminates the need for JSON serialization and resolves the congestion issues that occurred in bridge communication (*old architecture*) .


JavaScript Thread
JSI
Native Thread UI
Stepe 1) JavaScript has a direct reference to a native module
Step 2) It calls a method on this native module, via the JavaScript Interface



Fabrice

It's a critical component of the new JSI architecture. This rendering system replaces the current UI manager. In the previous bridge architecture, when the app runs, React executes code and generates a React Element Tree in JavaScript. Based on this tree, a shadow tree is created in C++, which the layout engine utilizes to calculate the position of UI elements for the screen. Once layout calculations are complete, the layout result is transformed into a Host View Tree, representing Native Elements.

ReactElementTree (JavaScript) ->ReactShadowTree(C++) ->HostViewTree(Native)

Indeed, relying on the bridge for rendering UI can introduce inefficiencies such as slow transfer rates and redundant data copying. Each node (whether on the JavaScript or native thread) store duplicate data separately, leading to increased memory usage and potential synchronization issues between the JS and UI threads. Consequently, this can result in perceived lagginess in the app, especially when performing tasks like scrolling through a large list of data in a FlatList

With the new rendering system, UI rendering on the host screen is optimized. Actions such as scrolling and gestures can be prioritized to execute synchronously in the main or native thread. API calls are executed asynchronously, and the resulting shadow tree is immutable and shared between the JS and UI threads, facilitating direct interaction from both ends. These improvements contribute to reduced memory consumption and enhance overall performance

Turbo Module

In the bridge architecture (turbo module), all native modules used by JavaScript, such as Bluetooth, Geolocation, and File Storage, are initialized before the app is opened. This can impact app performance because even if the user doesn't require a particular module, it still has to be initialized at startup.

However, in the JSI architecture (turbo module), enhancements are made to turbo modules. JavaScript can now hold references to these modules, allowing JavaScript code to load each module only when it is required. This significant improvement in startup time for React Native apps is achieved by deferring module initialization until it's actually needed, thus optimizing resource usage and enhancing overall performance.

Codegen

Codegen will utilize your typescript & flow types to generate more native code at build time , instead of run time. It's reduce the code size and lead faster execution or fewer errors. codegen ensure type-safety and compile time type safety ,

---

Creating a Native module which will used will detect image is blur or sharp and clear ,This native module will utilize in app

(with new architecture)

For Creating Native Module **Kotlin language** will be used with **new architecture**

What is openCV ?

OpenCV is one of the most popular libraries for computer vision tasks due to its comprehensive set of functionalities and ease of use .openCV stand for **open source computer vision library** ,it is used for real time computer vision processing. It utilize lot of algorithm for processing .it is used for facial recognition ,object detection, text extraction and many more use cases.

It is available in variety of programming languages including **C++,java** and **python**.

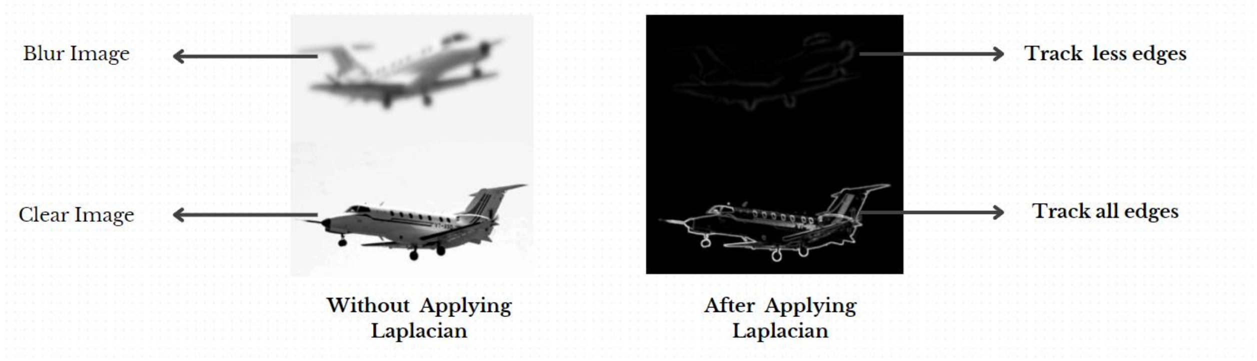
GrayScaling

By applying grayScaling .it reduce the dimensional which help in reducing the computation.

