

Authentication with AWS Cognito and NestJS



Jacob Do

Jun 8 · 4 min read ★



AWS Cognito

If you have landed here after a couple of incomplete guides on how to implement AWS Cognito into a NestJS application, then I hope you're in luck as I will try to fill in the gaps you might have.

Let us start from the very beginning.

Set up AWS Cognito account

Head over to <https://aws.amazon.com/cognito/> and sign in with your AWS account.

Then type in Cognito in the search box and click on the only search result there:

AWS services

Find Services

You can enter names, keywords or acronyms.



Cognito

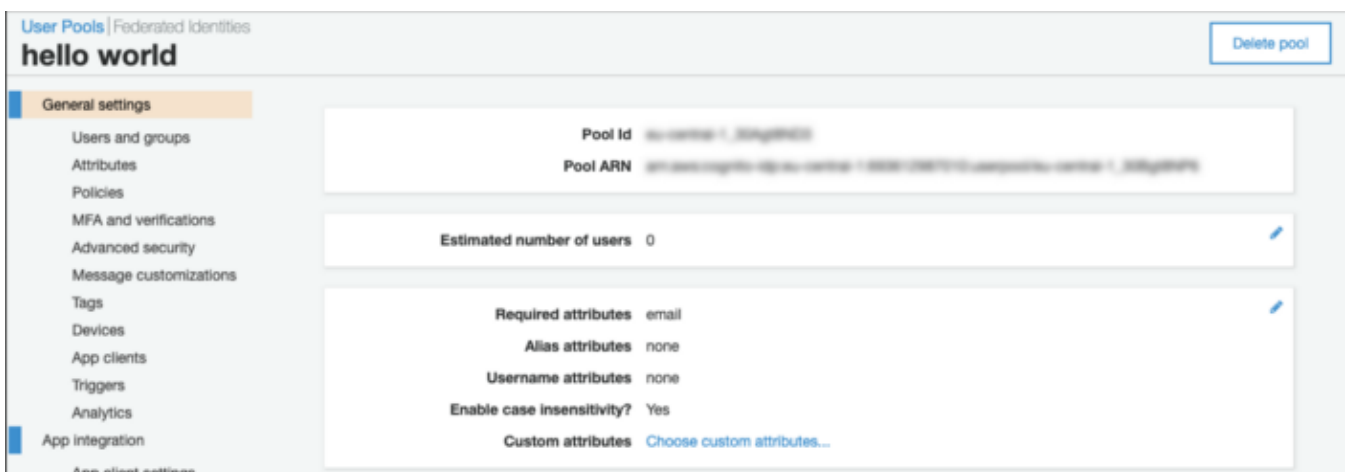
User Identity and App Data Synchronization

Finding Cognito

Then:

- Press “Manage user pools”
- Press “Create a user pool”
- Enter the name of your user pool (the name of your project will do)
- Press “Review defaults” as the other option takes longer and we really want to just get through this bit
- Finally, press “Create pool”

If all went well, you should be presented with a screen like this:



Cognito default dashboard

From here, find and click “App clients” in the sidebar.

- Press “Add app client”
- Enter the name of the app client, say “My project’s API”
- **IMPORTANT:** untick “Generate client secret”, otherwise this App client will not be usable in a NestJS application
- Then scroll to the bottom of the page and press “Create app client”

Again, if you leapt through the hoops with me, you should see the following:

The screenshot shows the AWS IAM console interface for configuring app clients. On the left, a sidebar lists navigation options: General settings, Users and groups, Attributes, Policies, MFA and verifications, Advanced security, Message customizations, Tags, Devices, App clients (highlighted), Triggers, Analytics, App integration, App client settings, and Domain name. The main panel is titled 'Which app clients will have access to this user pool?' and includes a sub-header: 'The app clients that you add below will be given a unique ID and an optional secret key to access this user pool.' Below this, there is a form for adding a new app client. The form contains the following fields: 'Name' (with the value 'Hello world API'), 'App client id' (with a generated ID), 'App client secret' (set to '(no secret key)'), and 'Refresh token expiration (days)' (set to '30').

App client information

From here we will use the “App client id” value, but we will come back for it later.

Create auth module

Next up, we need to create an auth module that will take care of all things related to our authentication. To do that, start by running the following command at the root of your NestJS project:

```
nest g module auth
```

This command will create a file called `auth.module.ts` inside of `src/auth`.

