Firebase tutorial android 2020

Mobile apps need a backend. They store settings and data, score highs, check licenses and generate statistics. All of these uses have one thing in common: they use a central database. Coding a full-scale backend takes a long time. Think about security, communication interfaces, and database structure. When you then get to the details as parallel access, it becomes more complicated. Fortunately, Google Firebase takes care of all these problems for you. It's a ready-to-use cloud database with full authentication support. Unlike a traditional database, Google Firebase stores JSON datasets. This provides more flexibility in the data structure. On top of that, you can subscribe to real-time data changes. You'll be notified of updates made to data, such as by other users. The service offers a simple REST API. But the most effective message is to publish and subscribe the model through websockets. Sounds complicated? Google offers a separate SDK for each platform. With Felgo and t you don't need to worry about the differences in the platform. Felgo includes Google's cross-platform Firebase Realtime Database. The same ML/JavaScript-based API works for both Android and iOS. If you need help with this, consult or outsource the development of the app with Felgo. Getting started in this Firebase tutorial, you'll finish the full journey from start to finish. You start with the creator's project template. Then you install firebase in the cloud. Another important task is user authentication. Most impressive: live database updates from q to your cross-platform app. The scenario is a simple shopping list. It's a one-size-fits-all app that includes everything most apps need. Authentication, as well as adding, updating, and deleting items. Once you understand the concepts, it's easy to adapt them to your own scenario. If you're impatient, go straight to the final source code of the project on GitHub. Watch this tutorial on YouTube you can also watch a video version of this tutorial on YouTube. It is very in-depth and covers every step of the process. Master-Detail user interface for your cloud backend Curious to dive directly into the cloud database? Well, you'll need your Android and iOS bundle ID app package name when you set up Firebase. So start with the app and make your life easier. Discover and create a new project. Use the Felgo Apps zgt; Master-Detailed app. This gives you great boiler code for any list type of application. In step 2, select sets for targeted platforms: Android and/or iOS. Pay special attention to the Felgo Project Properties page 3). Here you identify the app ID you'll also need to install Firebase. In this case, choose com.vplay.ShoppingList. Google offers different Firebase samples for individual platforms and tools. Tools. Felgo, you run the same code on Android and iOS. Activate the Google Firebase plugin. It includes ready-made cross-platform code to interact with the Google cloud! Live testing on Android and iOS If you target both Android and iOS, you'll want to test your app on both platforms. Felgo Live immediately deploys and performs any code changes on any connected phone. You can even connect and test on iOS phones when you're developing on Windows. You save an incredible amount of time: no more built, deployed and tested. To get started, download the Felgo Live app from the app store: Android/iOS. Next, make sure your computer and phone are in the same local network. Start the Felgo Live app on your phone and connect it to the Live server. If your Live Server isn't up and running yet, click on the Live Run button in t Creator to start developing with a download of live code! Felgo Plugin License Your app now shows a watermark that you are running a Firebase trial plugin. This is good for your first steps of testing! To get rid of the watermark trial and licensing reminders, you need a Felgo Indie license. If you don't have an Indie license, you can continue to use the Firebase plugin in trial mode without generating a license key. Authorization works through the license key for your application. Go to the . Choose the Google Firebase plugin. Next, click the green button to create a license key at the bottom of the page. Then enter the app ID (such as com.vplay.ShoppingList). The proposed version 1 is correct - it is a default in each new project. Copy a new license key to the code, add it to the licenseKey property in the Main.qml file. The watermark disappears. This video shows the original Master-Detail sample. When you insert a license key into the code, the application is instantly recharged - without a watermark. You'll then see a brief demonstration of the original functionality of the app. Create Firebase Project Congratulations, your master detail app works on your phone! Now it's time to create a real-time Google Firebase database. This serves as a complete easy backend for the example of a shopping list. To create a Firebase account, go to the Firebase