I'm not a robot

Steps to reproduce the issue or to explain in which case you get the issue Trying to render the pdf from a url, similar to the example provided in docs const source = { uri: ' , cache: true }; The pdf works perfect in Emulation or devices running older android version, but not in actual device running on Android 12. Interesting logs None. Show us the code you are using? const pdfUrl = {uri: 'actual_path',cache: true}; { this.pdf = pdf; }} trustAllCerts={false} source={pdfUrl} onLoadComplete={numberOfPages => { this.setState({ ..... }); }} onPageChanged={page => { this.setState({ .... }); }} onError={error => { this.setState({ . ..... }); }} style={styles.pdf} /> Device tested (Oneplus 9, Oneplus 10R, Android V12). Working on RealMe, Samsung devices running on Android V11. You can't perform that action at this time. Not so long ago UppLabs was working on a project which key functionalities included a PDF generation on mobile. In this material, we'll share some tips on how to generate a PDF document using React Native. The simplest way to create a PDF document in a React Native project includes the use of Expo Print plugin. The first step is to write HTML with a sample of content that should be in our PDF. You can insert CSS styles, custom fonts, images, links, etc to your markup. const htmlContent = ` Pdf Content body { font-size: 16px; color: rgb(255, 196, 0); } h1 { text-align: center; } Hello, UppLabs! `; Then you should write function, that creates a PDF document with defined HTML markup: import * as Print from 'expo-print'; const htmlContent = ` .... `; const createPDF = async (html) => { try { const { uri } = await Print.printToFileAsync({ html }); return uri; } catch (err) { console.error(err); } }; Notice: This function will create a PDF file and save it to the application cache folder. So if you want to replace or copy this file, you can use Expo Filesystem. If you want to save it to the Gallery on your device, you can use Expo MediaLibrary on Android platform and Expo Sharing on IOS platform. Example of function, that allows you to create PDF and save it to Gallery: import * as Print from "expo-print"; import * as MediaLibrary from "expo-media-library"; import * as Sharing from "expo-sharing"; .... const createAndSavePDF = async (html) => { try { const { uri } = await Print.printToFileAsync({ html }); if (Platform.OS === "ios") { await Sharing.shareAsync(uri); } else { const permission = await MediaLibrary.requestPermissionsAsync(); if (permission.granted) { await MediaLibrary.createAssetAsync(uri); } } } catch (error) { console.error(error); } }; You can use this functionality in your React Native components and that will be enough if you need to create a quick PDF document with simple content. If you're using the function Print.printToFileAsync, the resulting PDF document might contain page margins (it depends on WebView engine). They are set by @page style block and you can override them in your HTML content code: @page { margin: 20px; } If you insert a few content sections in your HTML markup, it may not fit together on the same document page. In such cases, the content of the second section breaks, and part of it is placed on the next page of the document. But sometimes such an approach is unacceptable when you want to avoid breaking the content section. To control the breaking of some HTML elements you can use the "break-inside" CSS property in your HTML slicing. section { break-inside: avoid; } So, if two sections are not fit on the same page, the second section will be placed on the next page of the document without breaking. You can use custom fonts, images, or other assets in your HTML without any troubles if it's placed remotely. But if you want to use the image from React Native app assets, it may be a problem, because such images can't be loaded to your document when your React Native app is built for production. In such cases, you should first copy the file from assets to your application's cache directory. Unfortunately, on the IOS platform, local files can't be loaded to PDF as well, so you should convert the local file to base64 string. After that, you can use the URL of the copied file (on Android platform) or base64 string (on IOS Platform) in your HTML markup. In such cases, you can use Expo Asset and Expo ImageManipulator. Example of code, that copies an image from assets and converts it to base64 for IOS platform before using it in a PDF document: import { Platform } from "react-native"; import { Asset } from "expo-asset"; import * as ImageManipulator from "expo-image-manipulator"; .... const copyFromAssets = async (asset) => { try { await Asset.loadAsync(asset); const { localUri } = Asset.fromModule(asset); return localUri; } catch (error) { console.log(error); throw err; } }; const processLocalImageIOS = async (imageUri) => { try { const uriParts = imageUri.split("."); const formatPart = uriParts[uriParts.length - 1]; let format; if (formatPart.includes("png")) { format = "png"; } else if (formatPart.includes("jpg") || formatPart.includes("jpeg")) { format = "jpeg"; } const { base64 } = await ImageManipulator.manipulateAsync(imageUri, [], { format: format || "png", base64: true }); return `${base64}`; } catch (error) { console.log(error); throw error } }; .... const htmlContent = async () => { try { const asset = require('./assets/logo.png'); let src = await copyFromAssets(asset); if(Platform.OS === 'ios') { src = await processLocalImageIOS(src); } return `` } catch (error) { console.log(error); } } So after calling this code you can use the returned URL of the asset image in your PDF document. If you want to place large images in your PDF document, you should optimize them before. It can greatly reduce the size of your PDF document. Optimizing the images is useful when you want to insert images that are made using the camera of your device. To optimize images you can use Expo ImageManipulator. Notice: This plugin works only with images placed locally on the device. You need to download the image first. Example of function, that optimizes large images: import * as ImageManipulator from 'expo-image-manipulator'; .... const optimizeImage = async (imageUri) => { try { const { uri } = await ImageManipulator.manipulateAsync(imageUri, [{ resize: { width: 600, }, },], { compress: 0.1 },); return uri; } catch (error) { console.log(error); return imageUri; } }; upplabs/upplabs-pdf-example upplabs/upplabs-pdf- We hope this article was useful! You can read more about mobile