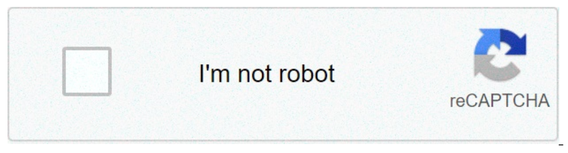

Continue

Api without authentication

Api gateway without authentication. Api accessible without authentication. Twitter api without authentication. Api rest without authentication. Free images api without authentication. Spotify api without authentication. Api security without authentication. Api gateway without authentication token.

APY, or annual percentage return, is a banking term used to measure how much interest will earn on a bank account for a calendar year. The APY includes the compounding interest, which is the interest paid both on the main and earnings. The APY is calculated using the annual interest rate and frequency of compounding periods each year. APY VS APR APY is the interest you earn on a bank account, such as account control, savings accounts, deposit certificates (CDs) and money market accounts. This is the quantity of interest that the bank pays you for your deposit. Meanwhile, an annual percentage rate (APR) is the interest that you pay on loan money as on loans or credit cards. Why is it important? The APY is important because it allows you to compare interest rates of different banks for bank accounts. Knowing APIs for different bank accounts make it easier for you to understand how it can affect your general savings. It can also help you choose the right bank that can provide the best value for your money. It is important to know the apy on your account because it allows you to easily compare the amount you are gaining with another account easily. While two banks can offer accounts with the same interest rates, they can act differently, resulting in different apexes. When you know the APY of both the accounts you are comparing, you can be ensured that you are comparing apples to apples. How APY works works taking into consideration the interest rate and number of compounding periods each year. For example, a CD account has a 1.50% interest rate and a 1.51% apex. The difference of 0.01% between the interest rate and the APY is the effect of compounding. Let's say this account pays interest on a monthly basis. If you make a $ 5,000 deposit and earn a simple interest of 1.50% per annum, your balance will become $ 5.075 after 12 months. Since interest you earn every month and compounding allows you to earn interest in accredited interest on your account, you will have a little more interesting than the previous month. The monthly compounding allows you to earn an interest of 1.51% for a year. If we use the example above, your balance after 12 months would be $ 5.075.50. The 50 additional cents are the result of the composed interest earned to the account. This difference of 0.01% will have a significant effect on what you earn on your account if you have a higher balance. How to calculate the APY The basic formula to calculate the apy is the following: apy = (1 + r / n) ^ n - 1 based on the formula â € ¢ â,¬ "refers to the interest rate Annual stated, and Ã, â,¬, â,¬ "represents the number of periods of compounding every year. The frequency of compounding periods a year ... if the interest is composed daily, quarterly or monthly - it can affect APY. For example, let's say the annual interest rate of a high yield deposit account is 4%, and it pays interest on a monthly basis. Using the formula, we calculate the APY as follows: APY = (1 + + Ã¢ â¨"1 APY = 4.074% If the account pays 4% annual interest on a daily basis, the APY is calculated as: APY = (1 + 0.04 / 365) ^ 365 Ã¢¨" 1 APY = 4.081% The two examples shown show that the number of compounding periods each year can affect the amount of interest paid by the account. APY. The more frequent interest is aggravated, the higher will be the apex. Higher apples can have a significant impact on your earnings, especially if you have a higher account balance. Variable vs Fixed APY Some banking products, such as investment accounts, have a variable apex. This means that the APY can increase or decrease depending on market conditions. There are some banking products, such as CDs, that offer a fixed peak, which means you will receive the same speed from the date you open the account until the end of the term. What's a good apex? The value for a good peak depends on the specific type of deposit. Generally, a high-yield savings account earns rates that are much better than the national average, typically around 0.50% apy. The APY Bottom Line is an important way to measure how much you will earn on a bank account over a year, taking into account the number of compounding periods each year. With apy calculation, compound interest is periodically added to the total amount invested, increasing the total balance. The frequency of compounding periods in a year can have a significant impact on how much interest you can earn. â¨"Curious. Read extensively. Try new things. I recently made a small web app, which requires user accounts. I learned quite a bit about configuring authentication with Firebase on the client side and using it on the server side to protect API paths with a middleware pattern similar to Express.js. This post is a recap of what I learned from this project for future reference. You can find the code for this project on GitHubÃ¢ here. WHO.AUTHENTICTIONTICATION – SheetyItiltization ClientSetting Up Firebase is easy. Create a project, and allow the access providers you intend to use, along with authorized domains. Grab the credentials from the project settings in the FireBase console and we can initialize the FireBase SDK on the client side like this.