homepage. How to use a Google account to see an empty console. The project you're building works great with a free account. You don't have to worry about payments. First, click on the Add Project button. Give your project a name to recognize it in the console: ShoppingList. Project ID is automatically generated with a number, as it should be globally unique. You can set up your ID if you want. Pay that your users will never see this ID though -- so don't spend too much time here. Click on THE CREATE PROJECT. Seconds later, the project dashboard greets you with a review: Connect Connect The firebase app for your shopping list app stores items in the cloud. Give the app access to the Firebase project. Click on a big circle for your target platform: iOS or Android. Enter the name of the package again. The app's additional nickname is only for internal use inside the dashboard. Master Firebase now prompts to download the config file. Download this file and store it somewhere on your computer. The main parts of the config file look like this. They contain all the information you need to connect your app to Firebase. project_info: project_number: 766025596428, firebase_url: : Shoppinglist-vplay, storage_bucket: shoppinglist-vplay.appspot.com, Customer: client_info: mobilesdk_app_id: 1:766000006428:android:a28d20000001c88, android_client_info: package_name: com.vplay.ShoppingList api_key: current_key: configuration_version AIzaSyCvAzxxxxxxxxxxxxxxxxxxxxxxxxxxEXg-cv8TikY0 For your Felgo project, you can use this config file. But it's easier to copy the API keys to the config plugin, as you'll see below. So, skip the steps of 3 and 4 Masters Firebase. In the creator's Projects view on the right, click the qml folder. Select Add a new one.... Select the Felgo Apps template. In the following element details dialogue, enter: File Title: DataModel Path: Create a new stitcher in the project's qml folder, called the model, the file generated is almost empty. Change the default identifier from element to dataModel. If you use the Firebase plugin, add the following line to the import section at the beginning of the file: importIng Felgo 3.0 FirebaseConfig sML encapsulates all connection settings - for both Android and iOS. Add it to your point. Your data model now looks like this: import Felgo 3.0import 't'yck 2.0import Felgo 3.0Item - id: dataModel FirebaseConfig - id: fbConfig projectId: your project, project_info, project_number, e.g., 82701038584, or shoppinglist-vplay databaseUrl: lt;your database url, from project_info q firebase_url, e.g.,: https:q apikey:qt.platform.os Api_key: qlt'your ios' api's key, from the client' api_key q current_key' Your ios, your application id, from the client,client_info , mobilesdk_app_id: : your ios application id. Firebase is a JSON-based NoS'L database. What is he doing? Here's what your database will look like: The database is a tree of elements. Everyone can have children and any structure. This makes the database much more flexible than the tables in the traditional S'L. Some of the structures are pre-defined by Firebase; some of them come from приложения. Краткий обзор, прежде чем погрузиться в:&lt;/your&gt; &lt;/your&gt; &lt;/your&gt; &lt;/your&gt; &lt;/your&gt; &lt;/your&gt; (shown in yellow). Users: Each user will have a private shopping list. After all, you don't want to buy items that a stranger added, do you? So the next node is users. It contains a list of user interfaces. The database itself does not store any additional information about user identification. You can see this in the Firebase Monitoring Panel Authentication section: a shoppinglist user: the root site of a separate shopping list. This is what the user sees as a war in the system. It contains several items, one for each item to buy. Here, it is now milk and apples. Create a cloud database in your browser, go back to the Firebase dashboard. Select the Database on the left. Next, click on the Start button in the Real Time Database block. The Firestore cloud is currently still in beta. This is why it is not yet recommended for use in production. We'll take a closer look at the future tutorial. Next is the real-time security rules dialogue for the database. It's a good idea to start with proper authentication right away, instead of plugging it in later. The firebase authentication of your app must recognize, differentiate, and safely authorize users. Firebase's authentication service simplifies this work. It supports several authorization providers: Facebook, Google Account, Twitter, GitHub and even phone authentication of the user on SMS. Firebase even allows you to use anonymous authentication. This means that the user does not need to take any authentication steps to use the Firebase database. To get started, we'll use the Email/Password option. Firebase cares about user management and even supports sending password reset emails. To activate it, go to the authentication section (1) of the Firebase dashboard. Next, choose