development on our website and ask UppLabs for help with developing your mobile app. I am using react-native-pdf in my project , It is working fine on Android simulator but not working on android device . I have found few warnings on command prompt while running . warn The following packages use deprecated "rnpm" config that will stop working from next release: on mobile it is showing error "Error : CLEARTEXT communication to ********.in not permitted by network security policy " PDF viewer for React Native. Implemented with platform native render functions for smallest deploy size impact, and restricted feature set to simplify integration with larger systems. Includes prefabricated full document viewer based on FlatList and a single page render component to use as a building block for your own fully custom viewer. Uses android.graphics.pdf.PdfRenderer on Android and CGPDFDocument on iOS. Unlike many native components in the wild, react-native-pdf-light provides full implementation of React Native shadow nodes. This simplifies UI development, since the component actually knows its own dimensions. Version 2.x, adds render support for annotations conforming the PAS standard v1. If annotation support is not needed, use 1.x for slightly reduced size. npm install react-native-pdf-light If iOS build fails with Undefined symbol: __swift_FORCE_LOAD_..., add an empty .swift file to the xcode project. import { Pdf, PdfUtil } from 'react-native-pdf-light'; PdfUtil.getPageCount(source).then(console.log); If creating your own custom viewer to manage pages: import { PdfView } from 'react-native-pdf-light'; For zoom support, install react-native-gesture-handler and use deep import. Note that react-native-gesture-handler is only required to use zoom features; the other display options do not have any dependencies. import { ZoomPdfView } from 'react-native-pdf-light/Zoom'; Props: annotation: string Optional: Path to annotation data. File assumed to be PAS v1. annotationStr: string Optional: Stringified JSON of annotation data. Data assumed to be PAS v1. onError: (error: Error) => void Optional: Callback to handle errors. onLoadComplete: (numberOfPages: number) => void Optional: Callback to handle pdf load completion. Passed the page count of the loaded pdf. onMeasurePages: (measurements: { itemHeight: number, offset: number }[]) => void Optional: Callback to receive layout details of all pages. shrinkToFit: 'never' | 'portrait' | 'landscape' | 'always' Optional: Size pages such that each page can be displayed without cutoff. Applies when device is in the specified orientation. source: string The following props are forwarded to the underlying FlatList component: initialScrollIndex ListEmptyComponent onMomentumScrollBegin onMomentumScrollEnd onScroll onScrollBeginDrag onScrollEndDrag refreshControl scrollEventThrottle Methods: scrollToIndex(index: number): void Scroll to the specified page (0-indexed). scrollToOffset(offset: number): void Scroll to the specified offset. Props: annotation: string Optional: Path to annotation data. File assumed to be PAS v1. annotationStr: string Optional: Stringified JSON of annotation data. Data assumed to be PAS v1. page: number Page (0-indexed) of document to display. resizeMode: 'contain' | 'fitWidth' Optional: How pdf page should be scaled to fit in view dimensions. source: string style: ViewStyle Optional: View stylesheet. Zoom interactions compatible with react-native-pager-view in horizontal display mode. Props: All props from PdfView onZoomIn: () => void Optional: Callback when view starts to zoom. onZoomReset: () => void Optional: Callback when view returns to non-zoomed state. maximumZoom: number Optional: Maximum allowed zoom. Default 2. refreshControl: Optional: A RefreshControl component, used to provide pull-to-refresh functionality for the internal ScrollView. PdfUtil.getPageCount(source: string): Promise Get the number of pages of a pdf. PdfUtil.getPageSizes(source: string): Promise Get the dimensions of every page. On Android API level < 26 when directly rendering pages with PdfView at a non-default aspect ratio (e.g. setting both width and height of the view such that the view's aspect ratio does not match the pdf page's aspect ratio) if a page in the pdf is cropped or rotated, the page may render in the wrong position. This is due to a bug in the native android.graphics.pdf.PdfRenderer. (If you are aware of a fix, pull requests welcome.) See the contributing guide to learn how to contribute to the repository and the development workflow. MIT The Portable Document Format (PDF) is one of the most popular formats used for documents. In this blog post, let's check how to show pdf files in react native.React Native Pdf is a third party that helps us to open and read pdf files from a URL or local assets in react native. It has important features such as drag and zoom, password-protected pdf support, etc. In short, you can create a pdf viewer with this library.To make React Native Pdf work you also need another library react native blob util. Install both libraries using the command given below.npm install react-native-pdf react-native-blob-util --saveAlso, don't forget to run the pod install command inside the ios folder.On Android, open android/app/build.gradle and add the following inside the android {} tag.packagingOptions { pickFirst 'lib/x86/libc++_shared.so' pickFirst 'lib/x86_64/libjsc.so' pickFirst 'lib/arm64-v8a/libjsc.so' pickFirst 'lib/arm64-v8a/libc++_shared.so' pickFirst 'lib/x86_64/libc++_shared.so' pickFirst 'lib/armeabi-v7a/libc++_shared.so' }Now, you can create a simple pdf viewer in react native as given below.import React from 'react'; import {StyleSheet, Dimensions, View, Platform} from 'react-native'; import Pdf from 'react-native-pdf'; const source = { uri: ' , cache: true, }; const App = () => { return ({ console.log(`number of pages: ${numberOfPages}`); }} onPageChanged={(page, numberOfPages) => { console.log(`current page: ${page}`); }} onError={error => { console.log(error); }} onPressLink={uri => { console.log(`Link presse: ${uri}`); }} style={styles.pdf} />); }; export default App; const styles = StyleSheet.create({ container: { flex: 1, justifyContent: 'flex-start', alignItems: 'center', marginTop: 25, }, pdf: { flex: 1, width: Dimensions.get('window').width, height: Dimensions.get('window').height, }, }); This 'Pdf' component displays the PDF file and provides callbacks for various events, such as when the PDF file is completely loaded, when a page