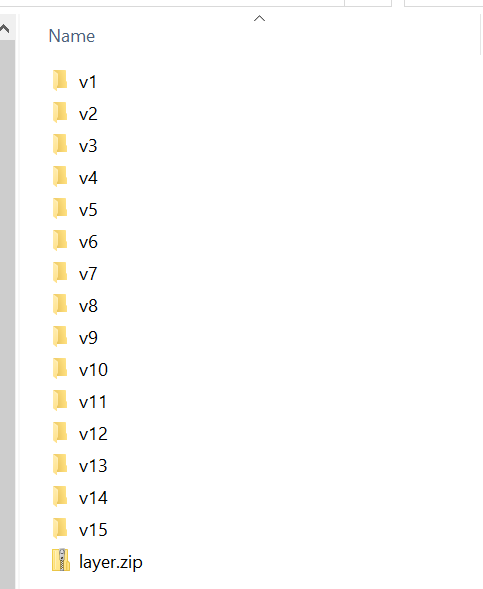
SECTION:1

1. Download the Resource files



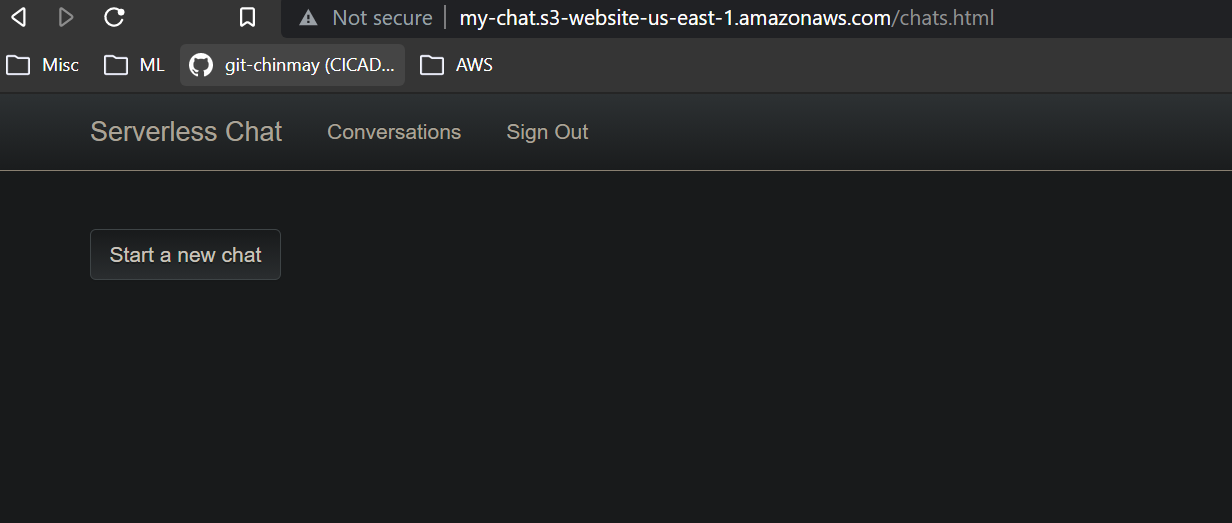
1. Create a S3 bucket (make sure remove uncheck block the public access)
2. Upload all the files from V1/Site/ folder

Note: Make sure you have enabled the Grant Public read access under predefined ACLs under Access Control List option

1. Once file uploaded enable the static website option under bucket properties

Give the index.html for enabling the site. We will create this file later but we can test the site using below link

<http://my-chat.s3-website-us-east-1.amazonaws.com/chats.html>

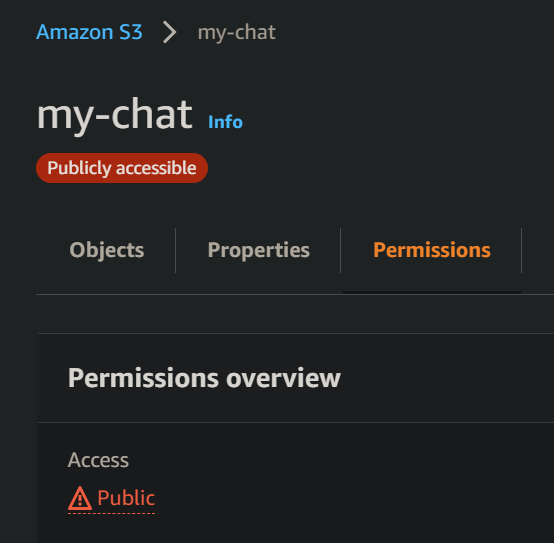


SECTION:2

Each time we upload the files we can not go and make it public instead we can add a policy to the bucket.

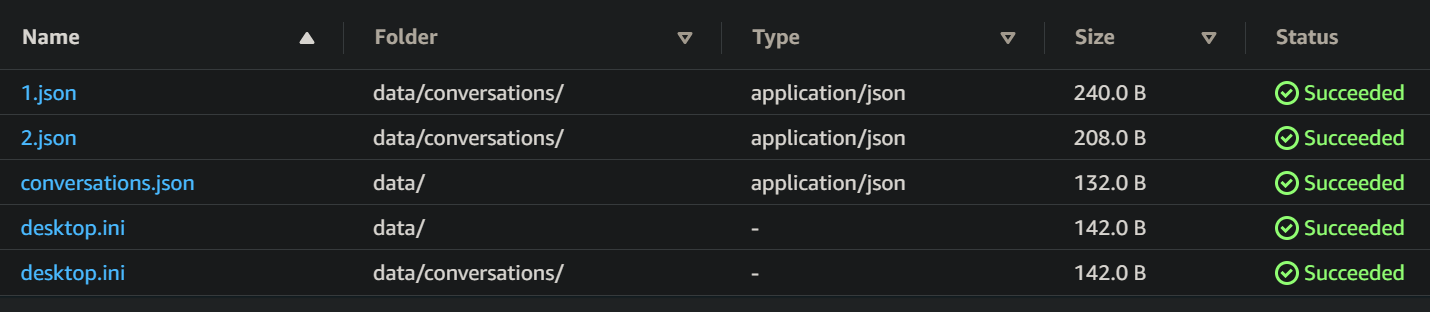
1. Go to your bucket policy under Permissions
2. We can copy the policy from resource/v2/policies folder and paste it in aws console
3. Replace the name with your bucket name and save.

You should see the publicly tag under your bucket



Lets see an example how our chat app will fetch the conversational data from S3 bucket

1. Upload the data folder from resource/v2/site/data to our bucket



If you open conversationa.json, you can see two chat participant details and 1.json and 2.josn contains what messages the participant sent

1. We can see the chat by going to our link add the data folder to it

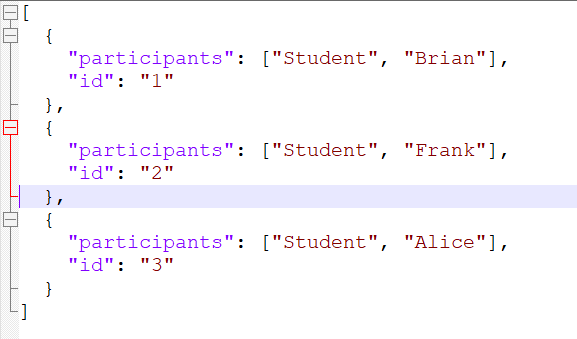
<https://my-chat.s3.amazonaws.com/data/conversations/1.json>



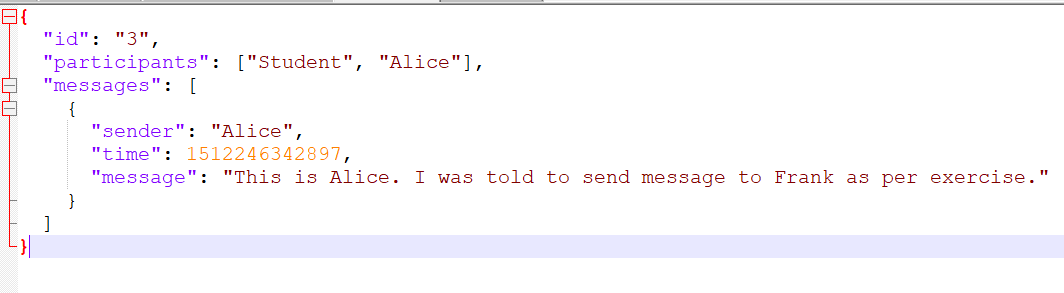
The above details our app will pull down and show us the message part when we have actual chat app implemented.

Exercise:-

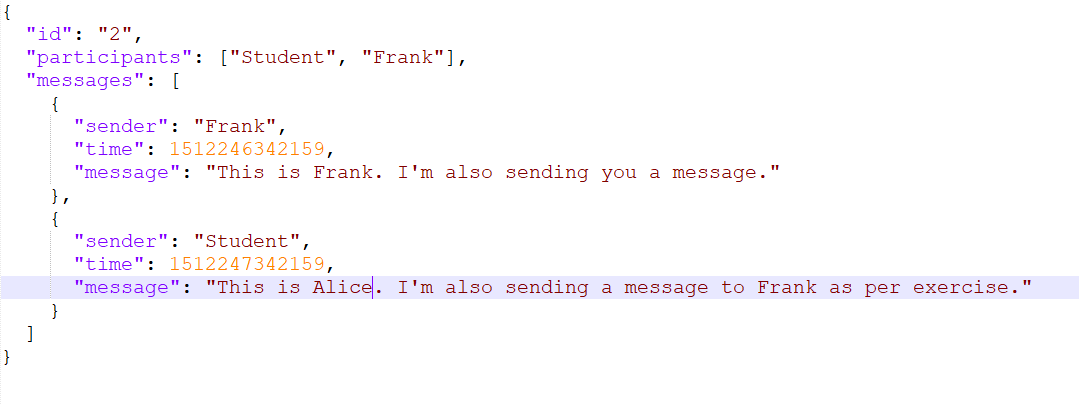
1. Add a message from Student to Frank
2. Add a new conversation
3. First add a new user to conversations.json



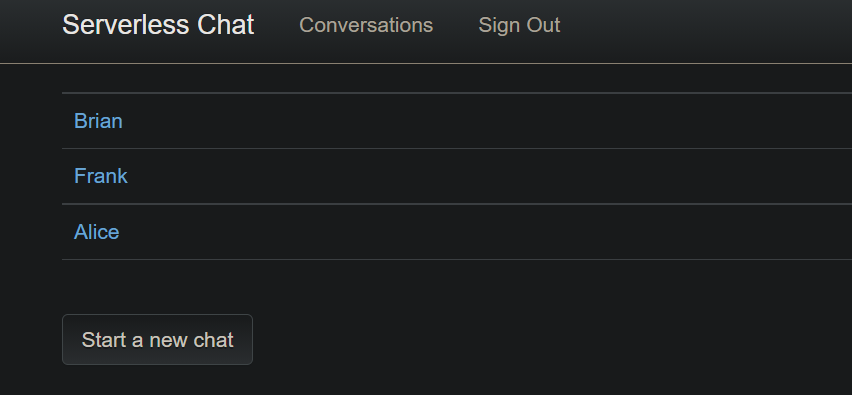
1. Now create a 3.json file and add the message Alice want to send to Frank



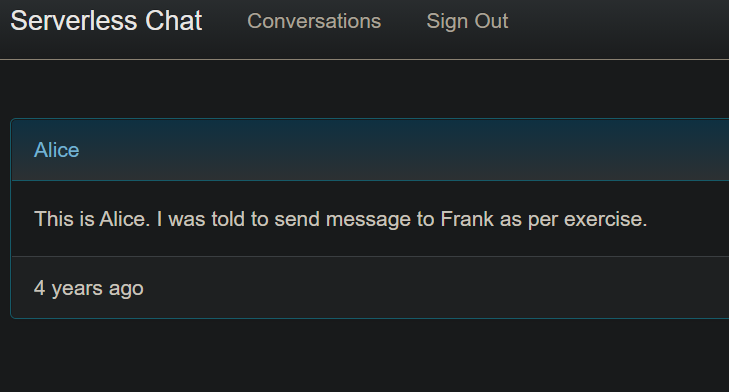
1. Now add the same message to Frank’s id file (2.json) so that it will reflect under Frank’s chart history



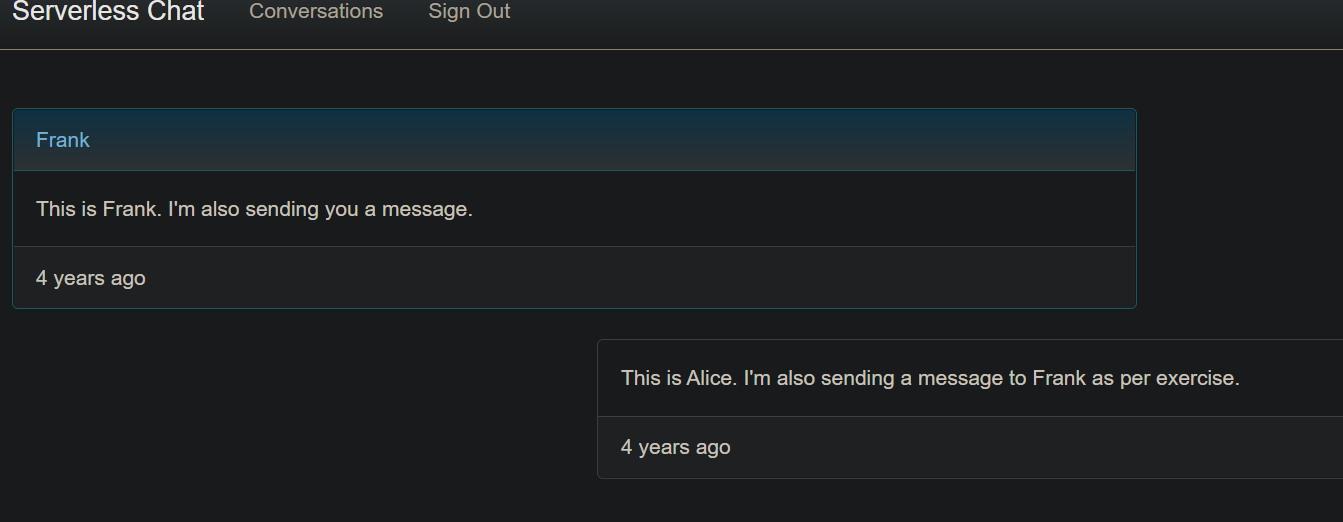
1. Now upload the updated files to s3 bucket under /data folder and refresh the page



