For the purposes of this sample setup, I have partially implemented the Fhir patient and fhir practitioner resources in dynamodb. These can be accessed through api gateways, and are authenticated by Cognito.

**Reading fhir spec:**

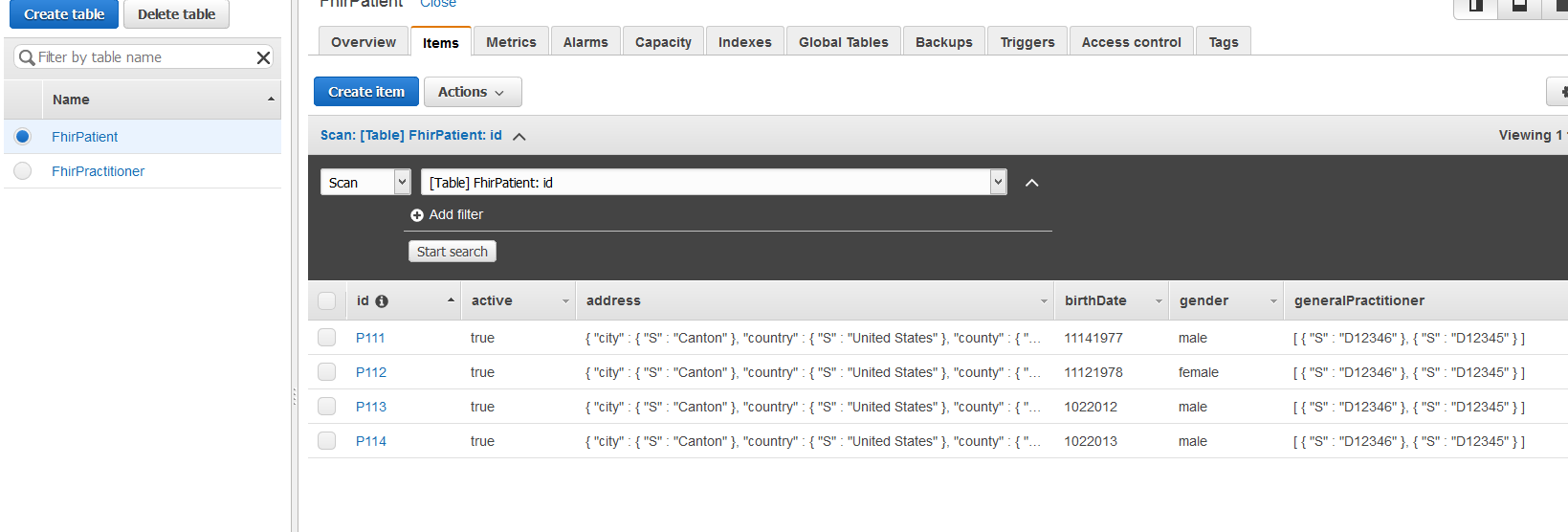
<https://www.hl7.org/fhir/patient.html>

As per standard spec, 0..\* signifies an optional field, of which there can be many. 0..1 signifies a single optional field, 1..1 signifies a required single field, and so on.

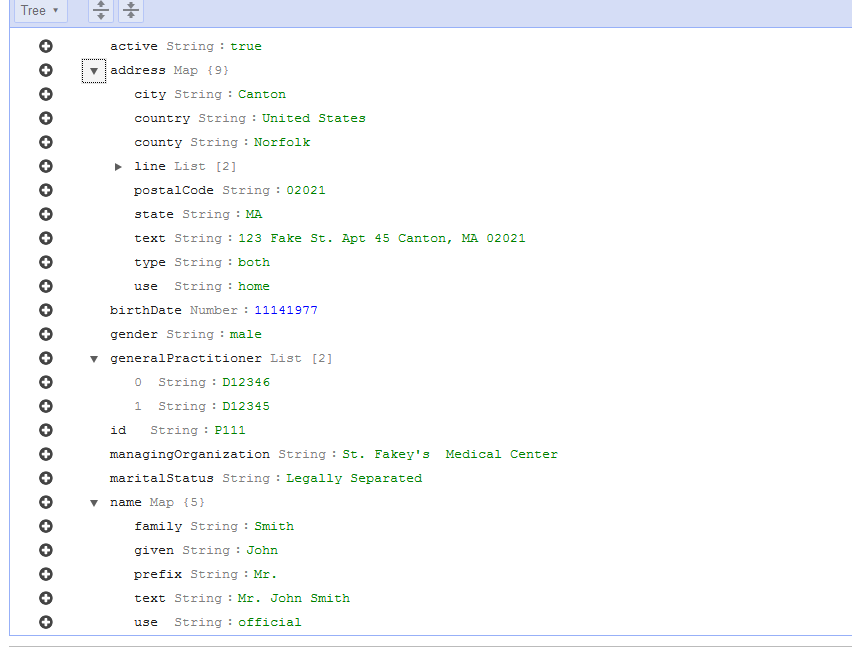
In this setup, the patient references the practitioner through the optional list of generalPractioners.

**Dynamodb setup:**

There are two tables, FhirPatient and FhirPractitioner.

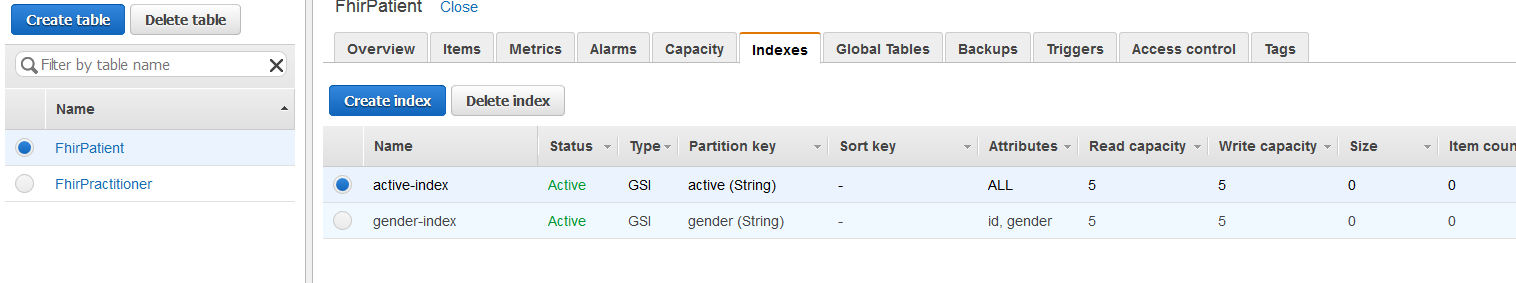


Each patient has the following schema:



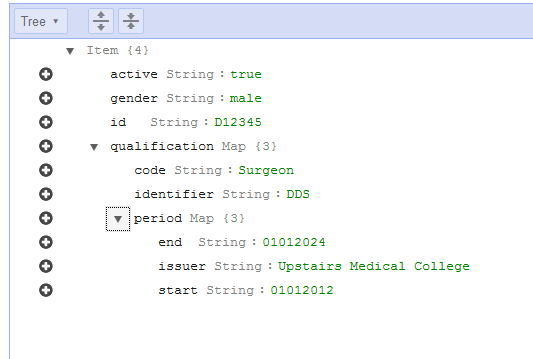
The primary key is id, same as Practitioner, and most api available resources in fhir. There is no sort key, since one should not be needed to retrieve a unique resource.

There are two indexes on the patient table: active and gender.



The practitioner is referenced as above.