is changed, when an error occurs, and when a link is pressed.The output of this react native pdf viewer example on Android is as given below.Following is the output in iOS.This is just a basic example and you can go through the official documentation of react native pdf library to do advanced things. I remember how was surprised when attempted to open a link to a PDF file on an Android device in my past React Native project. As usual in the RN world, everything was good on iOS but… on Android, I've only seen the message about successfully downloading the document to the file system. Great! I've researched a little bit and found that this is not a bug of react-native-webview (the component for rendering web pages) itself but that is how the Android browser works. Ok, let's find out how to fix it.Proposed SolutionA possible workaround is to parse the web page and redirect links to PDF files to a separate screen. This screen will render the PDF file using the react-native-pdf component.ImplementationAssuming that react-navigation and react-native-webview are already in use in your project, the first step is to follow the installation guide for react-native-pdf.Next, set up navigation to display web pages and PDF files on separate screens. You can do this in the following files:src/navigation/MainStack.tsxexport const MainStackNavigator = () => { return (({ title: route.params?.title })} /> ({ title: route.params?.title })} />);};src/screens/pdf/PdfScreen.tsxexport const PdfScreen = ({ route }: Props) => { useEffect(() => { console.log("PdfScreen", route.params.url); }, [route.params.url]); return ( );};*Note the line "trustAllCerts={Platform.OS === "ios"}" without this pdf won't be downloaded on Android, here is the issuesrc/screens/pdf/WebScreen.tsxexport const WebScreen = ({ route }: Props) => { return ( );}; src/screens/Home/HomeScreen.tsxexport const HomeScreen = ({ navigation }: Props) => { const onPdfLinkPress = useCallback(() => { navigation.navigate(screenNames.PdfScreen, { title: "dummy.pdf", url: " , }); }, [navigation]); const onWebLinkPress = useCallback(() => { navigation.navigate(screenNames.WebScreen, { title: "GitHub", url: " ", }); }, [navigation]); const onWebLinkWithPdfLinksPress = useCallback(() => { navigation.navigate(screenNames.WebScreen, { title: "Link to dummy.pdf", url: " }); }, [navigation]); return (Open web page Open pfd Open web page with links to pdf files);};At this point, if you tap on the "dummy.pdf" link in a Google page, it will open the PDF inside the WebView on iOS and download the file on Android.To improve this, let's update code of WebScreenexport const WebScreen = ({ navigation, route }: Props) => { const jsCode = useMemo(() => ` (function() { try { const links = document.getElementsByTagName("a"); for (let link of links) { const linkExt = link.href.split(".").pop().toLowerCase(); if (linkExt === "pdf") { link.onclick = () => { window.ReactNativeWebView.postMessage(JSON.stringify({ pdfLink: link.href })); return false; }; } } } catch (e) { window.ReactNativeWebView.postMessage(JSON.stringify({ error: e.message })); console.log(e); } })(); `, [],); const onMessage = useCallback((event: WebViewMessageEvent) => { const data = JSON.parse(event.nativeEvent.data); if (data.pdfLink) { navigation.navigate(screenNames.PdfScreen, { title: data.pdfLink.split("/").pop(), url: data.pdfLink, }); } }, [navigation],); return ( );};Here we are injecting JavaScript code into our WebView that adds an onClick handler to all links to PDF files on a page. After clicking on those links we send this PDF link from the WebView JavaScript context to React Native context and then we use it to redirect the user to a separate screen where we are rendering the PDF document properly.Now, when you tap on the "dummy.pdf" link in a Google page, the PDF file will open in a separate PdfScreen.ConclusionThis solution should help you handle PDF links in WebView in your React Native project. Hope it saves you some time in the future, happy coding:)PS. The code example is available here. In today's mobile landscape, the ability to display PDF documents within your application is crucial for many use cases. Whether it's displaying reports, invoices, ebooks, or any other document format, users expect a seamless experience. This blog post provides a deep dive into using react-native-pdf – a popular and versatile library for rendering PDFs in React Native applications.What is react-native-pdf?react-native-pdf is a React Native component that allows you to display PDF documents within your mobile application. It supports various features like:Local and Remote PDFs: Load PDFs from local file paths or URLs.Zooming and Panning: Allows users to zoom in and pan around the document for better readability.Pagination: Navigate through the PDF document page by page.Error Handling: Gracefully handle errors during PDF loading.Customization: Offers a range of props to customize the appearance and behavior of the viewer.Why Use react-native-pdf?Cross-Platform Compatibility: Works seamlessly on both iOS and Android platforms.Easy Integration: Simple and straightforward installation and usage.Highly Customizable: Adapt the viewer to match your application's design.Active Community: Benefit from a large community and regular updates.InstallationFirst, install the react-native-pdf package using npm or yarn:npm install react-native-pdf --save yarn add react-native-pdf After installation, you might need to link the native dependencies. For React Native versions 0.60 and above, auto-linking should handle this automatically. However, if you encounter issues, you can manually link them:react-native link react-native-pdf Important Note: If you are using Expo, the react-native-pdf library requires you to eject to the bare workflow as it utilizes native modules.Basic UsageHere's a basic example of how to display a PDF from a URL:import React from 'react'; import { StyleSheet, View, Dimensions } from 'react-native'; import Pdf from 'react-native-pdf'; const App = () => { const source = { uri: ' , cache: true, }; return ( ); }; const styles = StyleSheet.create({ container: { flex: 1, justifyContent: 'flex-start', alignItems: 'center', marginTop: 25, }, pdf: { flex: 1, width: Dimensions.get('window').width, height: Dimensions.get('window').height, }, }); export default App; Explanation:import Pdf from 'react-native-pdf';: Imports the Pdf component from the library.source object: Defines the source of the PDF. In this example, it's a remote