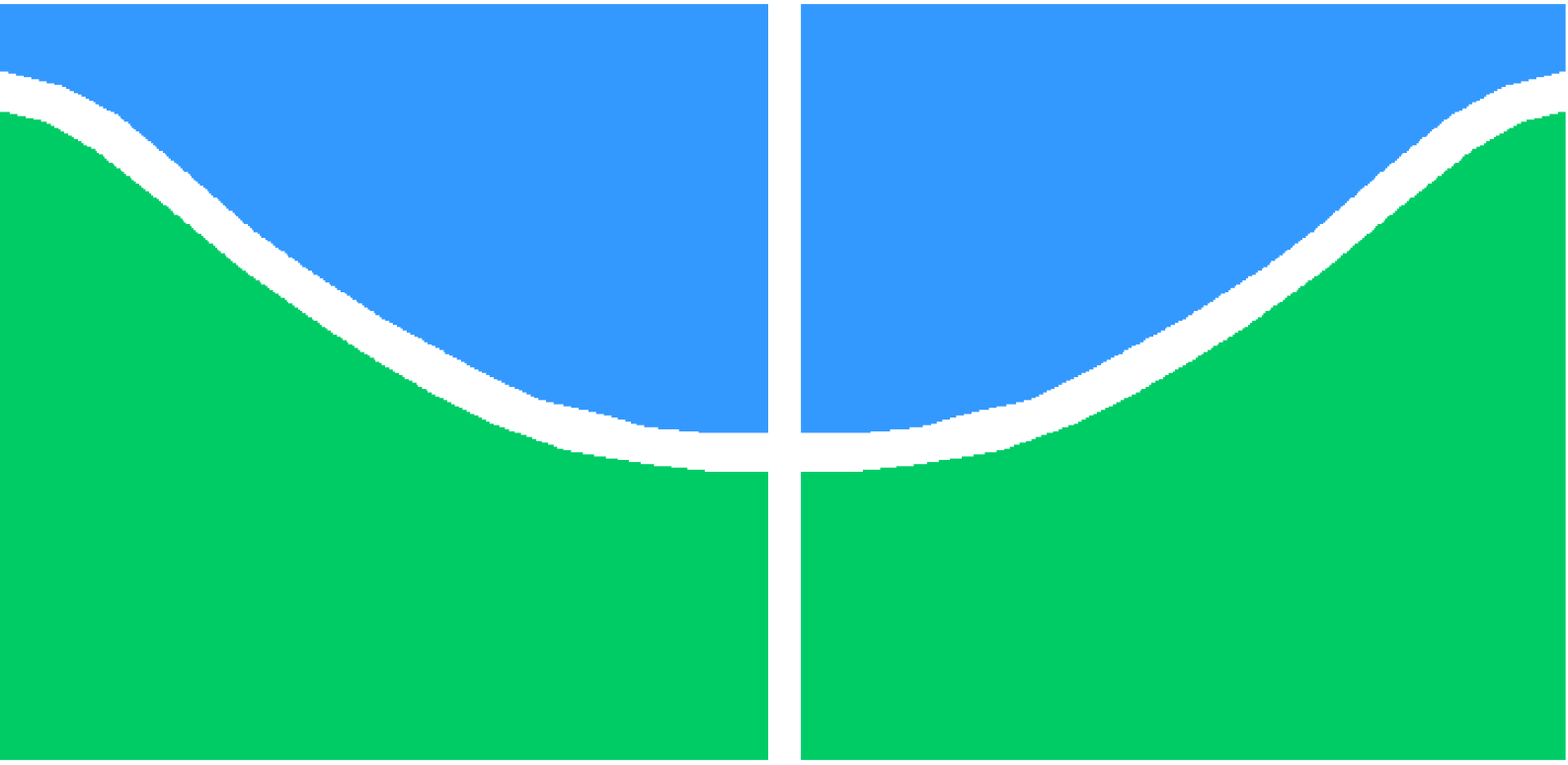


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Developing Native Modules in React Native Using Kotlin

Autor: Ezequiel De Oliveira Dos Reis

Orientador: Prof. Dr. Renato Coral Sampaio

Brasília, DF

2022

Ezequiel De Oliveira Dos Reis

Developing Native Modules in React Native Using Kotlin

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Supervisor: Prof. Dr. Renato Coral Sampaio

Brasília, DF

2022

Ezequiel De Oliveira Dos Reis
Developing Native Modules in React Native Using Kotlin/ Ezequiel De Oliveira Dos Reis. – Brasília, DF, 2022-
59 p. : il. (algumas color.) ; 30 cm.

Supervisor: Prof. Dr. Renato Coral Sampaio

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2022.

1. Engineering. 2. FGA. I. Prof. Dr. Renato Coral Sampaio. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Developing Native Modules in React Native Using Kotlin

CDU 02:141:005.6

Ezequiel De Oliveira Dos Reis

Developing Native Modules in React Native Using Kotlin

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 12 de maio de 2022:

Prof. Dr. Renato Coral Sampaio
Orientador

Profa. Dra. Carla Silva Rocha Aguiar
Convidado 1

Prof. Dr. Fernando William Cruz
Convidado 2

Brasília, DF
2022

Abstract

Mobile development can follow two ways, through Native or hybrid technologies. Native Technologies are focused on one platform with all the resources for a specific operating system. While hybrid technologies' purpose is to unify the code base, to create one APP for more than one platform.

A technology that has significant visibility in the market nowadays for the Hybrid development is React Native, a framework created by Facebook. The unified code is written in Javascript. The code is made above Java on Android and Objective-C on iOS. in the current scenario, two technologies that emerged recently to replace the previous native options are Kotlin for Android and Swift for iOS. These languages are currently on the rise.

Kotlin is a Java-based language that is more used than Java nowadays. The java code in React native is used in modules, which each module has its objective to handle. When we write modules using Kotlin, the gains are a simpler syntax that leads to a easier learning curve than Java. The problem to be analyzed is the integration of these two technologies.

This work proposes a study of Kotlin and React Native and the communication between them through native modules, starting with the thinking from native modules with Kotlin as a possibility, to create a complete app using different kinds of native modules using Koltin to interact with camera, gallery, calendar. At the beginning of this work, native modules in Kotlin were unofficial in the community. However, this communication became real during this work. The community started to adopt Kotlin, so this work is a guide to start with these kinds of native modules.

Currently, React Native uses the native modules to work correctly. Native modules for Android often use Java to create the native part of the app and communicate with the Javascript side. This work focuses on an approach to creating native modules as usual but using Kotlin on the native side of the app. The work shows the process of creating native modules using Kotlin, approaching common cases of necessities to native modules.

As the final result, we created an application using React Native with three native modules using Kotlin and Javascript resources to interact. The first native module to save events on the smartphone's calendar, the second module to use an Android native date picker, and the last one is the more complex native module with two functionalities related to images. The last module can get images from the gallery and take pictures using the camera.

Key-words: Kotlin. React Native. Javascript. Native Modules. Android.

Resumo

Atualmente o desenvolvimento de aplicativos pode ser feito através de tecnologias nativas ou híbridas. Tecnologias nativas são tecnologias focadas e com todos os recursos necessários para um sistema operacional específico. Já as tecnologias híbridas tem o propósito de unificar o código produzido, para criar aplicativos para mais de uma plataforma. Uma das tecnologias que está com grande visibilidade nos dias atuais, para o desenvolvimento híbrido, é o React Native, que é uma tecnologia criada pelo Facebook. O código unificado é escrito em Javascript, esse código único é convertido para código nativo de forma autônoma.No cenário atual, duas tecnologias que surgiram recentemente para substituir as opções nativas anteriores e estão em ascensão para Android e IOS, são elas Kotlin e Swift respectivamente.

Kotlin é uma tecnologia relativamente nova, baseada no seu antecessor o Java, que é usado atualmente, esse código Java no React Native é usado com base em módulos. Quando escrevemos módulos em Kotlin, temos um ganho tanto na facilidade com a sintaxe do Kotlin que leva a uma curva de aprendizado menor em relação ao Java quanto inserção dessa nova tecnologia junto a evolução do código híbrido.

Este trabalho propõe um estudo de Kotlin e React Native e a comunicação entre eles através de módulos nativos, para criar um aplicativo completo usando diferentes tipos de módulos nativos usando Koltin para interagir com câmera, galeria e calendário. No início deste trabalho, os módulos nativos em Kotlin não eram oficiais na comunidade. No entanto, essa comunicação se tornou real durante este trabalho.A comunidade começou a adotar o Kotlin, então este trabalho é um guia para começar com esses tipos de módulos nativos.

React Native necessita de módulos nativos para seu correto funcionamento. Módulos nativos para android geralmente usam Java para a parte nativa comunicação com o lado do Javascript. O foco deste trabalho é abordar a criação de módulos nativos tradicionais do React Native, porém usando Kotlin do lado nativo da aplicação. Este trabalho mostra o processo de criação de módulos nativos usando Kotlin, abordando casos comuns de necessidade de módulos nativos.

Como um resultado final, uma aplicação usando React Native e três módulos nativos usando Kotlin, foi criada. O primeiro módulo nativo é para salvar eventos no calendário do smartphone, já o segundo módulo tem como objetivo utilizar um seletor de datas nativo do android, o terceiro e mais complexo módulo criado para a aplicação tem duas funcionalidades principais, relacionadas a imagens. O último módulo pode pegar imagens da galeria do celular, e também pode capturar imagens através da câmera.

Palavras-chave: Kotlin. React Native. Javascript. Native Modules. Android

List of Figures

List of abbreviations and acronyms

API	Application Programming Interface
App	Application
CLI	Command Line Interface
IoT	Internet of things
iOS	iPhone operating system
JS	Javascript
JVM	Java virtual machine
KMM	Kotlin multiplatform mobile
OS	Operating system
SDK	Software development kit
UI	User interface

Contents

1 Introduction

In the past years, a vast part of the global population adopted smartphones for their lives, work, study, communicate, follow the news, and joy. This growth of smartphone use leveraged smartphone development to fill the user necessities. Big companies need to improve processes daily because this market is very competitive, needing to bring new functionalities and better tools for using their smartphones.

The growth of smartphone usage comes followed by gadgets and other kinds of devices, in general wearable and even to use the smartphone in IoT to serve as a control panel.

This development speed could cause many problems for the final user and the developers. The pressure on the developers for new features can bring bugs in the code, making Apps that run slowly consume more memory than what is waited or even crash during the usage.

Technologies like new languages or new frameworks of existing languages improve the development in many ways, avoiding errors, developing cross-platform, and offering better debug processes.

Some frameworks can use one or more programming languages. Some frameworks grew up to attend the mobile development scenario in the past years. For example, Cordova, Ionic, Xamarin, Flutter, and React Native are solid frameworks for mobile development.

Mainly in hybrid development, the framework needs to access the native layer of the devices using one or more languages. For example, React Native for access to the native layer through native modules uses Java, Objective-C, and others.

Technologies are evolving quickly, and companies are creating tools to improve mobile apps, developing new programming languages or new features for the existing languages. Talking about new programming languages, Kotlin and Swift, for example, are the new bet to the native development for Android and iOS.