Click on Alice to see what she sent

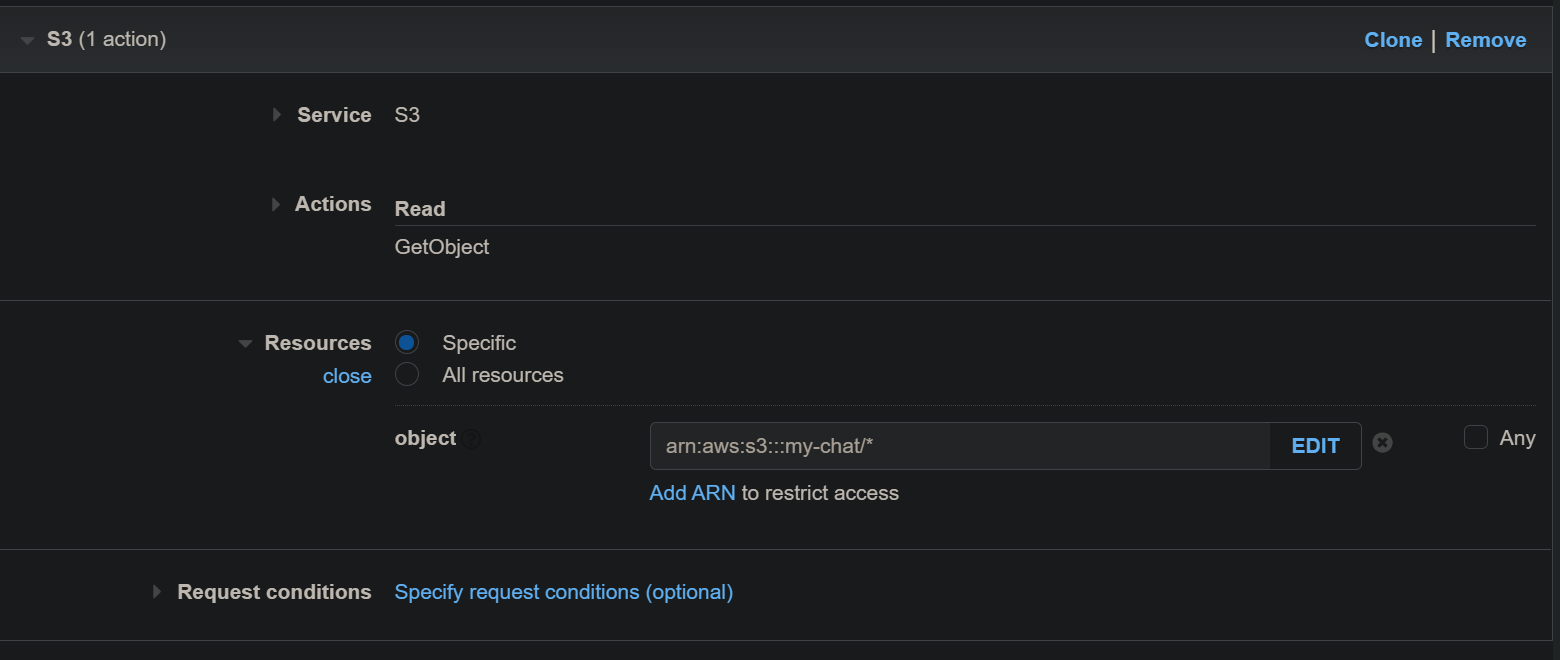


Click on Frank, we can see the message from Alice in his chat page

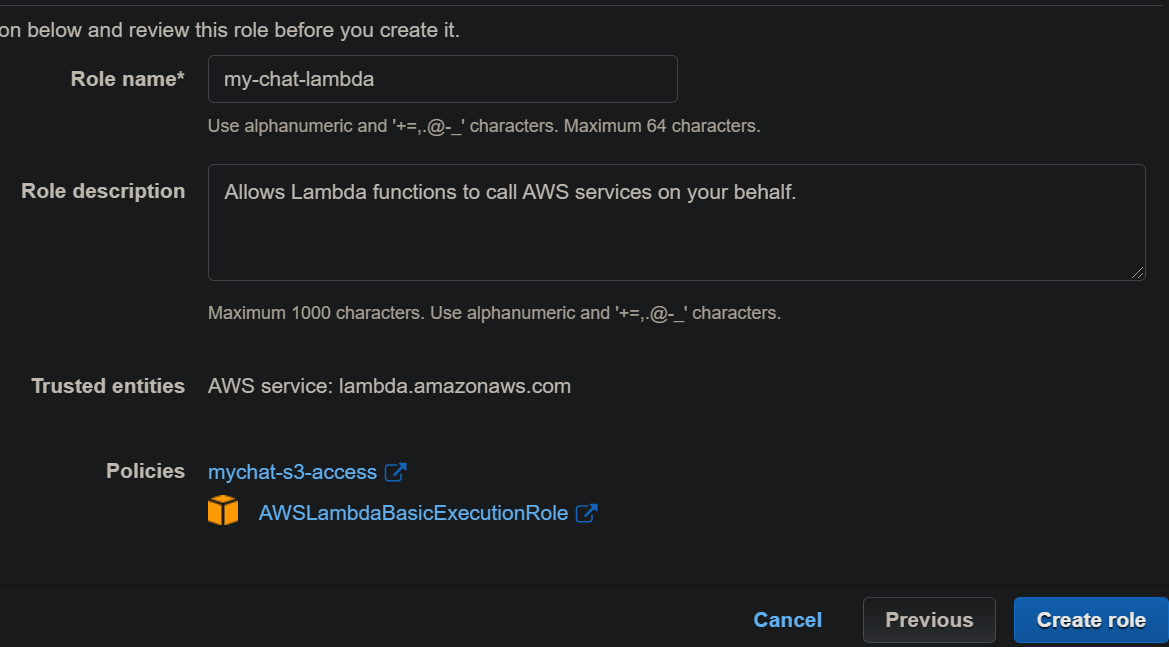


Now lets create a policy and attach it to S3 bucket so that our lambda function can access the S3 bucket and read the files in it.

1. Go to IAM
2. Create a Policy (Add ARN(Amazon Resource Number) with bucket name and object any)

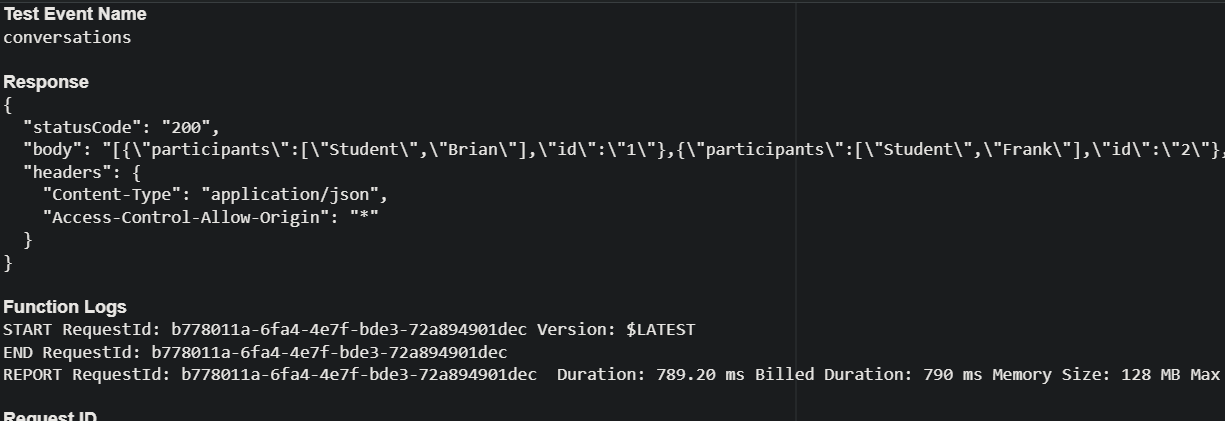


1. Give a name to policy and create it
2. Now we will create a role for Lambda and attach the Policy to the role
   1. Roles > Create a role
   2. Type of role AWS service >> Service type selct lambda
   3. Search these two policies and add to the role
      1. Chat-app-s3-access (policy for s3 bucket we created)
      2. AWSlambdabasicexecutionerole (Lambda need this for cloudwatch ..)
   4. Review the role and create it



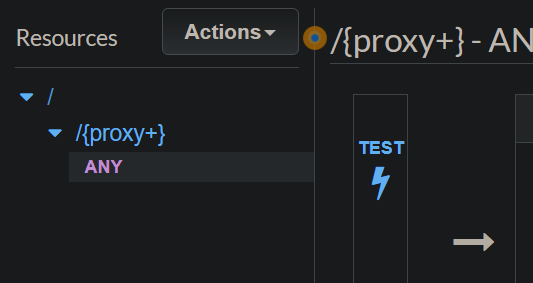
Now we will create the Lambda function

1. Go to lambda service, create function
2. Giva a name, language Node.js and select existing permission
3. Select the role we created for lambda and create the function
4. Get the code from V3/lambda folder and paste it in code function
5. Replace the bucket name with your bucket name , save/deploy it
6. Create a test event and test the lambda



Create an API Gateway for our chat application

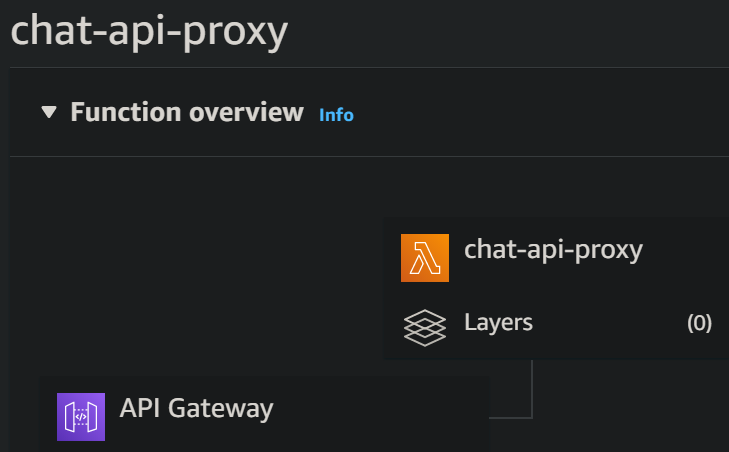
1. Build a public REST API from console
2. Select Endpoint type EDGE Optimised
3. Create Resource
   1. Enable the proxy resource & create the resource
   2. Select on ANY, select integration type Lambda function proxy, give region and lambda name and save



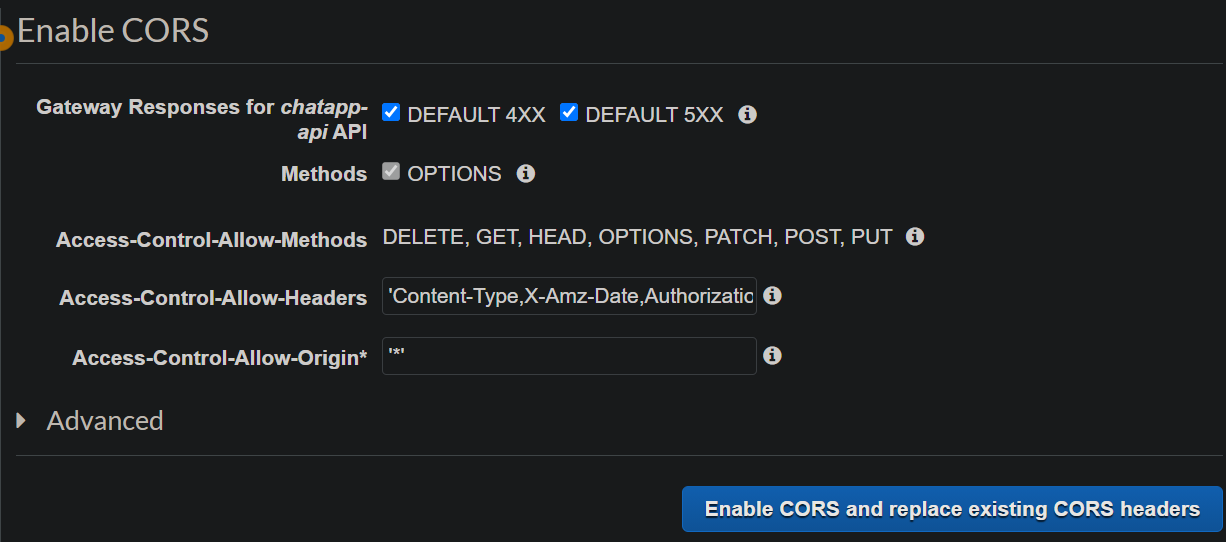
Proxy+ >> Indicating it will take all requests and pass it as it is

ANY >> Any method means e it will accept all kind of method call (GET, PUT, UPDATE)