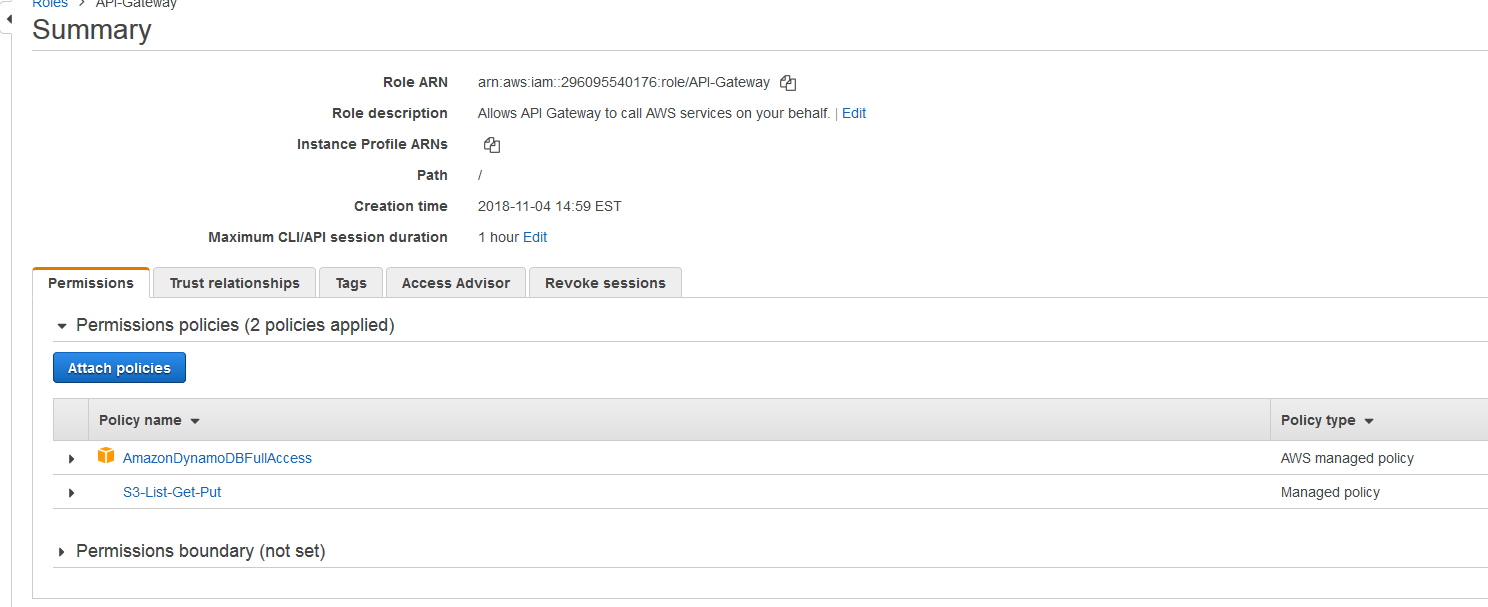
The practitioner schema is as such:



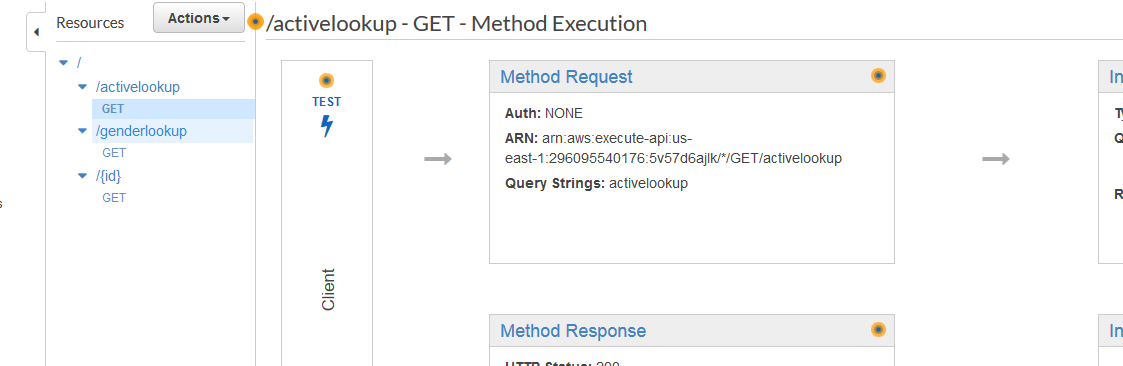
Next, these tables and data need to be hooked up to api gateways.

**API Gateway Setup**

We first need an IAM role set up to allow access to the db actions:

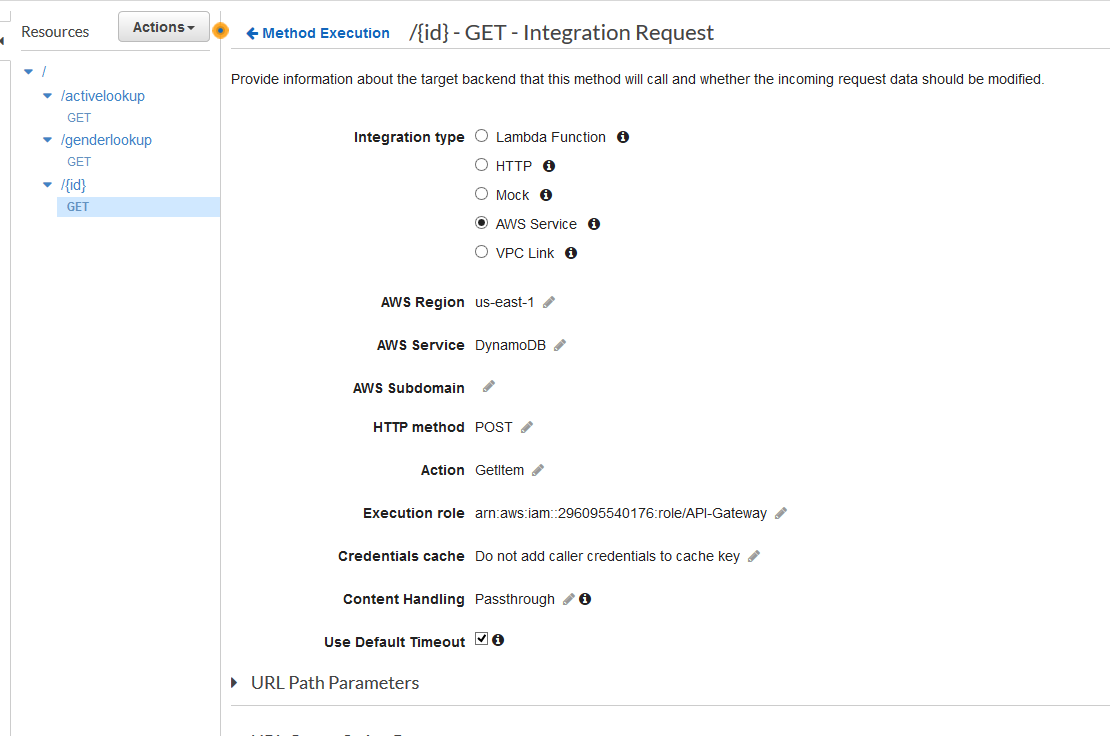


This will grant the api gateway access to manage all db requests. For more proper setup with reads and writes, there could be separate roles with readonly and writeonly access, or even access to only specific tables.

While every command specified in this project is a GET (get patient resource, get list of patient resources, get practitioner, etc), the actual api calls to dynamodb are POST, as in “post the request and receive the request result”.

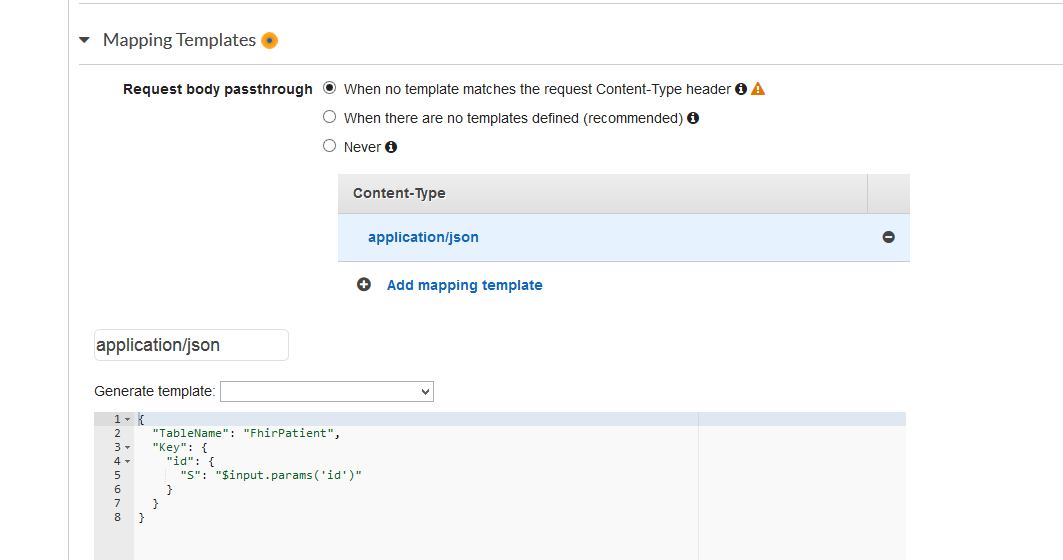
As such, both the actions and http methods are defined as post, though valid setup could be either, and for a larger setup with actual data posting, these api gateway actions would be defined as GET for better clarity. Only the http method in the integration request needs to be POST.

Most of the logic happens in the integration requests:



We specify the dynamodb aws service, region in which the data exists for better efficiency, the POST action as stated earlier, GetItem action to specify we want a single resource returned, the setup gateway IAM role set up for these apis, and passthrough handling.

