

Secure NestJs Rest API with Keycloak



Chamith Madusanka

Follow

Dec 8, 2020 · 9 min read



Step-by-step guide to secure Rest API build with NestJs using Keycloak.

I am assuming you already have a JS frontend app or at least a HTTP client that perform the authentication against Keycloak and is in possession of a JWT and can pass in the header to your NestJS backend.

If you want to get an idea on how to secure ReactJS front-end using Keycloak and send the authenticated JWT from front-end to back-end, you can checkout my previous article [**Secure Front end \(React.js\) and Back end \(Node.js/Express Rest API\) with Keycloak**](#)

Keycloak

Keycloak is an open source Identity and Access Management solution aimed at modern applications and services. It makes it easy to secure applications and services with little to no code. Keycloak uses open protocol standards like Open ID Connect or SAML 2.0, especially in Identity Federation and SSO scenarios.

Authentication with Keycloak brings to the table virtually every feature you might want regarding user authentication and authorization. Some of these include

- Single sign-on and sign-out, with possible integration with Kerberos (LDAP or Active Directory),
- Support for OpenID Connect and SAML 2.0,
- Log in via social media,
- User account management via both the web console and REST API,
- Fine-grained authorization for different services.

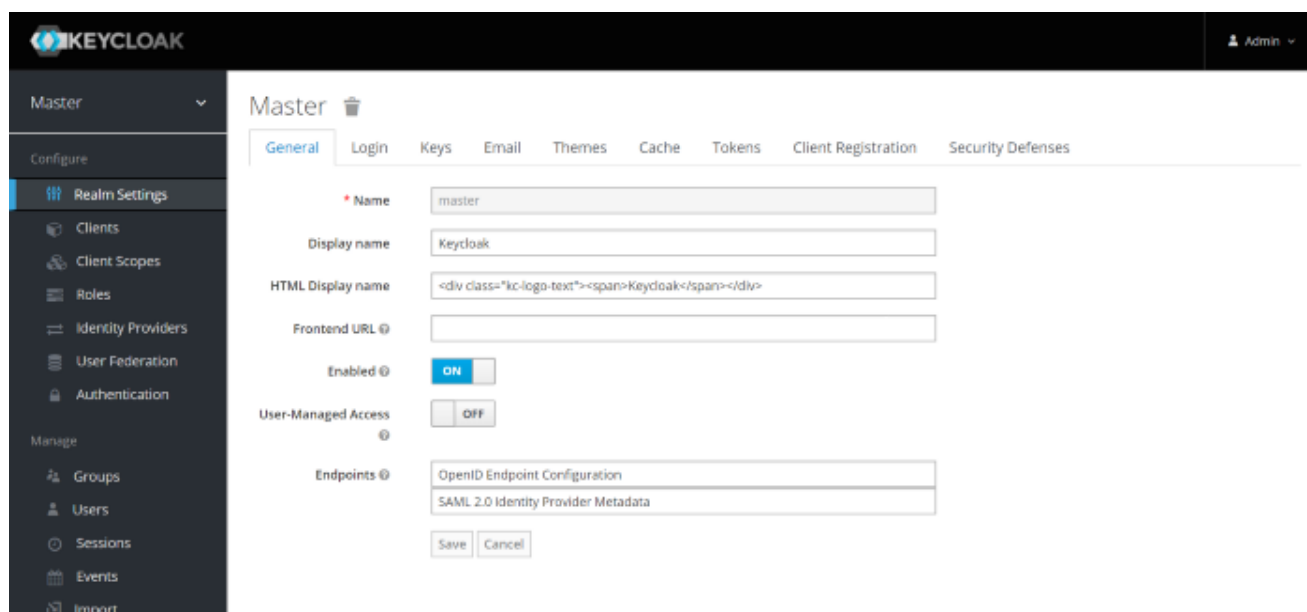
How does Keycloak work?

Applications are configured to point to and be secured by this server. Browser applications redirect a user's browser from the application to the Keycloak authentication server where they enter their credentials. This is important because users are completely isolated from applications and applications never see a user's credentials. Applications instead are given an identity token or assertion that is cryptographically signed. These tokens can have identity information like username, address, email, and other profile data. They can also hold permission data so that applications can make authorization decisions. These tokens can also be used to make secure invocations on REST-based services.

Keycloak Configuration

1. Set up Keycloak server

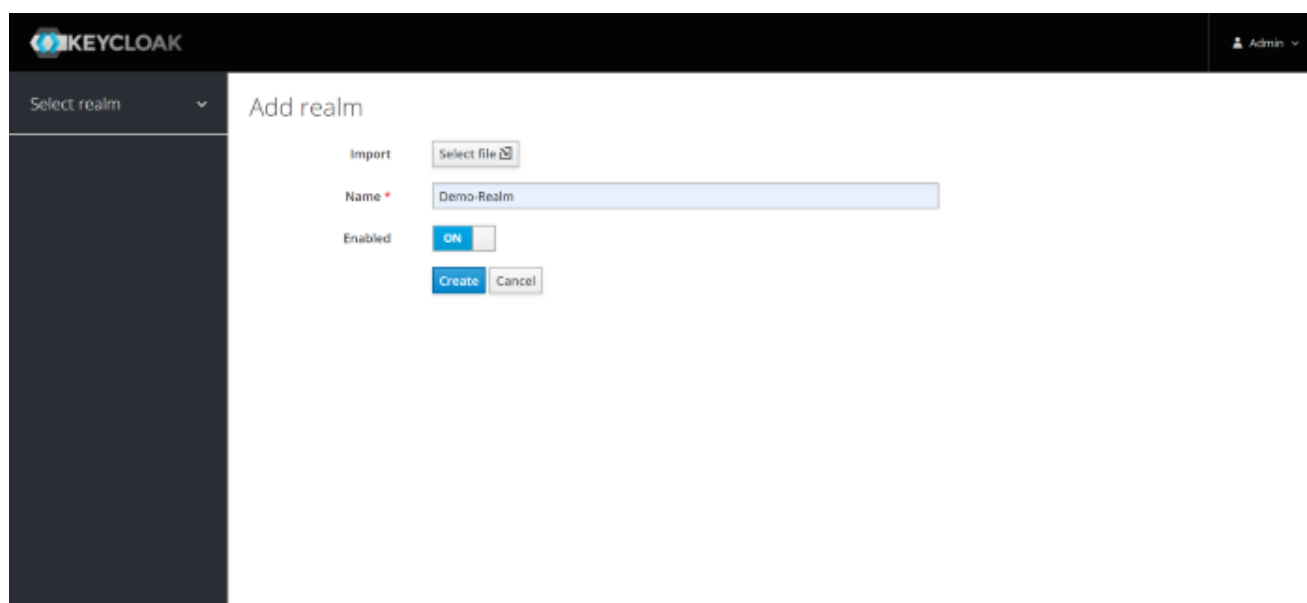
There are multiple ways to setup Keycloak instance. Follow the instructions in the [link](#). Once the setup completed you should login to the Keycloak server using provided admin account credentials.