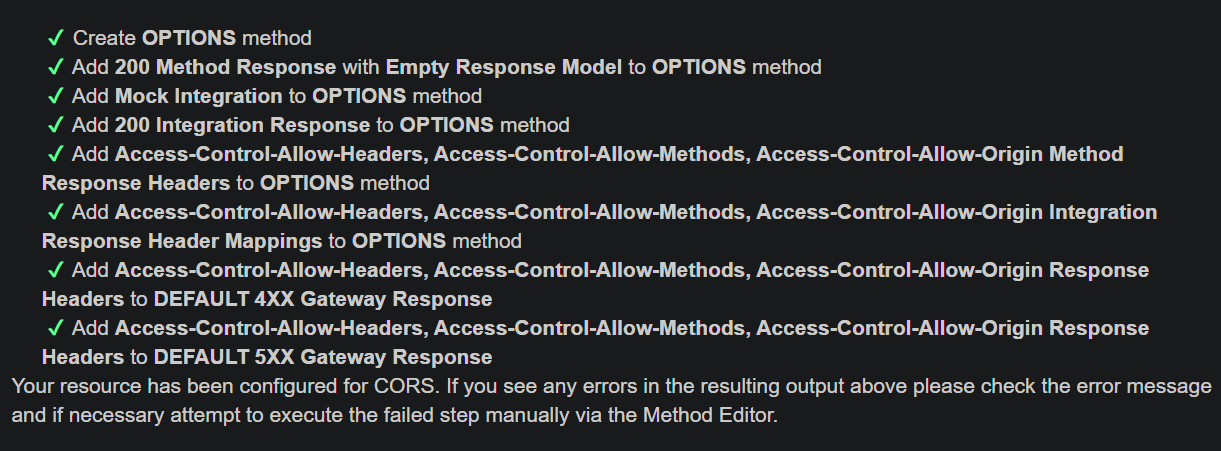
1. Do a Test call with GET method as we are reading the conversation data. We shoud get a 200 response with the JSON data.
2. If we got to Lambda function we should be able to see a trigger added to our function



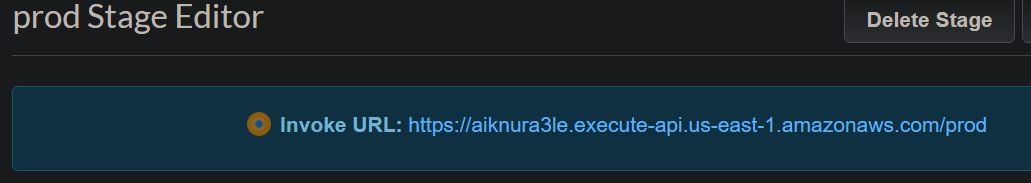
1. Now enable the CORS to our API. CORS is a complecated things but configuring it easy.
   1. Go to API Resource (For us Proxy+)
   2. Click on Actions and select enable CORS
   3. Select the default option and enable the cors



* 1. Give yes to replace existing values



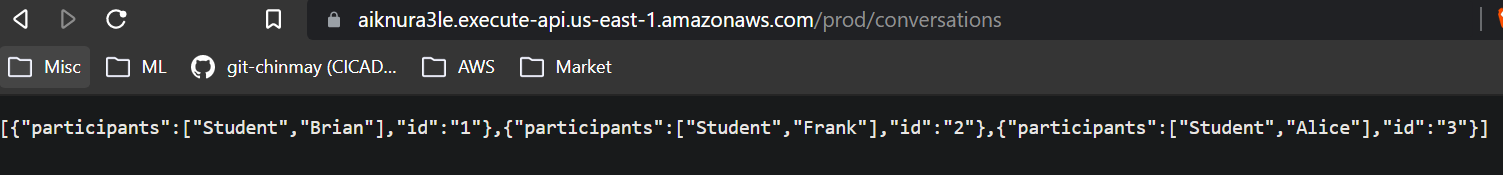
1. Now we will call the lambda via api so that we can use it in our chat application
   1. Got to api resource>> action>> then deploy with prod stage
   2. We should get an ugly url like this under stage



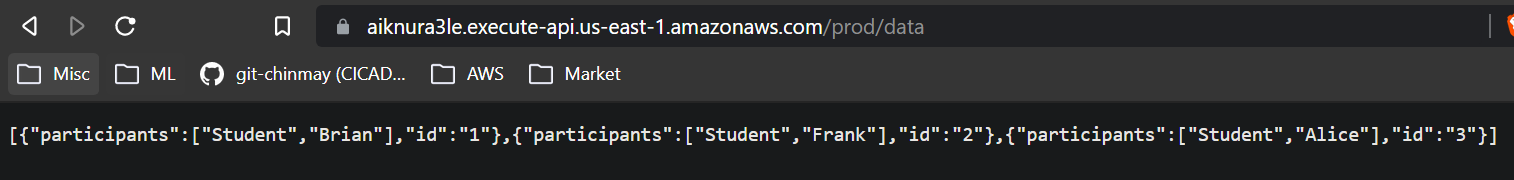
* 1. If we open the url it should throw error like below

{"message":"Missing Authentication Token"}

Add the directory conversation to the url and try again

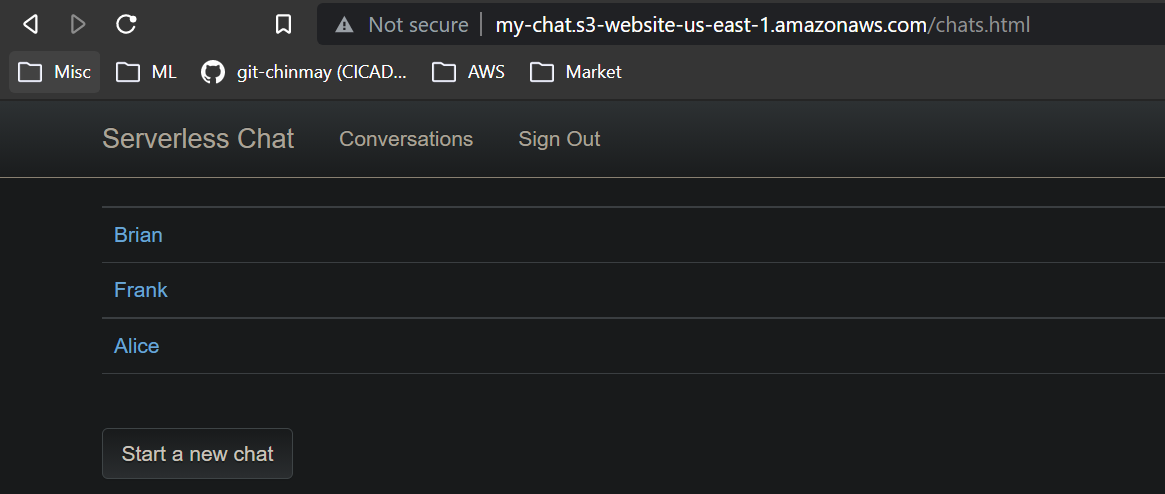


* 1. Even if you add just the data folder and call still it will work because the api just triggering the lambda and inside lambda we have mentioned to do getobject to path data/conversation.jaon which present in S3 and you can observe the 2nd time call will rertun faster because of EDGE.



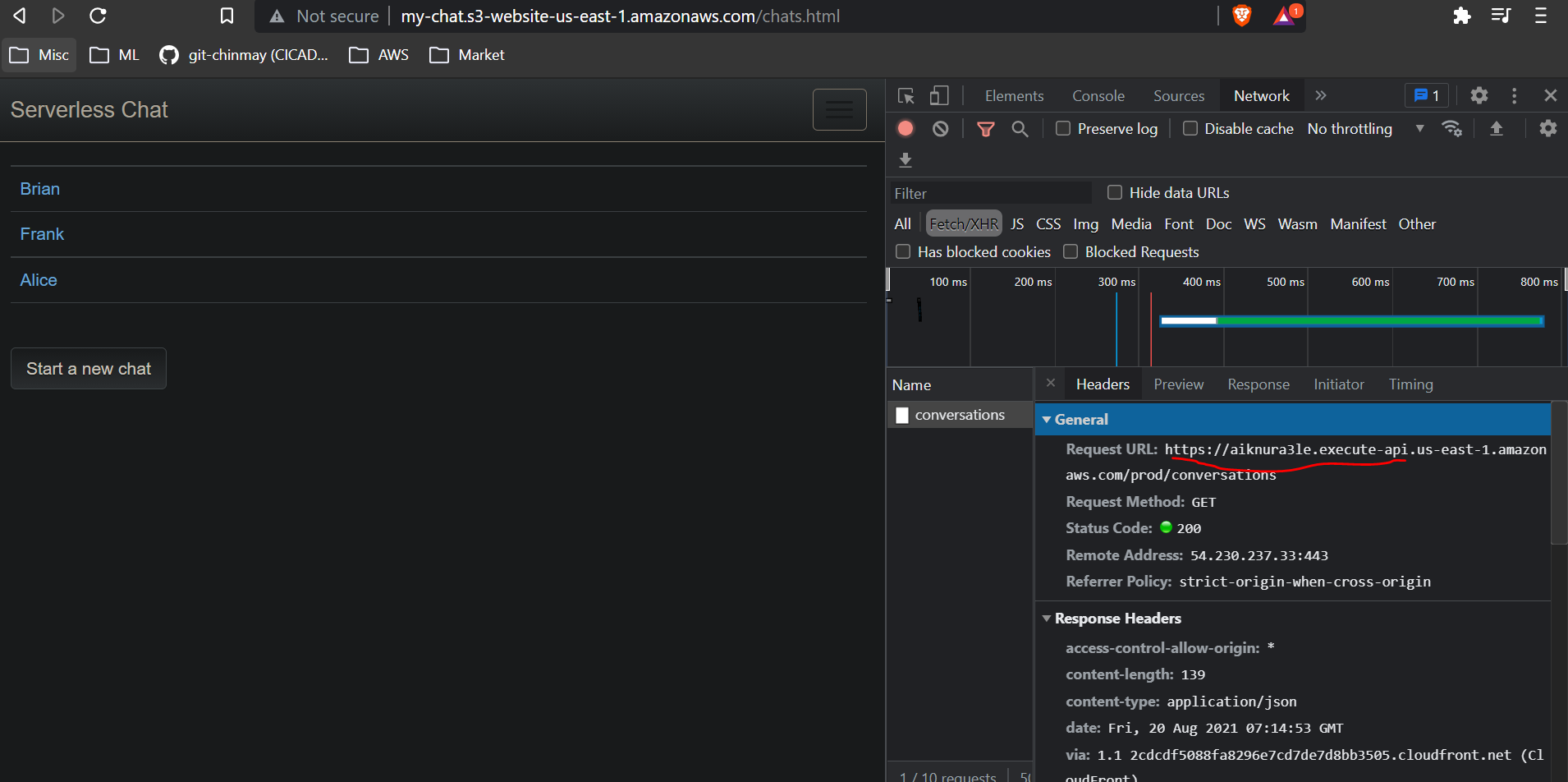
Now we will add the api link to our chat app javascript file

1. Copy the url from prod stage
2. Got to the V3/site/js and update the url in config.js file
3. Open the S3 bucket and got to JS folder, update both the site.js and config.js files
4. Open the chat-app url from static website under properties, we should see the conversation now



Now these details are directly coming via api gateway from lambda instead of static json page we created earlier.

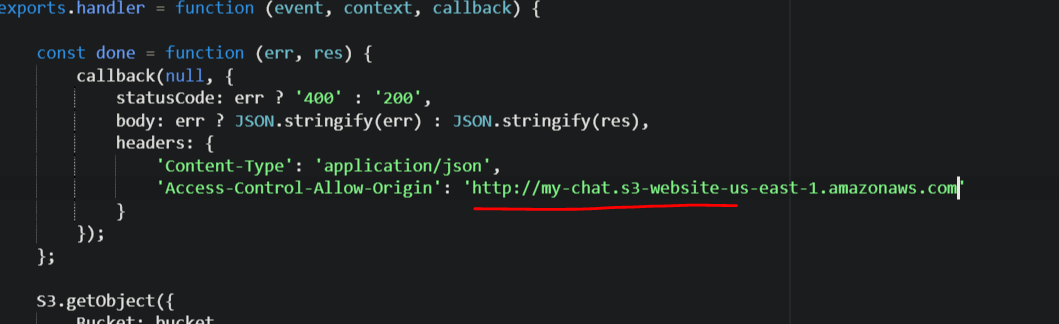
We can see that from page inspect of the browser



**Challenge: Making application more secure**

Currently our lambda is taking anything from CORS which is not secure. So make it only accepts our S3 bucket

Replace the \* with this <http://my-chat.s3-website-us-east-1.amazonaws.com>

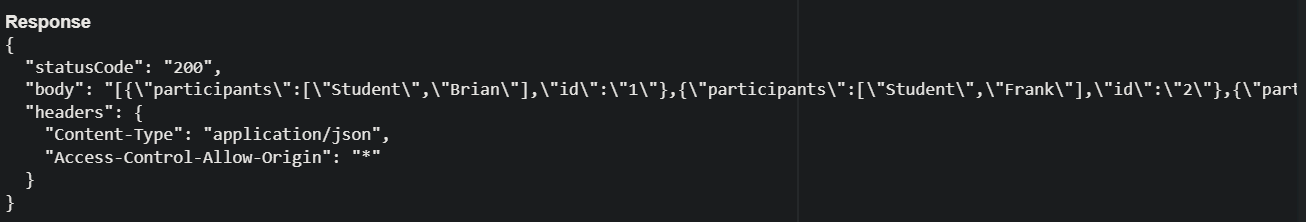


Previously we feched all the conversations via api now we will try to fetch individual conversations using api