Before anything will work, however, mapping templates need to be defined to specify the data and table.

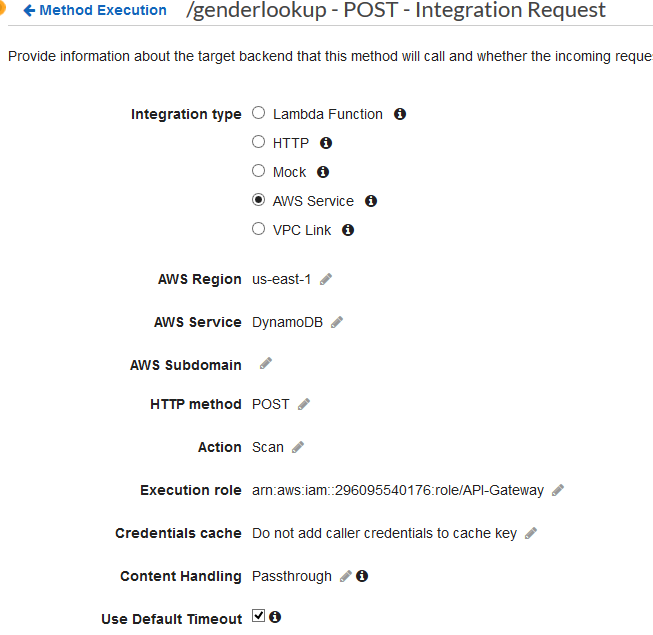
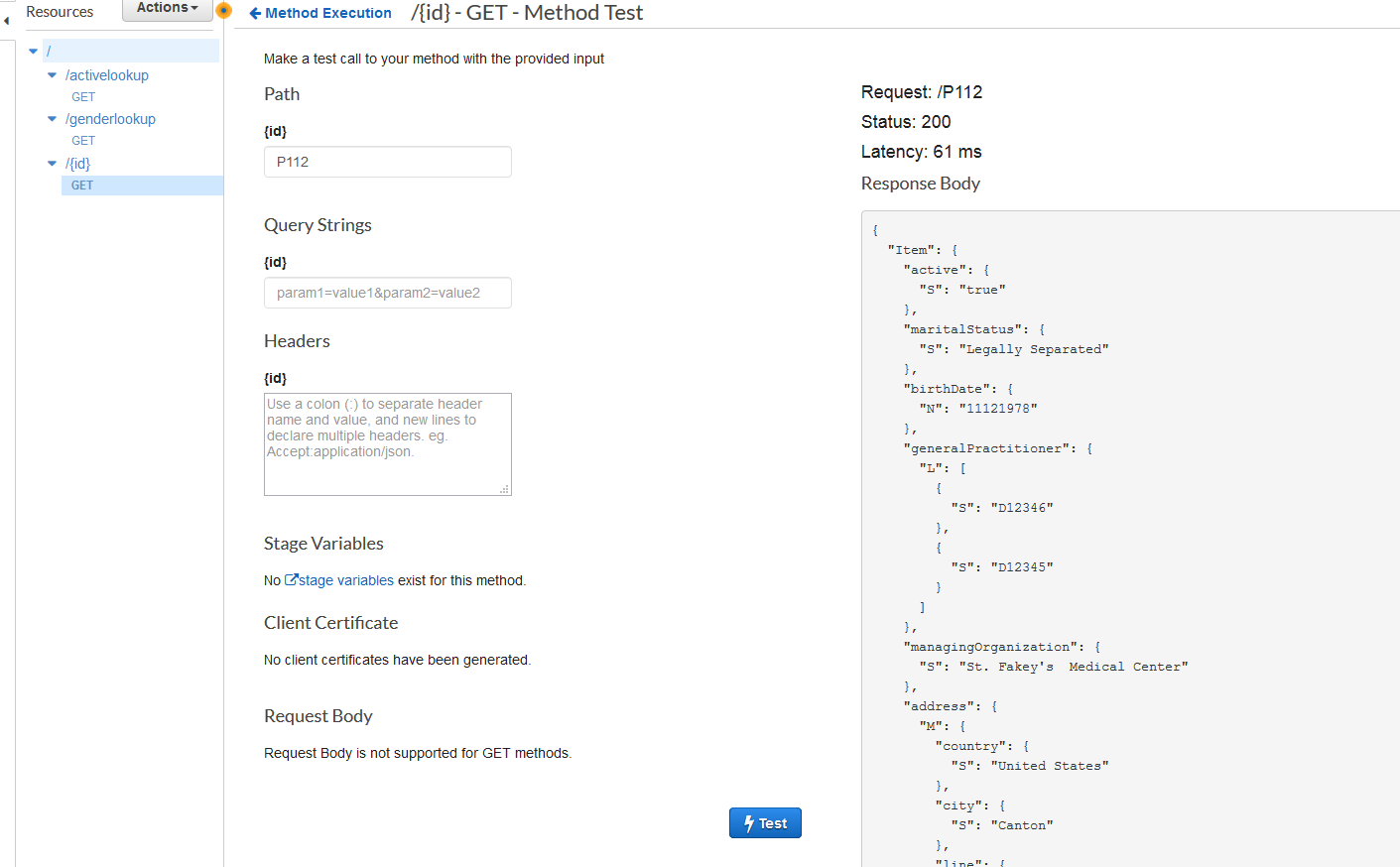


This is posted through the dynamodb api, and returns the resource at the passed in key.

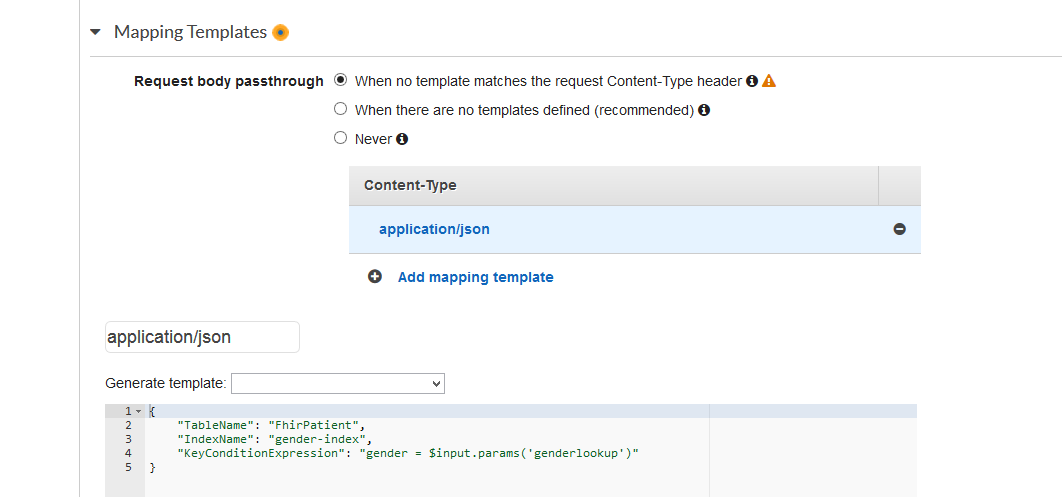
To test:

The same setup applies to the practitioner table, except with the TableName in the mapping set to FhirPractitioner.

For indexes, the setup changes. The method and most other configuration is the same, however the action is now Scan to signify returning a list of data:



and the mapping specifies a key condition search, with the given table and index:



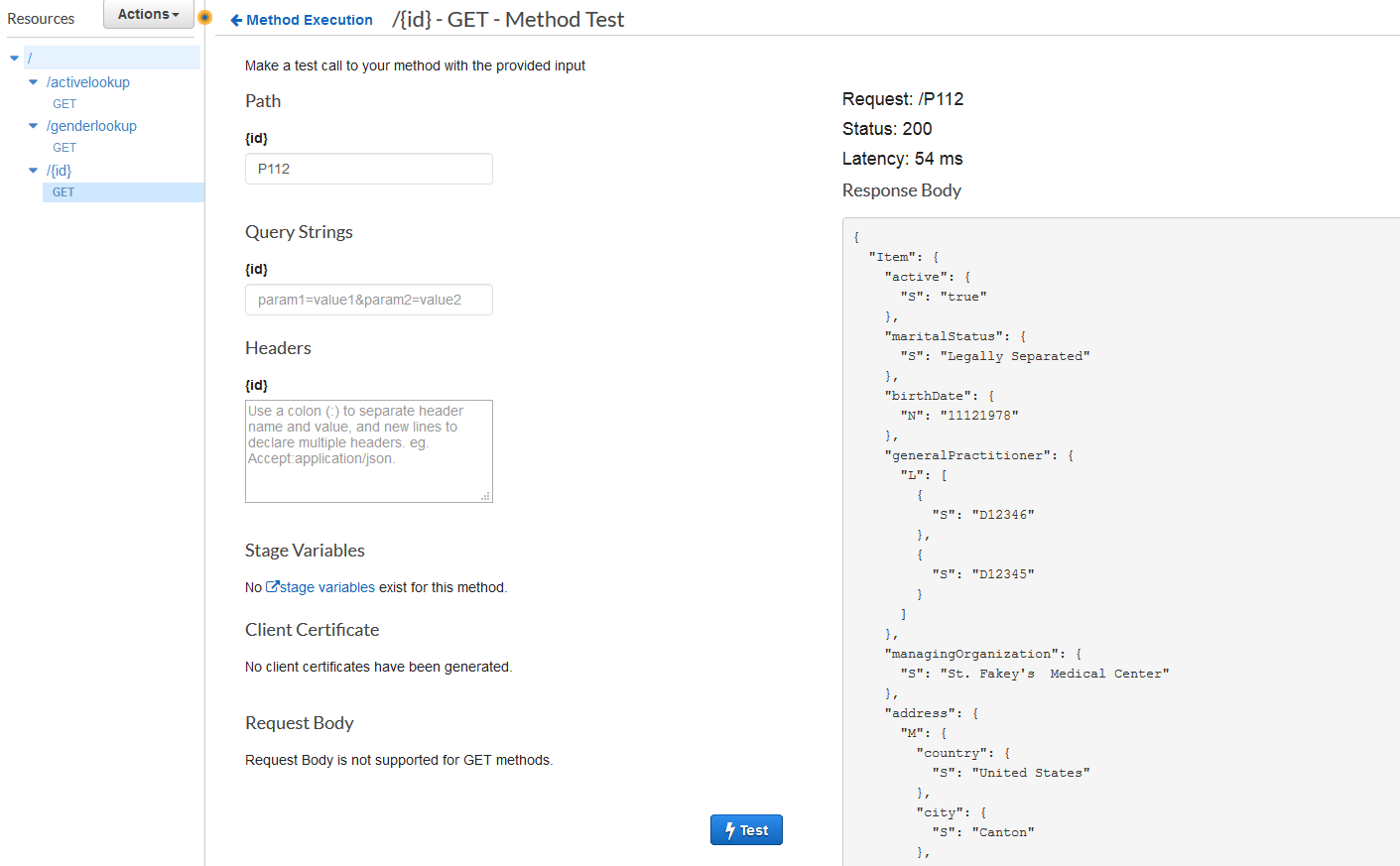
Since our searches happen through query parameters, these must also be set up in the medthod execution and integration request tabs:



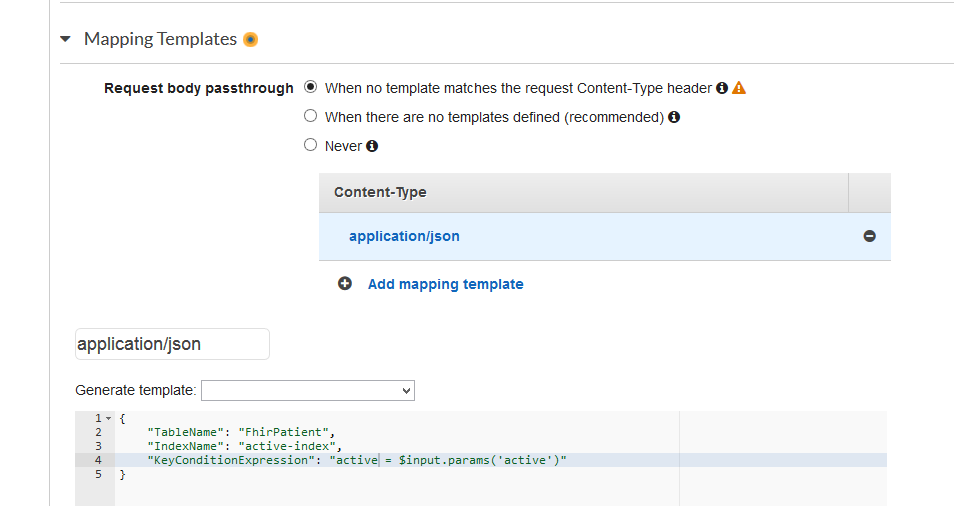


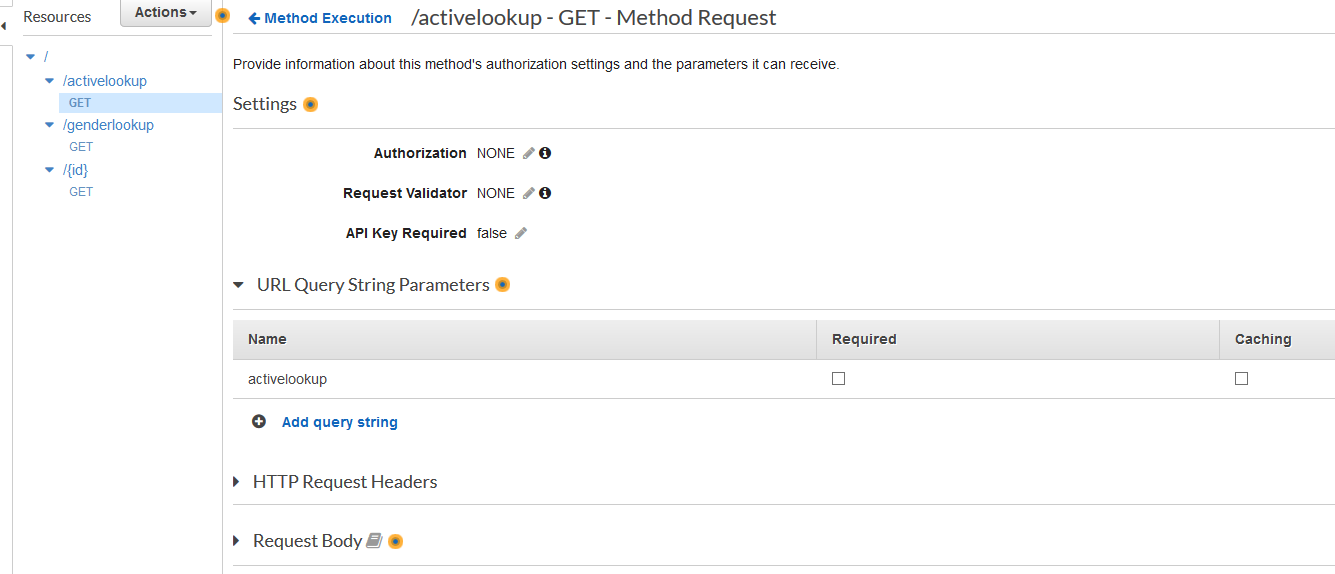
This posts a request to match all index entries where the index gender key is the passed value to genderlookup. To test:

The setup for the active index is similar, but the mapping specifies the index table and active key:









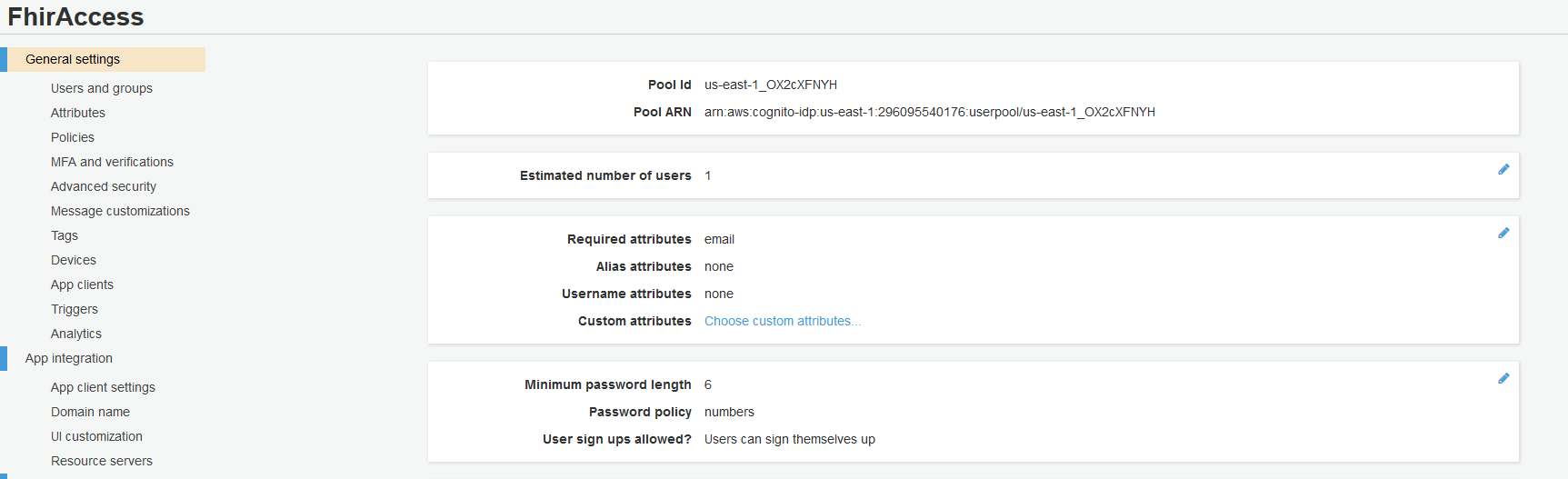
Finally, authentication needs to be set up.