PDF. You can also use a local file path or a bundle asset. The cache: true option tells the component to cache the PDF. component: Renders the PDF document. The source prop specifies the PDF source, and the style prop defines the styling.Loading Local PDFsTo load a PDF from your local file system, you can modify the source object. Here's an example:const source = require('./assets/my_document.pdf'); Important: Ensure that your PDF file is placed within your project's assets folder or any other location that your React Native application can access. For iOS, you might need to add the PDF to your Xcode project.Customizing the PDF ViewerThe react-native-pdf component provides various props for customizing its appearance and behavior. Here are some commonly used props:style: Applies styles to the PDF viewer container.source: Specifies the PDF source (URL, local file path, or base64 string).scale: Sets the initial zoom scale.minScale: Sets the minimum zoom scale.maxScale: Sets the maximum zoom scale.horizontal: Enables horizontal scrolling (pagination). Defaults to false for vertical scrolling.enablePaging: Enables page-by-page navigation (only effective when horizontal is true).fitPolicy: Determines how the PDF should be scaled to fit the container. Options are 'width', 'height', 'both'.onLoadComplete: Callback function triggered when the PDF is successfully loaded.onError: Callback function triggered when an error occurs during PDF loading.onPageChanged: Callback function triggered when the page changes.onPressLink: Callback function triggered when a link in the PDF is pressed.Example with Customization:import React, { useState } from 'react'; import { StyleSheet, View, Dimensions, Text } from 'react-native'; import Pdf from 'react-native-pdf'; const App = () => { const [currentPage, setCurrentPage] = useState(1); const [pageCount, setPageCount] = useState(0); const source = { uri: ' , cache: true, }; const handleLoadComplete = (numberOfPages, filePath) => { console.log(`PDF loaded successfully with ${numberOfPages} pages`); setPageCount(numberOfPages); }; const handleError = (error) => { console.error('Error loading PDF:', error); }; const handlePageChanged = (page, numberOfPages) => { console.log(`Current page: ${page}`); setCurrentPage(page); }; return ({currentPage} / {pageCount}); }; const styles = StyleSheet.create({ container: { flex: 1, justifyContent: 'flex-start', alignItems: 'center', marginTop: 25, }, pdf: { flex: 1, width: Dimensions.get('window').width, height: Dimensions.get('window').height - 50, }, pagination: { position: 'absolute', bottom: 10, left: 0, right: 0, alignItems: 'center', }, }); export default App; In this example:We added state variables (currentPage, pageCount) to track the current page and the total number of pages.We implemented onLoadComplete, onError, and onPageChanged callback functions to handle loading events and page changes.We enabled horizontal paging (horizontal={true} and enablePaging={true}).We set minScale and maxScale to control the zoom levels.We used fitPolicy="width" to fit the PDF width to the container.We added a simple pagination display at the bottom.Troubleshooting Common IssuesPDF Not Loading: Double-check the PDF source URL or file path. Ensure that the PDF file exists and is accessible. Verify network connectivity if you're loading a remote PDF. Also, make sure the PDF is a valid PDF file.Blank Screen: This can happen if the PDF is corrupted or if the styling is incorrect. Try setting explicit width and height values for the PDF style. Also, check the console for any error messages.Scaling Issues: Experiment with the minScale, maxScale, and fitPolicy props to adjust the scaling behavior.iOS Issues with Local Files: Make sure the PDF is included in your Xcode project's bundle resources. Right-click on your project in Xcode -> "Add Files to [Your Project Name]" and select the PDF. Ensure "Copy items if needed" is checked.Best PracticesError Handling: Always implement error handling to gracefully handle situations where the PDF fails to load.Loading Indicators: Provide a loading indicator while the PDF is loading to improve the user experience.Caching: Use the cache: true option to cache remote PDFs for faster loading on subsequent visits.Optimize PDF Size: Optimize your PDF files to reduce their size, especially for mobile devices. Smaller PDFs load faster and consume less bandwidth.Security: Be mindful of PDF security, especially when handling sensitive information. Consider using encryption or password protection if necessary.Advanced UsageBase64 Encoded PDFs: You can display PDFs from base64 encoded strings.const source = { uri: 'data:application/pdf;base64,...BASE64_ENCODED_STRING...' }; Programmatic Control: You can use the react-native-pdf's forwardRef to control the component programmatically. This allows you to programmatically scroll to a specific page, zoom in/out, etc. (Refer to the library's documentation for specific methods).Conclusionreact-native-pdf is a powerful and versatile library for displaying PDFs in your React Native applications. By following the steps outlined in this guide, you can easily integrate PDF viewing functionality into your mobile apps and customize the experience to meet your specific needs. Remember to handle errors, optimize PDF size, and consider security best practices for a seamless and secure user experience. Experiment with different props and configurations to create the perfect PDF viewer for your application! Good luck! #935 In wonday/react-native-pdf;#934 In wonday/react-native-pdf;#933 In wonday/react-native-pdf;#932 In wonday/react-native-pdf;#931 In wonday/react-native-pdf;#930 In wonday/react-native-pdf;#929 In wonday/react-native-pdf;#925 In wonday/react-native-pdf;#924 In wonday/react-native-pdf;#921 In wonday/react-native-pdf; You can't perform that action at this time. This tutorial compares two approaches for opening PDFs in React Native: react-native-pdf (free, open source) provides basic viewing with simple scrolling functionality, while Nutrient React Native SDK (commercial) offers advanced features like annotations, form filling, and document outline support. Choose react-native-pdf for simple viewing needs or Nutrient for applications requiring comprehensive document interaction capabilities. React Native lets you create mobile apps in JavaScript, but instead of building a cross-platform app, you use the same UI building blocks as regular iOS and Android apps. In this post, you'll use React Native