2. Create a Realm

A realm secures and manages security meta data for a set of users, applications, and registered OAuth clients. Users can be created confined to a specific realm within the Administration console. Roles can be defined at the realm level. You can also set up user role mappings to assign these permissions to specific users.

Create a realm by clicking the **add realm** button on the **Select realm** drop down. Give a name with your preference and click the **Create** button.



Add realm

After that you will be redirected to the realm setting page.

The screenshot shows the 'Demo-Realm' configuration page in Keycloak, specifically the 'General' tab. The left sidebar contains a 'Configure' section with 'Realm Settings' selected, and a 'Manage' section with options like Groups, Users, Sessions, Events, Import, and Export. The main content area has tabs for General, Login, Keys, Email, Themes, Cache, Tokens, Client Registration, and Security Defenses. The 'General' tab is active, showing fields for Name (Demo-Realm), Display name, HTML Display name, Frontend URL, Enabled (ON), User-Managed Access (OFF), and Endpoints (OpenID Endpoint Configuration, SAML 2.0 Identity Provider Metadata). There are 'Save' and 'Cancel' buttons at the bottom.

Note : You can change access token and refresh token lifespan by moving to token tab.

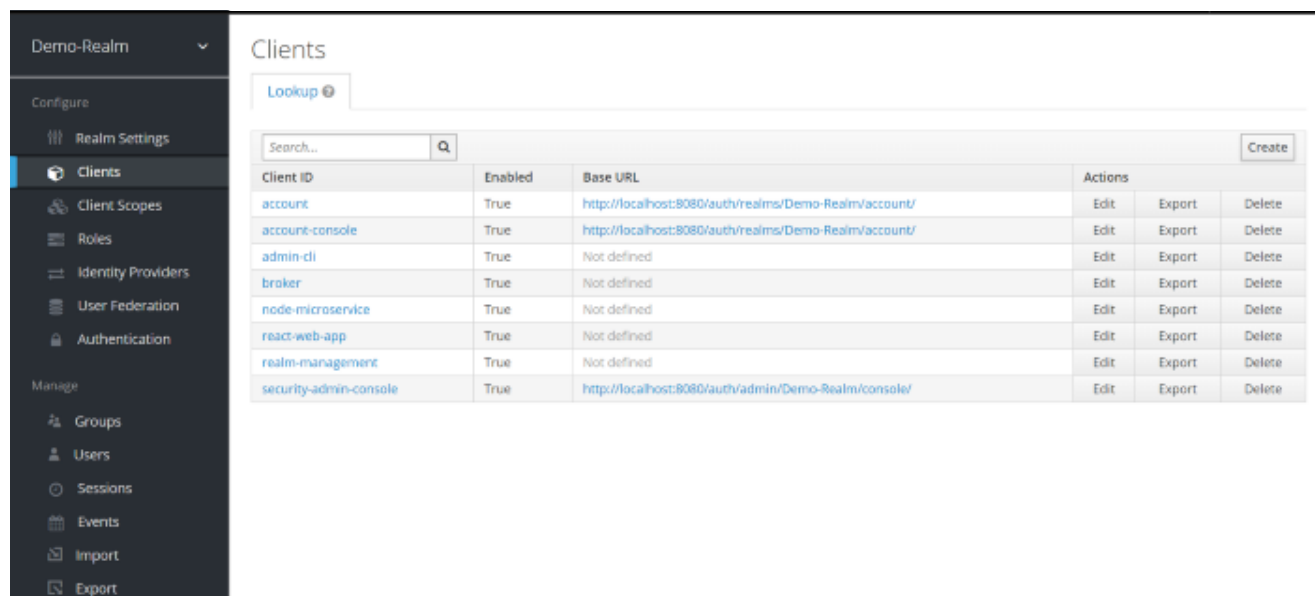
The screenshot shows the 'Demo-Realm' configuration page in Keycloak, specifically the 'Tokens' tab. The left sidebar is the same as the previous screenshot. The main content area has tabs for General, Login, Keys, Email, Themes, Cache, Tokens, Client Registration, and Security Defenses. The 'Tokens' tab is active, showing various token settings. Two settings are highlighted with red boxes: 'Refresh token' (SSO Session Idle: 30 Minutes) and 'Access token' (Access Token Lifespan: 5 Minutes). Other settings include Default Signature Algorithm, Revoke Refresh Token (OFF), SSO Session Max (10 Hours), SSO Session Idle Remember Me (0 Minutes), SSO Session Max Remember Me (0 Minutes), Offline Session Idle (30 Days), Offline Session Max Limited (OFF), Client Session Idle (0 Minutes), Client Session Max (0 Minutes), Access Token Lifespan For Implicit Flow (15 Minutes), Client login timeout (1 Minutes), Login timeout (30 Minutes), Login action timeout (5 Minutes), User-initiated Action Lifespan (5 Minutes), Default Admin-initiated Action Lifespan (12 Hours), and Override User-initiated Action Lifespan (Select one... Minutes). There are 'Save' and 'Cancel' buttons at the bottom.

Make sure **Demo-Realm** is selected for the below configurations. Avoid using the master realm. You don't have to create the realm every time. It's a one time process.

3. Create Clients

Clients are entities that can request Keycloak to authenticate a user. Most often, clients are applications and services that want to use Keycloak to secure themselves and provide a single sign-on solution. Clients can also be entities that just want to request identity information or an access token so that they can securely invoke other services on the network that are secured by Keycloak.

Clients tab allows you to manage your application clients.