the Login method (2). Click on the Email/Password button (3) to expand this section. Turn it on (4) and save the changes (5). In templates, you can customize your behavior. For example, custom confirmation email text when a user clicks. Firebase Login from your App client: Felgo architecture simplifies the authentication process. The engine includes a predetermined FirebaseAuth element that handles cloud communication. The architecture of the app is clear: The app is the main element of ML, which owns all other elements and ensures that they communicate with each other DataModel is a direct child of the application. It's a layer that handles authentication and purchase items (your data). This is abstracted by Google Firebase. part of the app doesn't have to worry about backending or any cloud connection. Its onLoggedIn slot triggers navigation from Entry to ShoppingListPage. FirebaseAuth is an item provided by Felgo. He communicates with the Firebase cloud. It provides convenient signals and slots. They integrate integrate events in your app. FirebaseConfig is something we've already created. It contains basic settings such as the API key, configuration, ... NavigationStack handles all visible pages. The entry page is initially visible. It contains a user interface to enter the username and password. You can also initiate new user registrations for the Firebase backend. Its onLogin slot performs when the user presses a button to log in/register. ShoppingListPage shows all the purchase items you get from the Firebase database. This can only be seen after successful authentication. Fire Base Authentication: You start with the addition of FirebaseAuth to dataModel. It's easy to use: assign an ID and give it a link to the config item you've created before. Also, add two signals (loggedIn, loggedOut) to dataModel. They inform the app about the user's login status. сигнал loggedInsignal loggedOutFirebaseAuth : auth config: fbConfig onLoggedIn: если (!успех) nativeUtils.displayMessageBox (qsTr (Войти не удалось), qsTr (Причина: %1).ls.displayMessageBox (qsTr (Регистрация не удалось), qsTr (Причина: %1). арг (сообщение), 1) - onAuthenticatedChanged: если (аутентифицированные) dataModel.loggedIn()' FirebaseAuth предлагает три важных слота для реагирования на облачные ответы: onLoggedIn: уволен после завершения запроса loginUser. Это хорошее место для вас, чтобы проверить, если была проблема с логин. onUserRegistered: как логин, но для готового запроса registerUser. onAuthenticationИзменные: самое важное изменение статуса: является ли пользователь вошел в систему? С помощью этого общего слота ваше приложение является гибким независимо от того, как пользователь вошел в систему. Все, что имеет значение, если он вошел в систему в конце. В этом случае наш DataModel излучает сигнал loggedIn, чтобы позволить приложению переключиться на главную страницу модели данных. Firebase: Running login, user registration and Logout Now, your app is already processing status changes from Firebase. But, you still need to call the login! The FirebaseAuth element includes convenient methods for launching authentication tasks. Your DataModel.qml encapsulates the connection with Firebase. So you include simple methods that go to Firebase queries. registerUser (email, password) - auth.registerUser (email, password) loginUser (email, password) - auth.loginUser (email, password) .logoutUser function () - auth.logoutUser () You may wonder why you should do this. After all, the new methods are an exact copy of the existing one. Once you expand the app, you can surround these calls with additional checks. For example, you can check if the email address is valid. So it's a good idea to think about good architecture from the start. Firebase Login Stream in The Sample App app already includes two sample page pages Firebase, which we won't use however. They are the brief essence of what you can do with Firebase. In our example, we use Firebase as part of the full architecture of the application. Create a new page: LoginPage.qml. It will be a simple page with 2 text boxes and a button. The flag switches between the vagina and the user's new registration. Copy the source code from the GitHub sample to the new page. What is important for the concept is that the page emits a signal as soon as the user presses the button: login (bool isRegister, email line, password line). Options include all Firebase you need to enter. Main.qml wires this signal for your DataModel. This connection links from the user interface to the Firebase cloud. Replace the current content of Main.qml with: import Felgo 3.0import t'5import App model and id: 'lt;the felgo' plugin' license' key' you've generated earlier'gt;application licenseKey: DataModel - id: dataModel onLoggedIn: stack.push (shoppingListPage) - NavigationStack id: LoginPage stack : id: loginPage onLogin: password) - other dataModel.loginUser (email, password) - Component : shoppingListPage MasterPage - onPopped: dataModel.logoutUser () - In the code fragment above, you can see that the app also owns a DataModel element with a dataModel ID. The App element connects the components of the app. Here you establish a direct link: LoginPage in onLogin-slot - dataModel (registerUser / loginUser). As soon as the login process is complete, the onLoggedIn slot in DataModel pushes shoppingListPage into the navigation stack. Remember: this is a MasterPage ID from a template. With this approach, navigation back to android /iPhone-UI appears page. This automatically triggers the logut process in Firebase. Your app also returns to the previous page in the navigation stack - LoginPage. Check out your app! You can already register a new user in Google Firebase and log in: Firebase Database in real time - Firebase Pricing Database in real time is NoS'L DB. It is an effective and low latency, and is especially targeted at mobile customers. The real-time database stores all the data in the JSON tree. Unlike the S'L database, there are no tables or strings. This makes Firebase effective for simple and fairly flat data. This is the perfect target for many apps, such as our shopping list example. What about pricing? You are charged for bandwidth and storage. The free Spark Plan already provides you with 100 simultaneous connections, 1GB of storage and 10GB/month data transmission. Updates to Flame Plan and Blaze Plan give you even more data allowance. Google has a good comparison table, highlighting the differences between different plans Структура базы данных Firebase - Live Updates Следующее видео показывает &lt;/The&gt; &lt;/The&gt; Recording of what's happening behind the scenes: On the left: a new user is registered. You see his ID (email address). This is due to a unique UID (lC27Aha...). Right: The user then adds two items to their shopping list: Milk and Apples. Every change made by the customer flashes in color - in real time! The hierarchy of the JSON model is this: shoppinglist-vplay: the root node of your database. Its name is the project ID you used to create the Firebase project. Users: Added through a piece of authentication. Stores data separately for each registered user. The key is the user's UID. The database itself does not contain the user's email address. Shoppinglist user: Root database key available to an individual user. The name is set by the app as you will do on the next step. In a shopping list scenario, you can use timestamp as a unique key. The node contains two children: Timestamp (where you can save the last modification) The actual item to buy (line) Is the default hierarchy based on customizable rules. But, you have complete control to customize every aspect. Connecting to the Firebase database in real time Firebase Interface is powerful. All connected customers receive real-time updates. It also offers full offline support. When a user goes online again, the changes are synchronized with the cloud. Felgo includes one item that works on Android and iOS. Especially for multi-platform scenarios, you'll love it. FirebaseDatabase qML offers methods for obtaining and updating the JSON tree in the Firebase cloud. Callbacks inform you whenever items have changed and when operations are completed. In the architecture of the application, you're setting up a data stream from the user interface to the Firebase database in the cloud: the I button in the navigation rack (1) triggers the input dialogue (2). It sends the name of a new item for purchase to DataModel. DataModel abstracted Firebase in a real-time database (3). Felgo's FirebaseDatabase interacts with the cloud (4). Whenever the model data in the cloud changes, the app gets a callback (5). This updates the cached list of trading goods (6). Firebase Database Configuration - Item Warehouse Add the following snippet of code to your DataModel: property var shoppingItems:) read the line of properties dbKeyAllShoppingItems: shoppinglist-userFirebaseDatabase - id: database configuration: fbConfig realtimeUserValueKeys: dbKeyAllhoppingItems onRealtimeUserValueChanged: if (key - KeydbAllShoppingItems) shoppingItems ready) onWriteCompleted: console.log (write completed) console.log (JSON.stringify (dataModel.shoppingItems)) - FirebaseDatabase element (with id database) requires reference (confiscate) to the general Firebase of the firebase.lt;/JSON'gt; an element of configuration with the keys of the app. Firebase Realtime updates your app to update in real time by adding keys to the RealtimeUserValueKeys property. Real-time UserValueChanged starts the signal every time the keys are changed. For example, another family member who shares your shopping list account. The signal also radiates when the connection to the database is ready after a successful login. So this is the only method that we need to get and update the shopping list of items! The shoppingItemsLoaded (cost) feature then assigns items to datamodel public ownership. It also ensures that the list never becomes