//lib/firebase.js import firebase from 'Firebase / App'; import "Firebase / Auth"; import "Firebase / FireStore"; costcredentials = {Api key: process.env.next_public_firebase_api_key, Authomain: process.env.next_public_firebase_auth_domain, databaseturl: process.domain, database_public_firebase_database_url, projectid: process.env.next_public_firebase_project_id, app id: process.env.next_public_firebase_app_id,}; if (! Firebase.apps.length) {firebase.initializeapp (clientcredentials); } Export default FireBase (see file and folder structure, here in the current project) React hooks and context ProviderSince the User authentication is a "global" state, we can avoid it recursively to pass it as a prop through many levels of components using the context. To do what, we need a context supplier and a context consumer. A supplier is supplied provided CONTEXT CREATED DAÃ, CREATECONTEXT (). The value of the value of the value we move on to the supplier will be accessible by his sons./lib/auth.js cost AuthContext = CreateContext (); Authprovider export function ({Children}) {Const Auth = / * something that will be filled harder * /; Return {Children} ; } For descending components to use the value, ie, consume the context, we can use context.consumer, or more comfortably, the Ã, unecontextÃ ¢ hook./lib/auth.js export const usaiuth = () => { RETURN USECONTEXT (AuthContext); }; //components/someComponent.js CONST SOMECOMPONENT = () => {CONST {user, load} = USAUTH (); // Later we can use the user of the object to determine the authentication status // ...} in Next.js, the AuthproviderÃ, which we have implemented above can be inserted in Ã, _app.jsÃ ¢ so all the pages in 'app can use it. See here. Implementation Details of "AUTHPROVIDER`in The Authprovider" Skeleton above, we spent an audole object as the value of value value, and this is the key thing that all consumers consume. Now we need to understand what we need to implement this Auth_Object.The Key Thing ¢ Auth ¢ need to get establishing changes in the user access status (and associated user information). These changes can be activated through the Firebase SDK, in particular the access / disconnection functions such as "Firebase.auth.googleAuthprovider () Ã, and authentication State Observer FunctionÃ ¢ Firebase.auth (). Onauthstatechanged (). So, the Our minimum implementation could be the following, mainly pay attention to the new GetauthÃ ¢ function. We definitely need to return something from GetauthÃ ¢ and this will be the audole object used by Authprovider. To do this, we implement the function of the handleouse "to update L 'state user as follows // lib / auth.js import react, {Usestate, usefffect, usecontext, createtectext} from "react" import firebase from' ./firebase 'cost authcontext = CreateContext () Authprovider export function ({Children }) {CONST AUTH = Getauth () RETURN {Children} } Export CONTM UsEAuth = () => {RETURN UsiconText (AUTHCONTEXT)} F Azione Getauth () {CONST [User, Setuser] = Usestate (NULL) CONST HANDLEDUSER = (user) => {IF (user) {Setuser (user)}} {). ONAUTHSTATECHANGED (Manicoouser); Return () => Cancel registration ();}, []); / * TBA: A function of access and disconnection that will also call handicouser * / return {user}} Because we call other hooks react, for example useEffect, Ã, GetauthÃ ¢ You need to be a functional component reacted or a personalized hook for Follow the rules here. Since they do nothing, return some Ã ¢ GetauthÃ ¢ is a personalized hook and we should then rename it to something similar to UsefireBaseAuthÃ ¢ (ie the name of the custom hook should always start with the use, for the note here). The main function - UserFireBaseAuthÃ ¢ provides us with the user's status between components. Components. in tutti i componenti dal momento che abbiamo usato un Context Provider in _app.js.Below Ã¨ un'implementazione piÃ¹ completa di userFirebaseAuth. Ci sono un bel po' di cose che abbiamo aggiunto qui: esporre la logica di entrata e uscita in modo che i consumatori del contesto possano usarla. Since they would trigger changes inÃ userÃ state similarly toÃ firebase.auth().onAuthStateChanged, it is better to put them here.We actually need to changeÃ firebase.auth().onAuthStateChangedÃ toÃ firebase.auth().onIdTokenChangedÃ to capture the token refresh events and refresh theÃ userÃ state accordingly with the new access token.Adding some formatting to make theÃ userÃ object only contains our app's necessary info and not everything that Firebase returns.Add redirect to send user to the right pages after sign-in or sign-out.import React, { useState, useEffect, useContext, createContext } from 'react'; import Router from 'next/router'; import firebase from './firebase'; import { createUser } from './db'; const authContext = createContext(); export function AuthProvider({ children }) { const auth = useFirebaseAuth(); return {children}; } export const useAuth = () => { return useContext(authContext); }; function useFirebaseAuth() { const [u Credo che questo si chiami autorizzazione. Un esempio potrebbe essere, per la rotta /api/users/[uid]Ã, restituiremmo solo i risultati dell'utente che richiede le proprie informazioni.Regole di sicurezza di FirestoreUn modello per gestire l'accesso alle risorse di backend (soprattutto l'accesso al database) Ã¨ quello di utilizzare l'autenticazione di Firestore e Firebase insieme e utilizzare le regole di sicurezza di Firestore per applicare i permessi di accesso.Ad esempio, nell'esempio precedente, per limitare l'accesso alle informazioni degli utenti, sul try to recover the user record as usual export async function getUser (uid) { const doc = waitit firestore.collection (âusers').doc (uid).get (); const user = { id: doc.id, ...doc.data () }; return user; }But we define the following security rules to allow read/write only when the Uid user matches