This work focuses on React Native and Kotlin, proposing developing a native module using Kotlin to aggregate to React Native instead of Java for Android. During the dissertation, it is possible to see that React Native is not the best option for all cases. For instance, we cite some languages to compare and illustrate other approaches. The main scope of this work is to create native modules to React Native to be accessed from the Javascript side. The focus is to create native modules of the React Native but using Kotlin instead of Java.

1.1 Objectives

The main goal of this work is to develop native modules in React Native using Kotlin instead of java. This goal drives these specific objectives:

- Analyze the current use of native modules in React Native community;
- Implement a native module using the current architecture of React Native;
- Create an example of a native module using Kotlin to use in React Native.

1.2 Work Structure

This work will be defined using a background that introduces relevant and necessary topics to provide an understanding of the problem, a methodology, describing which and how experiments should be made to achieve and understand the results and planning for the subsequent work, to describe what the following steps to finish the objective of the work.

2 Background

2.1 Mobile Development

Nowadays, it is very usual to see some companies offer their service or product at a long distance from the customer. These companies wish for a scalable service to reach the maximum number of users. For these operations to happen, the companies have to use a technological solution to keep in touch with customers worldwide.

Often websites are used for this purpose, but companies know this strategy is not enough. The user needs to open a browser, type the address, and wait to reach the site. Some users have problems with this kind of flow. To solve this, we have mobile applications.

Mobile applications offer the facility to access services or products from a company with a tap on the smartphone's screen and immerse the user in the platform. Furthermore, when the user installs an app on the phone, the company can show some announcements, discounts, and promotions to get the user's attention.

Mobile development needs to approach different limitations like different screen sizes, battery, network connection, life cycle, and other behaviors different from web development. That is why developers have to worry about the smartphone's properties.

There are many Operating Systems for smartphones, but we have two on the top of the most used operating systems for smartphones worldwide, iOS and Android, created by Apple and Google. This fact can be evident in Figure 1.

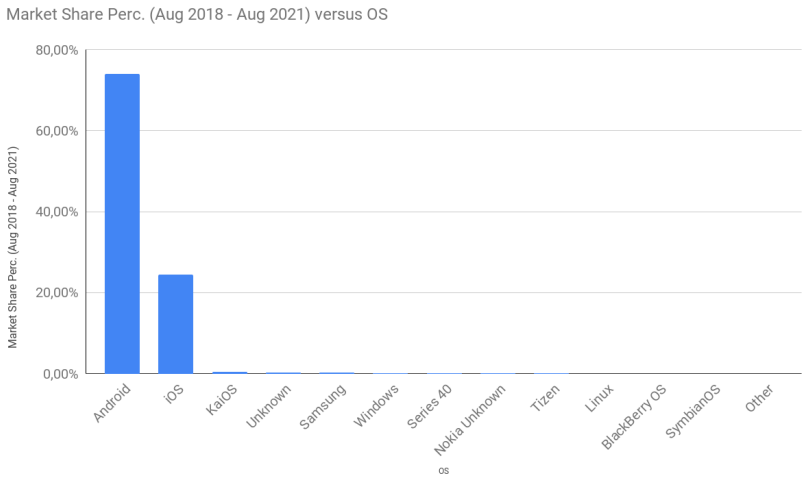


Figure 1 – Most used operating system (STATCOUNTER, 2021)

When we look at the current mobile development scenario, we soon remember iOS and Android. The new generation of smartphones uses one of these operating systems, and the developers that want to start with native development follow the path to learning these technologies.

2.2 iOS

iOS is a mobile OS created by Apple Inc. on June 29, 2007. Initially called iPhone OS from the first to the third version, the name changed officially to iOS at the release of the fourth version of the OS (COSTELLO, 2021).

Firstly referenced by OS X in 2007, iOS was developed for the company's hardware. The release happened alongside the first iPhone, and This OS has been developed and improved because nowadays, it is used to iPhone, iPad, and iPod.

2.3 Android

Android Inc. was founded in 2003, several years before Apple Inc. announced iOS. In 2005 the original company was acquired by Google, with some original developers continuing to work on the project under a new label. Android is an OS based on Linux, made to be offered to third-party manufacturers for free. Google's team could make money by offering services and including apps (CALLAHAM, 2021).

2.4 Native Mobile Development

Native mobile development use tools and programming languages provided for a specific mobile platform. These apps run only on mobiles with the target platform (EL-KASSAS et al., 2017).

When companies decide to create a mobile app, one point to consider is which public the company needs to reach. These applications run just on one platform and need to follow architectural patterns, life cycles, and SDKs provided for the platform.

Native apps have full APIs to access all the mobile devices features, high performance than other types of mobile development (EL-KASSAS et al., 2017). A problem with native apps is maintaining one app for each platform to reach a more significant number of users. Many apps mean more affords and costs.

Nowadays, to develop apps for Android, Google's recommendation is Kotlin, but still possible to use Java for Android development, while the development for iOS happens mainly using Objective-C and Swift.

2.5 Hybrid Mobile Development

The mobile hybrid development uses the web development technologies rendering inside a native app and using the device attributes through an abstraction layer (EL-KASSAS et al., 2017).

This development approach makes it possible to create apps across platforms using a single code base that works on more than one platform. When the companies develop a hybrid app, the developers have a smaller learning curve because they do not need to create many different apps, just one for many device platforms. The company can decrease costs to keep this hybrid solution.

Hybrid applications can access device resources using an abstraction layer. The interface is created once for many platforms and can reach a more extensive public than native applications. Unfortunately, the performance is a little worse than native applications.

Some big companies choose to continue the development using native technologies, giving up the unique app to offer a native experience and usability because native languages use patterns and ways to show an interface to the OS's user. Sometimes hybrid technologies do not follow the patterns of the OS and can disappoint the user.

2.6 Languages and Frameworks

This section talks about languages and frameworks that are important to develop this work or technologies related to the theme.

2.6.1 Kotlin

Kotlin is a statically typed programming language that targets the JVM, Android, JavaScript, and Native. Developed by JetBrains, the project started in 2010, and the first official release was in 2016. Kotlin is an open source[1] project from the start (JetBrains s.r.o., 2021d).

Kotlin is a programming language that supports object-orientation, functional programming, or even both. Kotlin has a clean syntax, becoming very quickly to learn and apply in many applications.

Kotlin is interoperable with Java, becoming an advantage to learn for Java developers. It is possible to call Kotlin in Java and the inverse because when targeting JVM, Kotlin produces Java-compatible Bytecode (JetBrains s.r.o., 2021d).

[1] <https://github.com/JetBrains/kotlin>

In Google I/O 2017, the support of Kotlin on Android was announced. In 2019 Google announced Android development as Kotlin-first (ANDROID, 2021). This change brought many looks to Kotlin and developing Android without harm to some before Kotlin Android developers.

Google and Jetbrains co-founded the Kotlin foundation to develop and promote Kotlin to be a more used programming language (JetBrains s.r.o., 2021e). This effort of these two giant companies enabled many improvements in the language environment, from debugging to more advanced topics like performance.

2.6.2 Flutter

Flutter is an open source[2] framework made by Google and contributors based in the programming language Dart (GOOGLE, 2021c). This framework focuses on developing cross-platform applications. In the beginning, Flutter was to develop just mobile applications, but now this framework has grown up to many platforms, enabling a single code base that works in many platforms and OS.

There are many options for mobile development. Fluter is one of the options that can be considered a rival of React native, so the citation is valid to talk about this competitor because this framework has gained much space in the past years and has a relation with Kotlin.

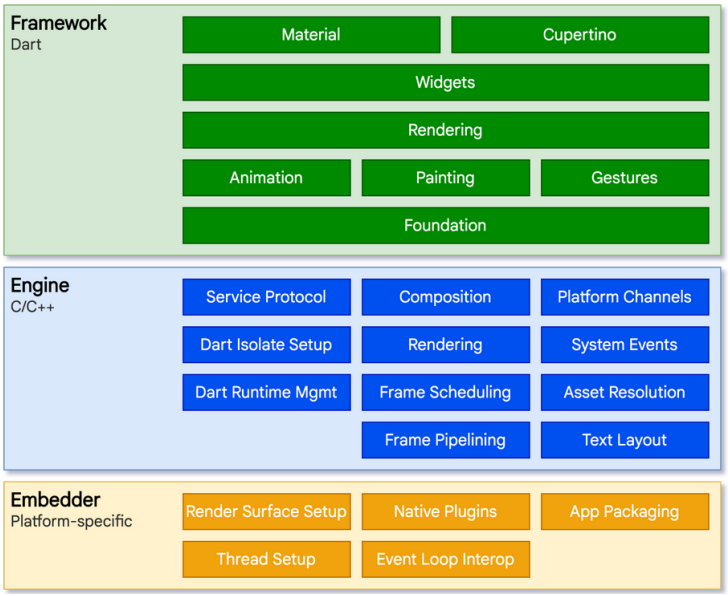


Figure 2 – Flutter architecture (GOOGLE, 2021d)

The first version of the framework was released in 2017 by Google, but the first stable version just was released in December 2018 (SACHINDANA, 2021). Since this

[2] <https://github.com/flutter/flutter>

date, Flutter has grown a lot, and the developers have begun to compare it with similar frameworks that have done a great job, like React Native. Companies like Nubank, BMW, eBay, and some Google services like Stadia, Google ads, and Google assistant (GOOGLE, 2021e).

Some companies decided to migrate to the framework for this extensive interoperability and focus primarily on the Native performance and animations. Thanks to the architectural solution implemented, this technology has an incredible performance compared to Native development.

Each layer of the architecture in Figure 2 has a responsibility, and it is its libraries, each piece of a layer is designed to be optional and replaceable. The first layer has to handle the user's gestures, render the app, and manage state animations. The second one is mainly written in C++, providing low-level implementations to Flutter's API. the last layer is for each specific platform code, enabling embed codes with existing code bases.

2.6.3 Kotlin and Flutter

Despite the differences, these two technologies have common points. Both of them have a Google's sponsorship and participation in the development.

Kotlin and Flutter have different approaches to development. Kotlin is a general-purpose programming language most widely used in Android development, and Flutter is Google's portable UI toolkit, according to the official documentation (GOOGLE, 2021f).

In terms of popularity, Flutter has the advantage. If we look at the Github stats, for example, forks and stars, Flutter and Kotlin have a considerable gap while Kotlin has 39.1k stars and 4.8k forks (GitHub, Inc., 2021a), Flutter has 131k stars, 19.2k forks (GitHub, Inc., 2021b).

The chart in Figure 3 created using Google's trends visually compares searches about Kotlin and Flutter in the past five years. It is easy to see the Flutter's popularity and growth over time.

About the development experience, as mentioned before about popularity, we can talk about performance, App size, and Depency of third party libraries. These technologies ended up turning references of performance in the market. The app size is smaller when using Kotlin because it is no need for as many third-party dependencies as Flutter. To develop apps, Kotlin has a hot reload that improves development speed. On the other hand, Kotlin has better bug detection.

There are many angles to see these comparisons. Each point of comparison has its winner, so it does not have any winner, just promising technologies with different use cases.

