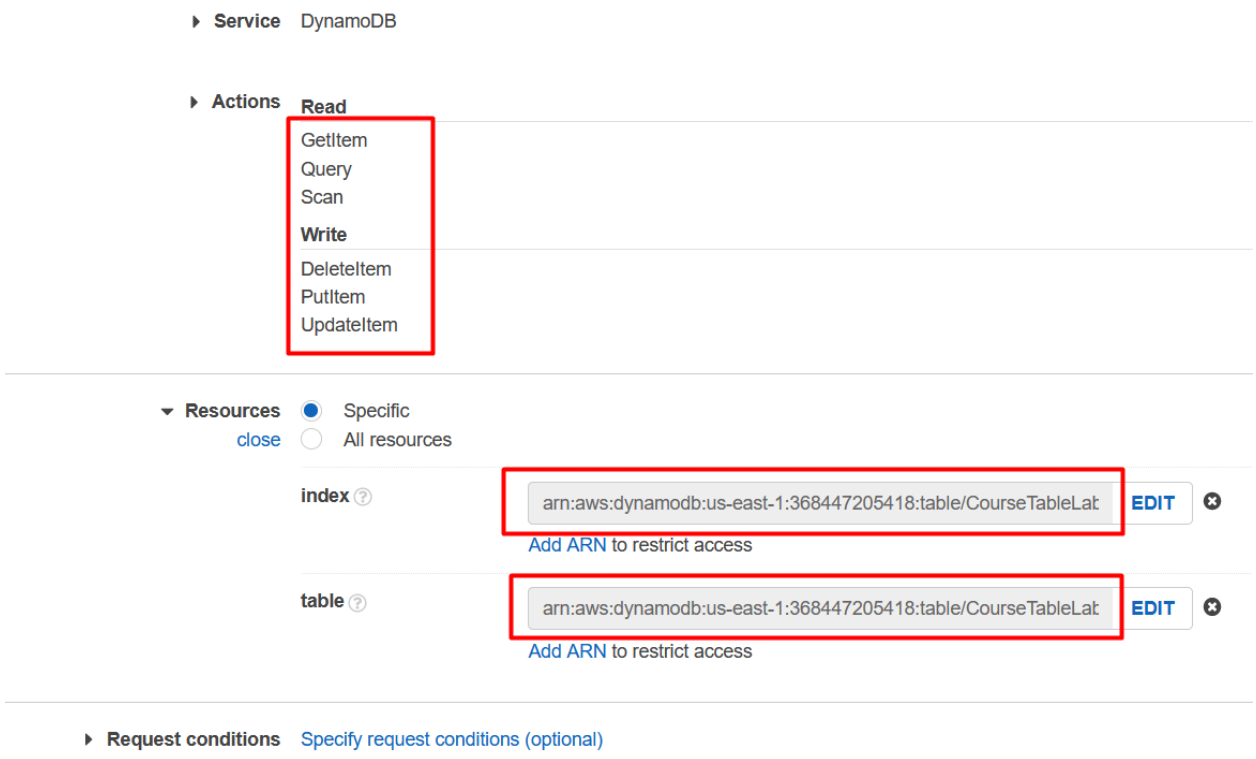
Assignment 7

PART I – Lambda and DynamoDB

Check out the "Be a better dev" channel. https://www.youtube.com/c/BeABetterDev/playlists

1. Create a Lambda called "CourseLambda".
2. Create a DynamoDB table called "CourseTable".
    a. courseCode -> Partition key
    b. teacherName -> Sort key
    c. courseName -> Global index
3. Add an inline policy to the LabRole so that it can do the CRUD operations.



4. Update the Course Lambda to do the CRUD operations.
    a. PutItem
    b. GetItem - get one item by a partition key (courseCode and teacherName)
    c. Query - on an index (courseName)
    d. Scan - Get all items with some criteria (teacher name, month, and year). Explore if you can search by an element in an array.
    e. UpdateItem - update an item
    f. In the Lambda, handle and console log out system and validation errors. Write if else for handling different endpoints based on the path and httpMethod.

Refer: https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html and https://medium.com/geekculture/become-a-dynamodb-ninja-d25b36ce765e

References

Inline policy for the lambda that gives it DynamoDB table access on the CourseTable.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PolicyToGiveLambdaAccessCourseTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query"
            ],
            "Resource": ["arn:aws:dynamodb:us-east-1:<<account-
id>>:table/CourseTable", "… the index ARN"]
        }
    ]
  }
```

The lambda code for the CourseLambda

```
const AWS = require("aws-sdk");
const dynamodb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });
const tableName = process.env.COURSE_TABLE;

exports.handler = async (event) => {
    console.log("Request received: " + JSON.stringify(event));

    const saveParams = {
        TableName: tableName,
        Item: {
            "courseCode": {
                S: "CS516"
            },
            "courseName": {
                S: "Cloud Computing"
            },
            "teacherName": {
                S: "Unubold"
            },
            "students": {
                SS: [
                    "Bipin",
                    "Ryan",
                    "Michael"
                ]
```