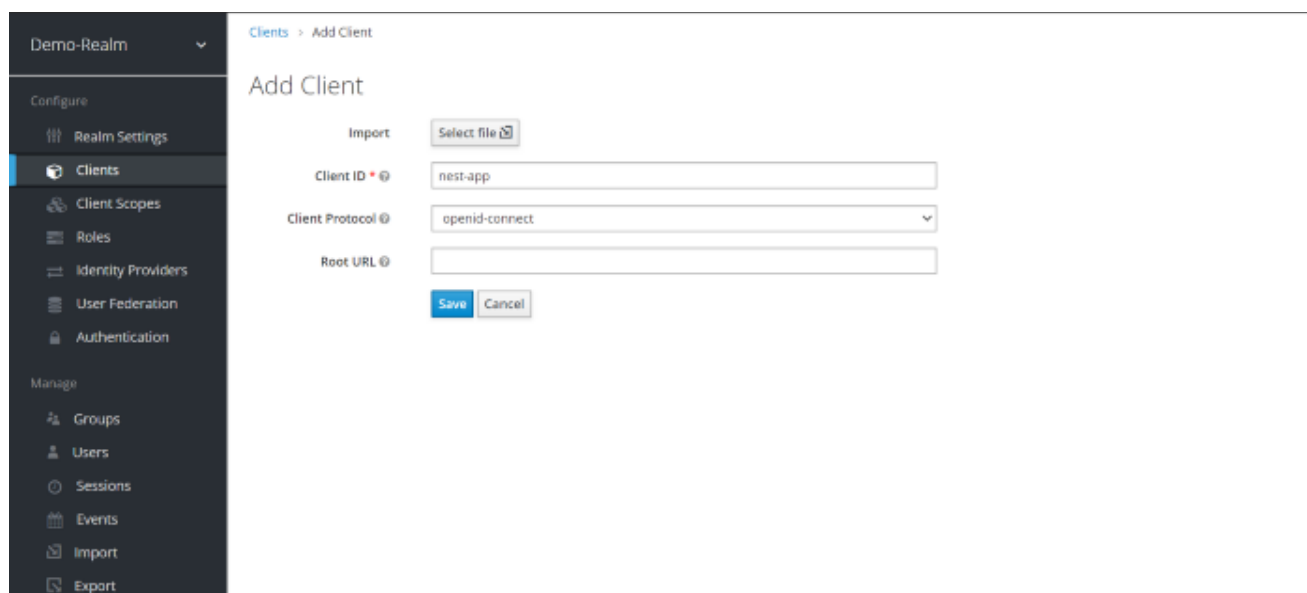


The screenshot shows the Keycloak Admin Console interface. On the left is a sidebar with a dark theme containing navigation links: Demo-Realm, Configure, Realm Settings, Clients (highlighted), Client Scopes, Roles, Identity Providers, User Federation, Authentication, Manage, Groups, Users, Sessions, Events, Import, and Export. The main content area is titled 'Clients' and includes a 'Lookup' button and a search bar. Below the search bar is a table listing existing clients.

Client ID	Enabled	Base URL	Actions		
account	True	http://localhost:8080/auth/realms/Demo-Realm/account/	Edit	Export	Delete
account-console	True	http://localhost:8080/auth/realms/Demo-Realm/account/	Edit	Export	Delete
admin-cli	True	Not defined	Edit	Export	Delete
broker	True	Not defined	Edit	Export	Delete
node-microservice	True	Not defined	Edit	Export	Delete
react-web-app	True	Not defined	Edit	Export	Delete
realm-management	True	Not defined	Edit	Export	Delete
security-admin-console	True	http://localhost:8080/auth/admin/Demo-Realm/console/	Edit	Export	Delete

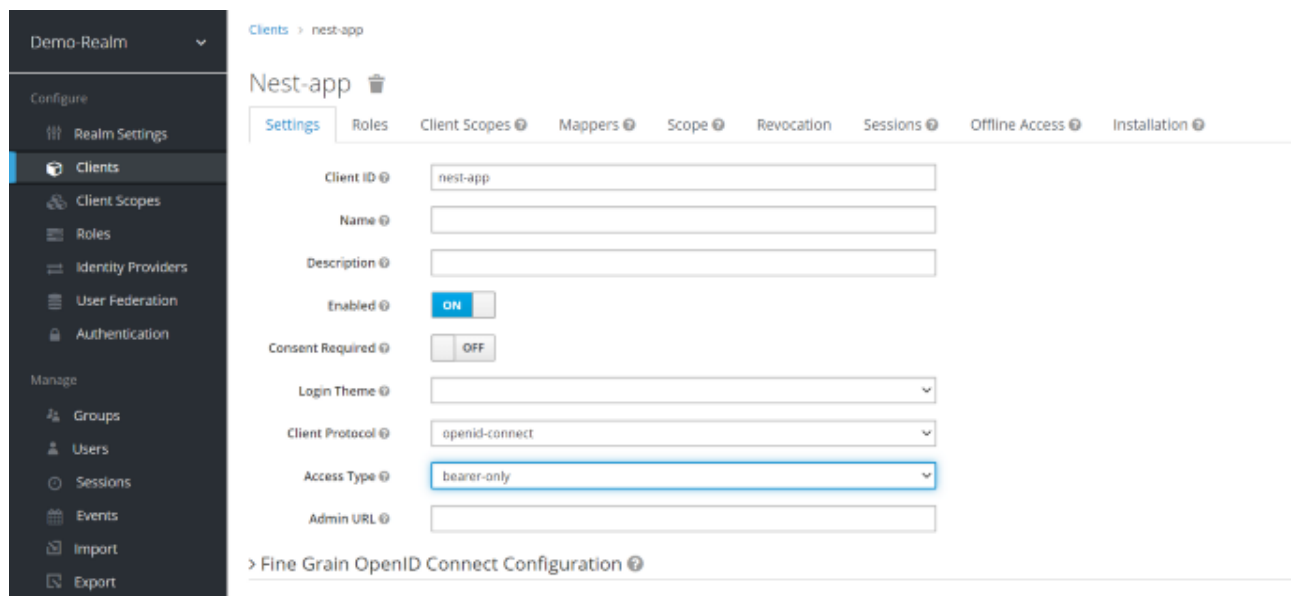
Here we have to register our NestJs application as a Keycloak client in Keycloak server.

Client : **nest-app**



The image shows the 'Add Client' form in the Keycloak administration console. The left sidebar contains the 'Demo-Realm' dropdown and a 'Configure' menu with options like 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', and 'Authentication'. The 'Clients' option is selected. The main area is titled 'Add Client' and includes an 'Import' button with a file icon, a 'Client ID' field with the value 'nest-app', a 'Client Protocol' dropdown set to 'openid-connect', and an empty 'Root URL' field. At the bottom are 'Save' and 'Cancel' buttons.

For the created client set the **Access Type** as bearer-only



The image shows the configuration page for the 'Nest-app' client in the Keycloak administration console. The left sidebar is the same as in the previous image. The main area has a breadcrumb 'Clients > nest-app' and a title 'Nest-app' with a trash icon. Below the title are tabs for 'Settings', 'Roles', 'Client Scopes', 'Mappers', 'Scope', 'Revocation', 'Sessions', 'Offline Access', and 'Installation'. The 'Settings' tab is active. The form includes fields for 'Client ID' (nest-app), 'Name', 'Description', 'Enabled' (ON), 'Consent Required' (OFF), 'Login Theme', 'Client Protocol' (openid-connect), 'Access Type' (bearer-only), and 'Admin URL'. A link '> Fine Grain OpenID Connect Configuration' is at the bottom.

Access Types explained,

- **Bearer-only** — this is for services that rely solely on the bearer token included in the request and never initiate login on their own. It's typically used for securing the back-end.
- **Confidential** — clients of this type need to provide a secret in order to initiate the login process. Mostly used in OAuth client credential flow.