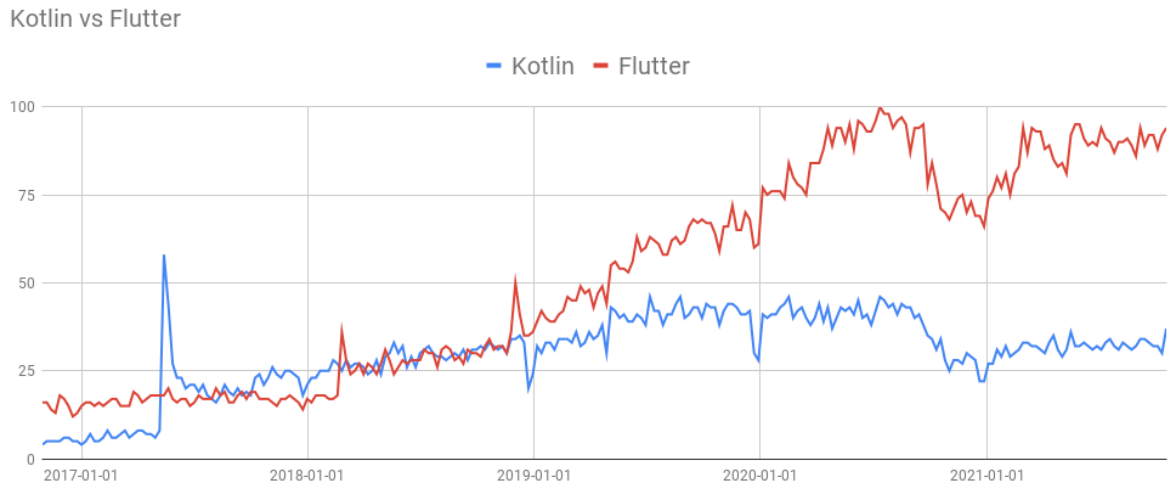


Figure 3 – Google Trends Kotlin and FLutter (GOOGLE, 2021a)

Currently, Kottlin has the support of Google, Jetbrains, and the community, but Jetbrains is the main contributor to Kotlin. In contrast, Flutter has the support of Google and the community, so the company is working hard to improve and promote more and more Flutter.

2.6.4 Javascript

Javascript is a scripting language that is part of the three pillars of web development with HTML and CSS. In just ten days, the Netscape programmer Brandan Eich created this language in 1995. Originally the language was called Mocha before using the name LiveScript and then becoming Javascript (DEGROAT, 2019).

Between 1996 and 1997, Netscape and Brandan Eich saw the rapid growth and adoption of the language. These results show them the need for better maintenance. Therefore they decided to pass the responsibility to ECMA then a new technical committee was created to handle the language (Refsnes Data, 2019).

Currently, Javascript is one of the most used languages (Stack overflow, 2021). This success is because Javascript changed the web development scenario working on the client-side to improve the experience and add new functionalities to the browser. Nodejs arrived to improve the coverage of the language, working on the server-side enabling work with Javascript in the Frontend and Backend.

In the past years, the number of Javascript frameworks has grown. Theses Frameworks were created to work even in the Backend and Frontend. Some were created and maintained for big companies and their community for being open source, like VueJs, ReactJs, and AngularJs.

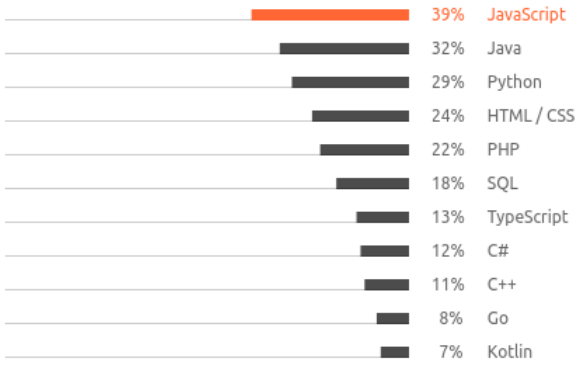


Figure 4 – The State of Developer Ecosystem 2021 (JetBrains s.r.o., 2021a)

Another excellent usage for Javascript is for Hybrid mobile development. Some frameworks use Javascript to create a mobile app for more than one platform. Usually, frameworks treat the Javascript to a specific platform.

2.6.5 React Native

React Native is an open source[3] Javascript framework created by Facebook using React.js, a library created by Facebook to develop websites. React Native's first release was in 2015 in a React.js Conf (RISINGSTACK, 2021).

React Native aims to develop mobile apps for Android and iOS, the development using React Native and React.js is a little similar because both use JSX (Facebook, Inc., 2021c) to render their components. The difference is that while React.js uses HTML elements like p, h1, and div, as in Figure 5, React Native uses elements based on native components in each platform, such as Text, View, and TextInput. There is an illustration of React Native syntax in Figure 6.

```
import React from 'react';
import ReactDOM from 'react-dom';

const App = () => {
  return(
    <h1>Hello World</h1>
  )
}

ReactDOM.render(<App />, document.getElementById('root'));
```

Figure 5 – React.js Example

When React Native arrived, the adoption grew because the framework offered developers beneficial tools to develop apps. The performance is suitable for a cross-platform

[3] <https://github.com/facebook/react-native>

```
import React from 'react';
import { Text, View, StyleSheet } from 'react-native';
import { AppRegistry } from 'react-native';
import { name as appName } from './app.json';

const App = () => {
    return (
        <View
          style={styles.container}>
          <Text>Hello, world!</Text>
        </View>
      )
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: "center",
        alignItems: "center"
    }
});


AppRegistry.registerComponent(appName, () => App);
```

Figure 6 – React Native Example

framework. It is possible to reach 60fps in animations and interactions in some cases.

React Native had many companies using it, but many new technologies arrived to compete for space in the market over time. The number of companies using was decreased, but even the number is smaller than before, great companies are using until now, for Example, Discord, Walmart, Instagram, and Facebook (Facebook, Inc., 2021d).

Currently, React Native has two threads. These threads are about JS, and UI (Facebook, Inc., 2021e). The Javascript thread is responsible for the business logic where the developer writes the code and implements significant parts of the app. At the same time, another one is called Main Thread, where the scroll events, transitions, and some animations happen. The Main Thread is used to listen to some events, for example, gestures. These threads communicate with one another via a mechanism called the bridge that is written in C++/Java. The bridge enables the communication between the two threads (REACTNATIVE.GUIDE, 2020) this communication has a good representation in Figure 7.

2.6.5.1 Native Modules

In some cases, React Native Apps need to use specific platform APIs that Javascript has no access to by default. Native modules enable to Access Apple or Google Play (Facebook, Inc., 2021f), for example.

Native modules are implementations in Objective-C/Java/Swift/C++ (Facebook,

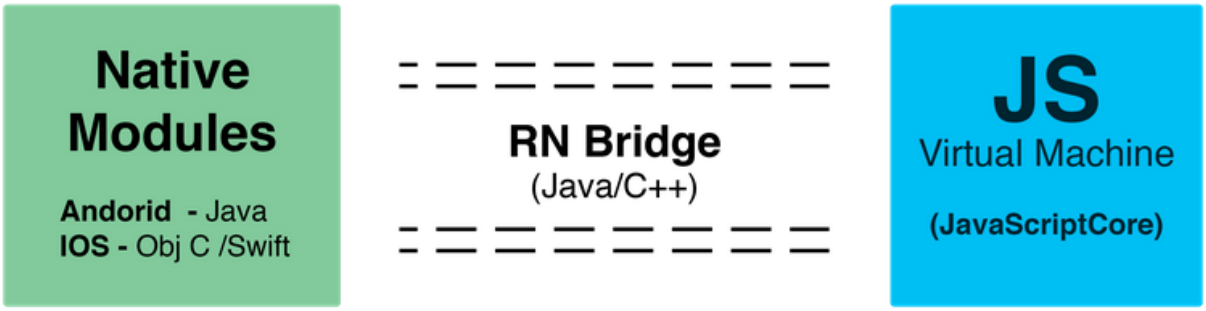


Figure 7 – React Native Threads Communication (REACTNATIVE.GUIDE, 2020)

Inc., 2021f) for exposing functionalities to Javascript in each specific platform. Usually, native modules are used for tasks that need high performance or multi-threading tasks.

React Native has, by default, some native modules created and used for its framework. Creating new native modules is unnecessary for primary usage and the usual development flow. New Native modules are necessary just for specific advanced features.

2.6.6 Kotlin and React Native

Since React Native launched, the base of React Native applications for android was Java because it was the primary technology for Android app development. However, when Google announced that Android development would be Kotlin-first, some developers migrated from Java to Kotlin, interested in its different vertices, but React Native kept using Java.

React Native to Android yet use Java, but the Kotlin conquered a large community and had attention from other audiences. A part of this audience is a React Native developer too. These communities began to join and compare technologies.

Kotlin offers some advantages like a smaller learning curve, especially for Java developers, a big community, support for Android libraries, and the significant point is the interoperability with Java (JetBrains s.r.o., 2021b). This interoperability makes it possible to work with Kotlin and Java in the same project, with the same code base using these two languages working together.

Kotlin has a branch of development called Kotlin Multiplatform Mobile (KMM). It is an SDK for cross-platform development for Android and iOS using the same codebase (JetBrains s.r.o., 2021c). This approach and React Native were compared for developing apps, but React Native generally won this discussion, and the primary usage of Kotlin still was the Android native development.

However, was no end to the history between Kotlin and React Native. That interoperability between Java and Kotlin allows work with both of them in the same language,

can make it possible to create native modules to React Native using Kotlin

3 Methodology

This chapter shows the steps to reach the objectives and results of this work.

3.1 Knowing the React Native Community

This section is a step of the study where the primary purpose is to understand the community organization and be updated with the React Native framework.

The first step to being inserted into the React Native community is to use React Native. This step seems obvious, but it is better to know a framework by using it. Alongside reading, the documentation is essential. There are a lot of instructions, descriptions, and some question that will help persons in this first contact with the framework.

Still talking about the documentation, there is a section called Community (Facebook, Inc., 2021g). It is possible to see official and recommended channels to ask for help, watch conferences, talk groups, and places to be up to date.

There are some official channels to stay up to date, like a Twitter account and a blog for React Native (Facebook, Inc., 2021g).

Conferences are an enormous source of information because tech talks happen, talking about improvements in the framework, new features, and ways to make better usage of some tool related.

There are many repositories related to React Native, the main repository that maintains the core of React Native[1]. Other than a deserving mention is the organization on Github called React Native community[2] that contains repositories with tools for React Native and a specific repository for discussions and proposals.

3.1.1 Main Repository

After using React Native and finding help and discussions, it is time to dive into the repository. The repository is where to find the source code, contribute to the React Native and keep in touch with maintainers and many other contributors.

3.1.2 How To Contribute

In the React Native repository is possible to see steps to become a contributor. Although the goal of this work does not to be a contributor to React Native core, this

[1] <https://github.com/facebook/react-native>
[2] <https://github.com/react-native-community>

reading is essential to understanding how the community organizes itself.

3.1.3 Issues

An incredible place to follow discussion is in the section issues, where discussions happen to improve the framework, problems reported, and questions answered. In the issues section, it was possible to learn about future improvements to the framework.