- Reducing the dimension of the images: RGB images have three color channels and constitute a three-dimensional matrix, while in grayscale images, there is no additional parameter for color channels and are only single-dimensional.
- Due to the dimension reduction, the information provided to each pixel is comparatively less.
- Reduces the complexity of the model: When there is less information provided to each pixel of the image, the input nodes for the neural network will also be considerably less. Hence, this reduces the complexity of a deep learning model.
- Difficulty in visualization in color images: Much more information is extracted for some images through gray scaling which might not be possible if the same algorithms or processes are applied to a color image. Features required for extraction become much more visible.
- For a better understanding of image processing

Laplacian

It is used to detect edges in an image. If the image is clear and sharp (**laplacian value will be maximum**), all edges are highlighted. However, if the image is blurry (**laplacian value will be minimum**), the Laplacian output image cannot accurately detect all edges. For instance, in the aeroplane images below, when the image is clear, edges are sharply detected. But when the image is blurry, the Laplacian method can only detect a few edges. If feature of **opencv** will be utilize in app for detecting image is blur or not.

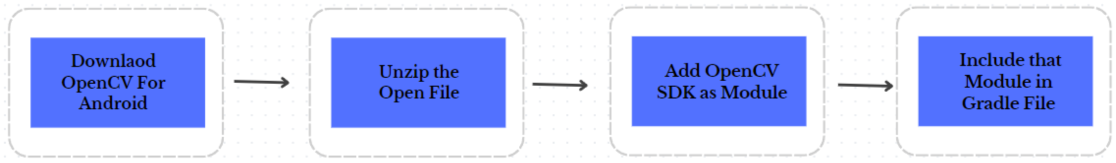


How to integrateOpenCV in app?

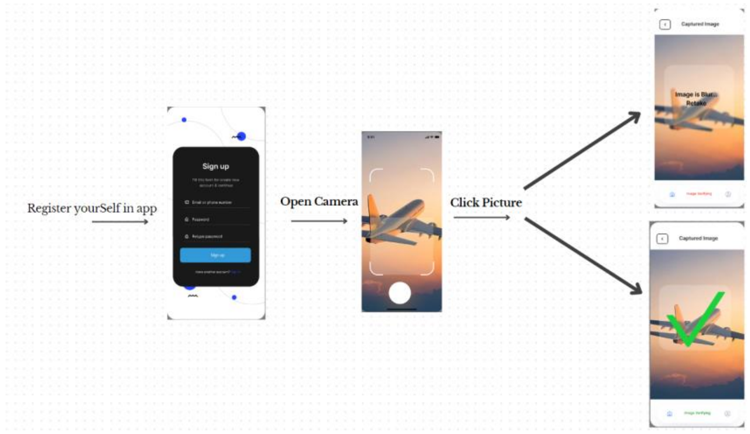
- First, download OpenCV for Android from the official website (https://opencv.org/releases/). After downloading, extract the SDK folder from the archive.
- Open Android Studio and create a new project or open an existing one.
- Add the OpenCV SDK to your Android Studio project:
- In Android Studio, navigate to File > New > New Module.
- Browse and select the OpenCV SDK folder that you extracted earlier.
- Click Finish to add the OpenCV module to your project.
- In your app's build.gradle file (usually located in the app directory), add the following line in the dependencies block to include the OpenCV module:

  Copy code **implementation project(':**opencv'**)**

- Sync your project with Gradle files by clicking on the "Sync Now" link that appears at the top right corner of the Android Studio window, or by navigating to File > Sync Project with Gradle Files.
- Now, you can use OpenCV in your Android project. Make sure to import the necessary OpenCV classes in your Java files.
- If you encounter any issues during the process, make sure to check the official OpenCV documentation or community forums for troubleshooting.


Downlaod OpenCV For Android
Unzip the Open File
Add OpenCV SDK as Module
Include that Module in Gradle File

"Now we will import the native module into the app and pass the clicked image. It will then analyze the image data and show information about whether it is blurry, sharp, or clear."



Conclusion

In Conclusion, our study shows that we can use clever methods in building mobile apps to detect when images are blurry. We did this by combining React Native, a popular framework for making mobile apps, with OpenCV, a powerful tool for image processing.

Our tests prove that this combination works well, giving accurate results quickly. It means we can now easily add advanced image analysis features to mobile apps, making them better for things like photography, medical use, and virtual reality.

Looking forward, we can make this system even better by refining it and adding more features. This could lead to even more exciting possibilities for using image analysis in mobile apps, making them more useful and enjoyable for users.

References

1. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3330011
2. https://www.irjmets.com/uploadedfiles/paper//issue_5_may_2023/38937/final/fin_irjmets1684308878.pdf
3. https://www.researchgate.net/publication/357435933_Comparison_of_Flutter_and_React_Native_Platforms