```
            },
            "month": {
                N: "7"
            },
            "year": {
                N: "2021"
            }
        }
    };

    await dynamodb.putItem(saveParams).promise();

    const response = {
        statusCode: 200,
        body: JSON.stringify('An item is saved.'),
    };
    return response;
}
```

PART II - API gateway and Cognito

- Create a CRUD API for the sample course app in API Gateway.

/course POST

/course/{courseName} GET – filter courses by course name. Query on the index. Get the course name as a path parameter.

/course GET – List all courses. Implement filter on non-key attributes. Get teacher name, month, and year values as query strings.

/course/item GET – it returns one item by the composite key. Get the course code and teacher name as query strings.

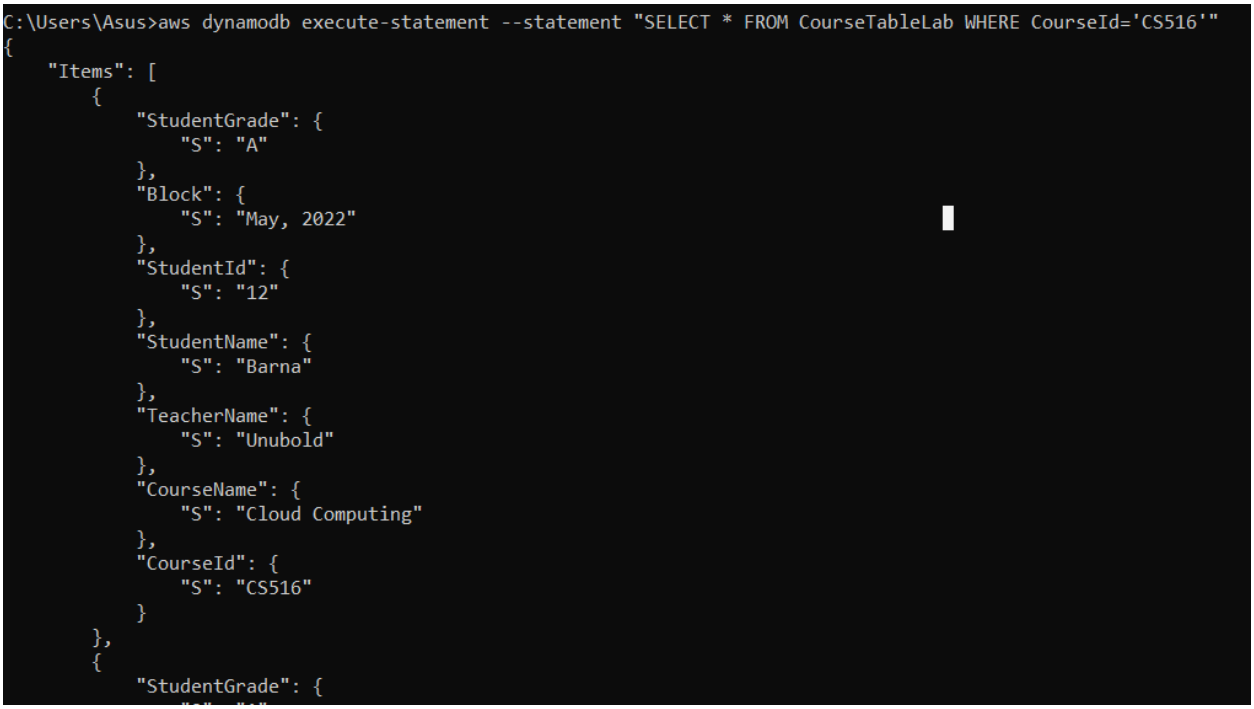
/course PATCH – That updates a course record. Don't overwrite, don't loose any data for example, when updating only one attribute.

/course DELETE – Delete