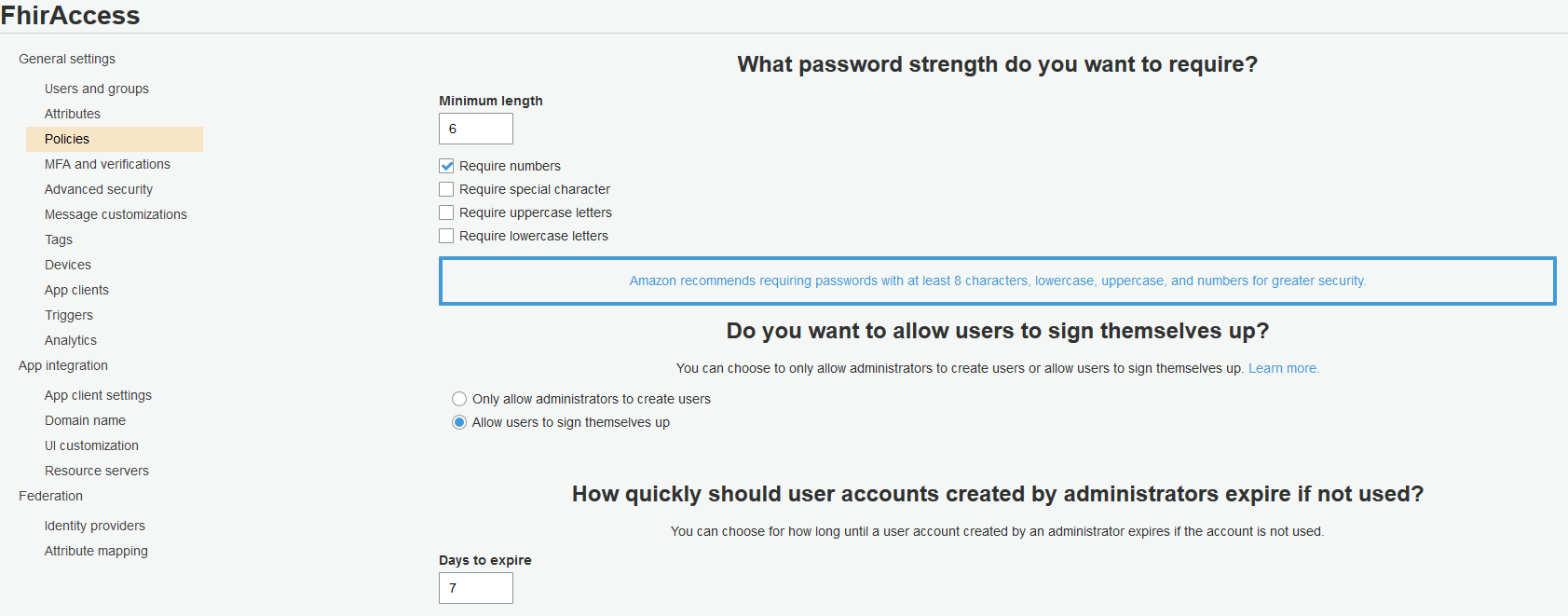
**Authentication:**

For the purposes of this fairly simple setup, I have a mostly default auth profile in Cognito. I’m simulating a verified provider signing users up through the application, and the administrator confirming each.

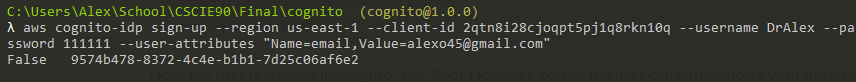
The user pool lets users log in and receive a token, with which they can receive a federated identity from the identity pool, and finally be granted access to aws services.

I have created this sample pool:



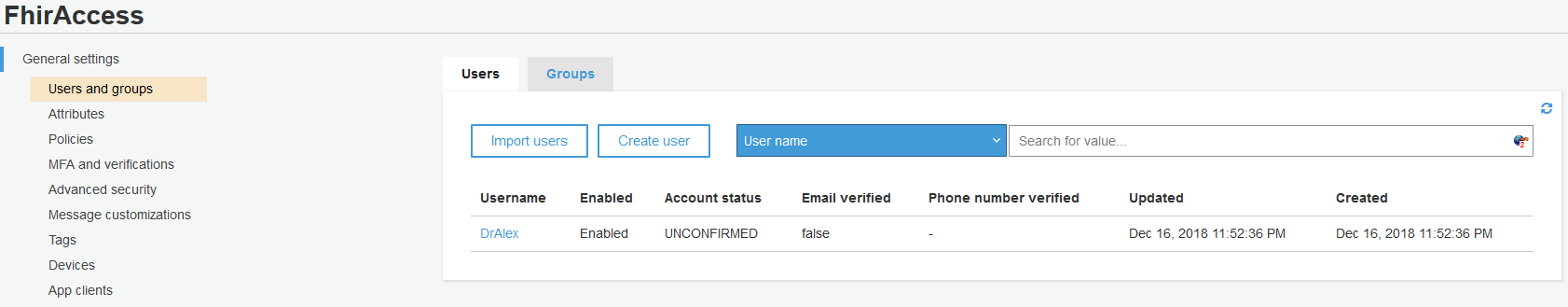


Now I can create a user:

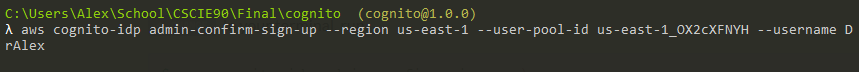


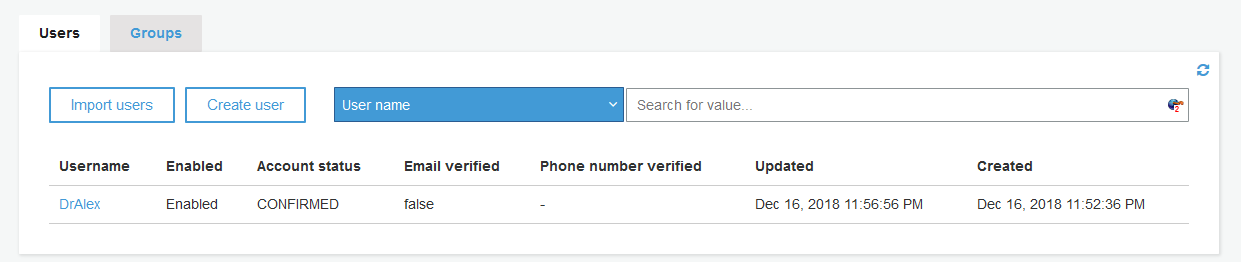
the username = DrAlex, and password = 111111.