to build an app that opens a PDF with the press of a button. Additionally, we'll address common issues related to the compatibility of React Native Expo with native modules when integrating PDF functionality. In the first part of this tutorial, you'll use wonday's react-native-pdf library to open a PDF in an app. In the second part, you'll learn how to view PDFs using the Nutrient React Native PDF library. Introduction to React Native PDF viewer A React Native PDF viewer is a crucial component for any mobile application that needs to display PDF files. It allows users to view and interact with PDF documents directly within the app, providing a seamless and integrated experience. With the growing demand for mobile apps that can handle PDF files, incorporating a React Native PDF viewer has become a popular choice among developers. React Native PDF viewers come with a variety of features that enhance the user experience. These include zooming, scrolling, and page navigation, making it easy for users to read and interact with PDF documents. Additionally, they support various PDF formats, including PDF 1.7 and later versions, ensuring compatibility with a wide range of documents. Some of the popular React Native PDF viewers include react-native-pdf, [react-native-pdf-view][] and Nutrient React Native PDF library. Each of these libraries offers unique features and capabilities, allowing developers to choose the one that best fits their needs. When selecting a React Native PDF viewer, it's important to consider factors such as performance, compatibility, and customization options to ensure it meets the specific requirements of your app. Choosing a React Native PDF library Selecting the right React Native PDF library is essential for ensuring a smooth and efficient user experience. Here are some key factors to consider when choosing a React Native PDF library: Performance — Look for a library that provides fast and smooth rendering of PDF files. This is crucial for maintaining a responsive and user-friendly app. Compatibility — Ensure that the library is compatible with your version of React Native and supports the required PDF formats. This will help avoid any integration issues. Customization — Choose a library that offers customization options to fit your app's design and functionality. This allows you to tailor the PDF viewer to match your app's aesthetic and user experience. Security — Consider a library that provides robust security features to protect sensitive PDF data. This is especially important for apps that handle confidential or personal information. Community support — Opt for a library with an active community and good documentation. This can be invaluable for troubleshooting and getting the most out of the library. Some popular React Native PDF libraries include: react-native-pdf — A widely used library that supports React Native 0.60 and above, known for its reliability and extensive features. [react-native-pdf-view][] — A lightweight library that provides fast and smooth rendering of PDF files, ideal for simpler use cases. Nutrient React Native library: A powerful, commercial-grade PDF SDK offering advanced functionalities, such as form filling, annotation, digital signatures, and real-time collaboration. Nutrient integrates smoothly with React Native, making it an ideal choice for projects requiring extensive PDF manipulation capabilities. By considering these factors, you can choose the best React Native PDF library for your project, ensuring a high-quality user experience. Opening a PDF in React Native with react-native-pdf Below are the steps for how to open a PDF in React Native with react-native-pdf. Step 1 — Installing the prerequisites You'll use yarn to install packages. If you haven't yet installed it, please follow the Yarn installation guide to set it up on your system. To create React Native apps from the command line, you also need to install Node.js and Watchman using Homebrew: brew install node brew install watchman Then download and install Android Studio and configure it following instructions from the official React Native guide. Windows users To manage Node.js versions, you can install Chocolatey, a popular package manager for Windows. It's recommended to use a long-term support (LTS) version of Node. If you want to be able to switch between different versions, you might want to install Node via nvm-windows, a Node version manager for Windows. React Native also requires the Java Development Kit (JDK), which can be installed using Chocolatey as well. To do this, open an administrator command prompt by right-clicking Command Prompt and selecting Run as Administrator. Then, run the following command: choco install -y nodejs-lts openjdk11 If you're using the latest version of the JDK, you'll need to change the Gradle version of your project so it can recognize the JDK. You can do that by going to {project root folder}/android/gradle/wrapper/gradle-wrapper.properties and changing the distributionUrl value to upgrade the Gradle version. Step 2 — Creating a new React Native app You can use react-native to create a new React Native app from the command line. This example uses the name OpeningaPDF for the app: npx react-native init OpeningaPDF npx is the npm package runner. You can read more about it here. For the rest of the tutorial, you'll work in OpeningaPDF: Step 3 — Installing Dependencies You'll use react-navigation components so that you can switch from one view to another in your app: yarn add @react-navigation/native @react-navigation/native-stack react-native-screens react-native-safe-area-context Next, add react-native-pdf, which includes a Pdf component: yarn add react-native-pdfStep 4 — Downloading a PDF document You can download a sample PDF document: curl -o Document.pdf Don't forget to move the document to the required folder after downloading it. For iOS Move or copy the document to ios/Document.pdf. Open the iOS project ios/OpeningaPDF.xcworkspace in Xcode and add the document to root of the OpeningaPDF project: For Android Move or copy the document to android/app/src/main/assets/Document.pdf: cp Document.pdf android/app/src/main/assetsStep 5 — Writing the app Now you can start implementing your app. First, import all the required packages and initialize your navigation stack in App.js: import React from 'react'; import Pdf from 'react-native-pdf'; import { Dimensions, StyleSheet, View, Button, Platform, } from 'react-native'; import { NavigationContainer } from '@react-navigation/native'; import { createNativeStackNavigator } from '@react-navigation/native-stack'; const DOCUMENT = Platform.OS === 'ios' ? require('./Document.pdf') : 'file:///android_asset/Document.pdf'; const PdfScreen = () => { return ; }; const HomeScreen = ({ navigation }) => { return (navigation.navigate('Pdf')} title="Open PDF" />); }; const Stack = createNativeStackNavigator(); const App = () => { return ( ); }; export default App; const styles = StyleSheet.create({ container: { flex: 1, justifyContent: 'center', alignItems: 'center', }, pdf: { flex: 1, width: Dimensions.get('window').width, }, }); HomeScreen contains an Open PDF button that navigates to PdfScreen. You need to put a Document.pdf file into the same path as App.js so that PdfScreen can show it. Next, define your App, which renders your navigation stack: const App = () => { return ( ); }; export default App; At the end of App.js, define your styles: const styles = StyleSheet.create({ container: { flex: 1, justifyContent: 'center', alignItems: 'center', }, pdf: { flex: 1, width: Dimensions.get('window').width, }, }); Center the Open PDF button and allow the PDF to fill the entire screen. The image below shows how it looks on Android. You can find the complete content of App.js on GitHub. Now, you can tap a button and scroll through a PDF document. However, you can't do anything else; there's no zooming, and there are no annotations. You only have the scrolling view mode. But that's where Nutrient comes in: It includes all of these features — and more — without you having to configure anything. Opening a PDF with Nutrient React Native PDF library To start, follow the integration guide for iOS and Android. Then, add a second button to HomeScreen that opens a PDF with Nutrient: var PSPDFKit = NativeModules.PSPDFKit; PSPDFKit.setLicenseKey(null); // Or your valid license keys using `setLicenseKeys`. const DOCUMENT = Platform.OS === 'ios' ? require('./Document.pdf') : { uri: 'bundle-assets://Document.pdf' }; // Simple screen containing an Open PDF button. class HomeScreen extends Component { _presentPSPDFKit() { PSPDFKit.present(DOCUMENT, { pageTransition: 'scrollContinuous', scrollDirection: 'vertical', }); } static navigationOptions = { title: 'Home', }; render() { const { navigate } = this.props.navigation; return (navigate('Pdf')} title="Open PDF with react-native-pdf" />); } } All you need is PSPDFKit.present('document.pdf') and you can view a PDF in Nutrient. Not only that, but you can also zoom, create annotations, look at the document's outline, and lots of other things. Additionally, you can customize the PDF viewer by passing a dictionary to PSPDFKit.present. Now, here's the React Native app powered by Nutrient, as seen on iOS. And here's the same thing, only on Android. You can find the source code for the entire project on GitHub. Troubleshooting common issues When working with React Native PDF viewers, you may encounter some common issues. Here are some troubleshooting tips to help you resolve them: PDF file not loading — Ensure that the PDF file path is correct and that the file is not corrupted or damaged. Double-check the file location and format to ensure it is accessible. PDF viewer not rendering — Verify that your React Native version is compatible with the PDF viewer library. Ensure that the library is properly configured and initialized in your project. PDF viewer crashing — Check the device's memory and storage capacity to ensure it can handle the PDF file size. Look for any memory leaks or performance issues in the PDF viewer. PDF viewer not responding — Ensure that the app's UI thread is not blocked or frozen. Verify that the PDF viewer is properly configured and initialized, and check for any performance bottlenecks. By following these troubleshooting tips, you can quickly identify and resolve common issues with your React Native PDF viewer, ensuring a smooth and seamless experience for your users. Conclusion As you saw, adding PDF support to your app with the react-native-pdf package is a breeze. However, if your React Native project requires more advanced PDF functionality, such as PDF annotations or PDF form filling, you should definitely look into using a commercial library. Our React Native PDF SDK comes with more than 30 out-of-the-box features and has well-documented APIs to handle advanced use cases. Try out our PDF library using our free trial, and check out our demos to see what's possible. Nutrient also comes with great customer support, so please reach out to us if you have any questions about our React Native integration. FAQ How can I open a PDF in React Native? You can open a PDF in React Native by setting up a React Native project, installing the react-native-pdf library or Nutrient, adding a PDF document to your project, and implementing a component to display the PDF. What are the steps to set up a React Native PDF viewer? To set up a React Native PDF viewer, create a new React Native app, install necessary dependencies (react-native-pdf or Nutrient), add a PDF document to the project, and write code to render the PDF in a component. You can't perform that action at this time. Library for displaying PDF documents in react-native android - uses Android PdfViewer. Targets minSdkVersion 21 (required by setClipToOutline) and above. By default stable version 2.8.2 is used. It's also possible to override it and use 3.1.0-beta.1 (this version allows to handle links, etc. and will be used when Android PdfViewer stable version is released). To change the version, define it in your build file: buildscript { ext { ... pdfViewerVersion = "3.1.0-beta.1" } ... } Barteksc PdfViewer uses JCenter, which should be read-only indefinitely, but in case the host project does not use it, there is a possibility to use mhiew/AndroidPdfViewer which is a fork published on mavenCentral. To use it, define the following configuration in your gradle script: buildscript { ext { ... pdfViewerVersion = "3.2.0-beta.1" pdfViewerRepo = "com.github.mhiew" } ... } ios - uses WKWebView. Targets iOS9.0 and above zero NPM dependencies Getting started $ npm install react-native-view-pdf --save Linking From RN 0.60 there is no need to link - Native Modules are now Autolinked Mostly automatic installation $ react-native link If it doesn't work follow the official react native documentation Using CocoaPods In your Xcode project directory open Podfile and add the following line: pod 'RNPDF', :path => '../node_modules/react-native-view-pdf' And install: pod install Android Open up android/app/src/main/java/[...]