- Create a **Cognito User pool** for the app. For hosted UI setup, refer: https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pools-app-integration.html
- Secure the API using tokens from the Cognito User pool.

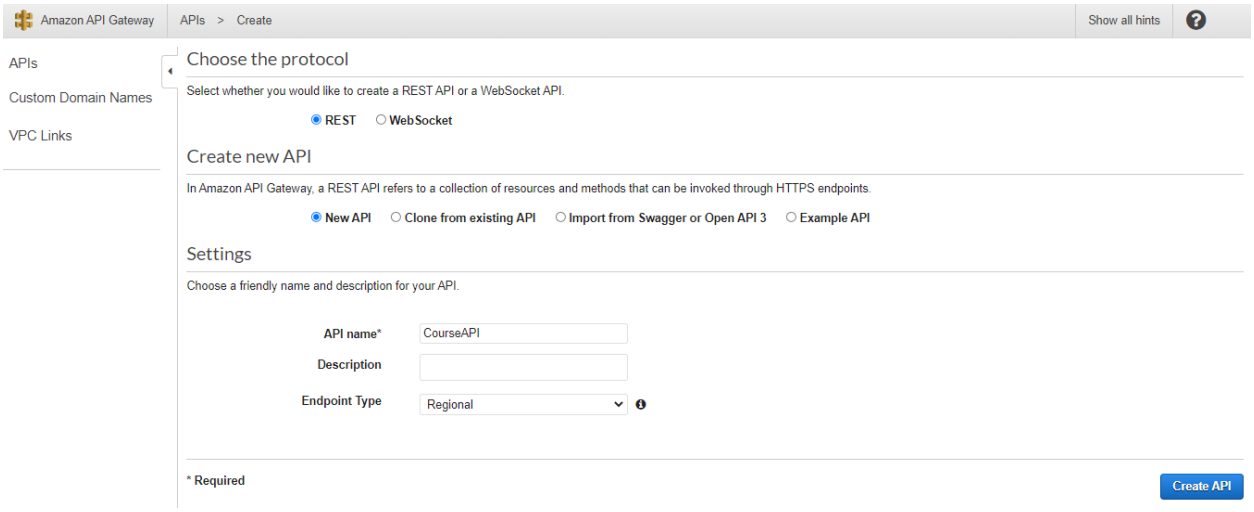
Extra

- Instead of Lambda, use StepFunctions to store data in DynamoDB.
- Practice the execute statement with SQL.



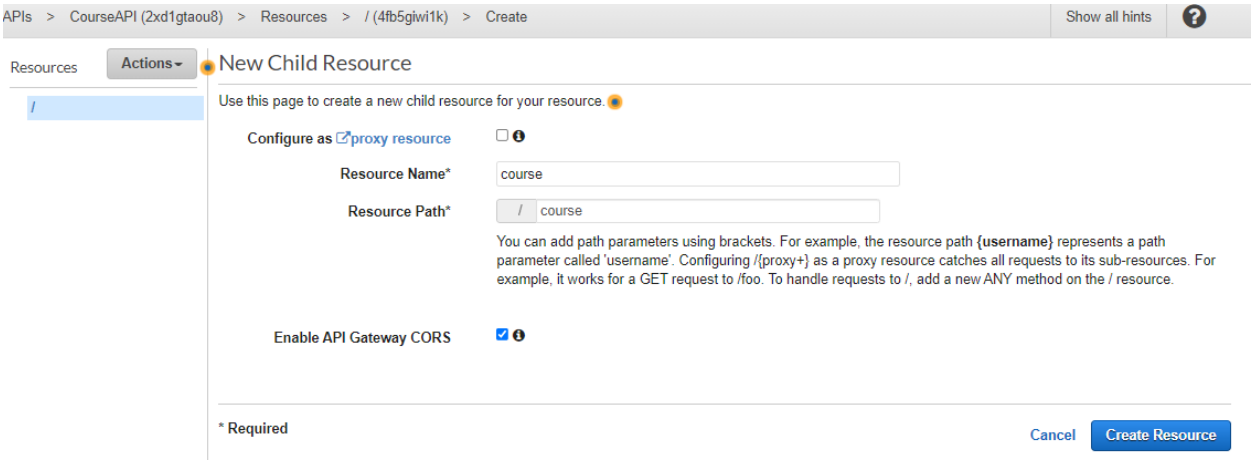
Instructions

1. Create a "**CourseAPI**" API on API Gateway in front of the "**CourseLambda**"
    a. Search on the top bar and go to the **API Gateway** on AWS Console.
    b. **REST API** (Not REST API private!!)-> click on the orange **Build** button.
    c. On the popup, press **OK.**
    d. In **Create new API**, select **New API** radio button.
    e. In **Settings, API name** is **CourseAPI**. Hit **Create API.**

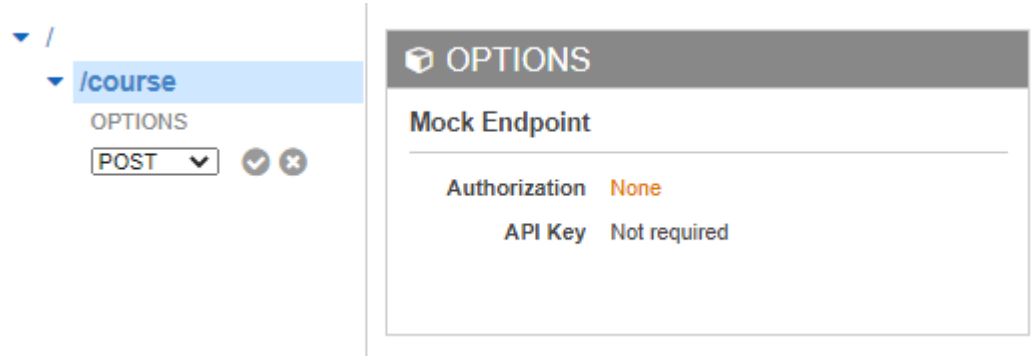


f. Click on **Actions** dropdown and hit **Create Resource.**

g. Resource Name is **course.** check **Enable API Gateway CORS.** Hit **Create Resource**.