1. Update the lambda code from resource/v4/lambda/ file
2. Change the test event as below



1. Test the lambda



1. Creating a new test event for individual chat



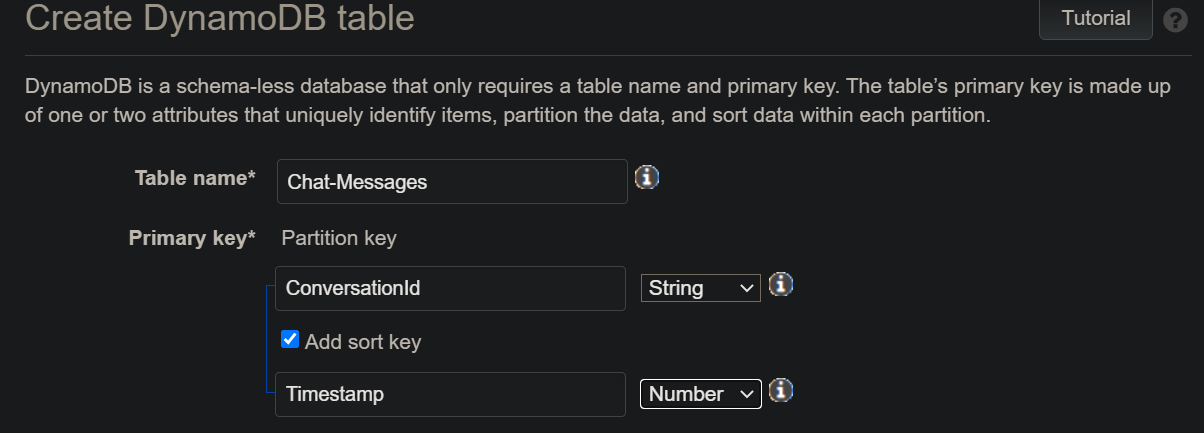
1. Test it, it should pull the 1.json file content now



1. Now upload the site.js from v4/site/js folder to S3 bucket js/ folder. Now we should see the individual conversations from api

**Creating a DynamoDB Table**

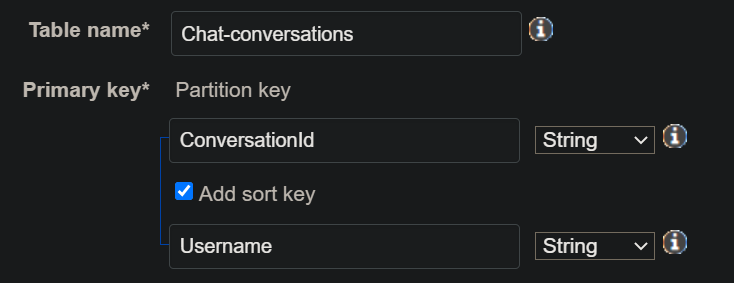
1. Got to DynamoDB & create table



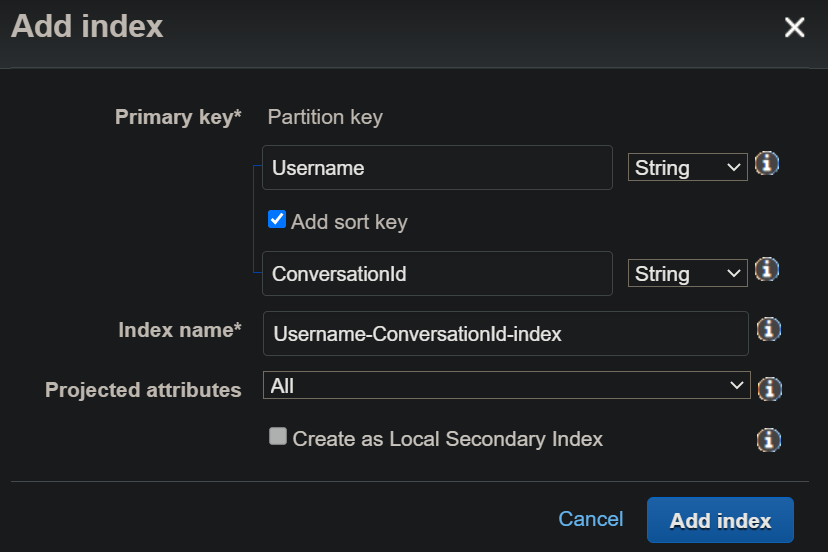
1. Uncheck default setting option
2. Remove Autoscaling Read and Write capacity
3. Set Provisioned read & write capacity for the table to 1 and 1
4. Create the table

Create another table

1. Create table with a new name



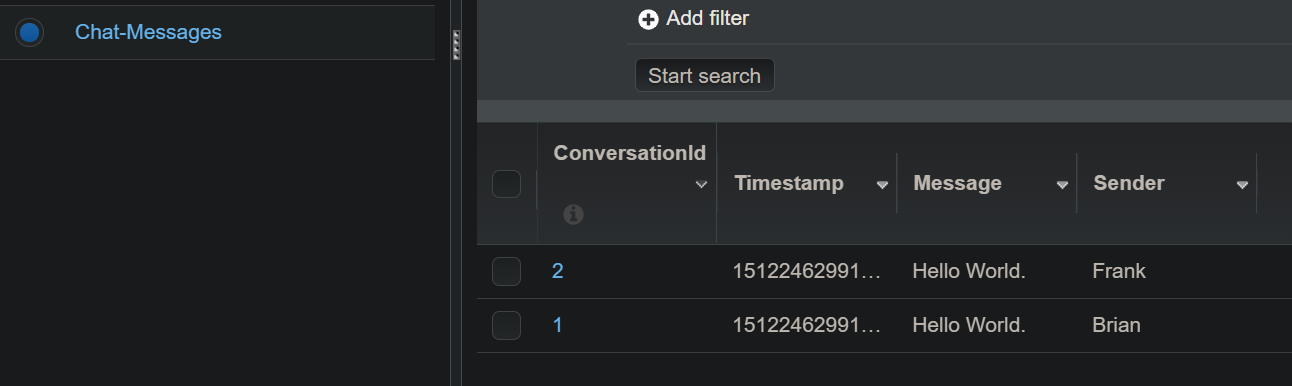
1. Add the secondary index

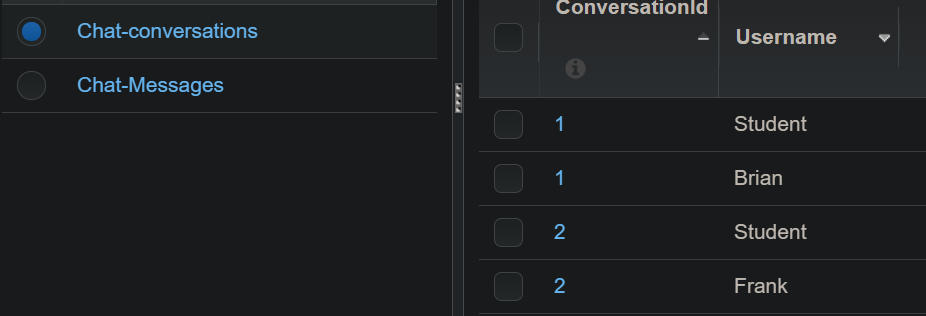


1. Remove auto scaling feature and set the read and write to 1 and create the table

**Feeding sample data to dynamodb**

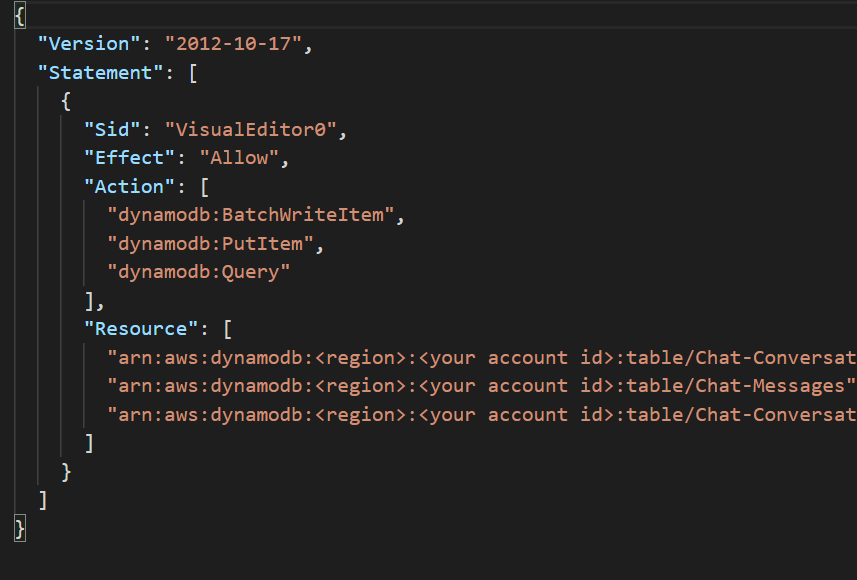
1. Open dynamodb and select the table
2. Create item
3. Update the value for Conversationid and Timestamp
4. We need to create attribute for Sender and Message (We can do this by append attribute)
5. Save it. We should see the row with our data in the table Chat-Messages
6. To add new rows we can select the row >> go to action>> duplicate >> then change the values and save it



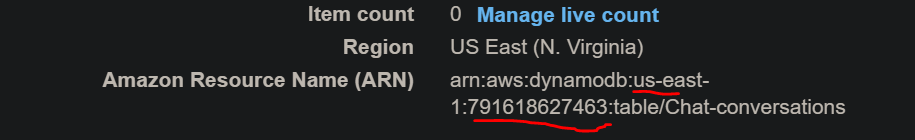
1. Now we will create sample data inside conversation table
   1. Create item and add the values as below
   2. 

**Update lambda policy to access DynamoDB**

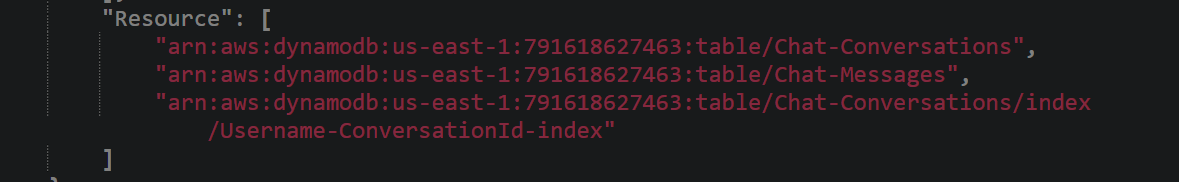
1. Go to the policy file under resource/V5/



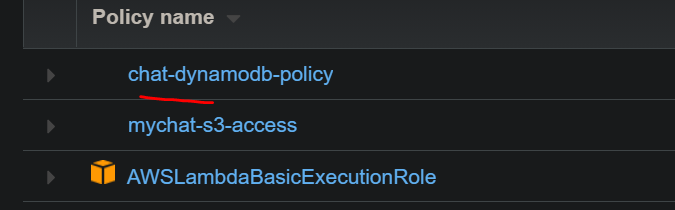
1. Get the region and account id name from general page of the table and update in policy



1. Go to Policy>>create policy>>json format>> paste the json query from resource>>update the region and id number

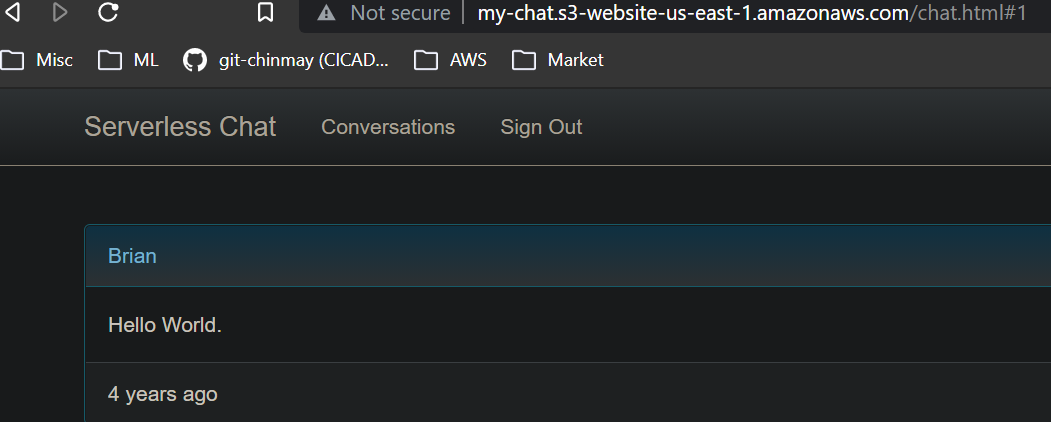


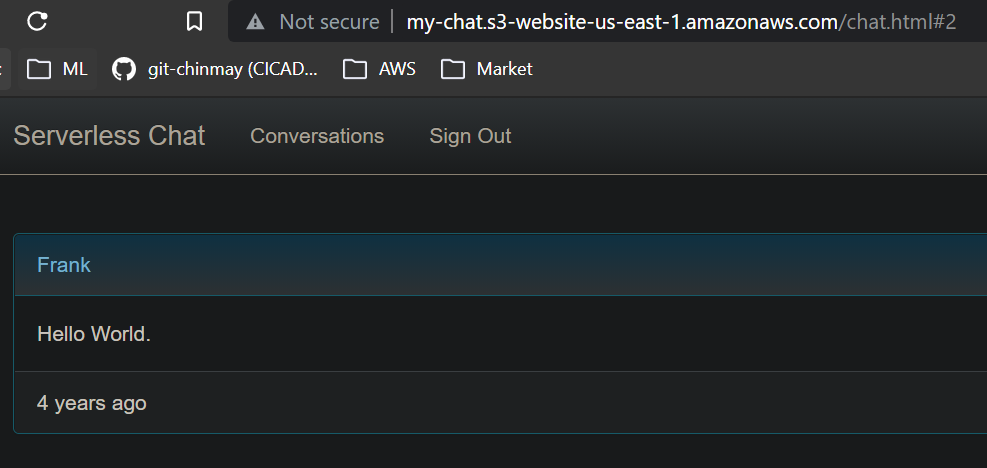
1. Create the policy
2. Now got to lambda role and attach the policy