Then we’ll create a file called `auth.config.ts` inside of `src/auth`. This file will hold all the values that we need to communicate with AWS Cognito:

```
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class AuthConfig {
5   public userPoolId: string = process.env.COGNITO_USER_POOL_ID;
6   public clientId: string = process.env.COGNITO_CLIENT_ID;
7   public region: string = process.env.COGNITO_REGION;
8   public authority = `https://cognito-idp.${process.env.COGNITO_REGION}.amazonaws.com/${process.env.COGNITO_USER_POOL_ID}`;
9 }
```

We also need to provide `node-fetch` in the global context of the app, so our `main.ts` file should look like this:

```
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app.module';
3
4 global['fetch'] = require('node-fetch');
5
6 async function bootstrap() {
7   const app = await NestFactory.create(AppModule);
8   await app.listen(3000);
9 }
10 bootstrap();
```

main.ts hosted with ❤ by GitHub

[view raw](#)

Adding node-fetch to main.ts

We will be using environment variables here and while setting up using them is not directly related to the task at hand, it is the prudent thing to do, so we'll quickly look at how to do that.

In order to use `.env` in our NestJS application, we first need to install `nestjs/config`:

```
yarn add @nestjs/config
```

When that's done, let's create a `.env` file at the root of the application and also enable our app to access values from the file by importing and setting up `ConfigModule` in `app.module.ts` as follows:

```
1 import { Module } from '@nestjs/common';
2 import { AuthModule } from './auth/auth.module';
3 import { ConfigModule } from '@nestjs/config';
4
5 @Module({
6   imports: [
7     AuthModule,
8     ConfigModule.forRoot({
9       isGlobal: true,
10     }),
11   ],
12 })
```

```
13 export class AppModule {}
```

app.module.ts hosted with ❤ by GitHub

[view raw](#)

Importing ConfigModule

Now our `.env` file is accessible in our app.

Next, let's add the following content to your `.env` file:

```
1 COGNITO_USER_POOL_ID=eu-central-1_AadYWQBJE
2 COGNITO_CLIENT_ID=1bk2sqm1453tu5g7ldkg5vk23u
3 COGNITO_REGION=eu-central-1
```

`.env` hosted with ❤ by GitHub

[view raw](#)

Declaring Cognito .env variables

`COGNITO_USER_POOL_ID` can be found in Cognito User Pools -> General Settings:



Pool Id eu-central-1_AadYWQBJE

Cognito user pool id

`COGNITO_CLIENT_ID` is the “App client id” I mentioned before in the article when we were creating our Cognito user pool.

Finally, `COGNITO_REGION` is just the first part of your `COGNITO_USER_POOL_ID`. I’ve broken it out into its own variable just to be able to re-use it conveniently.

For the next bit, we also need to install `amazon-cognito-identity-js`:

```
yarn add amazon-cognito-identity-js
```

Next, in order to be able to use our `AuthConfig` inside of `AuthService`, we need to declare the former as a provider inside of `auth.module.ts`:

```
1 import { Module } from '@nestjs/common';
2 import { AuthService } from './auth.service';
3 import { AuthConfig } from './auth.config';
4
5 @Module({
```

```
6   providers: [AuthService, AuthConfig]
7 })
8 export class AuthModule {}
```

auth.module.ts hosted with ❤ by GitHub

[view raw](#)

Making AuthConfig available in AuthModule

Now that that's out of the way, let's create our `AuthService` by running:

```
nest g service auth
```

```
1  import { AuthConfig } from './auth.config';
2  import { Inject, Injectable } from '@nestjs/common';
3  import {
4    AuthenticationDetails,
5    CognitoUser,
6    CognitoUserPool,
7  } from 'amazon-cognito-identity-js';
8
9  @Injectable()
10 export class AuthService {
11   private userPool: CognitoUserPool;
12   constructor(
13     @Inject('AuthConfig')
14     private readonly authConfig: AuthConfig,
15   ) {
16     this.userPool = new CognitoUserPool({
17       UserPoolId: this.authConfig.userPoolId,
18       ClientId: this.authConfig.clientId,
19     });
20   }
21
22   authenticateUser(user: { name: string; password: string }) {
23     const { name, password } = user;
24
25     const authenticationDetails = new AuthenticationDetails({
26       Username: name,
27       Password: password,
28     });
29     const userData = {
30       Username: name,
31       Pool: this.userPool,
32     };
33   }
```

```

34     const newUser = new CognitoUser(userData);
35
36     return new Promise((resolve, reject) => {
37         return newUser.authenticateUser(authenticationDetails, {
38             onSuccess: result => {
39                 resolve(result);
40             },
41             onFailure: err => {
42                 reject(err);
43             },
44         });
45     });
46 }
47 }

```

auth service is hosted with  by GitHub

[view raw](#)

authenticateUser method in AuthService

Finally, in order to be able to test out our functionality, we need to create a controller that will use the above service. We can create our `AuthController` by running:

```
nest g controller auth
```