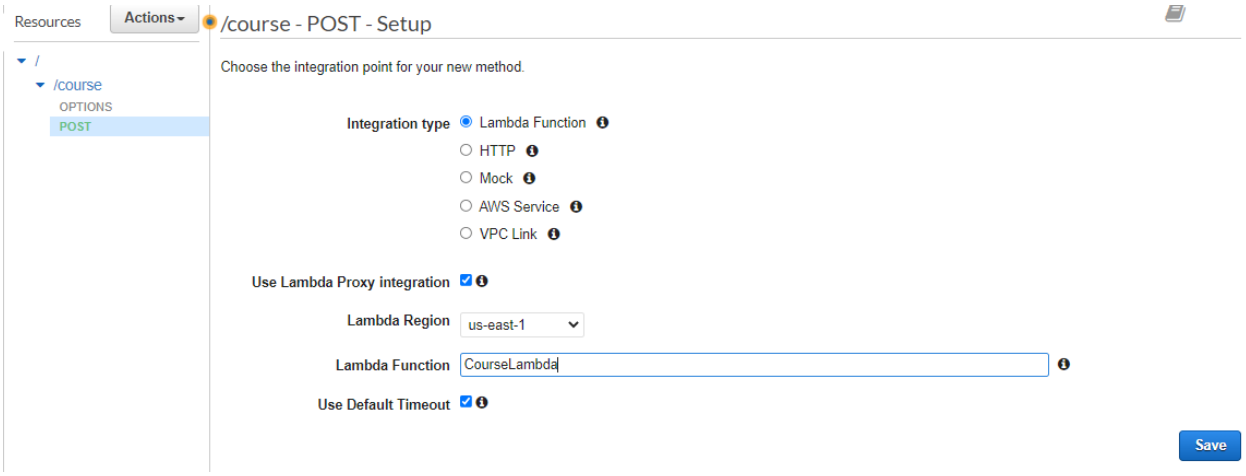
h. Click on **Actions** dropdown and hit **Create Method.** Select **POST** in the small dropdown under the resource. Click on the small OK icon.



i. Check **Use Lambda Proxy integration**

j. Type the lambda name **CourseLambda** as Lambda Function. Click **Save.**

k. There will be a popup. Read that and hit **OK**.



l. Click on the **Actions** dropdown and hit **Deploy API**

m. On the popup, the Deployment stage is **[New Stage]**. **Stage name** is v1. Hit **Deploy**.

2. Test your API with Postman.

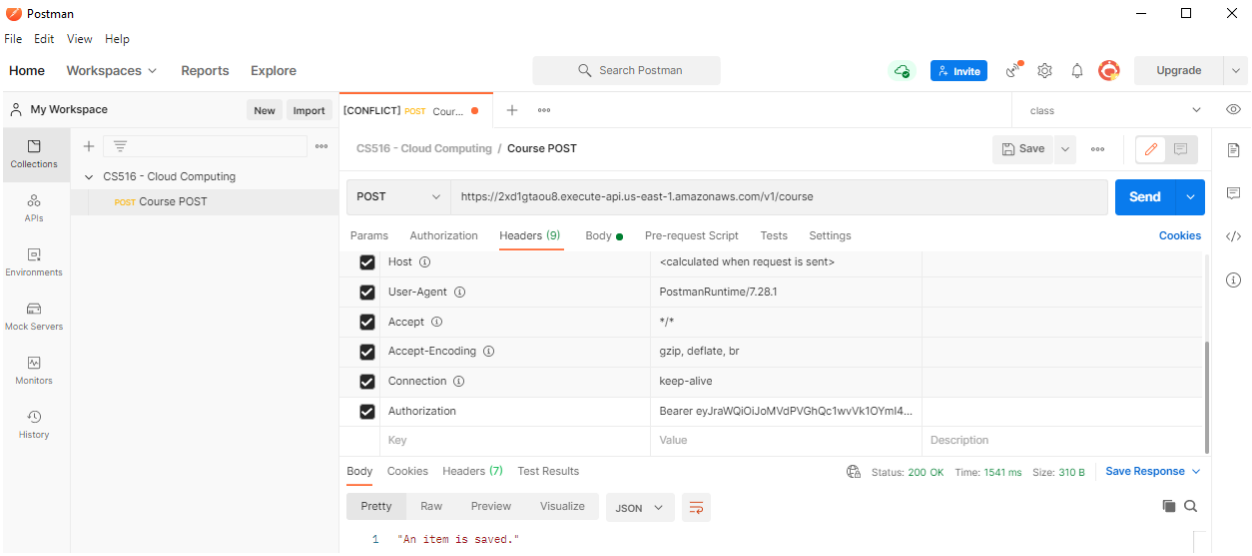
a. Click on **Stages** in left sidebar. Click on **v1**. Grab the **Invoke URL**.

b. Create a new **POST** request in postman. Provide the URL. Append the **course** resource. It will look like this: https://2xd1gtaou8.execute-api.us-east-1.amazonaws.com/v1/**course**

c. The body is below. Feel free to change the value. Body tab -> Select Raw -> Select JSON in the dropdown