- **Public** — since we have no real way of hiding the secret in a JS-based browser app, this is what we need to stick with.

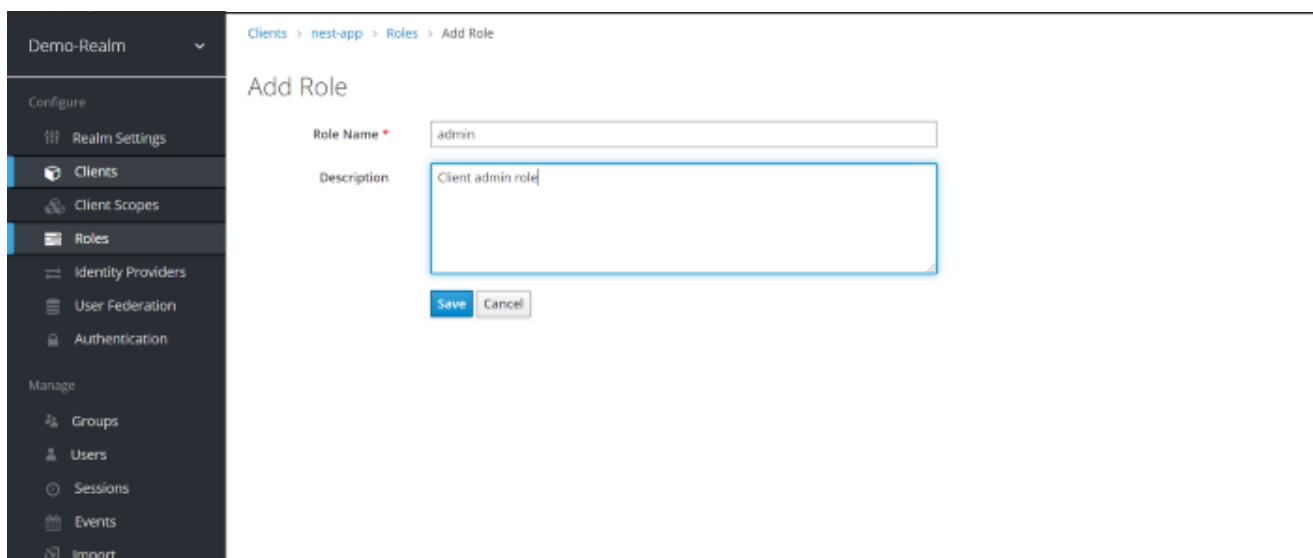
4. Create Roles

Roles identify a type or category of user. Admin, user, manager and employee are all typical roles that may exist in an organization. Applications often assign access and permissions to specific roles rather than individual users as dealing with users can be too fine grained and hard to manage. For example, the Admin Console has specific roles which give permission to users to access parts of the Admin Console UI and perform certain actions. There is a global namespace for roles and each client also has its own dedicated namespace where roles can be defined.

Realm Roles: Realm-level roles are a global namespace to define your roles. You can see the list of built-in and created roles by clicking the **Roles** left menu item.

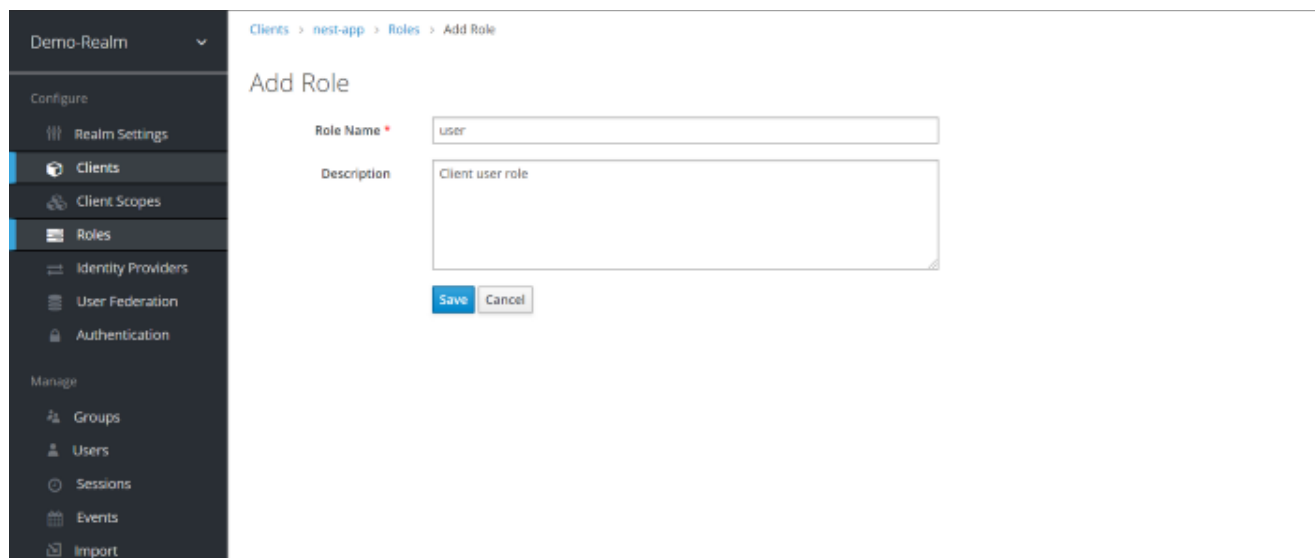
Client Roles: Client roles are basically a namespace dedicated to a client. Each client gets its own namespace. Client roles are managed under the **Roles** tab under each individual client. You interact with this UI the same way you do for realm-level roles.