**Read a conversation from DynamoDB**

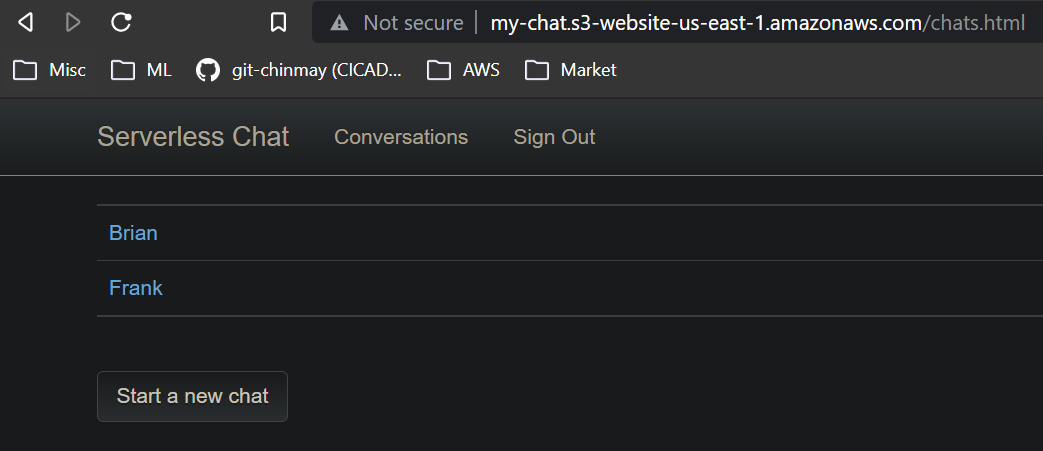
1. We need add the current time on chat message . To do that get the latest lambda code from V5 folder and replace it in lambda . (Make sure to update the bucket name)
2. We can see our chat application now and can see data is coming from database





**Read conversation list from DyanmoDB**

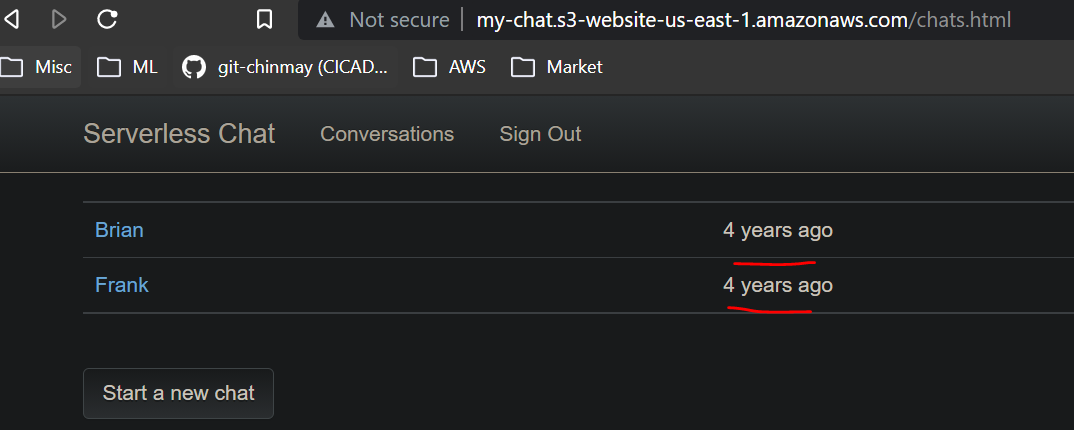
1. Go to V6/lambda folder and update the code (You can notice here we have removed the S3 bucket completely)
2. Now test the our chat app



1. We should see two users from table instead of 3 users from S3 bucket
2. Now we will update the new site.js file from V6 folder and will upload to S3 bucket/js folder.

(Remember we have just moved the conversation part to DynamoDb, the frontend still sits in S3 bucket)

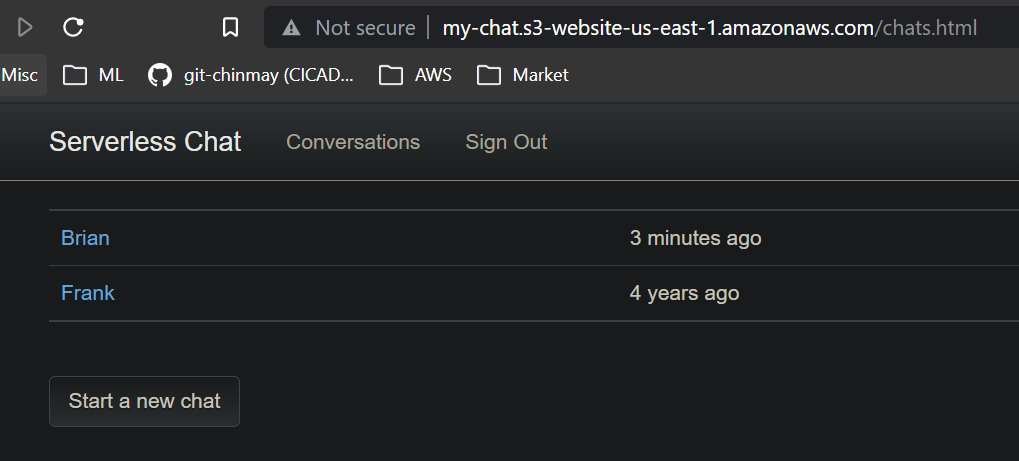
1. Do the hard reset the webpage and you can see the last chat time for each user on app



**Write New Messages to DynamoDb**

1. Get the lambda code from V7 and update the function
2. Upload the V7/Site/JS/site.js file to bucket
3. Upload the V7/Site/Chat.html file to bucket
4. Hard Reset the chat app page
5. Clickon any user (Do a hard rest again for safer side)
6. Send a message and we can see the our message with latest time

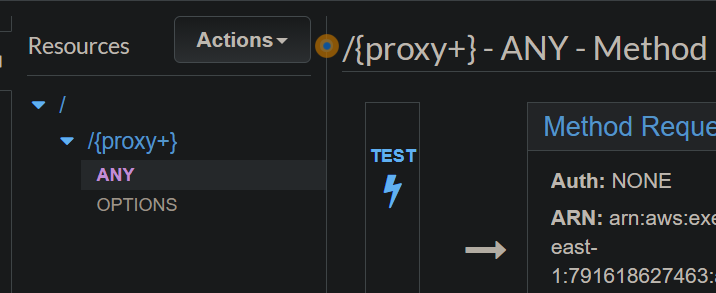




**Restructuring our chat API using API Gateway:**

Currenlty our api using proxy mode which we want to get rid off.

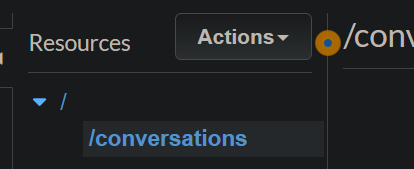
We can see that e have two resources / and /{proxy}, we cant do anything for / but we need to remove the proxy one.



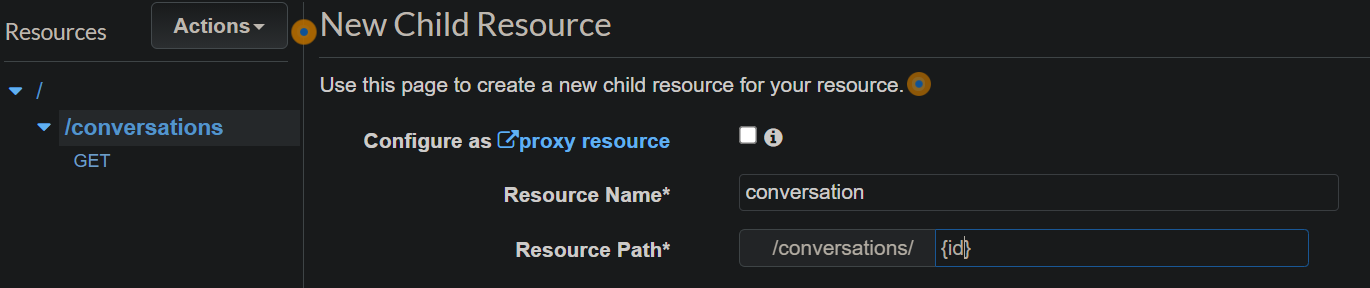
1. Select the proxy resource >> go to action >> delete the the resource

(Don’t delete the api , make sure to delete the resource)

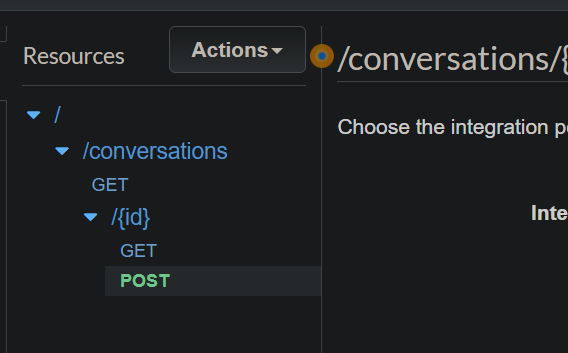
1. Create a new resource named conversations



1. Create a GET method in it (Don’t add any integration type now)
2. Create a new resource under conversations (child resource) , replace the auto created path with {id} and create it.



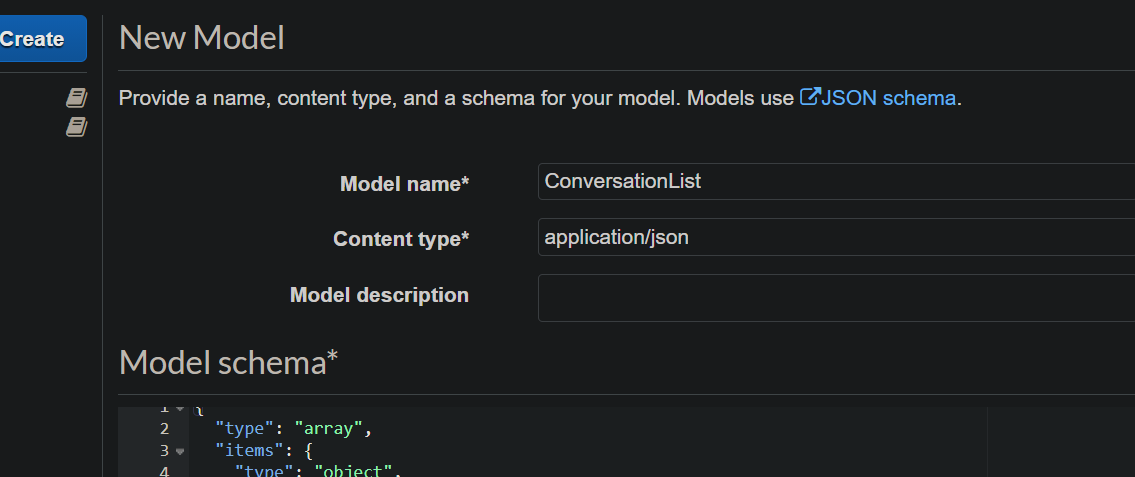
1. Create a GET and POST method for this child resource



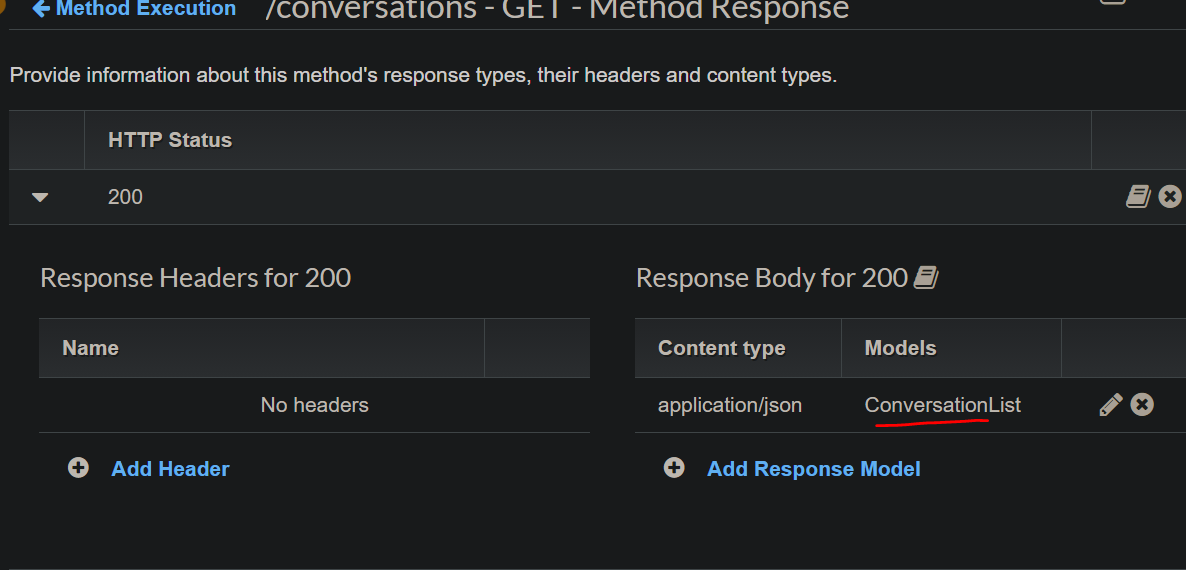
1. Now lets create lambda functions based on this resources. Go to lambda and create a function Chat-Conversations-GET (Make sure to use the existing Lambda role)
2. Now got to V8/Lambda/Chat-Conversations-GET .js code and paste it in Lambda
3. Create a test event test it. It should return all the conversations.
4. Now integrate it with our API gateway
   1. Got to GET method of Conversations resource
   2. Add Lambda function without proxy



* 1. We can see the Model type is empty, so we need to add a model
     1. Go to models and click on create
     2. Give a name and content type
     3. Get the schema from V8/Models/ConversationList folder and create it