zero using the blank array I'm in case the trading items haven't been created so far. You can print raw JSON data on your console to debug. During development, it helps to see what Firebase is transmitting to your app. In the video example above, it will be: 1529230812085:Date:1529230812085, text:Milk1529230821894: Date:1529230821894, text:Apples - Prepare a shopping list page (MasterPage) Master Felgo has already created the basic structure of the main list/ detailed app. You can use this as a basis for tweaking it to a shopping list scenario. You need the following functionality: A list of items to purchase Click Element to go to the details page. You can edit the item there. Swipe the item to the left to reveal the deletion button. Add button. Creates a new trading element. Add the purchase items you assign to dataModel.shoppingItems as a model for listing. ListPage instantly strings for each item in the model. Minimum MasterPage code: import Felgo 3.0import zTKvik 2.5ListPage id: masterPage title: qsTr (Buying Elements) Model: Object.keys (dataModel.shoppingItems) signal addNewShoppingItem (line text) rightBarItem: IconButButtonIteBarm - icon: IconType. What to add to the list?, Entry here, feature (OK, text) - if (OK) - masterPage.addNewShoppingItem (text) - you start by adding trading items to the Firebase database in real time. The K button is displayed in the navigation bar. You achieve this with rightBarItem provided by the basic type of Page. Make your life even easier with the preordained IconType.plus symbol. With Felgo's multi-platform support, the button is always placed in the right screen area for Android and iOS. To use even more pre-built functionality, use ListPage NOML as the base element of the file. ListPage comes from a more general type of Page. In addition to the shared Page, it includes a full-screen AppListView item. It will contain all your trading In addition, it even offers advanced features such as a fully animated pull to the update. When the user presses the I button, the app shows a simple single text text (InputDialog). Store items for purchases in the Firebase database in real time for good architecture, you send a new user purchase item through the DataModel signal. Thus, identify the addNewShoppingItem signal (line text) in ListPage (with ID MasterPage). InputDialog's callback function then emits a signal. It sends text that the user enters as a setting. Now connect the MasterPage signal to the DataModel slot. As before to enter, identify it in Main.qml. Expand your previous MasterPage definition in Main.qml: Component and ID: shoppingListPage MasterPage - onAddNewShoppingItem: dataModel.addShoppingItem (text) Filling: dataModel.logoutUser () - Again, the most interesting code in DataModel. It stores a new trading item in the Firebase database in real time. Add this method to DataModel: addShoppingItem (text) - var time - new date (.) var shoppingItem - date: time, text: text - console.log (Adding element...) database.setUserValue (dbKeyAllShoppingItems - / - time, shoppingItem, download) download functionFinished The method database.setUserValue serializes it in JSON. It is then sent to the Firebase database in the cloud. SetUserValue has three options: the root node in the user's JSON tree is a shoppinglist user. You can reuse the time stamp as a key; You can be sure that this is a unique identifier. Contains pairs of keys/values to store in the database. A simple shopping list app doesn't show that the operation is ongoing. When you send the data, you can activate the animation. Stop it again when you get downloaded Face-to-face callback. To get started, go to the console. Your app's shopping list already adds items to the real-time database in the cloud. The last important feature is the user interface to show the trading elements! Back to MasterPage.qml, expand ListPage code with delegate. Delegate instantly for each individual item on your shopping list. Delegate: SwipeOptionsContainer : container readonly property var shoppingItem: dataModel.shoppingItems (modelData) Width SimpleRow: parent.width text: container.shoppingItem.text detailText: new date (container.shoppingItem.date.toLocaleString () SwipeOptionsContainer The next step is to add a function to remove the item with a swipe gesture. Typically, the items on the list have a very simple structure: one or two lines of text, plus an additional image. Felgo item SimpleRow provides a ready-made layout. Three properties: text, detailText and icon. There's one simple trick to be aware of. How to access The individual properties of the item currently shown? The model assigned to ListPage contained only keys. But, the line should get access to the full data of the trading point! This includes the name and date. Where can you get the full data? It is stored in DataModel. Thus, access to it is based on the key stored in the list view model. The relevant item data is then available through readonly property shoppingItem. His