3.2 Study of Layers of Mobile Development

The core of this study is about React Native, and Kotlin then understands how both works are essential, which involved developing applications and trying to use Native APIs from Android.

About React Native, the studies were more focused on knowing better the lifecycle of the framework and more advanced topics like animations, gestures, and access in the existent modules of React Native.

In the studies about React Native, some questions appear when a developer has to handle with performance problems. Some of them are mentioned in topics of the community and repositories.

For the study for React Native, an essential part of this work was studying more about how the framework works with the native code from a specific platform focused on Android. This Native layer demanded a knowledge of Java to understand the communication between Javascript and Java.

On the other hand, the study about Kotlin to Android began barely from the beginning to understand the development flow, the lifecycle of Android, and how to access native APIs with Kotlin, Kotlin architecture.

Search about the current scenario for React Native and Kotlin was essential to stay up to date with new features, updates, and roadmaps. Both of them are open source and facilitated the study about this.

3.3 First experiment

The first experiment to complete this work is to create a native module to React Native Using Java. This module will serve as a base to create the native module using Kotlin.

The experiment aims to create a native module to create events on the smartphone's calendar. The first module uses Java, and when this module is completed, the

next step is to create a native module functionally equal but using Kotlin.

At the end of the experiment, the native module works using Java and Kotlin. The initial point of the work and the possibility of using Kotlin alongside React Native is prooved.

3.4 Native modules using Kotlin

The last step of this work focuses on creating native modules to React Native using Kotlin and approaching different resources of the Android environment and different approaches of interaction between the native side and the Javascript side. Will be some modules to use Camera, Gallery, Calendar, and Android Date Picker. Concepts of Javascript like Callbacks and Promises cover a good part of this kind of development on the native modules for React Native.

3.5 Tools

This section aims to show the necessary technologies to understand the development steps in this work.

3.5.1 Git

Git (2021) is one of the most used software for tracking versioning in a set of files. This tool is free and open source, initially created by Linus Torvalds. The first release was in 2005.

3.5.2 Android Studio

Android Studio is the official IDE for Android development, supporting Java, Kotlin, and C++ (Android developers, 2021). Google and Jetbrains created it in 2014.

4 Developing with React Native

4.1 Proposal

Merging the knowledge about React Native Java and Kotlin, the most significant proposal of this work is to create a native module to React Native using Kotlin. A long journey happened to reach the main objective and obtain results. The first idea was to use the interoperability between Kotlin and Java to create and document native modules with Kotlin, but a surprise occurred during this work. The React Native developers made available a way to use Kotlin. The proposal from this moment is to guide developers on how to use Kotlin to develop native modules in React Native using Kotlin.

4.1.1 Kotlin Available in React Native

The approach to using Kotlin in React Native would occur at any moment in the future because of the community and features of Kotlin compared to Java. However, this future occurred in version 0.68 of React Native, which began to demonstrate basic usages of Kotlin to create native modules (Facebook, Inc., 2021a).

A new issue was created on 16 March 2022 on the react-native-website repository, an issue called "Help us Kotlin-if the React Native Website" Figure 8. This issue is an excellent step to adopting the Kotlin by the community because it demonstrates the interest in Kotlin and lets the community help with this process.

Now we can focus on how to create these native modules using Kotlin and demonstrate more sophisticated examples to contribute to all of the React Native community.

4.2 Configuring React Native projects with Kotlin

Before developing native modules, it is necessary to enable Kotlin in the React Native project. This process is very similar to enabling Kotlin in the existing Java Android

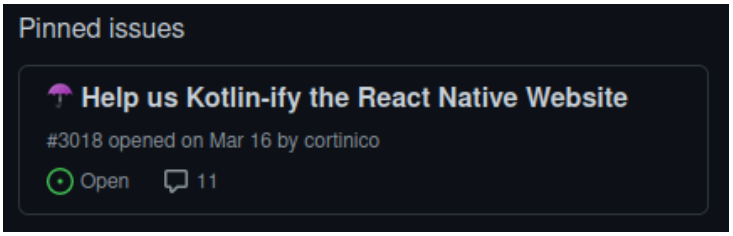


Figure 8 – React Native Kotlin-ify Website (CORTI, 2022)

project[1] because we cannot forget that below the Javascript, there is an Android project too.

Thinking about a new project created using the command ***npx react-native init kotlinsetup*** it is possible to open the ***android*** directory in Android Studio. One of the first things that it is possible to see is a notification on the right bottom corner of the screen, as in Figure 9, suggesting upgrading the Gradle plugin version. It is very recommended to upgrade it to avoid problems with versions.

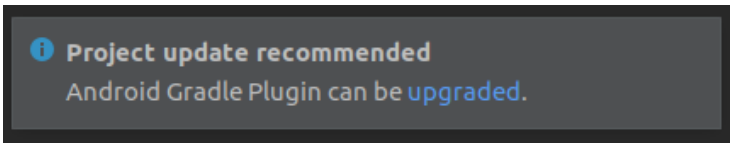


Figure 9 – update Gradle

After updating the Gradle version, it is time to put the Kotlin configuration in the project. The first one is on the ***android/build.gradle*** file, in ***buildscript*** we have some configurations. At first, in ***ext*** add ***kotlin_version*** and Kotlin plugin as a dependency to the project in ***dependencies*** like the Figure 10.

The second file that needs changes is the ***android/app/build.gradle*** the changes in this file, like in Figure 11, are to apply the Kotlin plugin to the created modules.

Now we are ready to take advantage of Kotlin. The project now can use what Kotlin has to offer because the environment was adequately created by applying the necessary plugin to use the interoperability between Java and Kotlin.

[1] <https://developer.android.com/kotlin/add-kotlin>

```
buildscript {
    ext {
        ...
        targetSdkVersion = 31
        kotlin_version = "1.6.20" // Add this line

       ...
    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version" // Add this line
        classpath("com.android.tools.build:gradle:7.0.4")
        ...
    }
}
```

Figure 10 – android/build.gradle

```
apply plugin: "com.android.application"
apply plugin: "kotlin-android" //add this line

...

dependencies {
    ...
    //noinspection GradleDynamicVersion
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version" //add this line
    implementation "com.facebook.react:react-native:+"  // From node_modules
    ...
}
```

Figure 11 – android/app/build.gradle

4.3 First native module with Kotlin

This section shows the journey to develop the first native module with Kotlin thinking about the proposal's viability. After studying Kotlin, Android development, and advanced React Native was time to use React Native's documentation as an inspiration to develop a first native module. It is essential to know that during the development of this module, React Native's documentation did not have examples using Kotlin. The approach was to create a native module based on React Native's documentation [2] using Java and create the same module using Kotlin.

The native module is a module that interacts with the default Android calendar, receiving params, calling the calendar, and filling its fields with the React Native provided data.

4.3.1 Native module using Java

There are some patterns to create native modules to React Native. Then the idea is to explain how to create them, starting with a native module in Java that serves as a base for the first module in Kotlin during this journey.

Inside the folder ***android/app/src/main/java/com/nameoftheapp*** create a new file called ***CalendarModule.java*** as Figure 12 it contains a Java class with the same name. This class extends ReactContextBaseJavaModule, a class used by React Native to their modules. The function ***getName*** needs to be overwritten, returning the module's name. In this case, the module's name is the same as the class name.

It is necessary to connect the Javascript environment and the Java environment. It is time to use an annotation provided by the ***React Native bridge*** to make both

[2] <https://reactnative.dev/docs/native-modules-android>

```java
package com.calendarnativemodulejava;

import com.facebook.react.bridge.ReactContextBaseJavaModule;

public class CalendarModule extends ReactContextBaseJavaModule {

    @Override
    public String getName() {
        return "CalendarModule";
    }
}
```

Figure 12 – Initial Calendar Module

```java
package com.calendarnativemodulejava;

import android.util.Log;

import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.ReactMethod;

public class CalendarModule extends ReactContextBaseJavaModule {
    @Override
    public String getName() {
        return "CalendarModule";
    }

    @ReactMethod
    public void createCalendarEvent(String name, String location) {
        Log.d("CalendarModule", "Create event called with name: " + name
                + " and location: " + location);

    }
}
```

Figure 13 – Creating calendar module method

sides share data. This annotation is called ***ReactMethod***. Functions are callable from the Javascript side with this annotation.

The next step is to create a function called ***createCalendarEvent*** like in Figure 13, passing two strings as a parameter to name and location. This function will be available on the Javascript side.

It is impossible to reach the native module because only in Android devices is it necessary to expose the package. We need to create a new file in the same directory called ***CalendarPackage.java*** like in Figure 14, in this case. It is necessary to implement the interface ***ReactPackage*** and override its methods ***createViewManagers*** and ***createNativeModules***. These methods return a list of native components and native modules. Then after creating this class, add the ***CalendarModule*** into the ***createNativeModules***.

There is a file called ***MainApplication.java*** in the same directory with a class

```java
package com.calendarnativemodulejava;

import com.facebook.react.ReactPackage;
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.uimanager.ViewManager;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class CalendarPackage implements ReactPackage {

    @Override
    public List<ViewManager> createViewManagers(ReactApplicationContext reactContext){
        return Collections.emptyList();
    }

    @Override
    public List<NativeModule> createNativeModules(ReactApplicationContext reactContext) {
        List<NativeModule> modules = new ArrayList<>();

        modules.add(new CalendarModule(reactContext));

        return modules;
    }
}
```

Figure 14 – CalendarPackage

```java
@Override
protected List<ReactPackage> getPackages() {
  @SuppressWarnings("UnnecessaryLocalVariable")
  List<ReactPackage> packages = new PackageList(this).getPackages();
  // Packages that cannot be autolinked yet can be added manually here, for example:
  packages.add(new CalendarPackage());
  return packages;
}
```

Figure 15 – Calendar module main application get packages

with the same name like Figure 15. The next step is to find a function called ***getPackages*** and add the ***CalendarPackage*** created before.

Finally, the native module created before is now available on the Javascript side and used it. To use the native module follow the Figure 16, it is just necessary to import ***NativeModules*** from react-native and get ***CalendarModule*** as a method.

React Native's document has a similar example of creating native modules [3]. Improving this example of a native module and reaching the initial objective of opening the default android calendar with data passed from the app. on the native module, an Android Activity will initialize an Intent for the calendar with the data received as a parameter.

After this journey, the first native module is ready and working as expected. The

[3] <https://reactnative.dev/docs/native-modules-android>

```
import { NativeModules } from 'react-native';
const { CalendarModule } = NativeModules;

...
 const onPress = () => {
  CalendarModule.createCalendarEvent('testName', 'testLocation');
}
...
```

Figure 16 – React Native component using calendar module

```
package com.calendarnativemodulejava;

import android.content.Intent;
import android.provider.CalendarContract;
import android.util.Log;

import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.ReactMethod;

public class CalendarModule extends ReactContextBaseJavaModule {
    ReactApplicationContext context;

    CalendarModule(ReactApplicationContext context) {
        this.context = context;
    }

    @Override
    public String getName() {
        return "CalendarModule";
    }

    @ReactMethod
    public void createCalendarEvent(String name, String location) {
        Intent intent = new Intent(Intent.ACTION_EDIT)
            .setData(CalendarContract.Events.CONTENT_URI)
            .putExtra(CalendarContract.Events.TITLE, name)
            .putExtra(CalendarContract.Events.EVENT_LOCATION, location)
            .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, System.currentTimeMillis())
            .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, System.currentTimeMillis() + (60 *
60 * 1000))
            .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

        context.startActivity(intent);
    }
}
```

Figure 17 – React Native Calendar module completed in Java (REIS, 2022a)

complete code of Figure 17 is on the author's GitHub [4].