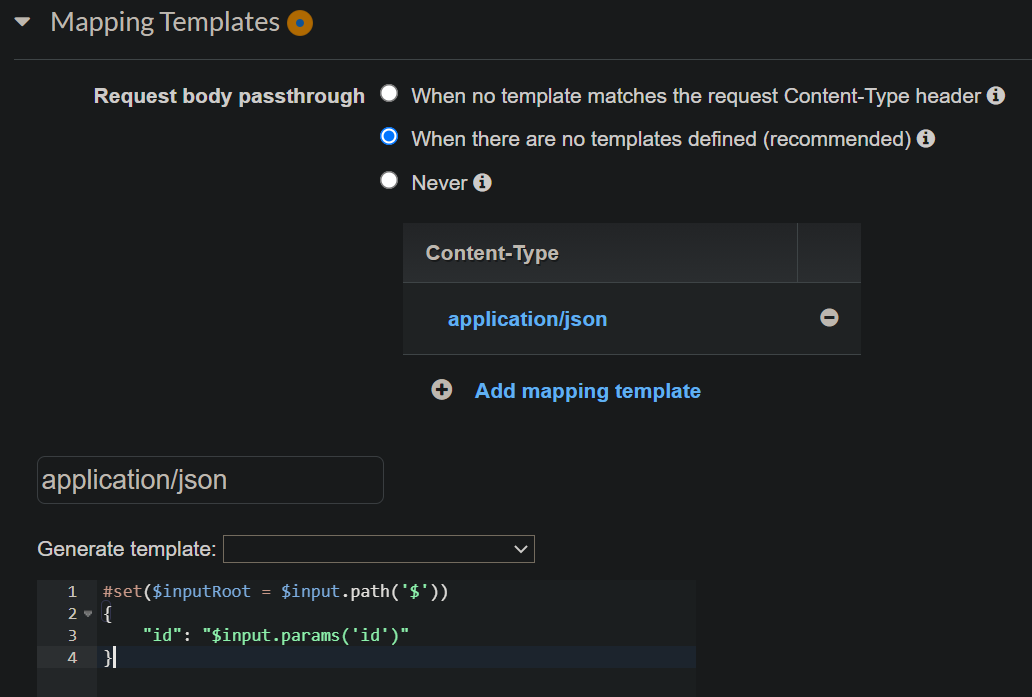


* 1. Now go back to Resource Method GET
     1. Click on Response Method
     2. Expand the Respond 200
     3. Add the Model under Response Body 200

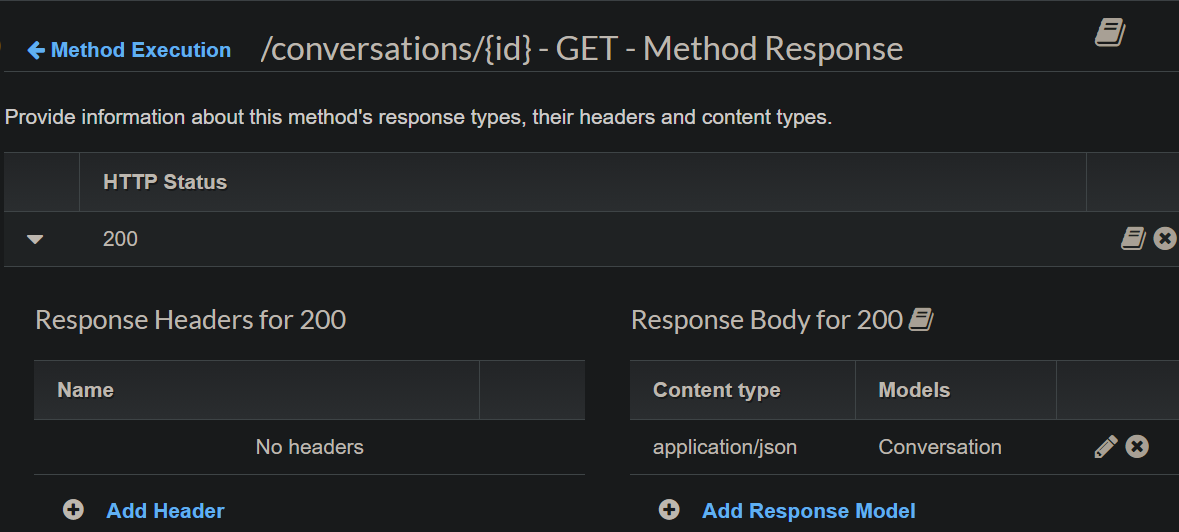


* + 1. Now do a Test using test link. It should return the conversations details as usually return

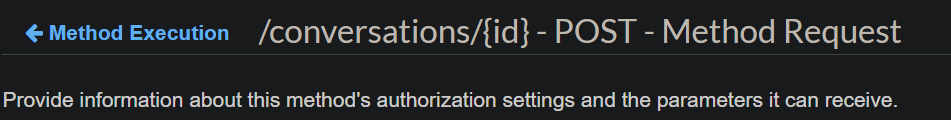
1. Now we will enable the CORS
   1. Go to the resource COnversations
   2. Action>> Enable the CORS
   3. Click on Enable the CORS and make sure all the points check marked
2. Now we will create Read and Write Lambda for individual message
   1. Create a function Chat-Messages-GET
   2. Copy the corresponding code from V8/Lambda
   3. Create a test event and test the code
   4. Create another lambda Chat-Messages-POST
   5. Copy its code from resource and deploy the function
3. Attached both lambdas to API gateway. (Don’t enable CORS for them)
4. Create a model called Conversation
5. Create another model called NewMessage
6. Now got to {id} resource and integration of GET Method
   1. Go to mapping template section
   2. Feed the details as below

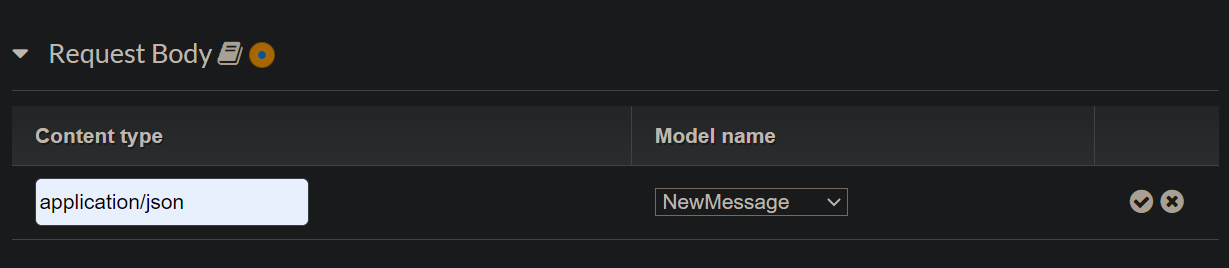


* 1. Get the generate template from V8/mapping template folder
  2. Do a Test by provide a id (1 or 2 as we have only 2 ids now) and we should get the response
  3. Add the Conversation Model to HTTP response body of GET

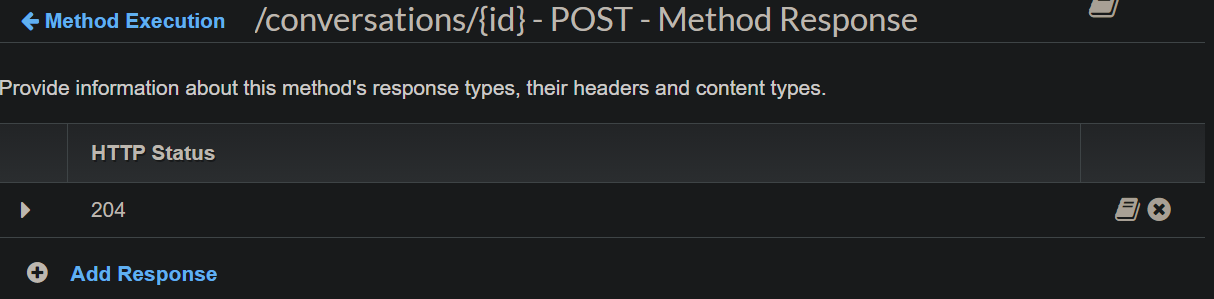


1. Perform the same for POST method of {id} resource
2. Add the Model to Request method not in Response Method as while posting the message we are not expecting any response

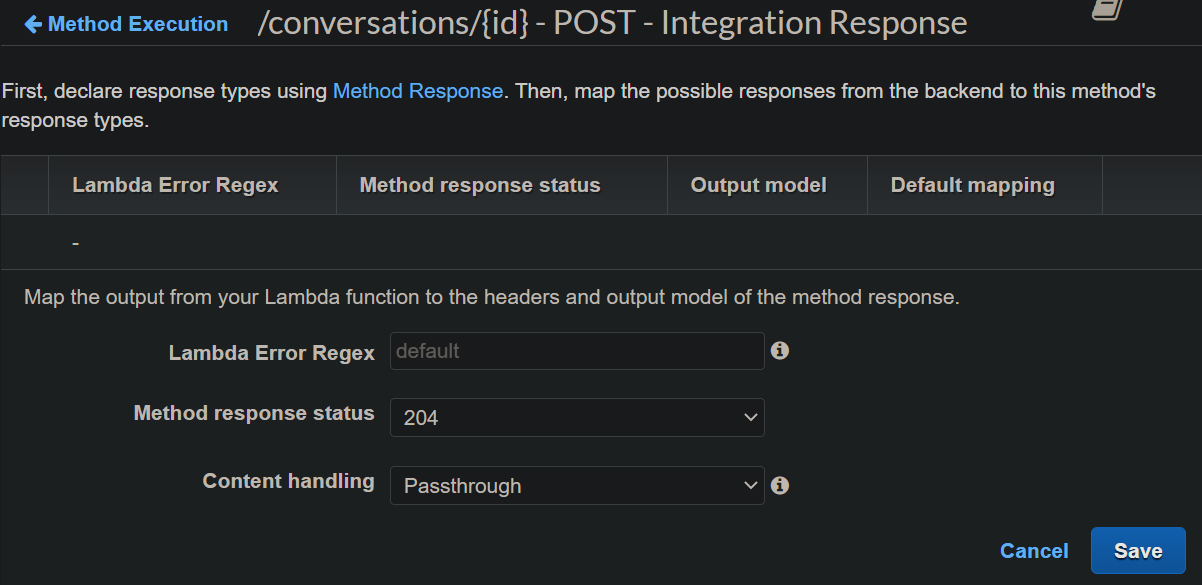




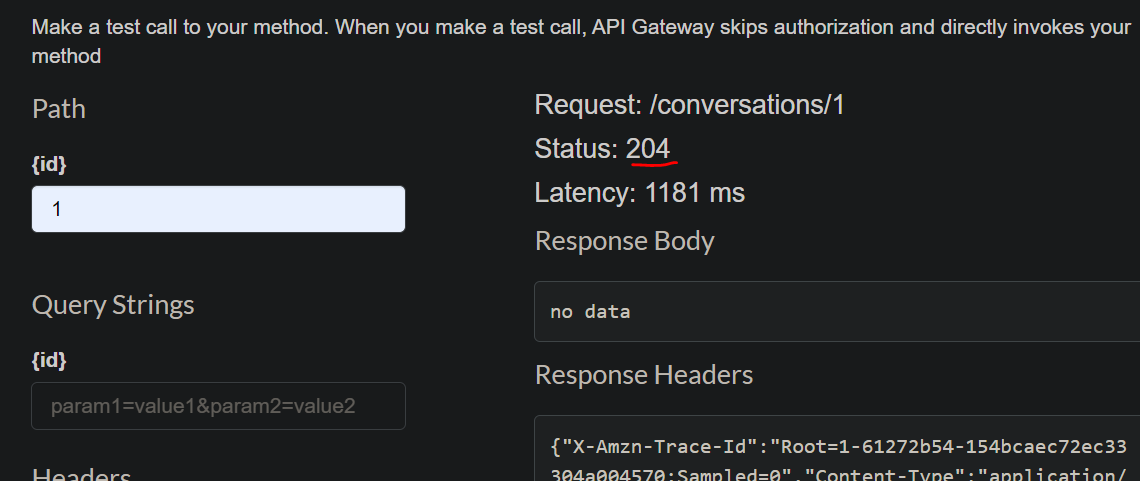
1. We want Response code 204 for successful POST message not 200
   1. Got to Method Response section of POST Method
   2. Remove the 200 response code
   3. Add 204 Response code



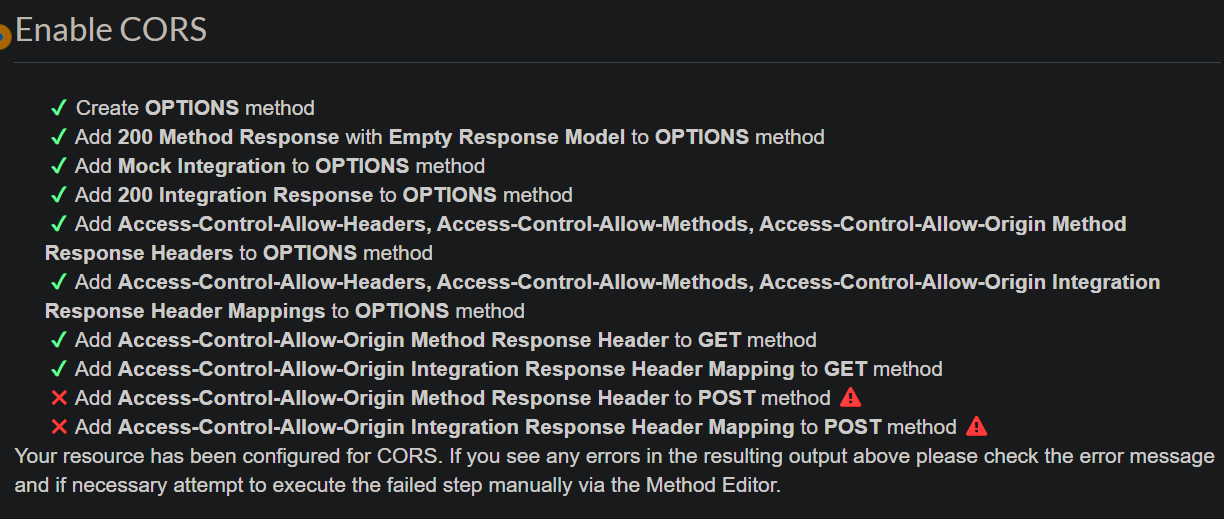
* 1. Add new code to Integration Response