```
{
    "courseCode": "CS100",
    "courseName": "My Course",
    "teacherName": "My Teacher",
    "month": 11,
    "year": 2022,
    "students": [
        "Student 1",
        "Student 2"
    ]
}
```

d. You should see the success response below.



3. Update your lambda to store the body we passed instead of hard-coded values.
    a. Go to Lambda -> Configuration -> Permission. In **Resource-based policy**, You will see a new statement. Explain what that is.
    b. Go to Lambda -> Monitor -> View logs in CloudWatch. Click on the orange **Search log group** button. Select **30m** in the top right corner. That will show the latest logs.
    c. We are logging the entire **event** object coming. Like this .log("Request received: " + JSON.stringify(event));. It shows you what was sent from API Gateway to the Lambda.

    d. If you scroll all way down, you will find the **body** that the user sent and we want to store that in DB instead of hard-coded values. So you can get the body in the code like this. **const body = JSON.parse(event.body);**
    e. Paste the code below and hit **Deploy.**

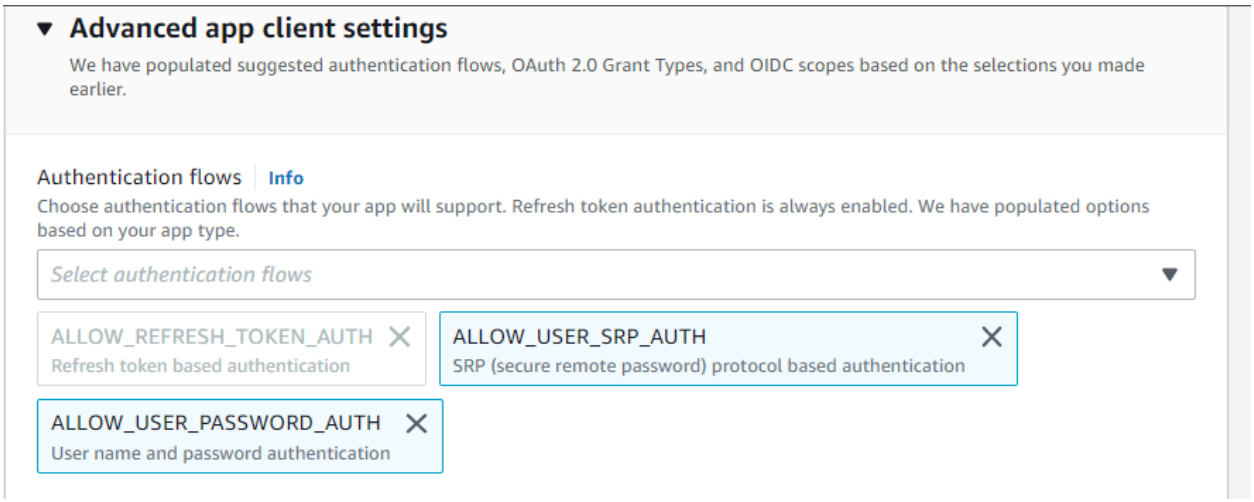
```
const body = JSON.parse(event.body);