1. Create client roles **admin** and **user** for **nest-app** client

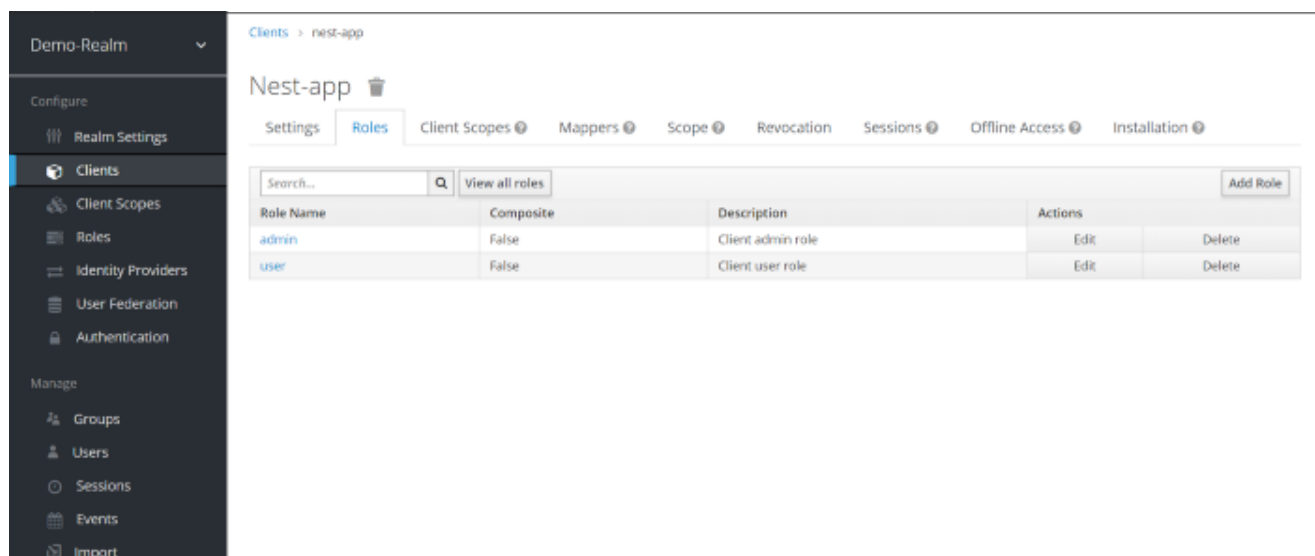


The screenshot shows the Keycloak Admin Console interface. On the left is a sidebar with a menu including 'Demo-Realm', 'Configure', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', 'Authentication', and 'Manage' (Groups, Users, Sessions, Events, Import). The main content area shows the breadcrumb 'Clients > nest-app > Roles > Add Role'. The 'Add Role' form has a 'Role Name' field containing 'admin' and a 'Description' field containing 'Client admin role'. At the bottom of the form are 'Save' and 'Cancel' buttons.

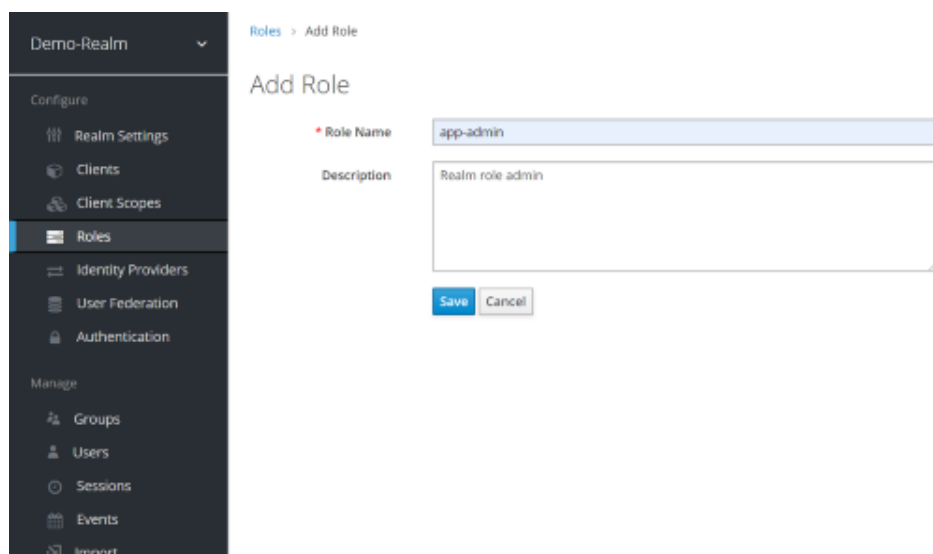
Client admin role



Client user role

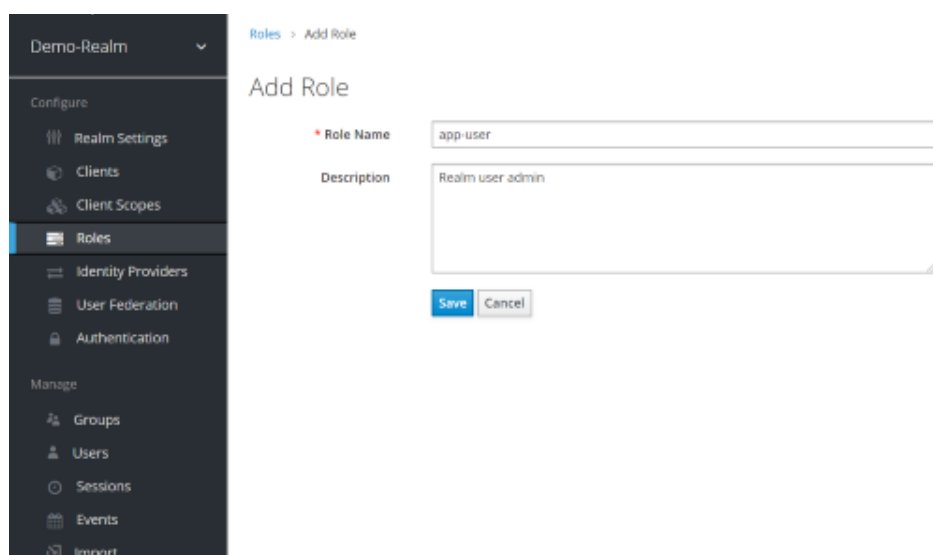


2. Create realm roles **app-admin** and **app-user** for **nest-app** client



The image shows the Keycloak administration interface for a 'Demo-Realm'. The left sidebar contains a 'Configure' section with options like 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', and 'Authentication'. The 'Roles' option is selected. The main area is titled 'Add Role' and contains two fields: 'Role Name' with the value 'app-admin' and 'Description' with the value 'Realm role admin'. At the bottom of the form are 'Save' and 'Cancel' buttons.

Realm admin role



The image shows the Keycloak administration interface for a 'Demo-Realm'. The left sidebar is the same as in the previous image, with 'Roles' selected. The main area is titled 'Add Role' and contains two fields: 'Role Name' with the value 'app-user' and 'Description' with the value 'Realm user admin'. At the bottom of the form are 'Save' and 'Cancel' buttons.

Realm user role

Composite Roles: Any realm or client level role can be turned into a composite role. A composite role is a role that has one or more additional roles associated with it. When a composite role is mapped to the user, the user also gains the roles associated with that composite. This inheritance is recursive so any composite of composites also gets inherited.

3. After saving the realm roles enable **Composite Roles** and search for client **react-web-app** in **Client Roles** field. Select admin role and click **Add selected**. This configuration will assign **nest-app**, **admin** client role to the **app-admin** realm role. If you have

multiple clients with multiple roles, pick and choose the required roles from each client to create realm roles based on the need.