* 1. Remove 200 from there
  2. Do a Test run by passing id and message body



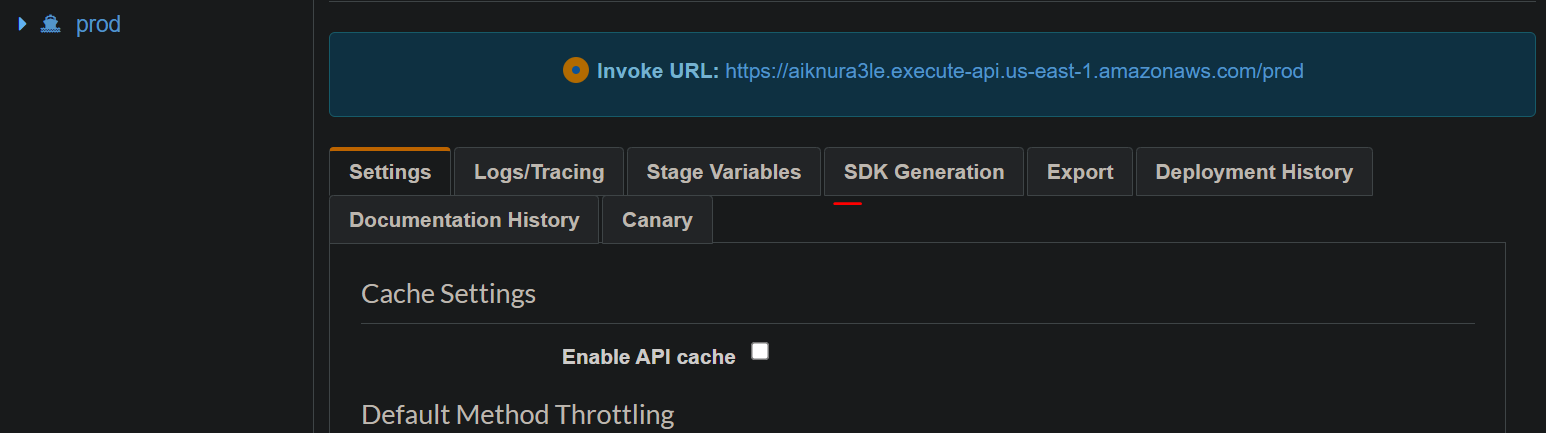
1. Enable CORS for {id} resource. You will see some failure but that is ok as they are one way communication



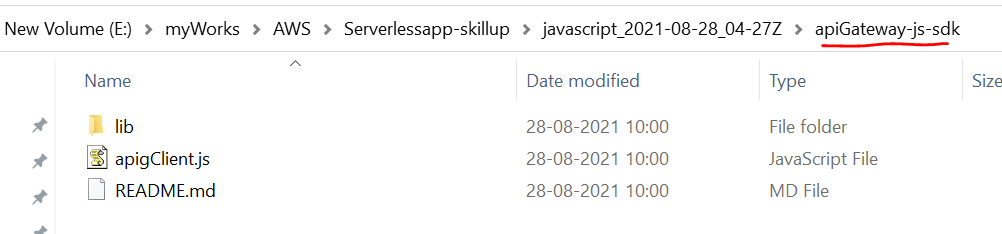
1. **STAGES**

(Will generate JavaScript SDK client from API Stages)

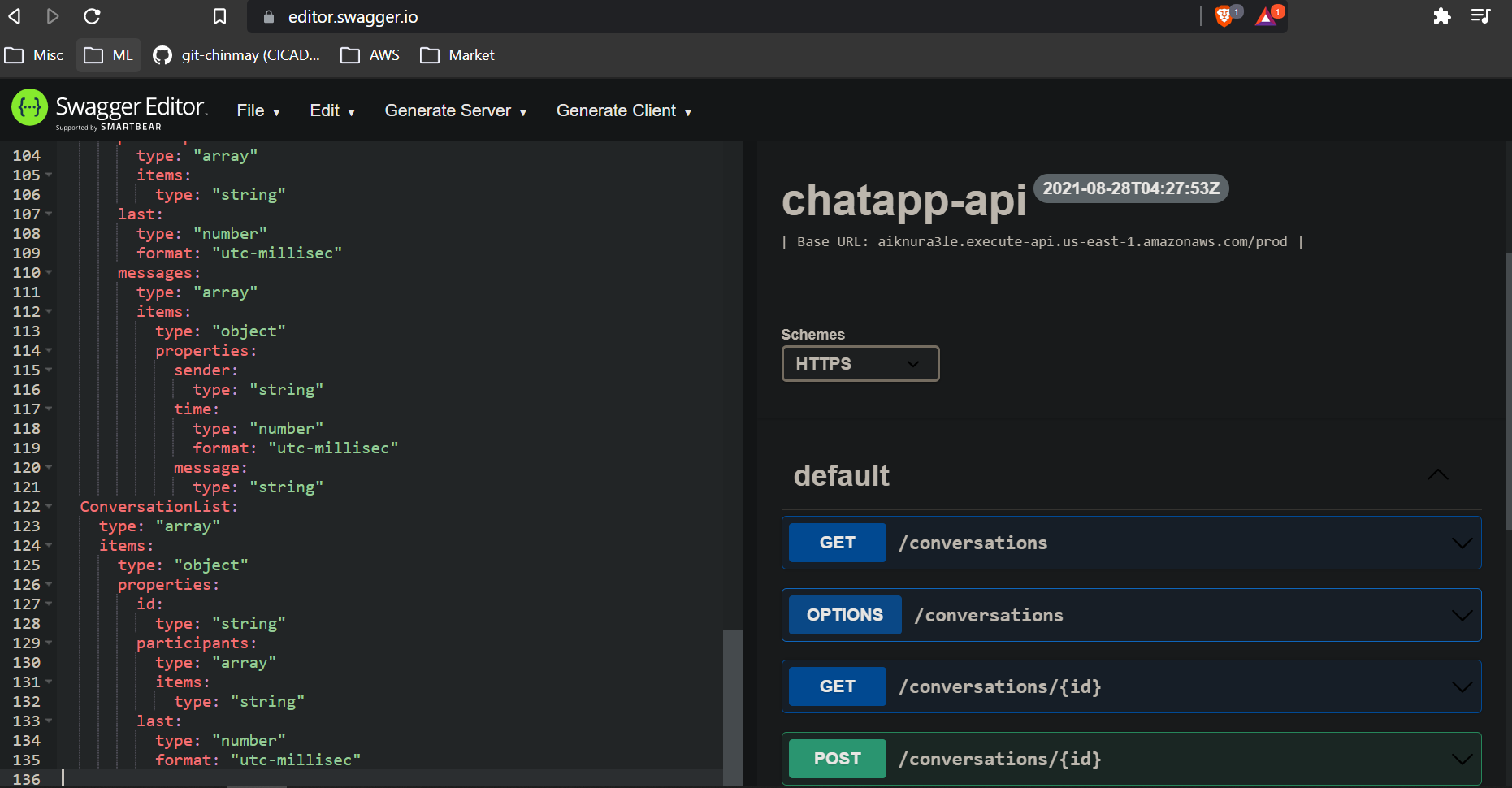
1. Got to your root resource(“/”) and deploy the API to a stage(Probably prod stage)
2. Once deployed go to the SDK Generation of Stage editor



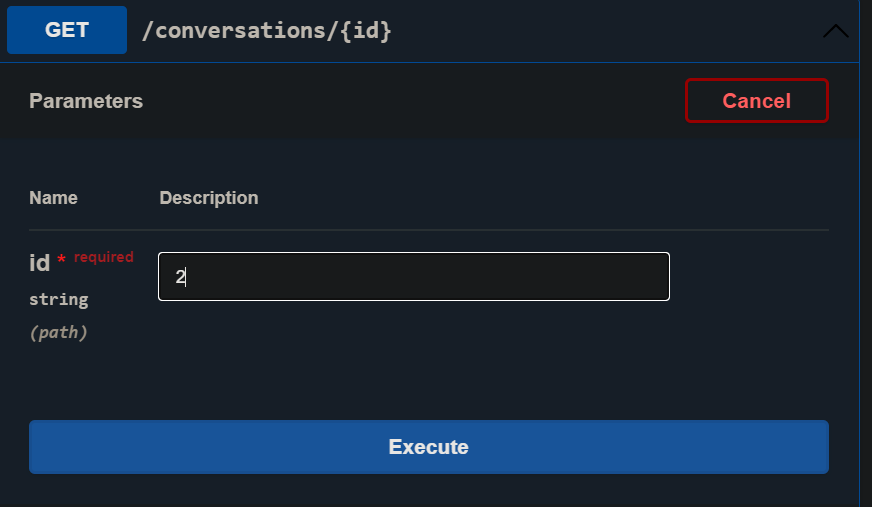
1. Select the JavaScript from drop down platform and Generate
2. A zip file will be downloaded and upon extracting you will found below



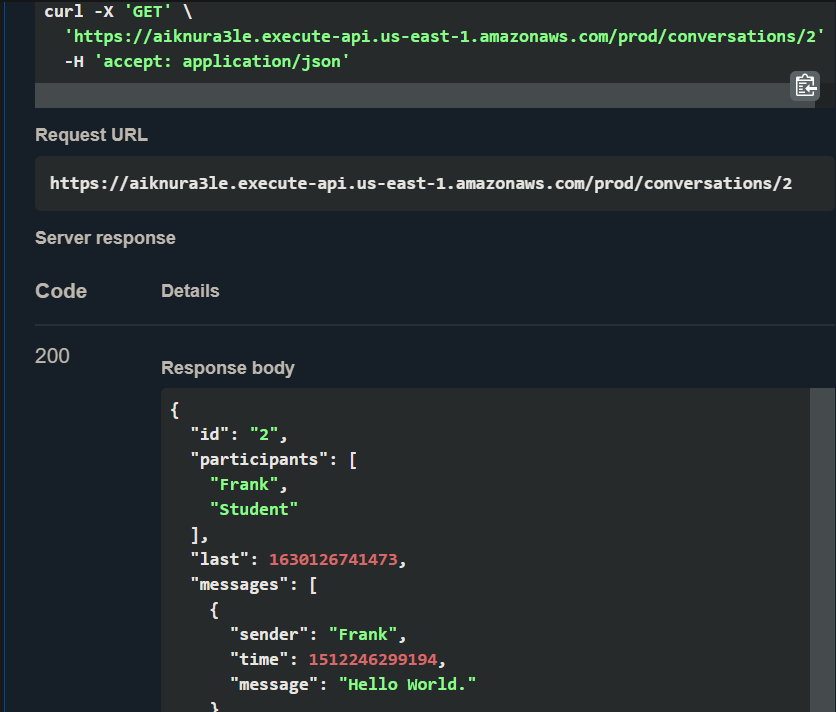
1. Now upload this apiGateway-js-sdk folder to S3/JS folder
2. Now go to root directory of the S3 bucket and upload the files & folder from Resource/V9/Sites directory
3. Once done check the chat app. Now our app is working from the code generated from SDK generator
4. We can generate the SWAGGER API or Postman code from our API gateway
   1. Got to the stage of the API
   2. Click on Export option and choose Swaage
   3. We will choose swagger YAML option and one file will get downloaded
   4. We also can take the code form window and open editor.swagger.io and paste it there



* 1. You may see some error on top but that’s ok as AWS generated code is not perfect
  2. We can try our api call from here
     1. Lets call individual conversation GET

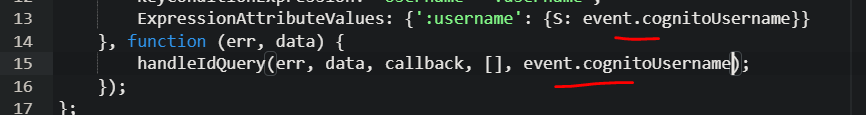


` ii. Execute and we will see the conversation lists from user id 2

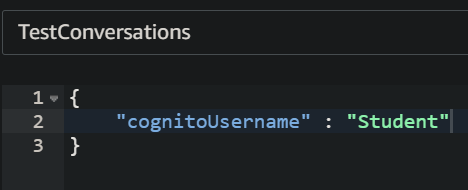


Replace the hardcoded Student with **cognitoUsername** variable

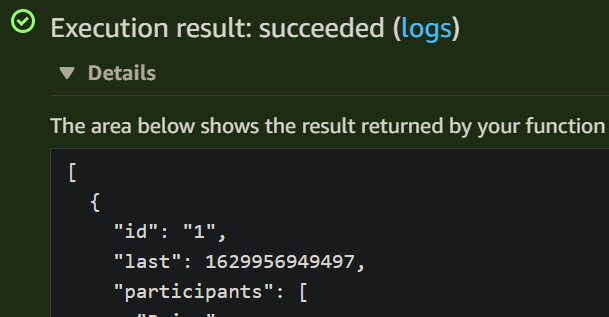
1. Open the lambda (Chat-Conversations-GET)
2. Replace the Student with the variable



1. Create a Testevent and test it

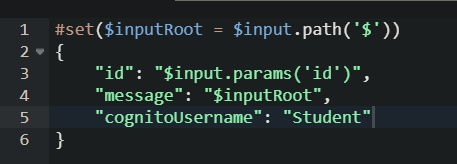


1. Test was successful



Now we have to map the variable to the api gateway

1. Go to the apigateway
2. Click on Resource {id} POST method
3. Go to Integration part
4. Got to mapping template and click on application/json
5. Add the new id to the json (currently we are agaion hardcoding student for time being but we will remove it later)
6. Save the chnages