Our controller will be simple and only have one method `login` , that will handle the `/auth/login` route:

```

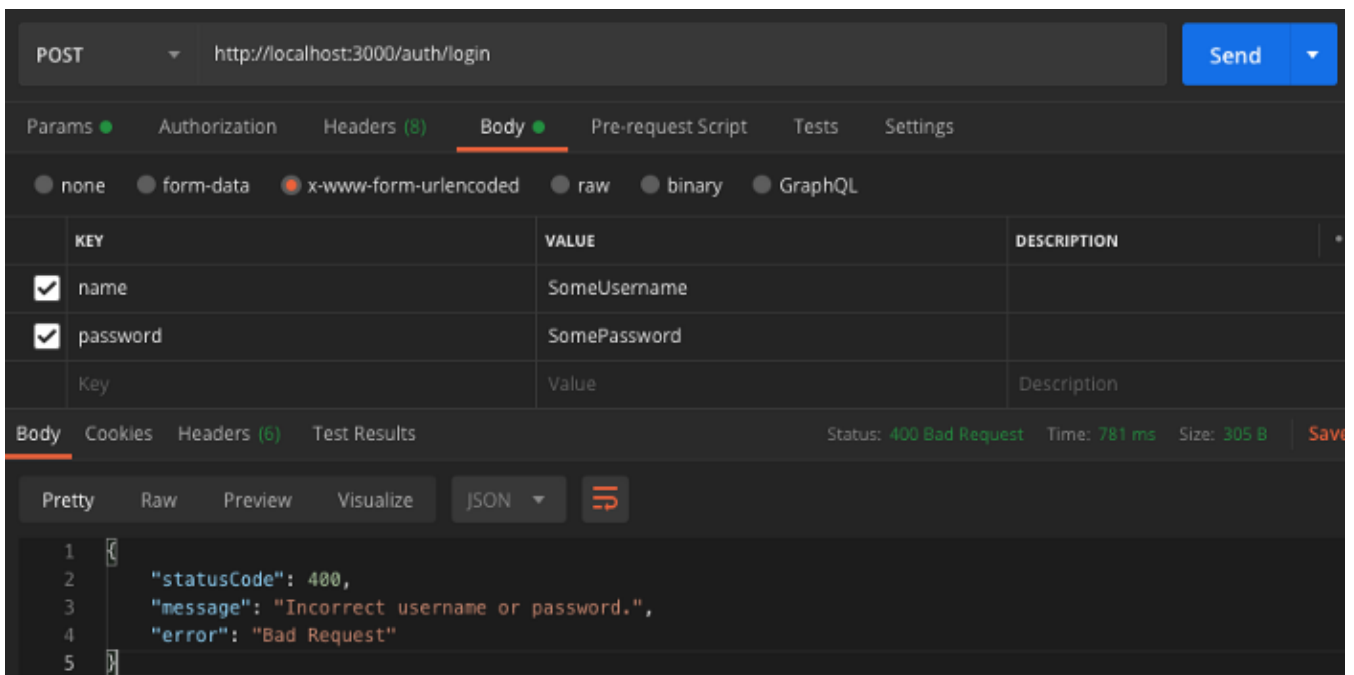
1  import { BadRequestException, Body, Controller, Post } from '@nestjs/common';
2  import { AuthService } from '../auth.service';
3
4
5  @Controller('auth')
6  export class AuthController {
7      constructor(private readonly authService: AuthService) {}
8
9      @Post('login')
10     async login(@Body() authenticateRequest: { name: string, password: string }) {
11         try {
12             return await this.authService.authenticateUser(authenticateRequest);
13         } catch (e) {
14             throw new BadRequestException(e.message);
15         }
16     }
17 }

```

In here, we simply call the `authenticateUser` method from `AuthService` and return the result as well as we catch any exceptions that return a `BadRequestException` along with the actual error message so that we can inform the consumer what exactly went wrong.

Now the only thing that is left is to test it out, I will use Postman <https://www.postman.com/downloads/> for this part and you're welcome to do the same.

If you got the following response for the request visible in the screenshot, it means that you have successfully attempted to verify credentials against your AWS Cognito user pool:



Validation response from AWS Cognito

A working GitHub example for the code produced in the course of this article can be found here: <https://github.com/jacobdo2/nestjs-cognito-example>

Hope it works for you 🙏 and let me know in the comments if it does not — we'll figure it out!

By Weekly Webtips

Get the latest news on the world of web technologies with a series of tutorial [Take a look](#)


Get this newsletter


Emails will be sent to zondagh@gmail.com.
[Not you?](#)

[AWS](#) [Cognito](#) [Nestjs](#) [Nestjs Tutorial](#) [Typescript](#)

[About](#) [Help](#) [Legal](#)

Get the Medium app

 A button that says 'Download on the App Store', and if clicked it will lead you to the iOS App store

 A button that says 'Get it on, Google Play', and if clicked it will lead you to the Google Play store