4.3.2 Native module using Kotlin

At this moment, a native module created in Java is ready to serve as a base for the first native module in Kotlin. This approach is because to elaborate on this native module, React Native's documentation did not have any example with Kotlin. The process was to create a module in Java and replicate it with Kotlin.

This step assumes that the project is appropriately configured with Kotlin, as mentioned before 4.2.

The first step is in the folder ***android/app/src/main/java/com/nameoftheapp*** create a new file called ***CalendarModule.kt*** like in Figure 18, it contains a Kotlin class with the same name. This class extends ReactContextBaseJavaModule. Yes, the name contains Java because the base class is in Java. Here is a case of interoperability between Kotlin and Java.

```kotlin
package com.calendarnativemodulekotlin

import android.content.Intent
import android.provider.CalendarContract
import android.provider.CalendarContract.Events.*
import android.util.Log
import com.facebook.react.bridge.ReactApplicationContext
import com.facebook.react.bridge.ReactContextBaseJavaModule
import com.facebook.react.bridge.ReactMethod

class CalendarModule(private val reactContext: ReactApplicationContext):
ReactContextBaseJavaModule(reactContext) {


    override fun getName(): String {
        return "CalendarModule"
    }

    @ReactMethod
    fun createCalendarEvent(name: String, location: String) {
        Log.d("CalendarModule", "Create event called with name: " + name
                + " and location: " + location + " Kotlin")

        val intent = Intent(Intent.ACTION_INSERT)
            .setData(CONTENT_URI)
            .putExtra(TITLE, "Kotlin")
            .putExtra(EVENT_LOCATION, "Remote")
            .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, System.currentTimeMillis())
            .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, System.currentTimeMillis() + (60 *
60 * 1000))
            .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK)

            context.startActivity(intent)
    }

}
```

Figure 18 – React Native Calendar module in Kotlin (REIS, 2022b)

Instead of going deeper into how to create a complete native module, this section aims to show the differences between the modules. When the environment is configured

[4] <https://github.com/EzequielDeOliveira/react-native-calendar-java-module>

```kotlin
package com.calendarnativemodulekotlin

import com.facebook.react.ReactPackage
import com.facebook.react.bridge.NativeModule
import com.facebook.react.bridge.ReactApplicationContext
import com.facebook.react.uimanager.ViewManager
import java.util.*
import kotlin.collections.ArrayList

class CalendarPackage : ReactPackage  {
    override fun createNativeModules(reactContext: ReactApplicationContext): List<NativeModule>
{
        val modules = ArrayList<NativeModule>()
        modules.add(CalendarModule(reactContext))
        return modules
    }

    override fun createViewManagers(reactContext: ReactApplicationContext) :
List<ViewManager<*, *>> {
        return Collections.emptyList<ViewManager<*, *>>()
    }
}
```

Figure 19 – React Native Calendar package in Kotlin (REIS, 2022b)

correctly is possible to write in Kotlin without worries about following the patterns to create native modules on React Native.

In this case, the ***CalendarPackage*** of Figure 19 shows how to create a ***ReactPackage*** that does the same as the previous using Kotlin.

To clarify one point about native modules: when the class extends ***ReactContextBaseJavaModule***, the constructor receives a parameter usually called ***reactContext*** of the type ***ReactApplcationContext***. The class receives the parameter from the ***CalendarPackage***, and it is necessary for some operations using Android Context. For instance, the ***CalendarModule*** needs the context to start the activity.

This module is enough to run perfectly, and the next steps are the same as the native module created before because the files using Java now can import Kotlin files without problems. The next step is to add it to the ***MainApplication.java*** and uses it on the Javascript. side.

The first native module in Kotlin is ready and working as expected too. Based on the previous native module in Java, this native module can work well in the React Native environment.

The Javascript side calls the native module called ***createCalendarEvent***, and provided by the native modules, the function receives the params name and location, creates an ***Android Intent*** and finally starts this intent as an ***Android Activity***. Our data from the Javascript side is ready to create an event in the default calendar. This code is on the author's GitHub [5].

After creating the first module in Kotlin, it concludes the initial challenge. It is

[5] <https://github.com/EzequielDeOliveira/react-native-calendar-kotlin-module>

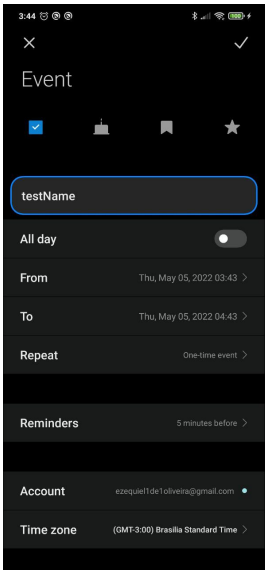


Figure 20 – React Native Calendar package in Kotlin

time to go ahead with more complex examples of native modules in Kotlin, thinking about maximizing this study for the community.

4.4 Android Widgets

At the first moment, the objective was to contribute to the community of React Native and Kotlin developers. The idea was to create a library to React Native using Kotlin and publish it on npm. This library would help to create Android widgets (GOOGLE, 2021b) in a React Native app.

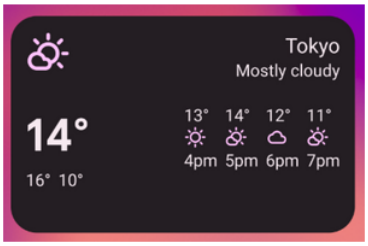


Figure 21 – Information Widget Example (GOOGLE, 2021b)

Starting to develop a solution for the proposed problem was notable that this kind of component needs a strong knowledge of the whole Android environment and a long time to understand how to use it as a library in React Native.

The first step was to create a widget for a React Native app in the traditional way, using the Android project in the Android studio.

The first step was to create a widget in Kotlin for a React Native app in the traditional way, using the Android project in the Android studio. on the folder ***android/ap-***

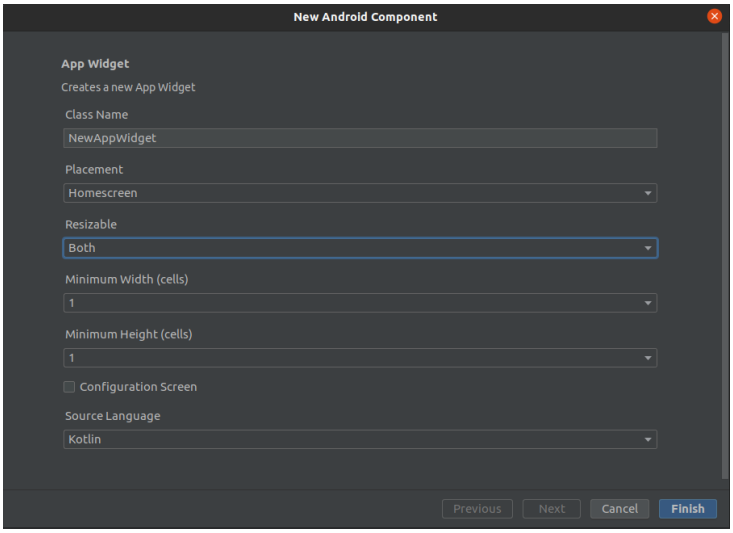


Figure 22 – Create widget

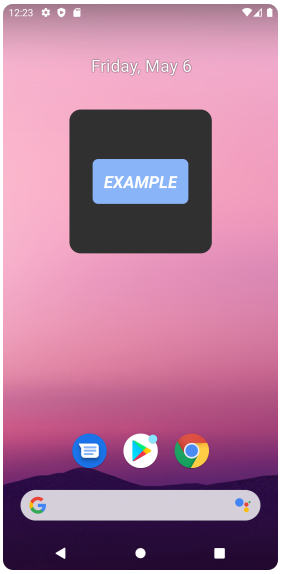


Figure 23 – Simple widget

p/src/main/java/com/nameoftheapp using Android studio, will show a menu like the Figure 22, click with the right button of the mouse in the folder and go to ***new*** > ***Widget*** > ***App Widget***

Click to finish to create an initial structure for a simple widget with a default design like Figure 23.

After doing this basic widget in React Native was possible to see how to work with this kind of application. Analyzing how to create the library, the study ended up with a limitation: the design of the app widget is created entirely with XML was then impossible to create a file using JSX to the widget such as a component in React Native.

Alongside the study and development phase of the library for React Native, the

community started to ***kotlin-ify*** 4.1.1 the documentation. This movement with Kotlin and React Native happening was an excellent opportunity to contribute by showing how to create native modules using Kotlin. Then instead of creating a library that needs the React Native developer to create XML to style their widgets, showing how to create native modules became the way to the aggregate more value.

5 Developing a React Native App Using Native Modules in Kotlin

Although the idea was to create the library to create widgets in React Native, studying more about React Native to create more complex native modules using Kotlin during the current phase of React Native's documentation update with Kotlin is an opportunity to learn and teach.

Applying the knowledge about native modules in React Native is nothing better than creating an app using native modules in Kotlin. The app is a scheduling app to manage events and meetings. The main screens are represented by Figures 24 25 26.

The development followed practices and patterns to React Native and Kotlin for Android apps. The complete app project is open on Github's repository [1].

The app has three native modules and, as a bonus, a native component. This chapter aims to describe each native module used in the project. The modules developed for this project are:

[1] <https://github.com/EzequielDeOliveira/React_native_kotlin_modules>

Figure 24 – Home page

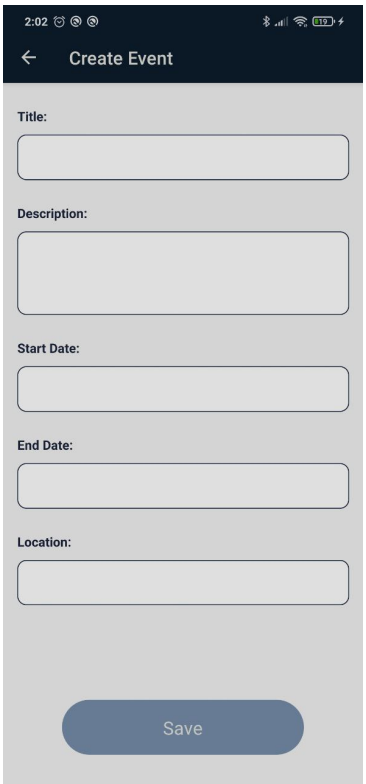


Figure 25 – Create page

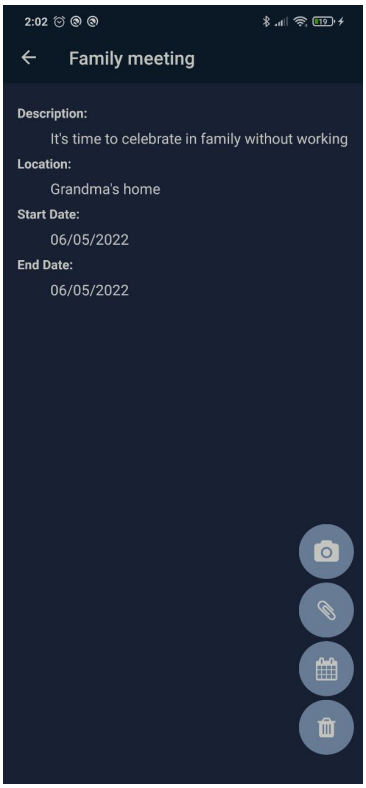


Figure 26 – Event page

- Calendar native module.
- Date picker native module.
- Image picker native module.
- Image view native component

The project setup is the same as mentioned before but using the name ***schedule_kotlin_modules***. After creating the project and config, Kotlin Plugin was in the React Native development phase. This phase aims to develop the app's design and what kind of data is used here and used in the native modules.