    const saveParams = {
```

```
        TableName: tableName,
        Item: {
            "courseCode": {
                S: body.courseCode
            },
            "courseName": {
                S: body.courseName
            },
            "teacherName": {
                S: body.teacherName
            },
            "students": {
                SS: body.students
            },
            ...
        }
    };
```

f. Send the postman request once more, you should be able to see the item you submitted in DynamoDB.

Add GET method in the API Gateway and that returns all courses in the DB.

4. Create a user pool for the Course API in AWS Cognito.
    a. Go to Cognito -> click on **Manage User Pools** -> Top right corner, click on **Create a user pool**.
    b. In **Name** section, **CourseUserPool** as Pool name. Click on **Step through settings.**
    c. In **Attributes section,** Select **Email address or phone number.** In **Which standard attributes do you want to require?,** check **email** and **name.** Click on **Next step.**
    d. In **Policies** section, nothing to change. Click on **Next step.**
    e. In **MFA and verifications** section, nothing to change. Click on **Next step.**
    f. In **Message customization** section, nothing to change. Click on **Next step.**
    g. In **Tags** section, nothing to change. Click on **Next step.**
    h. In **Devices** section, nothing to change. Click on **Next step**
    i. In **App Clients** section, click on **Add an app client. CourseApiClient** as **App client name.**
        i. Uncheck **Generate client secret.**
        ii. Uncheck **Enable lambda trigger based custom authentication.**
        iii. Check **Enable username password based authentication.** Then click on **Create app client.**
    j. **Enable hosted-ui.**

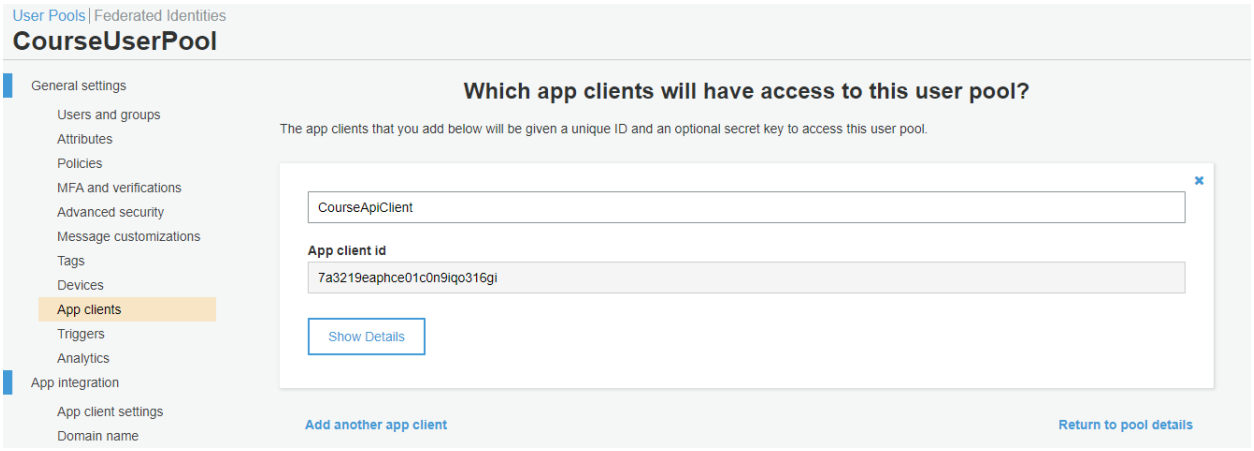


Disable Secret key

Only enable Username password auth

k. Click on **return to pool details.**
l. Hit **Create pool**.

5. Grab the **App client id** and store it somewhere. You will need it in the next steps.



6. Create a user in your user pool with hosted UI or AWS CLI.

Hosted UI:

https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pools-app-integration.html

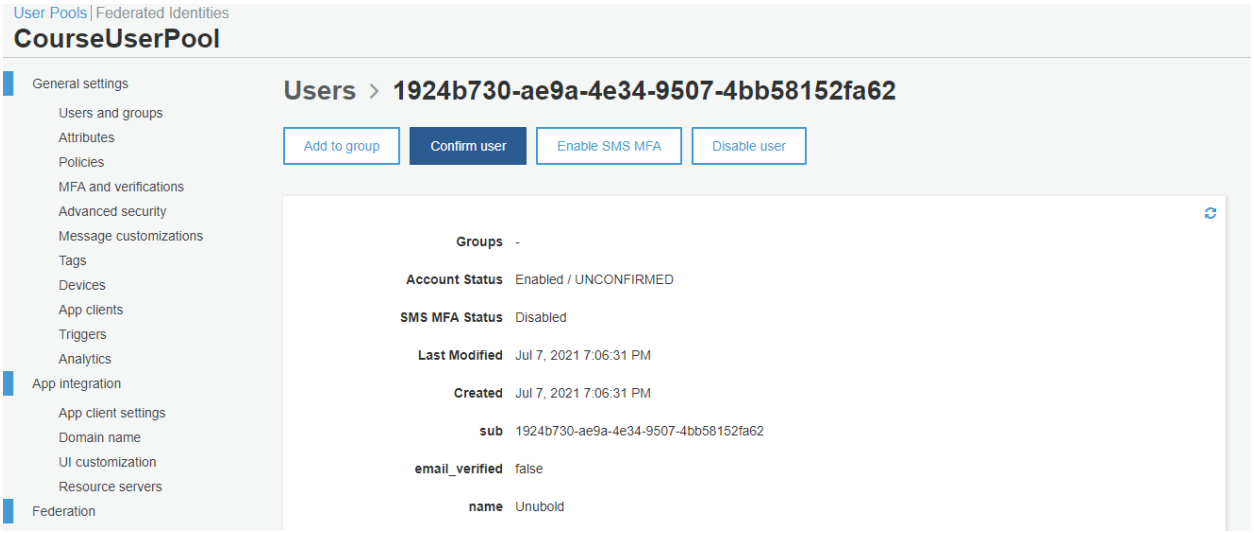
https://<your_domain>/login?response_type=code&client_id=<your_app_client_id>&redirect_uri=http://localhost:3000

OR

CLI: aws cognito-idp sign-up --client-id <<app_client_id>> --username <<your_email>> --password Test123 --user-attributes Name=email,Value=<<your_email>> Name=name,Value=<<your_first_name>> --region us-east-1

```
C:\Users\admin>aws cognito-idp sign-up --client-id 7a3219eaphce01c0n9iqo316gi --username utumenbayar@miu.edu --password
Test!123 --user-attributes Name=email,Value=utumenbayar@miu.edu Name=name,Value=Unubold --region us-east-1
{
    "UserConfirmed": false,
    "CodeDeliveryDetails": {
        "Destination": "u***@m***.edu",
        "DeliveryMedium": "EMAIL",
        "AttributeName": "email"
    },
    "UserSub": "18157ff9-47b1-43c7-9f40-8066cbca7e16"
}