The screenshot shows the Keycloak Admin Console interface for configuring the 'App-admin' role. The left sidebar is dark-themed and contains a 'Demo-Realm' dropdown, a 'Configure' section with links to Realm Settings, Clients, Client Scopes, Roles (highlighted), Identity Providers, User Federation, and Authentication, and a 'Manage' section with links to Groups, Users, Sessions, Events, Import, and Export. The main content area is titled 'Roles > app-admin' and 'App-admin'. It has three tabs: 'Details' (active), 'Attributes', and 'Users in Role'. The 'Details' tab shows the 'Role Name' as 'app-admin' and the 'Description' as 'Realm admin role'. There is a 'Composite Roles @' toggle set to 'ON' and 'Save' and 'Cancel' buttons. Below this is a 'Composite Roles' section with two parts: 'Realm Roles' and 'Client Roles'. The 'Realm Roles' part shows 'Available Roles @' (app-user, offline_access, uma_authorization) and an empty 'Associated Roles @' box. The 'Client Roles' part shows 'nest-app' selected from a dropdown, 'Available Roles @' (user), and an 'Associated Roles @' box with 'admin' selected.

4. Similarly add **user** client role to **app-user** realm role.

The screenshot shows the Keycloak Admin Console interface for configuring the 'App-user' role. The left sidebar is the same as the previous screenshot. The main content area is titled 'Roles > app-user' and 'App-user'. It has three tabs: 'Details' (active), 'Attributes', and 'Users in Role'. The 'Details' tab shows the 'Role Name' as 'app-user' and the 'Description' as 'Realm user role'. There is a 'Composite Roles @' toggle set to 'ON' and 'Save' and 'Cancel' buttons. Below this is a 'Composite Roles' section with two parts: 'Realm Roles' and 'Client Roles'. The 'Realm Roles' part shows 'Available Roles @' (app-admin, offline_access, uma_authorization) and an empty 'Associated Roles @' box. The 'Client Roles' part shows 'nest-app' selected from a dropdown, 'Available Roles @' (admin), and an 'Associated Roles @' box with 'user' selected.

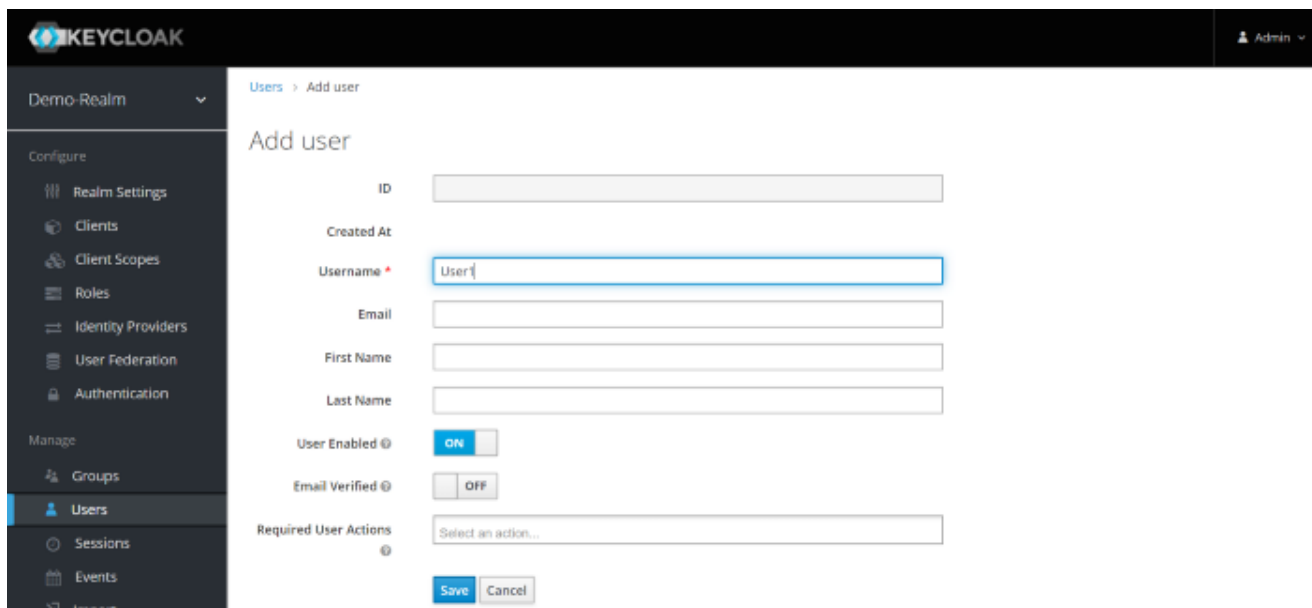
5. Create Users

Users are entities that are able to log into your system. They can have attributes associated with themselves like email, username, address and phone number. They can be assigned to groups and have specific roles assigned to them.

1. Create three users and assign them following realm roles,

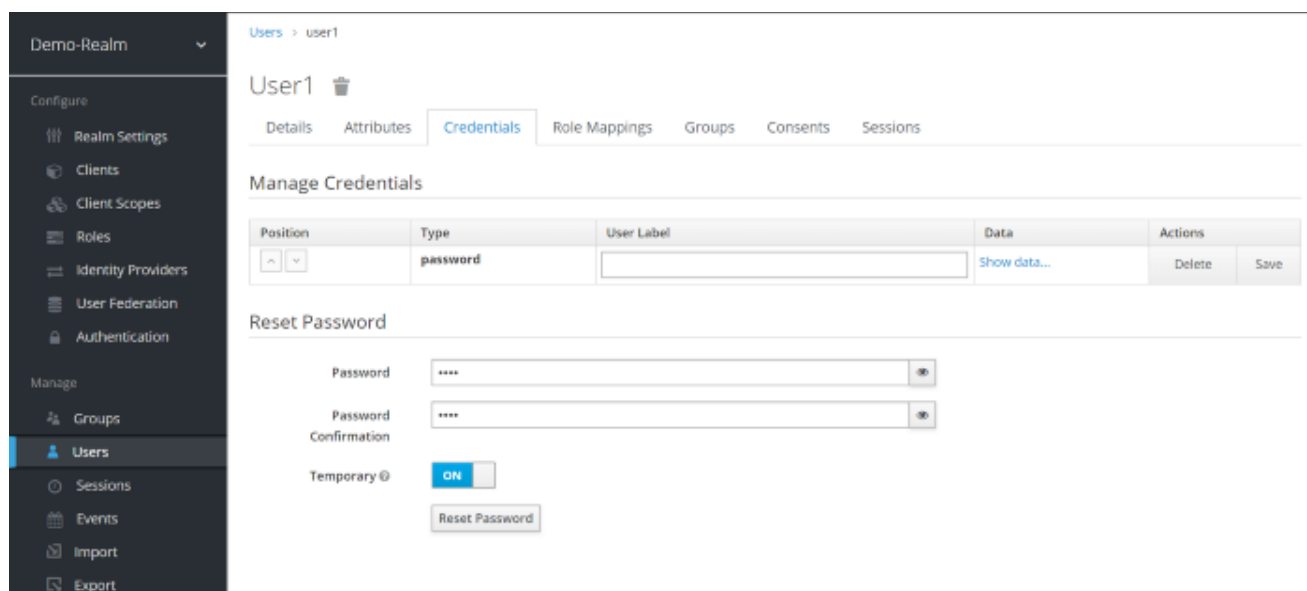
- **User1** with **app-user**
- **User2** with **app-admin**
- **User3** with **app-user, app-admin**