The type of data is essential because we are limited to the support of the React Native Bridge in this case. The bridge offers some accepted types of parameters listed in Figure 27.

JAVA	KOTLIN	JAVASCRIPT
Boolean	Boolean	?boolean
boolean		boolean
Double	Double	?number
double		number
String	String	string
Callback	Callback	Function
ReadableMap	ReadableMap	Object
ReadableArray	ReadableArray	Array

Figure 27 – Accepted parameters in native modules (Facebook, Inc., 2021a)

For a better organization create the folder ***kotli*** in the directory ***android/app/src/main/java/com/schedule_kotlin_modules/kotlin*** to store the native modules. To use these native modules on the Javascript side, create another folder called ***NativeModules***

5.1 Calendar native module

As mentioned before, the first native module of the app is the first native module created in Kotlin. It was improved to be part of the final application.

Create a file called ***CalendarModule.kt*** represented by Figure 28 inside the folder ***kotlin*** created before 4.3.

```kotlin
package com.schedule_kotlin_modules.kotlin

import android.content.Intent
import android.provider.CalendarContract
import android.provider.CalendarContract.Events.*
import android.util.Log
import com.facebook.react.bridge.ReactApplicationContext
import com.facebook.react.bridge.ReactContextBaseJavaModule
import com.facebook.react.bridge.ReactMethod
import java.text.SimpleDateFormat

class CalendarModule(private val reactContext: ReactApplicationContext): ReactContextBaseJavaModule(reactContext) {
    override fun getName() = "CalendarModule"

    @ReactMethod
    fun createEvent(title: String, location: String, startDate: String, endDate: String) {
        try {
            val intent = Intent(Intent.ACTION_INSERT)
            .setData(CONTENT_URI)
            .putExtra(TITLE, title)
            .putExtra(EVENT_LOCATION, location)
            .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, SimpleDateFormat("dd/MM/yyyy").parse(startDate).time)
            .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, SimpleDateFormat("dd/MM/yyyy").parse(endDate).time + (60 * 60 * 1000))
            .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK)

        reactContext.startActivity(intent)
        } catch(t: Throwable) {
            Log.d("CalendarModule", "Error to create event")
        }
    }
}
```

Figure 28 – Calendar native module (REIS, 2022c)

The changes here are the parameters passed to the method to create an entire event and serve a better experience in the app. The signature of the class keeps the pattern to create native modules. The function ***getName*** returns the name of the native module using a short syntax provided by Kotlin.

The method ***createevent*** receives the title, location, start date, and end date of the event. The first step is to create an Android Intent using the Android ***CalendarContract*** (GOOGLE, 2022a), setting info to the Intent. The ***SimpleDateFormat*** created dates from the strings provided as parameters for dates.

If the initialization happened without error, the context of the application starts the activity using the calendar intent.

To use on the Javascript side, create a file called ***CalendarModule.js*** like in Figure 29 this file will import our native module and export the component to the Javascript side.

```javascript
import { NativeModules } from 'react-native';

const { CalendarModule } = NativeModules;

/**
 * DatePickerModule
 * Avaliable functions:
 * - createEvent(title, location, startDate, endDate)
 *      - title: String
 *      - location: String
 *      - startDate: String dd/mm/yy
 *      - endDate: String dd/mm/yy
*/
export default CalendarModule;
```

Figure 29 – Javascript Calendar native module (REIS, 2022c)

Now it is possible to use the function exported on the Javascript side to create events. In the app, this function saves our event in the calendar. The result is in Figures

30, 31.

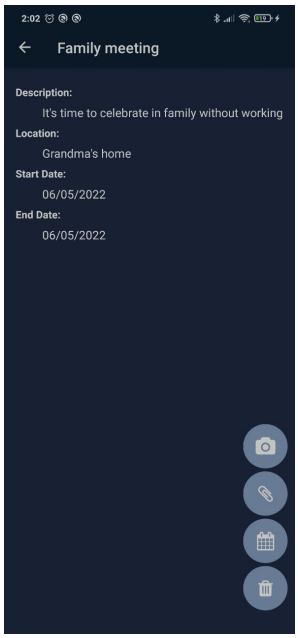


Figure 30 – Event page

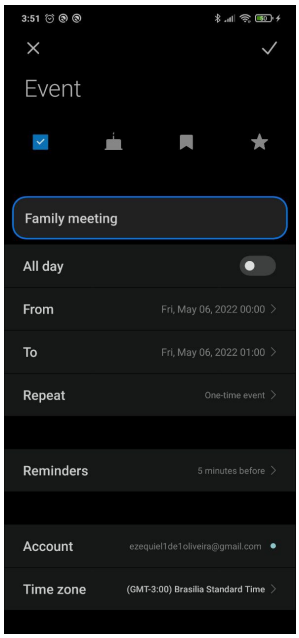


Figure 31 – Booking event

5.2 Date picker native module

The second native module is to handle a date picker from Android. Date pickers are components used in various applications because it is usual to use dates.

Until now, the developed modules receive data from the Javascript side, handle this data and take some path, but probably will appear a case that is necessary to send data from Javascript to Kotlin and get a response. The module will use the concept of callback to send the data back to Javascript.

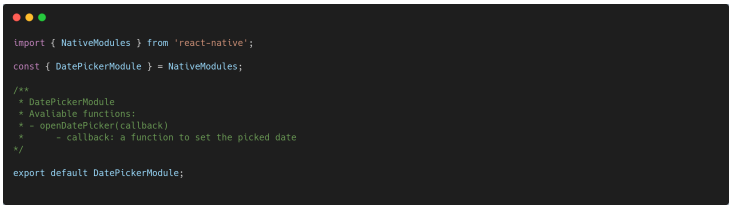
Create a file called ***DatePickerModule.kt*** like Figure 32, inside the folder ***kotlin*** created before 5.

This native module has a function marked with ***@ReactMethod*** it receives just one parameter called callback of type Callback. This callback is a Javascript function to be called in the native module to send data to Javascript. A peculiarity is that this callback has an id. Then each callback received as a parameter only can be called one time (Facebook, Inc., 2021a).

The method initializes by getting an instance of the phone's calendar and the current activity. The next step is to create a listener to handle the events from the date picker. After handling the date selected, the listener uses the callback to return the result to the Javascript side.

```kotlin
class DatePickerModule(
    reactContext: ReactApplicationContext
) : ReactContextBaseJavaModule(reactContext) {
    override fun getName() = "DatePickerModule"

    @ReactMethod
    fun openDatePicker(callback: Callback) {
        val calendar = Calendar.getInstance()
        val activity = currentActivity

        val dateSetListener = DatePickerDialog.OnDateSetListener{ _ , year, month, day ->
            calendar.set(Calendar.YEAR, year)
            calendar.set(Calendar.MONTH, month)
            calendar.set(Calendar.DAY_OF_MONTH, day)

            val locale = Locale("pt", "BR")
            val sdf = SimpleDateFormat("dd/MM/yyyy", locale)
            val result = sdf.format(calendar.time);
            callback.invoke(result)
        }

        if(activity == null) {
            return
        }

        DatePickerDialog(activity,
            R.style.SpinnerDatePickerDialog,
            dateSetListener,
            calendar.get(Calendar.YEAR),
            calendar.get(Calendar.MONTH),
            calendar.get(Calendar.DAY_OF_MONTH)
            ).show()
    }
}
```

Figure 32 – Date picker Kotlin native module (REIS, 2022d)



```javascript
import { NativeModules } from 'react-native';

const { DatePickerModule } = NativeModules;

/**
 * DatePickerModule
 * Avaliable functions:
 * - openDatePicker(callback)
 *      - callback: a function to set the picked date
*/

export default DatePickerModule;
```

Figure 33 – Javascript side date picker module (REIS, 2022d)

After the setup, the ***DatePickerDialog*** (GOOGLE, 2022b) calls the function ***show()*** to instantiate it, and the user inserts the desired date.

To use on the Javascript side, create a file called ***DatePickerModule.js*** represented by Figure 33, this file will import our native module and export the component to the Javascript side.

The result is in Figure 34. From now it is possible to import this function and use it like this:

```
DatePickerModule.openDatePicker(date => console.log(date))
```

5.3 Image picker native module

The last native module is so far the most sophisticated because this native module uses a concept called promise, where a function returns a response. However, the response can be negative or positive, and the response time can change on each call.

On the Kotlin side, this time, we have some assistant functions to help us to react to the objective of this native module.

The main objective of this module is to get images from the gallery and camera of the smartphone using a native module to handle this kind of process.

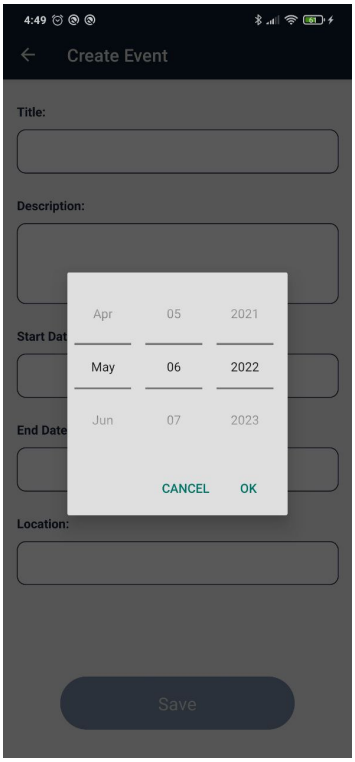


Figure 34 – Date to the event

Create a file called ***ImagePickerModule.kt*** inside the folder ***kotlin*** created before.