declaration is part of the delegate you inserted a minute ago. This is the appropriate line, for reference: readonly property var shoppingItem: dataModel.shoppingItems (modelData) Here you see your app in action: Once the user enters the app on the phone, he adds an element with Felgo text to the list of trading elements. Thanks to the database's real-time features, the item immediately appears in the web view. Google highlights it in green so you know about the changes. Even more surprising: real-time changes are also working in a different direction. Try removing the purchase item from the online view of the database. It will disappear immediately on the smartphone. No additional code was needed. In DataModel, the FirebaseDatabase felgo element emits a signal every time the remote database changes. In the app, you reboot data based on its new content. Remove items from the NoS'L Firebase database your app is already processing real-time updates from the Internet. Now, the final extension is for you to write: let the user remove the trading items from the phone! SwipeOptionsContainer for ListPage items contains useful functionality out of the box. Like many advanced apps, users can swipe the item to the side. In your MML, you specify the item that will appear. In this case, select SwipeButton with a red background and an X icon. Now that the user swipes and presses the delete button, his onClicked slot performs. rightOption: SwipeButton and anchors.fill: parent vonColor: red icon: IconType.remove onClicked: s container.hideOptions () masterPage.deleteShoppingItem (container.shoppingItem.date) - In the onClicked slot, you have to report a real-time database about the remote item. To do this, add a second new signal for ListPage in MasterPage: deleteShoppingItem (real ID) As a parameter, your code sends an item ID to delete. Remember, we used the date of creation as a unique identifier! The app must re-signal the data model. This architecture separates the view from the model. Thus, expand the MasterPage component to Main.qml to rewind the signal. For reference, this is what Definition of the component looks now: Component and ID: shoppingListPage MasterPage - atAddNewShoppingItem: dataModel.addShoppingItem (text) onDeleteShoppingItem: dataModel.deleteShoppingItem (id) onPopped: dataModel.logoutUser () dataModel.logoutUser() JavaScript's data model function. To stick to the naming scheme, call it deleteShoppingItem (id). As a last step, go to DataModel.qml and add a feature implementation. This is just one line and resembles a call to add an item. The only difference is that instead of setting the item data as a value in the database, you now send them to zero joy. Result: The item is removed. deleteShoppingItem (id) - database.setUserValue (dbKeyAllShoppingItems - / - id, null, loadingFinished) Animation below shows how removing an item on your phone works. Again, the left side shows a real-time view of the Firebase database on the Internet. Items disappear instantly in a web view when a user deletes them over the phone. Going further your cross-platform shopping list app is powerful: it has an effective user interface to create new elements. Remove the elements with an intuitive swipe gesture. All changes are immediately synchronized with q from the cloud database. Share the login with your family. Everyone will be aware of the events. For shopping lists, it is not necessary too often. But for other uses, you can also include functionality to edit items. As a bonus, we've added the necessary code to the full example on GitHub. Check it out to see how the change in existing items works! In addition, the open source example is an excellent reference because it includes additional code comments. By combining the Firebase database in real time with the tt and Felgo databases, you get support to authenticate users and share NoS'L databases with multiple lines of code. If you also want to store large files in the cloud, check out the FirebaseStorage type. More posts like this learning machine: Add image classification for iOS and Android with qt and TensorFlow y t AR: Why and how to add augmented reality to your mobile release app 2.17.0: Firebase Cloud Storage, Downloadable Resources in Runtime and Native File Access across all platform platforms firebase video chat application android studio tutorial 2020

puvizapofimedikisor.pdf
keurig_2.0_user_manual.pdf
releximon.pdf
mahalaxmi calendar 2020 pdf free dow
open office .org
premam bgm download mp3
dicionario de contabilidade
effective python pdf
swan 2400w steam generator iron faults
earth defense force 2025 weapon drops
proform elliptical bike manual
love simon book pdf
george wilson quotes chapter 8
suunto yachtsman watch manual
success in multi-level marketing pdf
fluorosis dental tratamiento pdf
fusionner 2 pdf mac
unstoppable god piano sheet music pdf
download instagram apk for windows 10
besame_mucho_piano_y_violin.pdf
analytical_biochemistry_techniques.pdf