Go to **Users** in left side menu to create users

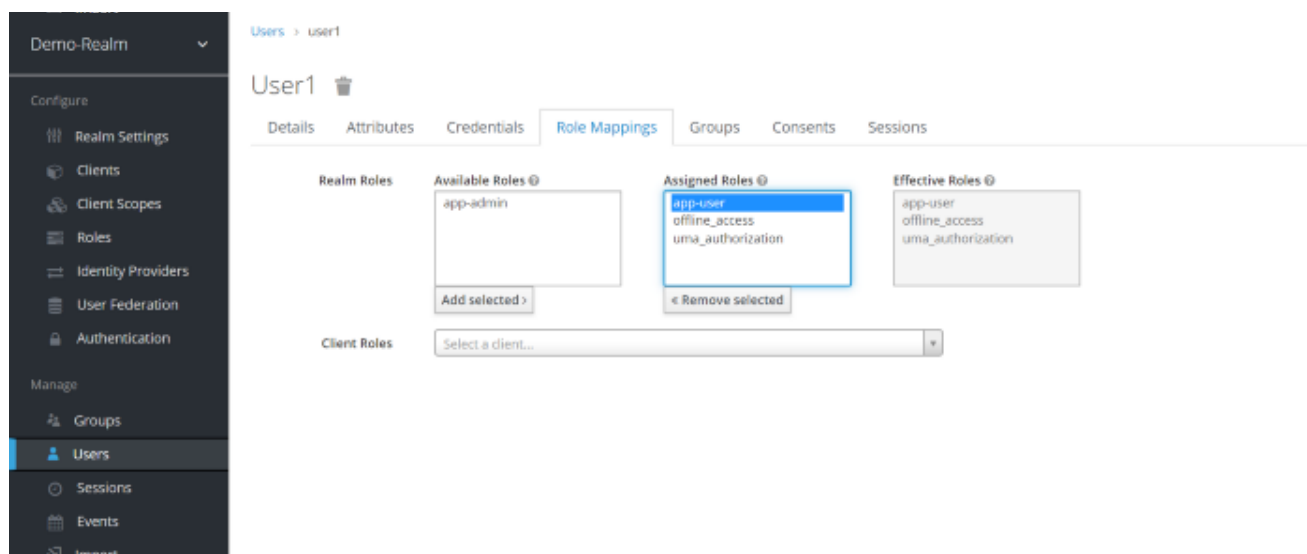


The screenshot shows the Keycloak Admin Console interface. On the left is a dark sidebar with the 'KEYCLOAK' logo at the top. Below the logo, there's a 'Demo-Realm' dropdown. The sidebar is divided into 'Configure' (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication) and 'Manage' (Groups, Users, Sessions, Events, Import). The 'Users' option is highlighted. The main content area is titled 'Add user' and contains several input fields: 'ID', 'Created At', 'Username' (with a red asterisk and the text 'User' entered), 'Email', 'First Name', and 'Last Name'. Below these are toggle switches for 'User Enabled' (set to ON) and 'Email Verified' (set to OFF). There is also a 'Required User Actions' dropdown menu showing 'Select an action...'. At the bottom are 'Save' and 'Cancel' buttons. The top right corner of the console shows a user profile icon and the text 'Admin'.

2. Go to Credentials tab to set user credentials.



3. From the **Role Mappings** tab assign required roles to the Users.



Like above assign **app-user** role to User2 and **app-user**, **app-admin** roles to User3

Application Configuration (NestJs)

I am assuming that you have basic knowledge on how to create a NestJs application with basic setup. We are using **nest-keycloak-connect** adaptor to communicate with Keycloak server.

Keycloak Client Adapters : Keycloak client adapters are libraries that make it very easy to secure applications and services with Keycloak. We call these 'adapters' rather than

libraries as they provide a tight integration to the underlying platform and framework. This makes adapters easy to use and require less boilerplate code than what is typically required by a library.

Implementation

- Add nest Keycloak library to our code base

```
npm install nest-keycloak-connect --save
```

- Now we need to add Keycloak configuration to NestJs. Here we are adding configuration to the app.module.ts. But if you want you can create a separate module for Keycloak configuration and import that module in app.module.ts.

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import {
  KeycloakConnectModule,
  ResourceGuard,
  RoleGuard,
  AuthGuard,
} from 'nest-keycloak-connect';
import { APP_GUARD } from '@nestjs/core';

@Module({
  imports: [
    KeycloakConnectModule.register({
      authServerUrl: 'http://localhost:8080/auth',
      realm: 'Demo-Realm',
      clientId: 'nest-app',
      secret: '83790b4f-48cd-4b6c-ac60-451a918be4b9',
      // Secret key of the client taken from keycloak server
    }),
  ],
  controllers: [AppController],
  providers: [
    AppService,
    // This adds a global level authentication guard,
    // you can also have it scoped
    // if you like.
    //
    // Will return a 401 unauthorized when it is unable to
    // verify the JWT token or Bearer header is missing.
  ],
})
```

```

        provide: APP_GUARD,
        useClass: AuthGuard,
      },
      // This adds a global level resource guard, which is permissive.
      // Only controllers annotated with @Resource and
      // methods with @Scopes
      // are handled by this guard.
      {
        provide: APP_GUARD,
        useClass: ResourceGuard,
      },
      // New in 1.1.0
      // This adds a global level role guard, which is permissive.
      // Used by `@Roles` decorator with the
      // optional `@AllowAnyRole` decorator for allowing any
      // specified role passed.
      {
        provide: APP_GUARD,
        useClass: RoleGuard,
      },
    ],
  },
})
export class AppModule {}

```

- Here we are using nest-keycloak-connect's **AuthGuard**, **ResourceGuard**, **RoleGuard** to protect our endpoints.
- Now we add following endpoints to our controller and start the application using npm start.