The first assistant function called ***createFile*** like Figure 35, is responsible for using the resource nullable from Kotlin and creating the file necessary for getting images from the camera.

```
private fun createFile(): File {
        val timestamp: String = SimpleDateFormat("yyyy_MM_dd_mm_ss").format(Date())
        val dir: File? = currentActivity?.getExternalFilesDir(Environment.DIRECTORY_PICTURES)
        return File.createTempFile(
            "JPEG_${timestamp}_",
            ".jpg",
            dir
        )
    }
```

Figure 35 – Create image file (REIS, 2022e)

The second one is responsible for handling the response of the activities opened before from other methods. Kotlin is essential for this function to handle null values and apply resources like "let" and "when" to improve the method's readability. In the end, this method receives the image's address and responds to the promise positively, or if it does not have any address, it sends a negative response. The activity listener is represented in Figure 36.

```
private val activityEventListener = object : BaseActivityEventListener() {
        override fun onActivityResult(
            activity: Activity?,
            requestCode: Int,
            resultCode: Int,
            data: Intent?
        ) {
            pickerPromise?.let { promise ->
                when (resultCode) {
                    Activity.RESULT_CANCELED ->
                        promise.reject(E_PICKER_CANCELLED, "image picker was cancelled")
                    Activity.RESULT_OK -> {
                        when (requestCode) {
                            REQUEST_IMAGE_PICK -> {
                                uri = data?.data

                                uri?.let {
                                    promise.resolve(uri.toString())
                                }

                                    ?: promise.reject(E_NO_IMAGE_DATA_FOUND, "No image data found")
                            }
                            REQUEST_CAMERA_PICK -> {
                                uri?.let {
                                    promise.resolve(uri.toString())
                                }

                                    ?: promise.reject(E_NO_IMAGE_DATA_FOUND, "No image data found")
                            }
                        }
                    }
                }
                pickerPromise = null
            }
        }
    }
```

Figure 36 – Activity Event listenter (REIS, 2022e)

```
@ReactMethod
    fun pickFromGallery(promise: Promise) {
        val activity = currentActivity
        pickerPromise = promise

        try {
            val intent = Intent(Intent.ACTION_PICK).apply {
                type = "image/*"
            }
            activity?.startActivityForResult(intent, REQUEST_IMAGE_PICK)
        } catch (t: Throwable) {
            pickerPromise?.reject(E_FAILED_TO_SHOW_PICKER, t)
            pickerPromise = null
        }
    }
```

Figure 37 – Pick images from gallery (REIS, 2022e)

After explaining the assistant methods, it is time to create the methods exposed to the Javascript side. Figure 37.

This method receives a promise of type Promise as a parameter and creates an activity for the gallery. The id used to start the activity return it to the listener. If all happen as expected, the functions open the gallery. If an error occurs, the promise function rejects the promise. Figure 38

This method receives a promise of type Promise as a parameter. It creates an

```kotlin
@ReactMethod
    fun pickFromCamera(promise: Promise) {
        val activity = currentActivity
        pickerPromise = promise

        if (activity?.packageManager == null) {
            pickerPromise?.reject(E_FAILED_TO_SHOW_PICKER, "Error to get Activity")
            pickerPromise = null
            return
        }

        try {
            Intent(MediaStore.ACTION_IMAGE_CAPTURE).also { takePictureIntent ->
                takePictureIntent.resolveActivity(activity.packageManager).also {
                    val photoFile: File? = createFile()

                    photoFile?.also {
                        uri = FileProvider.getUriForFile(
                            reactContext,
                            BuildConfig.APPLICATION_ID + ".provider",
                            it
                        )
                        takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, uri)
                        activity?.startActivityForResult(takePictureIntent, REQUEST_CAMERA_PICK)
                    }

                }
            }
        } catch (t: Throwable) {
            pickerPromise?.reject(E_FAILED_TO_SHOW_PICKER, t)
            pickerPromise = null
        }
    }
```

Figure 38 – Pick images from camera (REIS, 2022e)

```javascript
import { NativeModules } from 'react-native';

const { ImagePickerModule } = NativeModules;

/**
 * ImagePickerModule
 * Avaliable functions:
 * - pickFromGallery() ~ Promise
 *      - promise: used to get image data
 * - pickFromCamera() ~ Promise
 *      - promise: used to get image data
*/

export default ImagePickerModule;
```

Figure 39 – Javascript pick images module (REIS, 2022e)

intent for the camera, checking if the image is created correctly and starting the activity for this camera.

To use on the Javascript side, create a file called ***ImagePickerModule.js*** like in Figure 39, this file will import our native module and export the component to the Javascript side.

Finally, using the native module on the Javascript side is possible, getting images from the gallery or the camera. This functionality was applied to the application to save moments for the meets.

An emulator is not a good option for picking images from the camera. Then the smartphone was a ***Xiaomi Redmi 8***

The app gets images from the gallery and camera and saves them with a specific event like Figures 40 - 43.

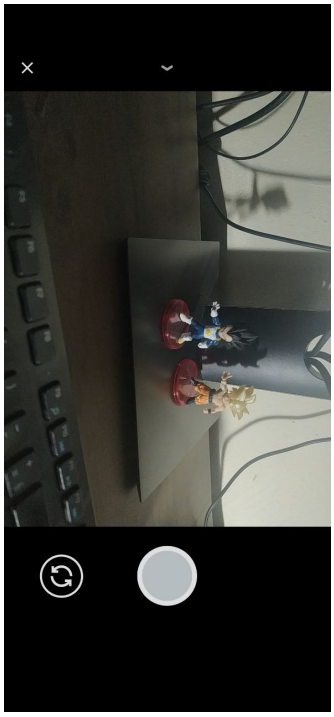


Figure 40 – Camera intent

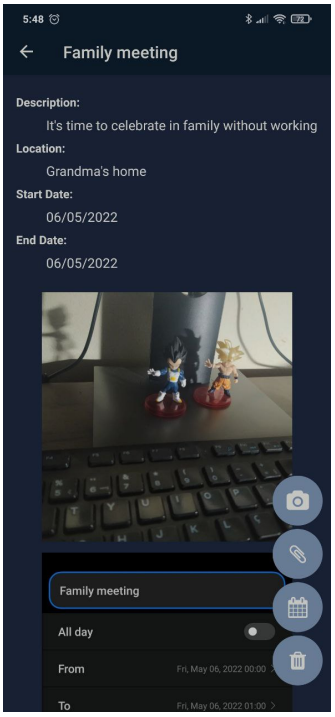


Figure 41 – Saved photo

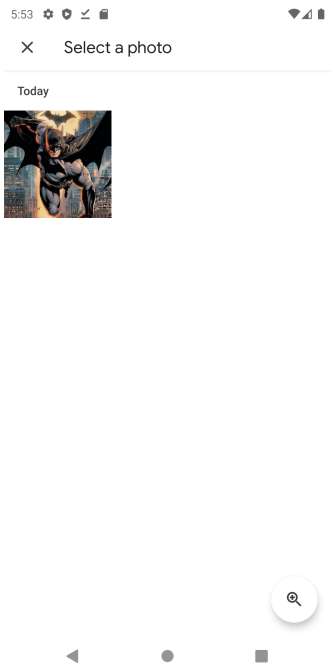


Figure 42 – Gallery intent

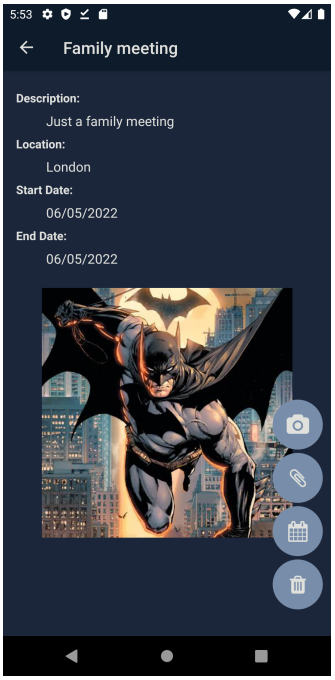


Figure 43 – Saved photo

5.4 Image view native component

After creating these native modules as an extra is not a bad idea to create a simple native component. Let's create a simple Image component to start creating a file called ***NativeImageView.kt*** like Figure 44 inside the folder ***kotlin*** created before.

Unlike the native modules, native components classes need to extend ***SimpleViewManager***, override ***getName***, and create a view instance for the native component.

This component needs to receive the image's address. To pass properties to native components, the annotation is ***@ReactProp***

```kotlin
package com.schedule_kotlin_modules.kotlin

import com.facebook.drawee.backends.pipeline.Fresco
import com.facebook.react.bridge.*
import com.facebook.react.uimanager.SimpleViewManager
import com.facebook.react.uimanager.ThemedReactContext
import com.facebook.react.uimanager.annotations.ReactProp
import com.facebook.react.views.image.ReactImageView

class NativeImageView(
    private val reactContext: ReactApplicationContext
) : SimpleViewManager<ReactImageView>() {
    override fun getName() = "CustomNativeImage"

    override fun createViewInstance(context: ThemedReactContext) =
        ReactImageView(context, Fresco.newDraweeControllerBuilder(), null, reactContext)

    @ReactProp(name = "src")
    fun setSrc(view: ReactImageView, uri: ReadableMap){
        val sources = WritableNativeArray()
        sources.pushMap(uri)
        view.setSource(sources)
    }
}
```

Figure 44 – Native component (Facebook, Inc., 2021b)

There is a list of images for each event in the app. Insert image via other native modules.

5.5 Package

The last step to using all of these native modules is to create the package to create the view managers and native modules. Inside the folder ***kotlin*** yet, create a file called ***KotlinAppPackage.kt*** like Figure 45, in this package. We will put all of the modules exposed to the Javascript side.

The package contains all of the modules receiving the context. Now it is time to go to ***android/app/src/main/java/com/schedule_kotlin_modules/MainApplication.java*** Figure 46, and add the package in the ***getPackages*** method

```kotlin
package com.schedule_kotlin_modules.kotlin

import android.view.View
import com.facebook.react.ReactPackage
import com.facebook.react.bridge.NativeModule
import com.facebook.react.bridge.ReactApplicationContext
import com.facebook.react.uimanager.ReactShadowNode
import com.facebook.react.uimanager.ViewManager

class KotlinAppPackage : ReactPackage {
    override fun createNativeModules(reactContext: ReactApplicationContext):
            MutableList<NativeModule> = listOf(
        DatePickerModule(reactContext),
        ImagePickerModule(reactContext),
        CalendarModule(reactContext)
    ).toMutableList()

    override fun createViewManagers(reactContext: ReactApplicationContext) = listOf(NativeImageView(reactContext)).toMutableList()
}
```

Figure 45 – Kotlin app package (REIS, 2022f)

```java
        ...
        @Override
        protected List<ReactPackage> getPackages() {
          @SuppressWarnings("UnnecessaryLocalVariable")
          List<ReactPackage> packages = new PackageList(this).getPackages();
          // Packages that cannot be autolinked yet can be added manually here, for example:
          packages.add(new KotlinAppPackage());
          return packages;
        }
        ...
```

Figure 46 – Add kotlin app package (REIS, 2022g)

The package is created correctly with all of the native modules. Now it is possible to use all of them as desired in the application developed in this project and have good use of each native module <https://github.com/EzequielDeOliveira/React_native_kotlin_modules>.

6 Conclusion

Currently, mobile development is growing fast, and companies need to improve their processes every day to improve user experience, attract more customers, and stay one click distance from the user. However, the competition is wild for this business, so using the best technologies puts companies one step ahead of others. Inappropriate use of this kind of technology can cause financial injury to the company.

There are many languages and frameworks to improve a business or solve a problem, but in this work, the focus goes to React Native and Kotlin.