C:\Users\admin>
```

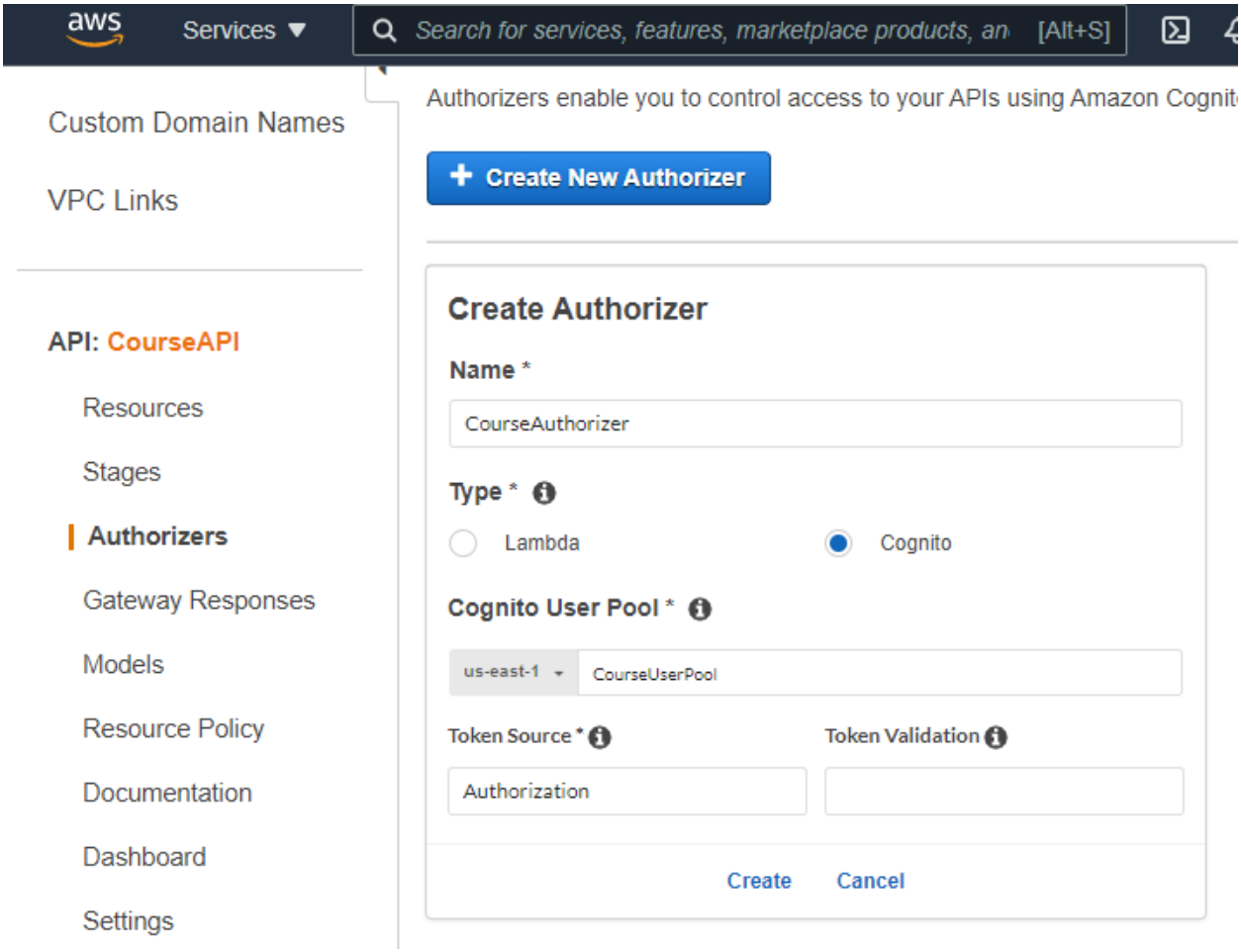
a. Go to your user pool and click on **Users and groups** in the left sidebar. Hit refresh icon on top right corner. That will pull the newly-created user. Click on the username which is UUID hyperlink. Click on **Confirm user** button.



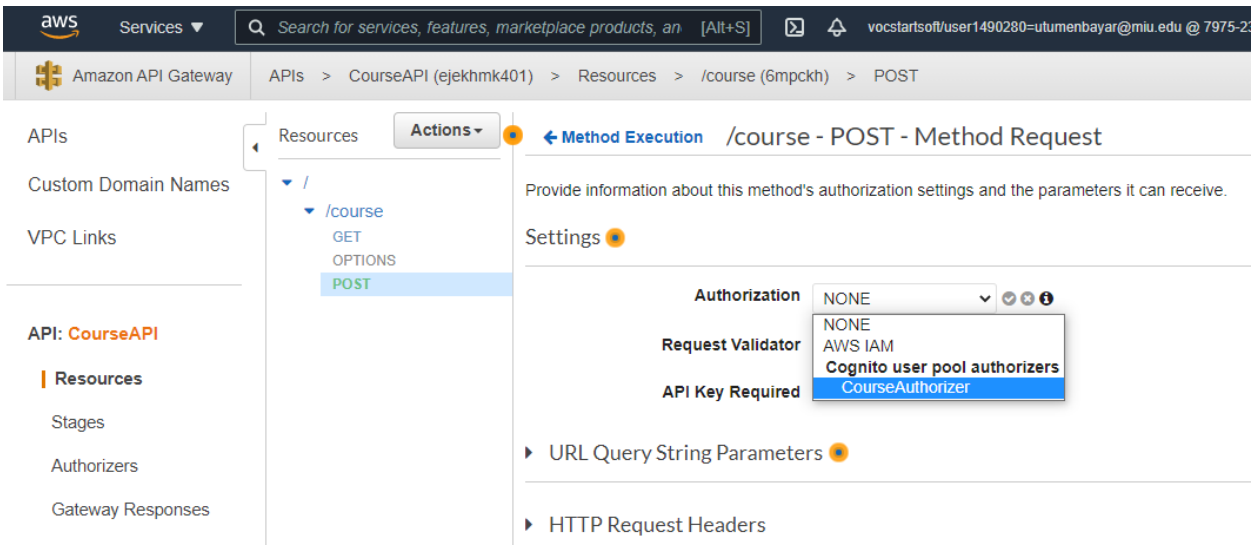
b. Execute the command below that returns token associated with the user. That you need to provide after securing the API to store and retrieve data from the back-end or lambda. You may need to re-execute this command to get the new tokens in case it is expired. Based on how you configured the custom attributes, it could be slightly different.

aws cognito-idp initiate-auth --auth-flow USER_PASSWORD_AUTH --client-id <<app_client_id>> --auth-parameters USERNAME=<<your_email>>,PASSWORD=Test123# --region us-east-1

7. Secure the POST endpoint.