/MainApplication.java Add import com.rumax.reactnative.pdfviewer.PDFViewPackage; to the imports at the top of the file Add new PDFViewPackage() to the list returned by the getPackages() method Append the following lines to android/settings.gradle: include ':react-native-view-pdf' project(':react-native-view-pdf').projectDir = new File(rootProject.projectDir, '../node_modules/react-native-view-pdf/android') Insert the following lines inside the dependencies block in android/app/build.gradle: compile project(':react-native-view-pdf') Note for Android The Android project tries to retrieve the following properties: compileSdkVersion buildToolsVersion minSdkVersion targetSdkVersion from the ext object if you have one defined in your Android's project root (you can read more about it here). If not, it falls back to its current versions (check the gradle file for additional information). Windows ReactWindows N/A Demo Android iOS Quick Start // With Flow type annotations (import PDFView from 'react-native-view-pdf'; // Without Flow type annotations // import PDFView from 'react-native-view-pdf/lib/index'; const resources = { file: Platform.OS === 'ios' ? 'downloadedDocument.pdf' : '/sdcard/Download/downloadedDocument.pdf', url: ' , base64: 'JVBERi0xLjMKJcfs...', }; export default class App extends React.Component { render() { const resourceType = 'url'; return ({/* Some Controls to change PDF resource */} console.log(`PDF rendered from ${resourceType}`)} onError={(error) => console.log('Cannot render PDF', error)} />); } } Use the demo project to: Test the component on both android and iOS Render PDF using URL, BASE64 data or local file Handle error state Props Name Android iOS Description resource ✓ ✓ A resource to render. It's possible to render PDF from file, url (should be encoded) or base64 resourceType ✓ ✓ Should correspond to resource and can be: file, url or base64 fileFrom ✗ ✓ iOS ONLY: In case if resourceType is set to file, there are different way to search for it on iOS file system. Currently documentsDirectory, libraryDirectory, cachesDirectory, tempDirectory and bundle are supported. onLoad ✓ ✓ Callback that is triggered when loading is completed onError ✓ ✓ Callback that is triggered when loading has failed. And error is provided as a function parameter style ✓ ✓ A style fadeInDuration ✓ ✓ Fade in duration (in ms, defaults to 0.0) to smoothly fade the webview into view when pdf loading is completed enableAnnotations ✓ ✗ Android ONLY: Boolean to enable Android view annotations (default is false). urlProps ✓ ✓ Extended properties for url type that allows to specify HTTP Method, HTTP headers and HTTP body onPageChanged ✓ ✗ Callback that is invoked when page is changed. Provides active page and total pages information onScrolled ✓ ✓ Callback that is invoked when PDF is scrolled. Provides offset value in a range between 0 and 1. Currently only 0 and 1 are supported. Methods reload Allows to reload the PDF document. This can be useful in such cases as network issues, document is expired, etc. To reload the document you will need a ref to the component: ... render() { return (this._pdfRed = ref} />); } And trigger it by calling reloadPDF: reloadPDF = async () => { const pdfRef = this._pdfRef; if (!pdfRef) { return; } try { await pdfRef.reload(); } catch (err) { console.err(err.message); } } Or check the demo project which also includes this functionality. Development tips On android for the file type it is required to request permissions to read/write. You can get more information in PermissionsAndroid section from react native or Request App Permissions from android documentation. Demo project provides an example how to implement it using Java, check the MainActivity.java and AndroidManifest.xml files. Before trying file type in demo project, open sdcard/Download folder in Device File Explorer and store some downloadedDocument.pdf document that you want to render. On iOS, when using resource file you can specify where to look for the file with fileFrom. If you do not pass any value, the component will lookup in two places. First, it will attempt to locate the file in the Bundle. If it cannot locate it there, it will search the Documents directory. For more information on the iOS filesystem access at runtime of an application please refer the official documentation. Note here that the resource will always need to be a relative path from the Documents directory for example and also do NOT put the scheme in the path (so no file://.....). You can find an example of both usage of the Bundle and Documents directory for rendering a pdf from file on iOS in the demo project. In demo project you can also run the simple server to serve PDF file. To do this navigate to demo/utils/ and start the server node server.js. (Do not forget to set proper IP adress of the server in demo/App.js) License MIT Authors Other information -keep class com.shockwave.** ❤ 6.7.7 • Public • Published 4 months ago ReadmeCode Beta2 Dependencies37 Dependents96 Versions A react native PDF view component (cross-platform support) read a PDF from url, blob, local file or asset and can cache it. display horizontally or vertically drag and zoom double tap for zoom support password protected pdf jump to a specific page in the pdf We use react-native-blob-util to handle file system access in this package, So you should install react-native-pdf and react-native-blob-util The table below shows the supported versions of React Native and react-native-blob-util for different versions of react-native-pdf. React Native 0.4x - 0.56 0.57 0.60+ 0.62+ 0.62+ react-native-pdf 4.x.x - 5.0.x 5.0.9+ 6.0.0+ 6.2.0+ 6.4.0+ react-native-blob-util 0.13.7+ Expo: This package is not available in the Expo Go app. Learn how you can use this package in Custom Dev Clients via the out-of-tree Expo Config Plugin. Example: with-pdf. # Using npm npm install react-native-pdf react-native-blob-util --save # or using yarn: yarn add react-native-pdf react-native-blob-util Then follow the instructions for your platform to link react-native-pdf into your project: iOS details React Native 0.60 and above Run pod install in the ios directory. Linking is not required in React Native 0.60 and above. React Native 0.59 and below react-native link react-native-blob-util react-native link react-native-pdf Android details If you use RN 0.59.0 and above, please add following to your android/app/build.gradle** android { + packagingOptions { + pickFirst 'lib/x86/libc++_shared.so' + pickFirst 'lib/x86_64/libjsc.so' + pickFirst 'lib/arm64-v8a/libjsc.so' + pickFirst 'lib/arm64-v8a/libc++_shared.so' + pickFirst 'lib/x86_64/libc++_shared.so' + pickFirst 'lib/armeabi-v7a/libc++_shared.so' + } } React Native 0.59.0 and below react-native link react-native-blob-util react-native link react-native-pdf Windows details Open your solution in Visual Studio 2019 (eg. windows\yourapp.sln) Right-click Solution icon in Solution Explorer > Add > Existing Project... If running RNW 0.62: add node_modules\react-native-pdf\windows\RCTPdf\RCTPdf.vcxproj If running RNW 0.62: add node_modules\react-native-blob-util\windows\ReactNativeBlobUtil\ReactNativeBlobUtil.vcxproj Right-click main application project > Add > Reference... Select progress-view and in Solution Projects If running 0.62, also select RCTPdf and ReactNativeBlobUtil In app pch.h add #include "winrt/RCTPdf.h" If running 0.62, also select #include "winrt/ReactNativeBlobUtil.h" In App.cpp add PackageProviders().Append(winrt::progress_view::ReactPackageProvider()); before InitializeComponent(); If running RNW 0.62, also add PackageProviders().Append(winrt::RCTPdf::ReactPackageProvider()); and PackageProviders().Append(winrt::ReactNativeBlobUtil::ReactPackageProvider()); To add a test.pdf like in the example add: true in the app .vcxproj file, before . FAQ details Q1. After installation and running, I can not see the pdf file. A1: maybe you forgot to excute react-native link or it does not run correctly. You can add it manually. For detail you can see the issue #24 and #2 Q2. When running, it shows 'Pdf' has no propType for native prop RCTPdf.acessibilityLabel of native type 'String' A2. Your react-native version is too old, please upgrade it to 0.47.0+ see also #39 Q3. When I run the example app I get a white/gray screen / the loading bar isn't progressing . A3. Check your uri, if you hit a pdf that is hosted on a http you will need to do the following: iOS: add an exception for the server hosting the pdf in the ios info.plist. Here is an example : NSAppTransportSecurity NSExceptionDomains yourserver.com NSIncludesSubdomains NSTemporaryExceptionAllowsInsecureHTTPLoads NSTemporaryExceptionMinimumTLSVersion TLSv1.1 Android: see here Q4. why doesn't it work with react native expo?. A4. Expo does not support native module. you can read more expo caveats here Q5. Why can't I run the iOS example? 'Failed to build iOS project. We ran "xcodebuild" command but it exited with error code 65.' A5. Run the following commands in the project folder (e.g. react-native-pdf/example) to ensure that all dependencies are available: yarn install (or npm install) cd ios pod install cd .. react-native run-ios ChangeLog details v6.7.7 1. Added: add support for customizable scroll indicators in PdfView component (#904) 2. Fixed: fix field values not being visible on android. issue #864 (#896) v6.7.6 Fixed: Add missing 'enableDoubleTapZoom' to fabric codegen source (#832) Fixed: added missing 'scrollEnabled' prop (#842) Fixed: java.lang.IllegalStateException: Tried to access a JS module before the React instance was fully set up (#840) Fixed: an issue that crashes when cancel is not present (#852) Added: add load method (#861) Fixed: encoded accented character is decoded incorrectly (#873) Fixed: enableDoubleTapZoom bugfix v6.7.5 Added progressContainerStyle prop Improved: Added enableDoubleTapZoom option Fixed: Fix app crash with this.lastRNBFTask.cancel is not a function (#827) Fixed: Remove override to fix 'no matching methods to override' error (#822) v6.7.4 Fixed: fix Android crash issue v6.7.3 Fixed: fix android package name v6.7.2 Fixed: fix iOS double tap zoom scrolling Fixed: fix RN 73 compatibility Fixed: bump crypto-js to avoid critical vulnerability v6.7.1 Fixed: fix ios project setting Fixed: fix typo in RNPDFPdfViewManagerInterface interface causing android build error v6.7.0 Fixed: fix(iOS): center page at tap point after double tap to zoom Fixed: add PDFKit to podspec to make ios compile Improved: Update build.gradle to support RN 0.71 on new arch Fixed: fix some small bugs and documents. v6.6.2 Fixed: Migrate to ViewPropTypes exported from 'deprecated-react-native-prop-types' Added: Decode File Path for iOS Improved: prefer current page for calculating scale factor on fit v6.6.1 depresed v6.6.0 depresed Fixed: Migrate to ViewPropTypes exported from 'deprecated-react-native-prop-types' Added: Decode File Path for iOS Improved: prefer current page for calculating scale factor on fit Improved: Typescript version source v6.5.0 Fix: replace mavenCentral with maven Breaking Change(Android): replace deprecated repository: jcenter() Fix: loading progress Add: Typed "source" prop Remove: dependency to fbjs v6.4.0 Remove sample for reducing NPM package size Add support for setting a filename for the cached pdf file Use react-native-blob-util instead of rn-fetch-blob Add blob support remove progress-view dependency v6.3.0 Add windows support Fixed some bugs [more] /** * Copyright (c) 2017-present, Wonday (@wonday.org) * All rights reserved. * * This source code is licensed under the MIT-style license found in the * LICENSE file in the root directory of this source tree. */ import React from 'react'; import { StyleSheet, Dimensions, View } from 'react-native'; import Pdf from 'react-native-pdf'; export default class PDFExample extends React.Component { render() { const source = { uri: ' , cache: true }; //const source = require('./test.pdf'); // ios only //const source = {uri:'bundle-assets://test.pdf' }; //const source = {uri:'file:///sdcard/test.pdf'}; //const source = {uri:"data:application/pdf;base64,JVBERi0xLjcKJc..."}; //const source = {uri:"content://com.example.blobs/xxxxxxxx-...?offset=0&size=xxx"}; //const source = {uri:"blob:xxxxxxxx-...?offset=0&size=xxx"}; return ({ console.log(`Number of pages: ${numberOfPages}`); }} onPageChanged={(page,numberOfPages) => { console.log(`Current page: ${page}`); }} onError={(error) => { console.log(error); }} onPressLink={(uri) => { console.log(`Link pressed: ${uri}`); }} style={styles.pdf}/>) } } const styles = StyleSheet.create({ container: { flex: 1, justifyContent: 'flex-start', alignItems: 'center', marginTop: 25, }, pdf: { flex:1, width:Dimensions.get('window').width, height:Dimensions.get('window').height, } }); Property Type Default Description iOS Android Windows FirstRelease source object not null PDF source like {uri:xxx, cache:false}. see the following for detail. ✔ ✔ ✔