However, the user is still not authorized:

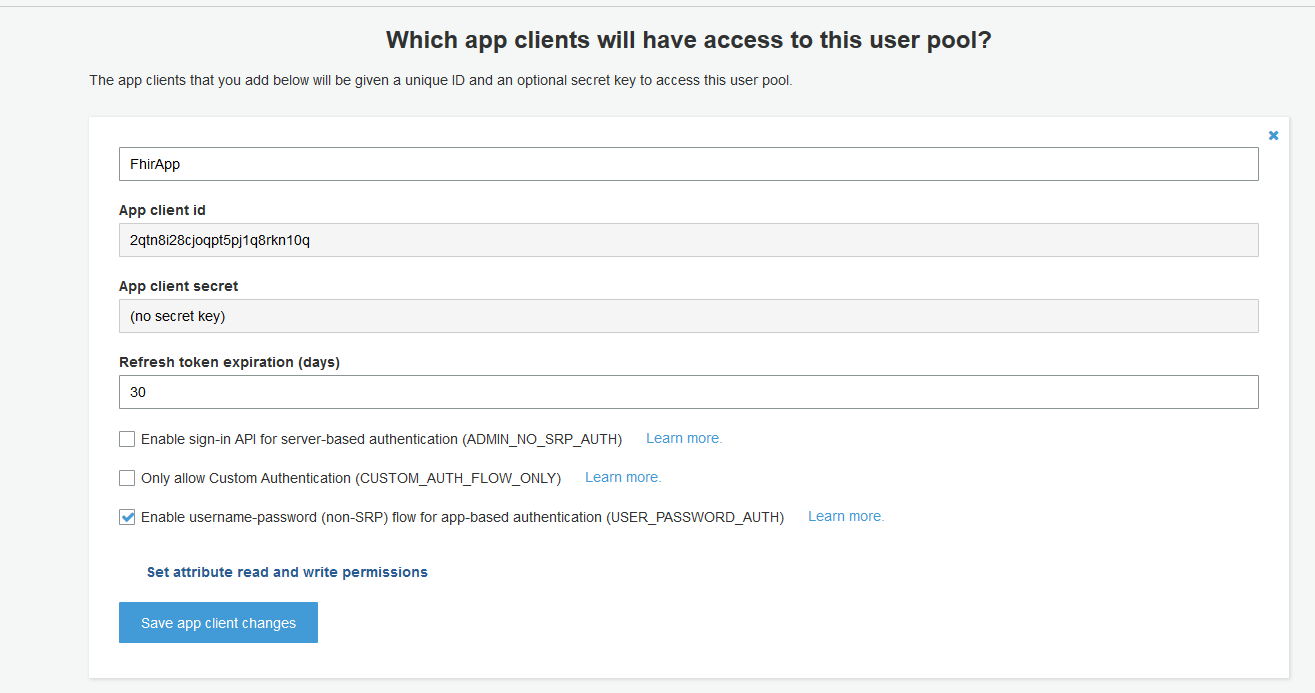


We perform the authorization with the cli:

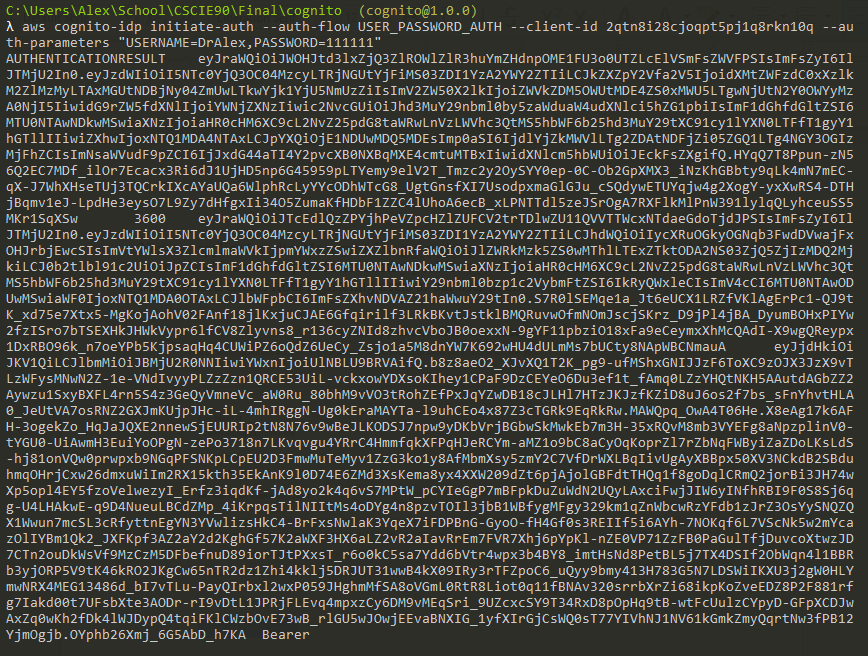




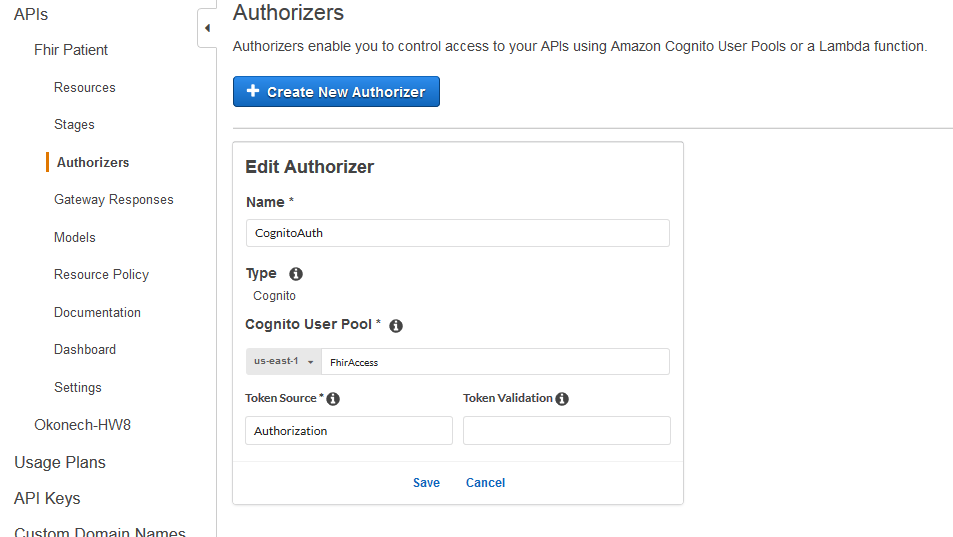
Next, an application client is required to pre-register with the user pool. This is needed for a token to be returned, which we can pass to an identity pool for aws access. The client id is used to prevent unauthorized clients from registering users to our user pool.



Finally, the user can perform a sample credential check, and receive a bearer token from the user pool:

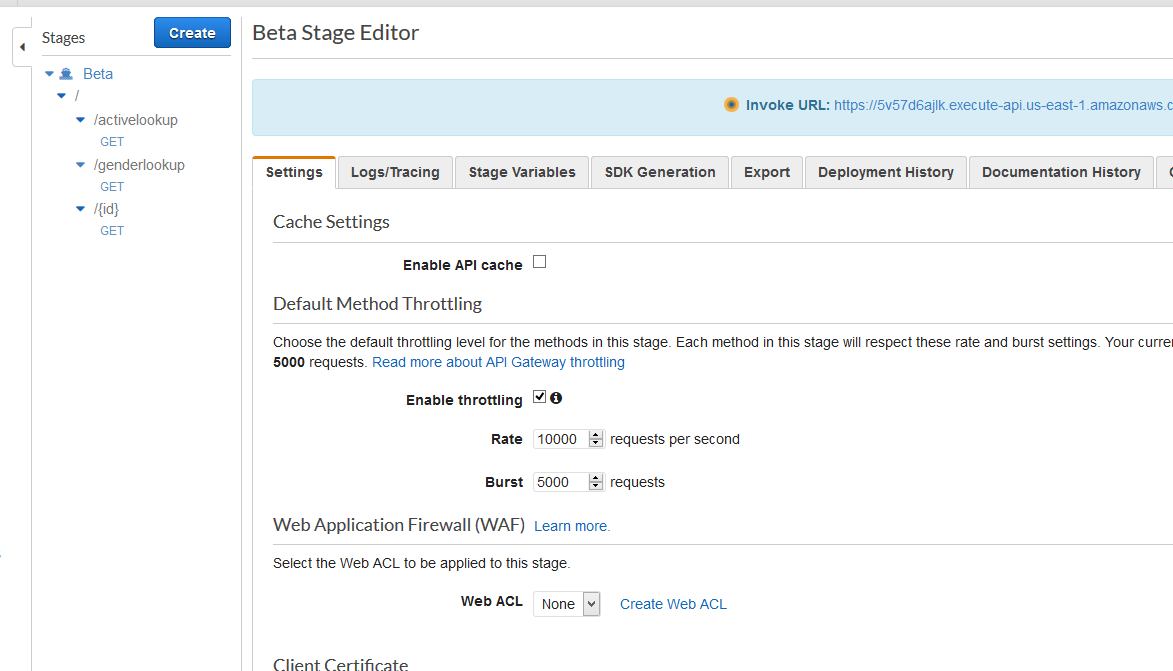


We can go through setting up a federated identity with an identity pool, but this would be more in depth. Instead, we can simply add the existing user pool to the api gateway, and allow the token to be used in api calls without an identity pool:



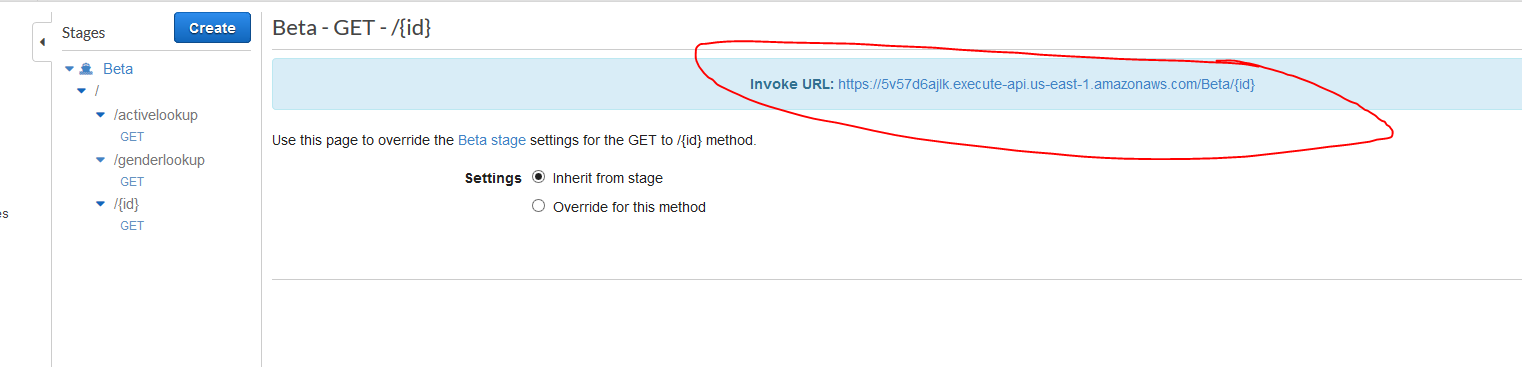
This way, the signed in user will receive a bearer token, and can pass it directly to the api call to authenticate and authorize.

Finally, we publish the patient api for public calls, and do the same with the practitioner api. They are both published with the stage name ‘beta’.

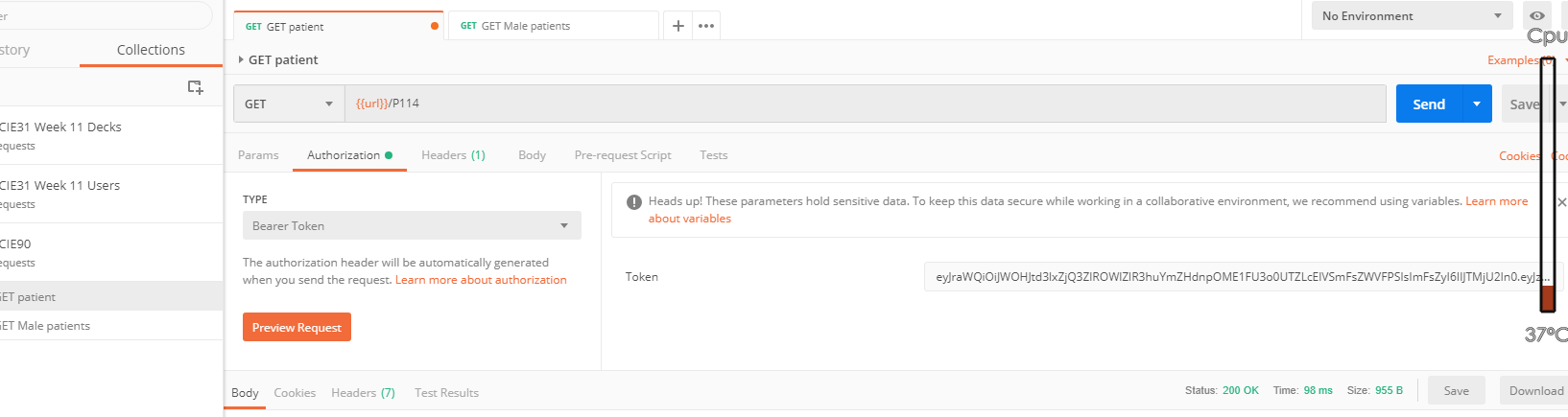


We can now access the apis through a real app, but this will instead be done through postman.

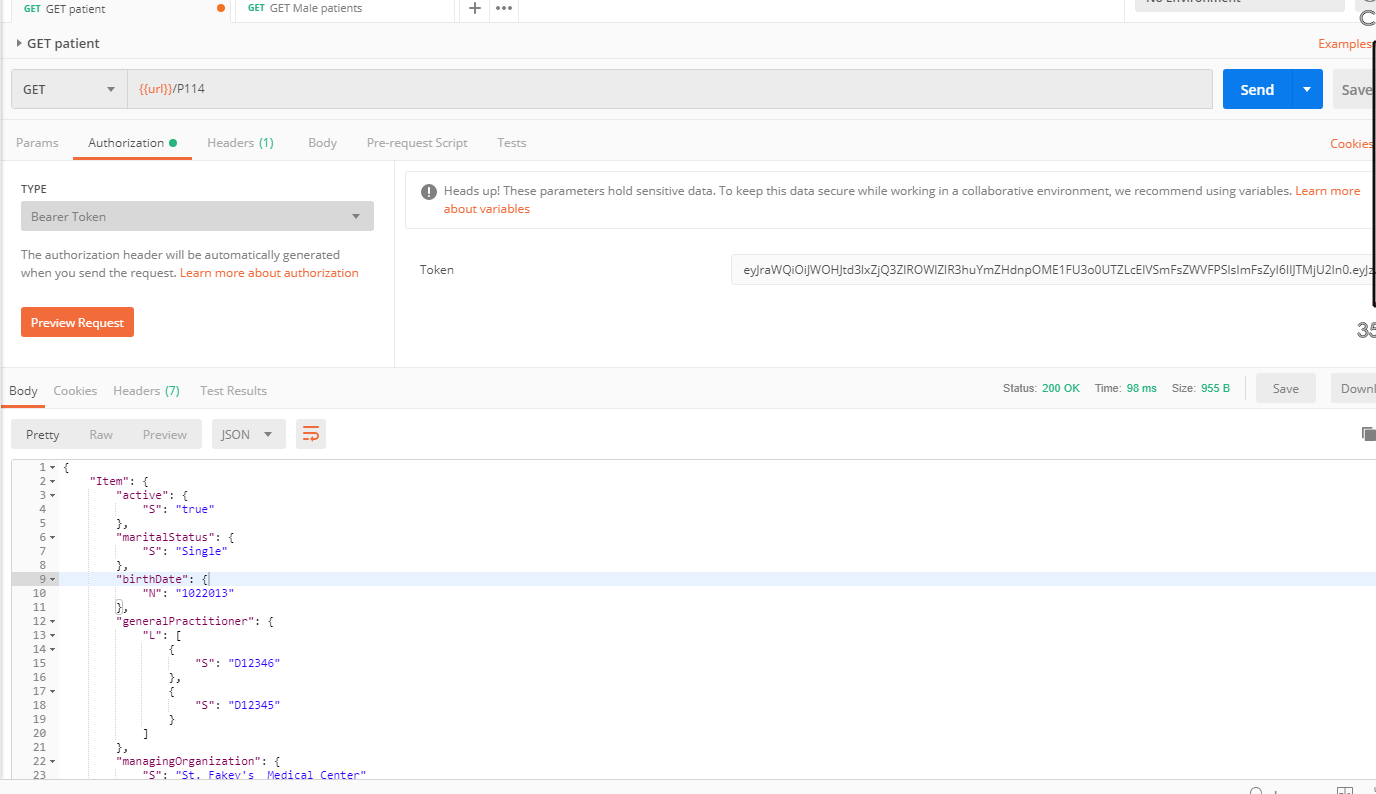
We first get the url to invoke:



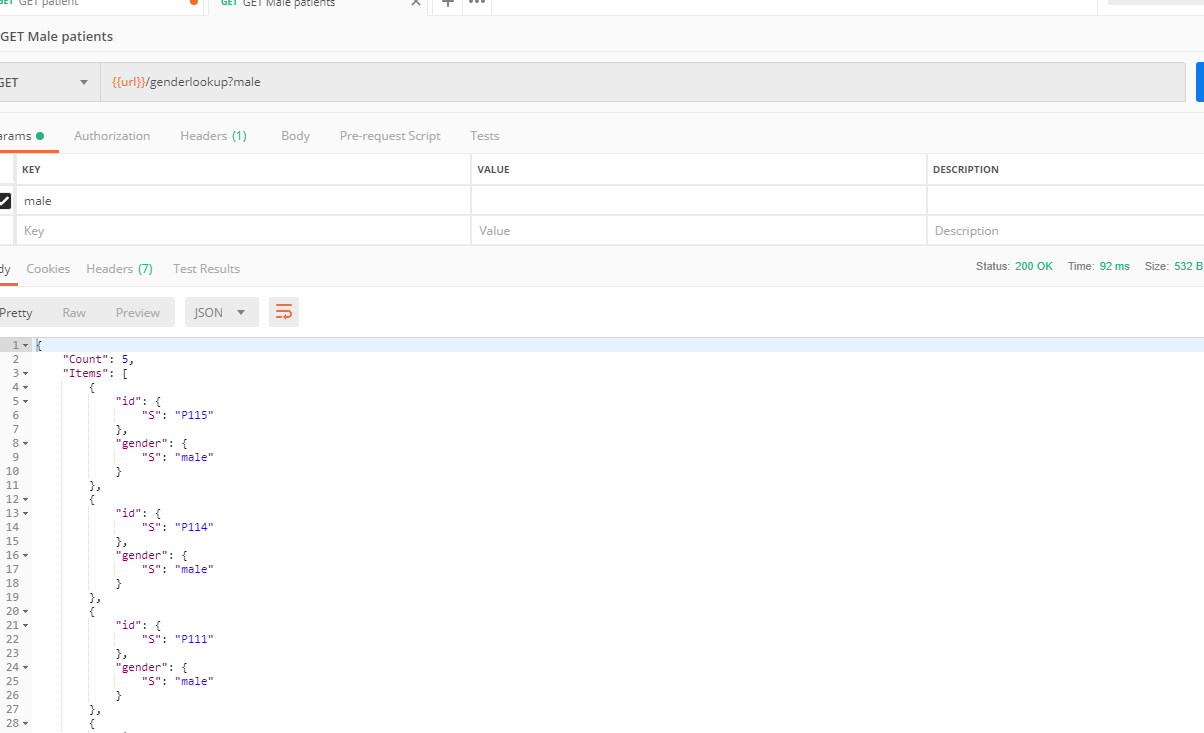
Then create a postman request, using ‘Bearer token’ as the authorization method, and post the bearer token received from the initiate-auth cli result above:



When executed, it returns the data we expect:



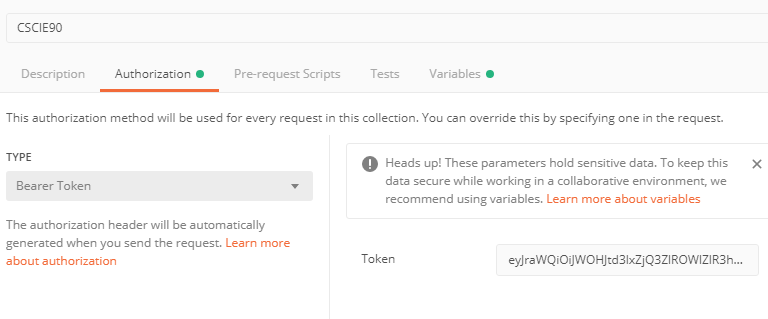
Similarly, the indexes return the data when the searched:

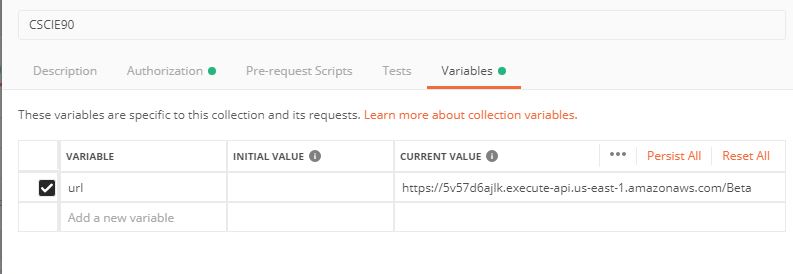


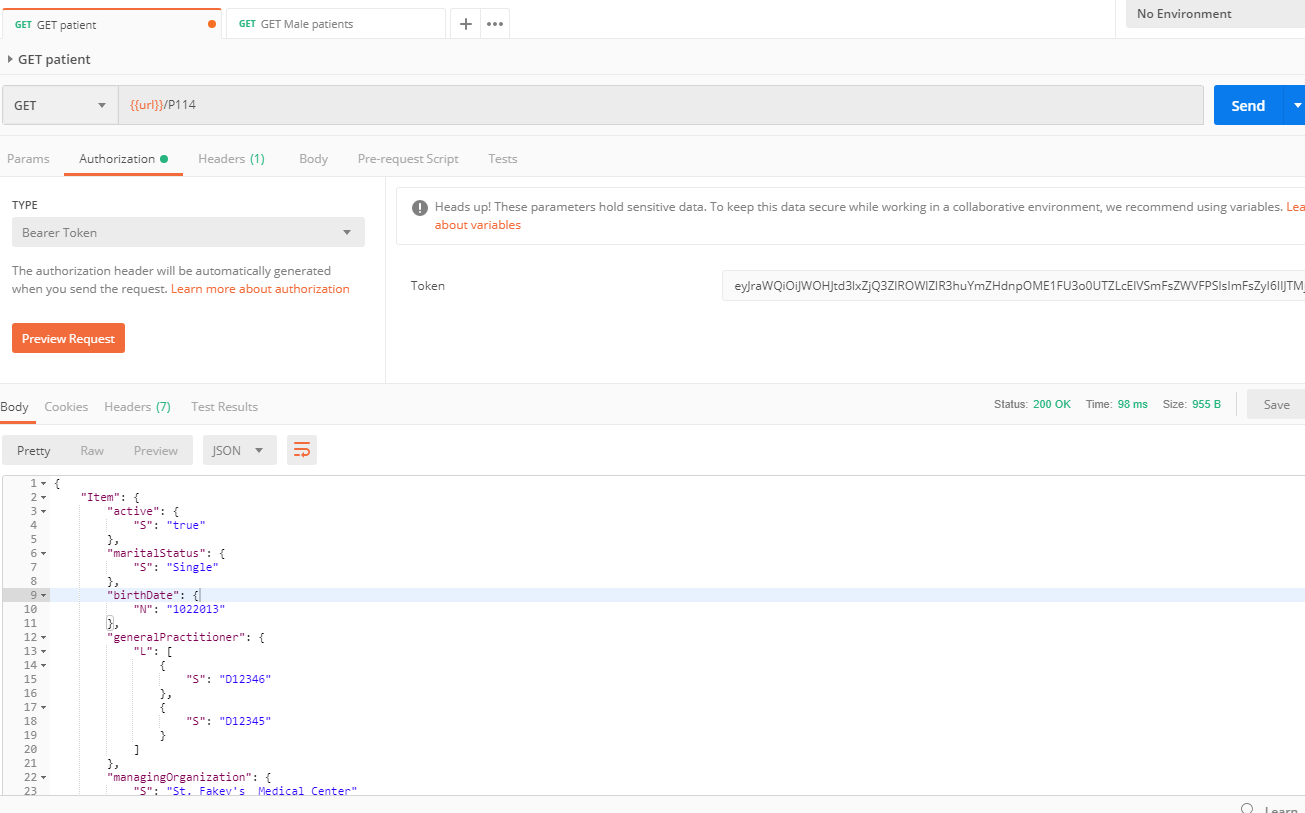
The end result being, we can make authenticated api requests, using Cognito as the authorization service, when we log in as users with username/passwords.

The Api gateway published the apis for getting resources and index searches, and is connected to a dynamodb back end, where the data persists.

To end, we clean up the postman configuration, export it, and provide it with this document.







So now the collection contains the base url for all requests, and also the bearer token passed to all requests. The url is invoked with a global variable.

**CLI commands used:**

**Simulate registering user through an application:**

aws cognito-idp sign-up --region us-east-1 --client-id 2qtn8i28cjoqpt5pj1q8rkn10q --username DrAlex --password 111111 --user-attributes "Name=email,Value=alexo45@gmail.com"

**Confirm user registration manually as an admin:**

aws cognito-idp admin-confirm-sign-up --region us-east-1 --user-pool-id us-east-1\_OX2cXFNYH --username DrAlex

**Start the authn process and receive a bearer token:**

aws cognito-idp initiate-auth --auth-flow USER\_PASSWORD\_AUTH --client-id 2qtn8i28cjoqpt5pj1q8rkn10q --auth-parameters "USERNAME=DrAlex,PASSWORD=111111"

Summary link:

<https://youtu.be/zDwWJLwb-zw>

Full review link:

https://www.youtube.com/watch?v=VzqyUookhUI