    a. Go to API Gateway. Go to your API. Click on **Authorizers** in the left sidebar.Click on **Create New Authorizer.**
    b. Name as **CourserAuthorizer.** Type is **Cognito.** Select the user pool you created. **Token Source** is **Authorization**.

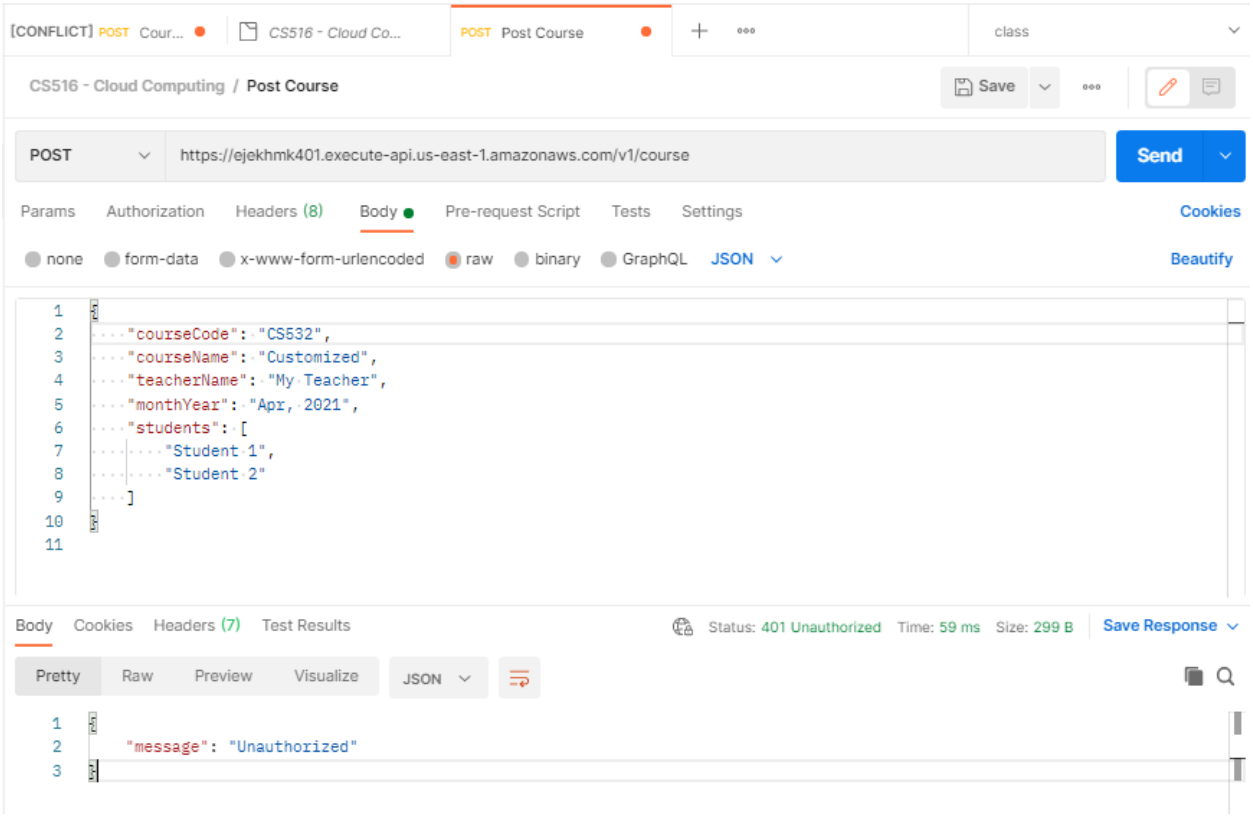


c. Go to **Resources.** Select the **POST** method under course resource.
d. Refresh the whole page. Click on **Method Request. Authorization** is the authorizer you just created. Click on OK icon.



e. Secure the GET endpoint as well by using the authorizer you created earlier. Do the step c and d on the GET.
f. Actions -> Deploy API -> Go with the existing stage.

8. Test.
   a. As see you below. Your endpoint is secured. You must provide the tokens that we generated in previous steps in Authorization header.



   b. Copy the **ID Token**. Provide it in the header as **Authorization.**