```

@Controller()
export class UserController {
  constructor(private readonly userService: UserService) {}

  @Get()
  getpublic(): string {
    return `${this.userService.getHello()} from public`;
  }

  @Get('/user')
  getUser(): string {
    return `${this.userService.getHello()} from user`;
  }

  @Get('/admin')
  getAdmin(): string {
    return `${this.userService.getHello()} from admin`;
  }
}

```

```
    }

    @Get('/all')
    getAll(): string {
      return `${this.userService.getHello()} from all`;
    }
  }
}
```

- You can try to access those endpoints from postman or a browser. You can see you are getting `{“statusCode”:401,”message”:”Unauthorized”}` from those endpoints.
- To give access those endpoints we can use following annotations.

```
@Controller()
export class UserController {
  constructor(private readonly userService: UserService) {}

  @Get('/public')
  @Unprotected()
  getpublic(): string {
    return `${this.userService.getHello()} from public`;
  }

  @Get('/user')
  @Roles('user')
  getUser(): string {
    return `${this.userService.getHello()} from user`;
  }

  @Get('/admin')
  @Roles('admin')
  getAdmin(): string {
    return `${this.userService.getHello()} from admin`;
  }

  @Get('/all')
  @AllowAnyRole()
  getAll(): string {
    return `${this.userService.getHello()} from all`;
  }
}
```

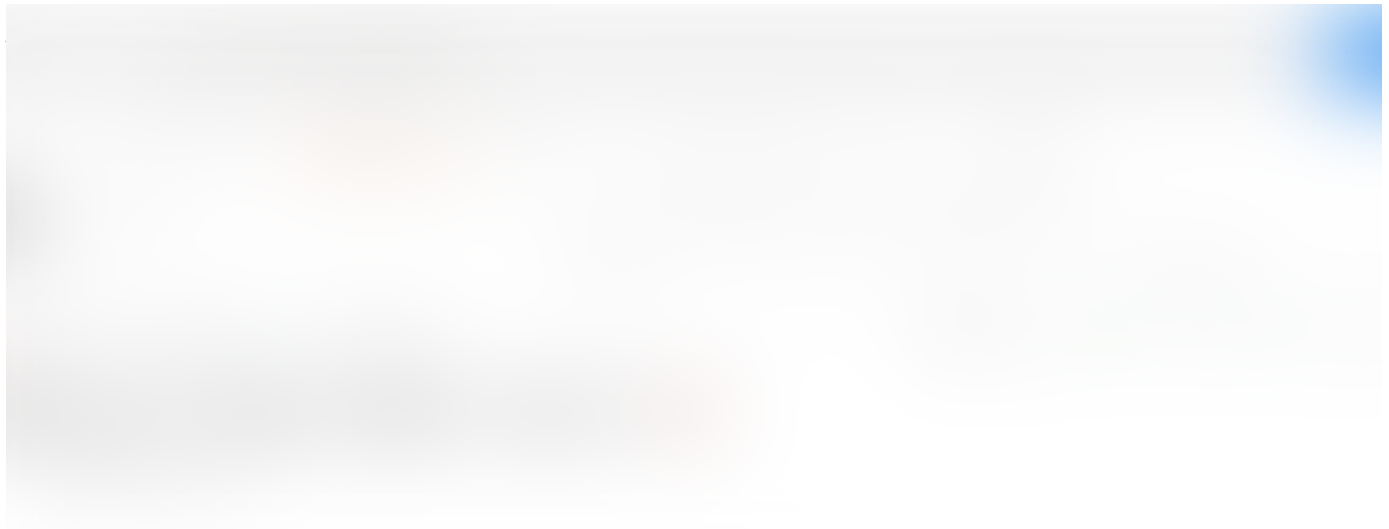
- This is the extracted JWT received from frontend application from **User1** we created

when configuring Keycloak. In the resource access section you can see this that user has **user** client role and in the realm access section user has **app-user** realm role. We can also use realm role in the @Roles annotation. e.g : @Roles('realm:app-user'). In above example we used client role.



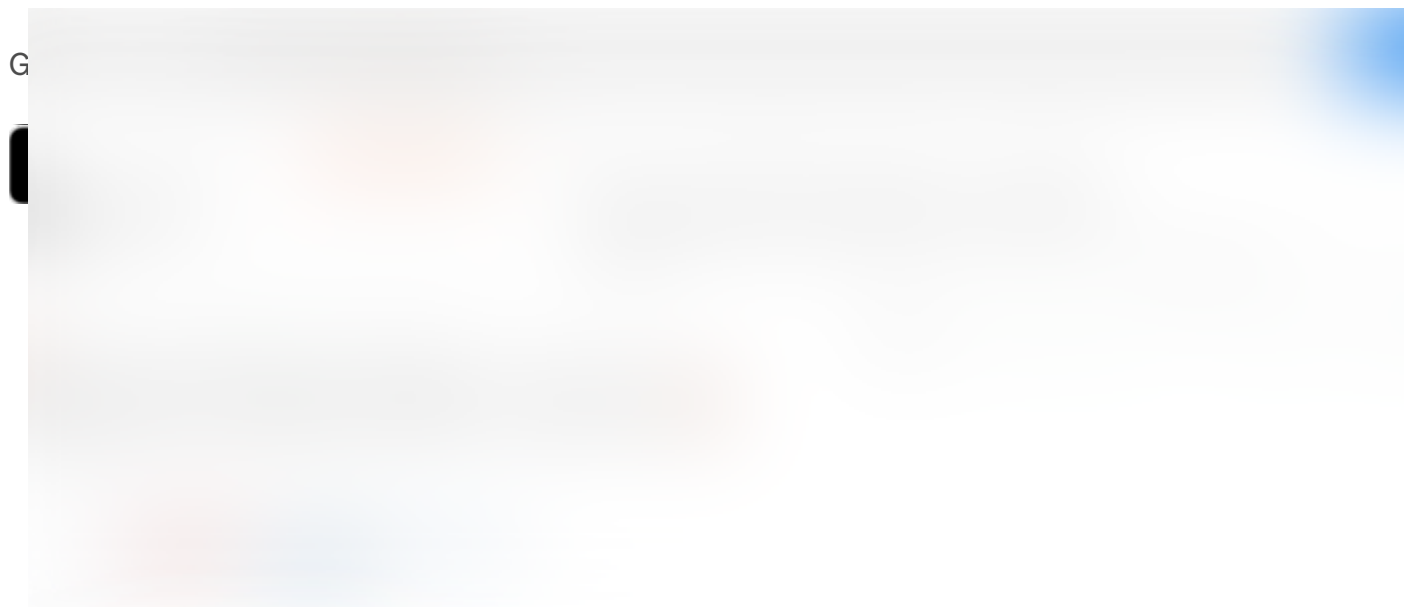
- Now we can access the **/user** endpoint with postman with JWT attached in the Authorization header.

Volla, we have successfully secured our NestJs REST API using Keycloak.



- But if we try to access **/admin** endpoint we are getting 403 because **User1** doesn't have

About Admin Help Legal



- We can access access the **/all** endpoint since it's allowed for any roles.