Kotlin has an excellent experience for new users. Some points noticed are that The language is easy to learn. The community is enormous and always available to help. The code is safe, making it easy to handle errors. Different from Java, null exceptions are more difficult to appear in Kotlin. The interoperability between Java and Kotlin makes it possible to migrate from Java anytime or even work with both languages in the same project.

Interoperability was an essential point in finishing this proposed work. A difficulty encountered was using this interoperability between Java and Kotlin and enabling the right plugins for the languages to work together.

React Native is another example of how big a community can be, and this framework is very receptive too because the new users will have to learn Javascript and some concepts of style then. Starting in React Native is an excellent experience.

There is a moment in the life of a React Native developer when using Javascript is not enough to make great apps. to go through this point, Native modules are essential because the developer has many options for using native resources.

Working in this project alongside a movement of the React Native community to adopt Kotlin was an excellent opportunity to learn and contribute to developers interested in the topic. Because when this work started, Kotlin was just a discussion in Github, but in the middle, the Kotlin was real to React Native.

Using Kotlin in React Native can increase the visibility of Kotlin and attract more contributors to the core of React Native, given that learning Kotlin can be easier than learning Java.

This work delivers all of the initial objectives of analyzing the current technologies and implementing and creating many valuable examples of native modules using Kotlin.

Looking at the results is possible to see the potential of using React Native with

native modules, this work focus on Android, but the process is the same in iOS. Using native modules can change to the level of any application making it possible to use the maximum of the native resources of the operating system.

An essential thing in any mobile application is the feeling of the user using the application. Creating an app unified to iOS and Android has advantages. However, when the app is very unified, users of an OS can feel uncomfortable because the app does not look to be part of the OS used.

The effort to create a hybrid application using React Native and make this look and feel part of the OS for the user requires more knowledge and costs for the development team. The company will need to invest in developers who have knowledge about Javascript and another native language instead of only Javascript.

There are different cases. This work is essential to see how robust a framework can be, using a native language as a base.

Although the good results and the process of completing this work, some problems appear. The main problem with completing it is the content about this. This work brings a new thing, so some problems do not have an easy solution on the internet. Kotlin, React Native, and Android documentation was essential to solve the faced problems. Following these languages' patterns and reading documentation could reach a good result.

6.1 Future works

This work is not the end of the journey using native modules, Kotlin, React Native, and tools for Android and even iOS. During this project's time, it was possible to keep in touch and close to the Kotlin and React Native communities.

Currently, there are contributions in progress besides this work. As part of the community, the central plan is to continue contributing to the whole community with code, documentation, and content to teach and learn better.

The point of continuing this work or other related works and subjects is to explore more subjects of the native development for Android. The modules created in this work cover more used use cases about the native resources on Android. For instance, the native resources cited here were Camera, Gallery, Calendar, and Android Date Picker. There are different resources to approach based on the result of this work, resources as GPS, maps, and Bluetooth.

Bibliography

ANDROID. *Android*. 2021. Available at: <https://developer.android.com/kotlin/first>. Accessed on: out, 17. 2021. Cited on page 20.

Android developers. *Android Studio*. 2021. Available at: <https://developer.android.com/studio>. Cited on page 29.

CALLAHAM, J. *History of Android*. 2021. Available at: <https://www.androidauthority.com/history-android-os-name-789433/>. Accessed on: out, 11. 2021. Cited on page 18.

CORTI, N. 2022. Available at: <https://github.com/facebook/react-native-website/issues/3018>. Accessed on: Mar, 16. 2022. Cited 2 times on pages 9 and 31.

COSTELLO, S. *History of iOS*. 2021. Available at: <https://www.lifewire.com/ios-versions-4147730>. Accessed on: out, 1. 2021. Cited on page 18.

DEGROAT, T. 2019. Available at: <https://www.springboard.com/blog/data-science/history-of-javascript/>. Accessed on: out, 28. 2021. Cited on page 22.

EL-KASSAS, W. S. et al. Taxonomy of cross-platform mobile applications development approaches. *Ain Shams Engineering Journal*, Ain Shams University, v. 8, p. 163–190, 6 2017. ISSN 20904479. Cited 2 times on pages 18 and 19.

Facebook, Inc. 2021. Available at: <https://reactnative.dev/docs/native-modules-android>. Accessed on: Apr, 20. 2022. Cited 4 times on pages 9, 31, 44, and 46.

Facebook, Inc. 2021. Available at: <https://reactnative.dev/docs/native-components-android>. Accessed on: Nov, 2. 2021. Cited 2 times on pages 10 and 52.

Facebook, Inc. 2021. Available at: <https://reactjs.org/docs/introducing-jsx.html>. Accessed on: out, 31. 2021. Cited on page 23.

Facebook, Inc. 2021. Available at: <https://reactnative.dev/showcase>. Accessed on: Nov, 1. 2021. Cited on page 24.

Facebook, Inc. 2021. Available at: <https://reactnative.dev/docs/performance#what-you-need-to-know-about-frames>. Accessed on: Nov, 1. 2021. Cited on page 24.

Facebook, Inc. 2021. Available at: <https://reactnative.dev/docs/native-modules-intro>. Accessed on: Nov, 2. 2021. Cited 2 times on pages 24 and 25.

Facebook, Inc. 2021. Available at: <https://reactnative.dev/help>. Accessed on: Nov, 11. 2021. Cited on page 27.

Git. *Git*. 2021. Available at: <https://git-scm.com/>. Cited on page 29.

GitHub, Inc. 2021. Available at: <https://github.com/JetBrains/kotlin>. Accessed on: out, 25. 2021. Cited on page 21.

GitHub, Inc. 2021. Available at: <https://github.com/flutter/flutter>. Accessed on: out, 25. 2021. Cited on page 21.

GOOGLE. 2021. Available at: <https://trends.google.com/trends/explore?date=today%205-y&q=Kotlin,Flutter>. Accessed on: out, 25. 2021. Cited 2 times on pages 9 and 22.

GOOGLE. 2021. Available at: <https://developer.android.com/guide/topics/appwidgets/overview>. Accessed on: Nov, 11. 2021. Cited 2 times on pages 9 and 39.

GOOGLE. *Flutter*. 2021. Available at: <https://flutter.dev/docs/resources/faq#what-is-flutter>. Accessed on: out, 17. 2021. Cited on page 20.

GOOGLE. *Flutter Architecture*. 2021. Available at: <https://flutter.dev/docs/resources/architectural-overview#architectural-layers>. Accessed on: out, 17. 2021. Cited 2 times on pages 9 and 20.

GOOGLE. *Flutter Showcase*. 2021. Available at: <https://flutter.dev/showcase>. Accessed on: out, 17. 2021. Cited on page 21.

GOOGLE. *Oficial documentation*. 2021. Available at: <https://flutter.dev/docs/resources/faq#what-is-flutter>. Accessed on: out, 25. 2021. Cited on page 21.

GOOGLE. 2022. Available at: <https://developer.android.com/reference/android/provider/CalendarContract.Events>. Accessed on: Apr, 1. 2022. Cited on page 45.

GOOGLE. 2022. Available at: <https://developer.android.com/reference/android/app/DatePickerDialog>. Accessed on: Apr, 26. 2022. Cited on page 47.

JetBrains s.r.o. 2021. Available at: <https://www.jetbrains.com/lp/devecosystem-2021/#Main_what-are-your-primary-programming-languages-choose-no-more-than-3-languages>. Accessed on: out, 28. 2021. Cited 2 times on pages 9 and 23.

JetBrains s.r.o. 2021. Available at: <https://kotlinlang.org/docs/android-overview.html>. Accessed on: Apr, 23. 2022. Cited on page 25.

JetBrains s.r.o. 2021. Available at: <https://kotlinlang.org/lp/mobile>. Accessed on: Apr, 25. 2022. Cited on page 25.

JetBrains s.r.o. *History of Kotlin*. 2021. Available at: <https://kotlinlang.org/docs/faq.html>. Accessed on: out, 17. 2021. Cited on page 19.

JetBrains s.r.o. *Kotlin Foundation*. 2021. Available at: <https://kotlinlang.org/docs/kotlin-foundation.html>. Accessed on: out, 17. 2021. Cited on page 20.

REACTNATIVE.GUIDE. 2020. Available at: <https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html>. Accessed on: Nov, 1. 2021. Cited 3 times on pages 9, 24, and 25.

Refsnes Data. 2019. Available at: <https://www.w3schools.com/js/js_history.asp>. Accessed on: out, 28. 2021. Cited on page 22.

REIS, E. D. O. D. 2022. Available at: <https://github.com/EzequielDeOliveira/react-native-calendar-java-module>. Accessed on: Mar, 13. 2022. Cited 2 times on pages 9 and 36.

REIS, E. D. O. D. 2022. Available at: <https://github.com/EzequielDeOliveira/react-native-calendar-kotlin-module>. Accessed on: Mar, 23. 2022. Cited 3 times on pages 9, 37, and 38.

REIS, E. D. O. D. 2022. Available at: <https://github.com/EzequielDeOliveira/React_native_kotlin_modules/blob/main/android/app/src/main/java/com/schedule_kotlin_modules/kotlin/CalendarModule.kt>. Accessed on: May, 1. 2022. Cited 2 times on pages 9 and 45.

REIS, E. D. O. D. 2022. Available at: <https://github.com/EzequielDeOliveira/React_native_kotlin_modules/blob/main/android/app/src/main/java/com/schedule_kotlin_modules/kotlin/DatePickerModule.kt>. Accessed on: Apr, 29. 2022. Cited 2 times on pages 9 and 47.

REIS, E. D. O. D. 2022. Available at: <https://github.com/EzequielDeOliveira/React_native_kotlin_modules/blob/main/android/app/src/main/java/com/schedule_kotlin_modules/kotlin/ImagePickerModule.kt>. Accessed on: Apr, 30. 2022. Cited 5 times on pages 9, 10, 48, 49, and 50.

REIS, E. D. O. D. 2022. Available at: <https://github.com/EzequielDeOliveira/React_native_kotlin_modules/blob/main/android/app/src/main/java/com/schedule_kotlin_modules/kotlin/KotlinAppPackage.kt>. Accessed on: May, 1. 2022. Cited 2 times on pages 10 and 53.

REIS, E. D. O. D. 2022. Available at: <https://github.com/EzequielDeOliveira/React_native_kotlin_modules/blob/main/android/app/src/main/java/com/schedule_kotlin_modules/MainApplication.java>. Accessed on: Apr, 28. 2022. Cited 2 times on pages 10 and 53.

RISINGSTACK. 2021. Available at: <https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/>. Accessed on: out, 31. 2021. Cited on page 23.

SACHINDANA, S. *Introduction to Flutter*. 2021. Available at: <https://dev.to/sudarasach/intro-to-flutter-2odk>. Accessed on: out, 17. 2021. Cited on page 20.

Stack overflow. 2021. Available at: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-programming-scripting-and-markup-languages>. Accessed on: out, 28. 2021. Cited on page 22.

STATCOUNTER. *Comparing operational systems*. 2021. Available at: <https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201808-202108>. Accessed on: Set, 18. 2021. Cited 2 times on pages 9 and 17.