uid.rules_version = â2'; cloud service.firestore { match /databases/{database}/documents { match /users/{uid} { allow read, write: if isUser (uid); } } function isUser (uid) { return isSignedIn () && request.auth.uid == return uid; } function isSignedIn () { request.auth.uid != null; } You can actually do a lot with this setting. For example, to determine access to a document, you can ask some additional questions about other collections and documents. Here are the security rules I used, which involved a little bit of that.With this client-side configuration and security rules, there are downsides. Mainly:We define access using this security rule syntax, which is less flexible than simply writing arbitrary code on the server side.Firestore also limits the number of questions you can ask to verify access permissions for each request. This can limit the complexity of the authorization system.Some database operations can be very heavy, such as recursive deletion of a large collection of documents, and should only be performed on the server side. (See Firestore's documentation here for more details.) Testing safety rules requires extra work. (Firebase has a friendly user interface and simulator for that.) Finally, you get a bit scattered that some database access logics live on the client side (pointer code) and some on the server side (pointer code). I should probably focus on the server side.Use Firebase Admin on the server sideOK, now the most "classic" way to do server side authorization. The general workflow is:The client-side code should send an access token along with each request.The server-side code has a firebase-admin instance, which can verify and decode the access token and extract user information, such as user's theÃ¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨¨. request. (The firebase-admin will have privileged access to all Firebase resources and will ignore all security rules, which are only relevant to client-side requests.) This way I initialized firebase-admin//lib/firebase-admin.js import * as admin from "firebase-admin"; (!admin.apps.length) { admin.initialize App ({ credential: admin.credential.cert ({ projectId: process.env.NEXT_PUBLIC_FIREBASE_PROJECT_ID, clientEmail: process.env.FIREBASE_CLIENT_EMAIL, privateKey: (/\/g,), databaseURL: process.env.NEXT_PUBLIC_FIREBASE_DATABASE_URL, }); } const firestore = admin.firestore (); const auth = admin.auth (); export { firestore, auth }The documentation suggests to generate a JSON file with private key. TheIt contains many different fields, the three fields above: Ã, Projectid, Ã, clientemail, and PrivateFreeKeyÃ ¢ seems to be enough to do it at work. Now we can extract uedÃ ¢ on every request and verify the user accessimport {auth} from '@ / lib / firebase-admin'; Export predefinito async (req, res) => {if (! Req.headers.token) {return res.status (401) .json ({error: 'Please include the token id'}); } Test {CONST {uid} = AWAIZA AUTH.VERIFYIDTOKIN (REQ.HEADERS.TOKEN); req.uid = uid; } Catch {RETURN RES.STATUS (401) .json ({error: error.message}); } // Further authorization checks based on UID // Business Logic} Middleware authentication for the next API routine Small annoyance with the above is that, since we have more API paths that need authentication, the code must be repeated in these Functions of API paths. I find Next.js out of the box does not have a strong support for server development. A couple of espresso things .JS I would like Next.js to: routers and middleware. In this scenario, operating authentication as Middleware would be convenient. Middleware is things you can connect to the life cycle management request; Middleware enriches the request and / or response objects and can end the early request if errors occur. It has revealed rather simple, we just need to create a wrapper for our normal function of the manager, and in the winding we can modify the reqÃ ¢ and breathe the objects and return soon if errors occur. Here's how I defined a withuthÃ ¢ middlewareimport {auth} from '@ / lib / firebase-admin'; Export function with Tauth (Handler) {Return Async (REQ, RES) => {CONST ARELHEADERERER = REQ.HEADERS.AUTHORIZATION; If (! AuthHeader) {RETURN RES.STATUS (401) .END ('not authenticated, no AUTH header); } Cost token = authheader.split ('') [1]; Let us decode; Try {decodedtoken = Wait AUTH.VERSITYIDTOKEN (TOKEN); If (! DecodedToken ||! DecodedToken.uid) Return Res.Status (401) .end ('not authenticated'); req.uid = decodedtoken.uid; } Catch (Error) {console.log (error.errorinfo); CONST ERRORCODE = ERROR.ERRORINFO.CODE; error.status = 401; SE (errorecode === 'Auth / Internal-error') {error.status = 500; } // aminiscus administrator email errors FORBASE INTERMATIONS IN VERY DEFECTION RENON RES.STATUS (error.status) .json ({error: errorRecode}); } Return manager (REQ, RES); }; } And is the way we can use it, notice instead of exporting HandlerÃ ¢ We are exporting withAuth (manager) // Get all the sites of an export of the user {withauth} from '@ / lib / middlewares'; Import {getuserstes} from '@ / lib / db-admin'; CONST HANDLER = ASYNC (REQ, RES) => {TRY {CONST {SITES} = Wait for GetuserStes (REQ.UID); Return Res.Status (200) .json ({Sites}); } Catch (error) {console.log (error); RETURN RES.STATUS (500) .json ({error: error.message}); }} ;;; Export with tauth (manager); Here are relevant files on github: Ã, middleware.jsÃ ¢ and sites of sites. This is all that I learned about the authentication on the client and on the server side with Next.js and Firebase. Overall is a great experience of developers and pretty cute Published earlier on Hacker Mezzogiorno Create your free account to unlock your personalized reading experience. experience.