1. Do the same to Conversations-GET resource
2. But here we have to add a new mapping template and then add the above content



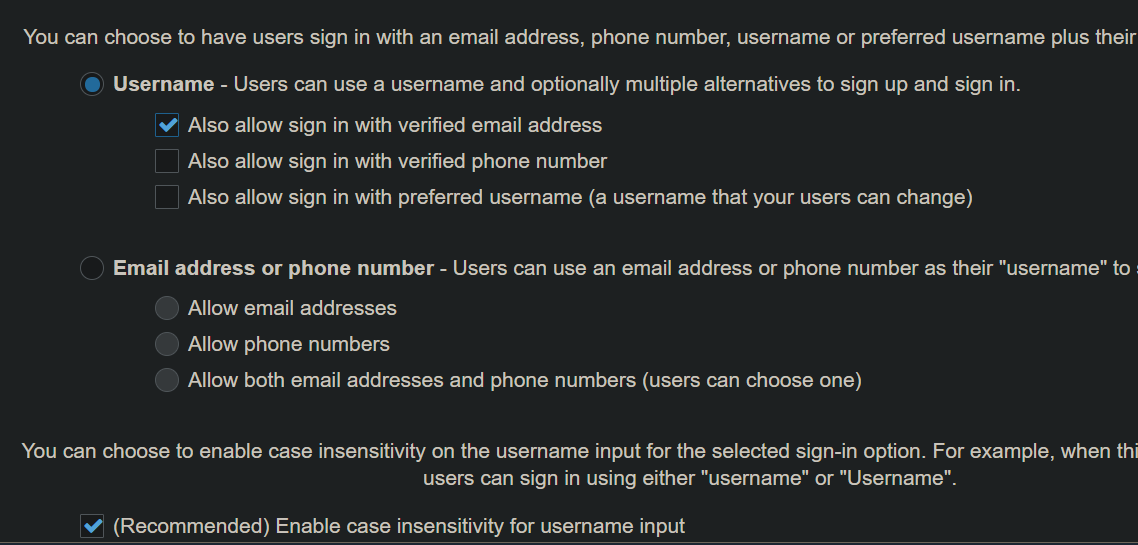
1. Save the changes and deploy the API
2. As we redeploy the API with changes we have to re generate the SDK code
3. Upload the sdk file to the s3 bucket/JS folder
4. Now check the chat app its working as usual or not

**COGNITO**

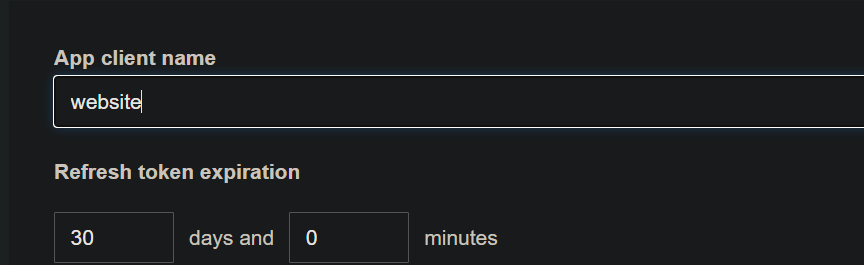
We will add the identity to our app and for that we will use AWS Cognito service

We will first create a Cognito user pool

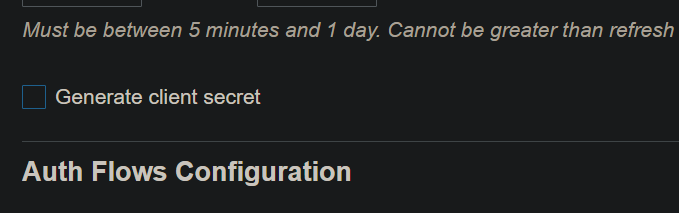
1. Go to AWS Cognito
2. Select Manage User pool
3. Create a User Pool
4. Give a name and Select Step through settings
5. For us we will go with default



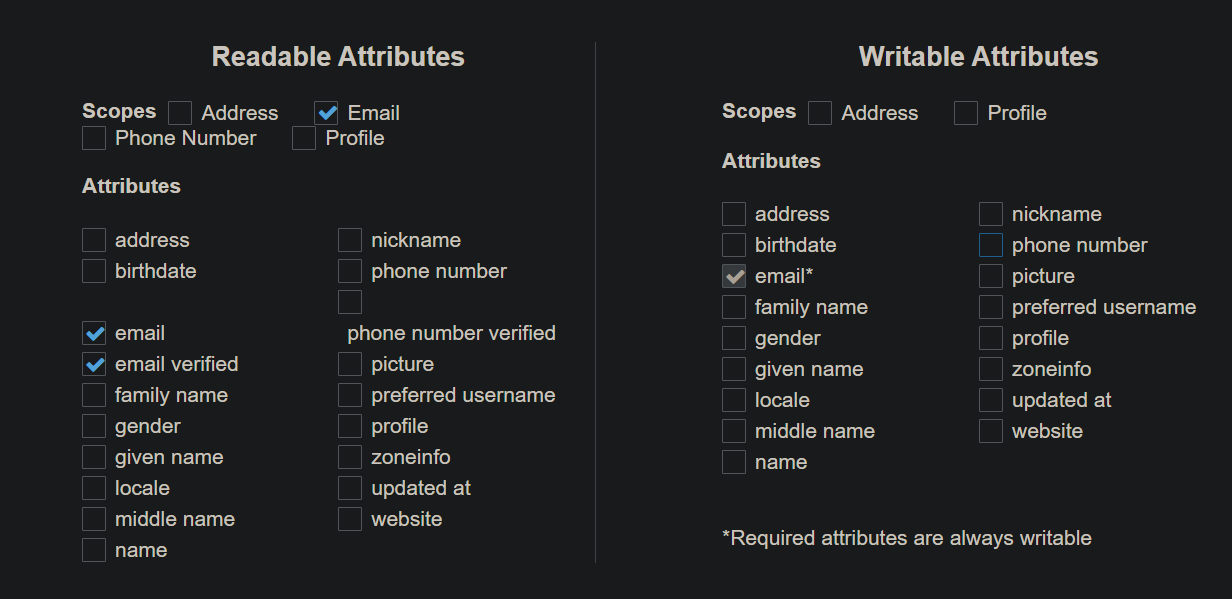
1. Next step. Change value as you wish or go with default
2. Next step. We will leave this as it is bcz setting up MFA will cost money
3. Next step. Can customise the email sendings if you wise
4. Net step. Add tags if you want or go default
5. Next step. Remember devised. Not required forour test app
6. Next step. Add a client to access the app



Uncheck Generate client secret as we don’t need forour app



Remove everything from Read and Write attribute except email



For Everything else Go with default setting

Create app client

1. Next step. Trigger we don’t need.
2. Next step. Review
3. Next step. Create the user pool

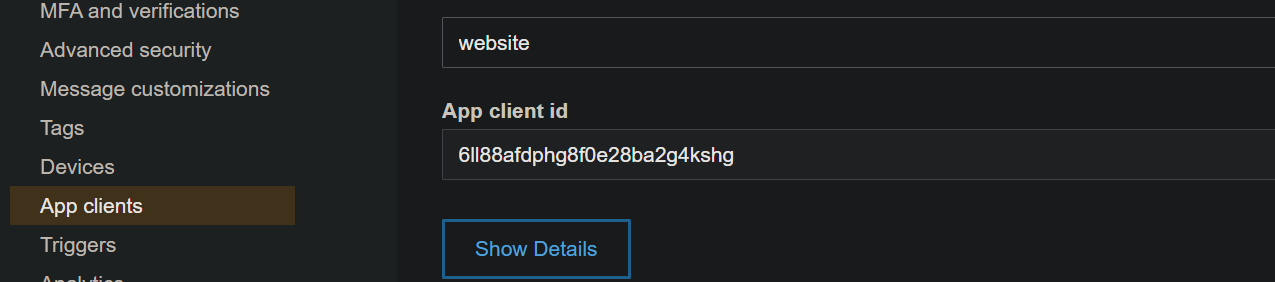
Pool Id us-east-1\_Wkfe51ChZ

Pool ARN arn:aws:cognito-idp:us-east-1:791618627463:userpool/us-east-1\_Wkfe51ChZ



1. Get the client id from app client

6ll88afdphg8f0e28ba2g4kshg

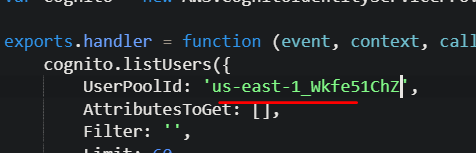


Now we want our Lambda able to listouts the users from Cognitouser pool. For that we have to create a policy.

1. Get the Userpool ARN details
2. Replace the placeholder arn with above one at V10/Policies/lambda-cognito.json
3. Create a new policy in IAM , paste the json code , give a name(Lambda-Cognito), Create
4. Create a new role for Lambda (Lambda-Cognito)
   1. Create role
   2. Attach the above policy along with AWSLambdaBasicExecutionRole

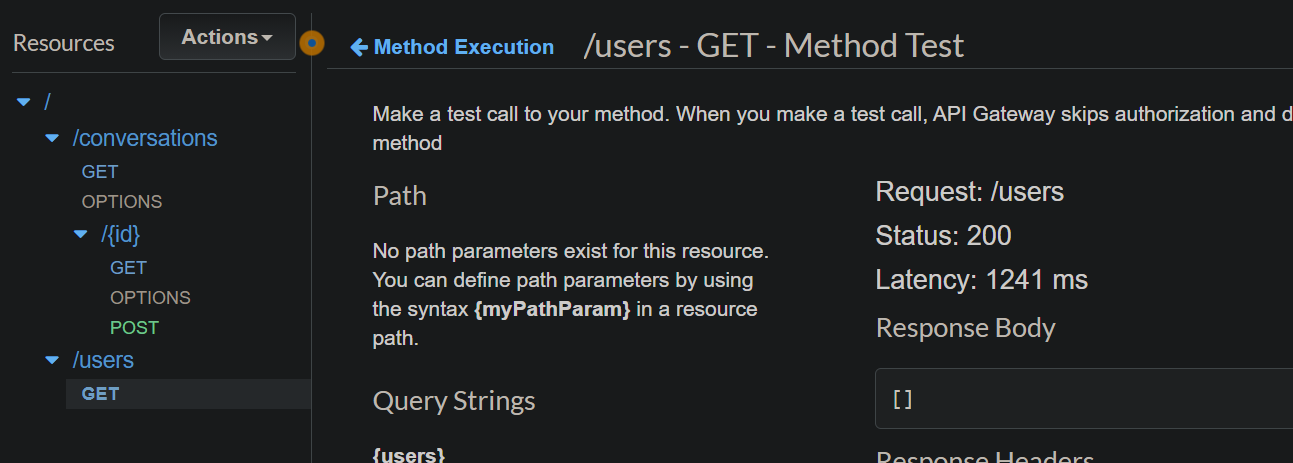
Now we will create a Lambda to get the Cognito user details using above Roles

1. Create a function named Chat-Users-GET
2. Get the code from V10/lambda
3. Update the user pool id

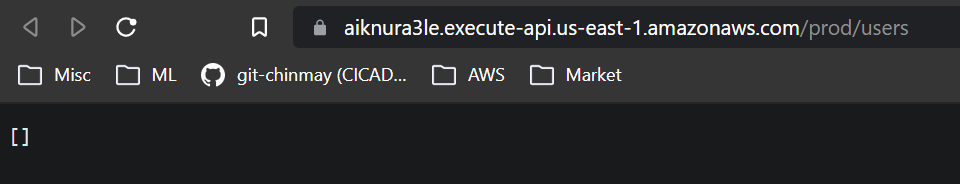


Now we will link the lambda with our api gateway

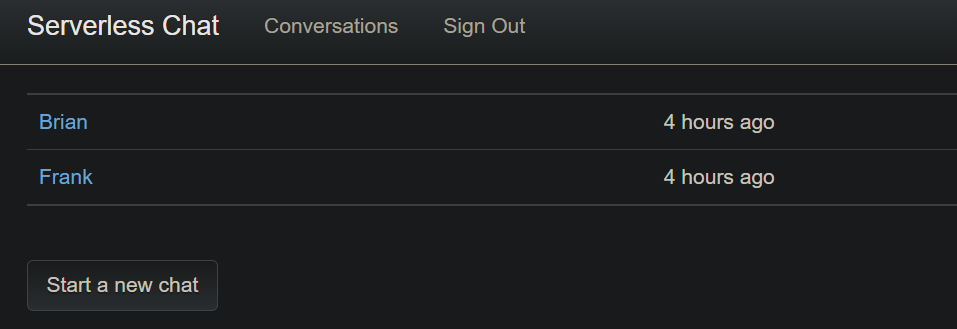
1. Create a Model named UserList and use the schema from V10/Model
2. Now create a Resource named /users under root resource & a GET method to it
3. Add above lambda to it
4. Add the UserList model to the Response Method
5. Test the Method, we should get 200 with empty output as we don’t have any users in users pool as off now



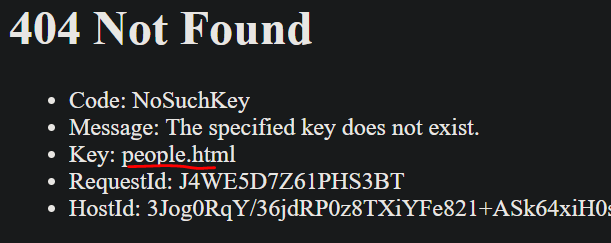
1. Enable CORS for the /users
2. Deploy the API
3. Test the API , it should work



Now as we have Lambda that is able to listing out Cognito users, lets go and tied it with Chat app.

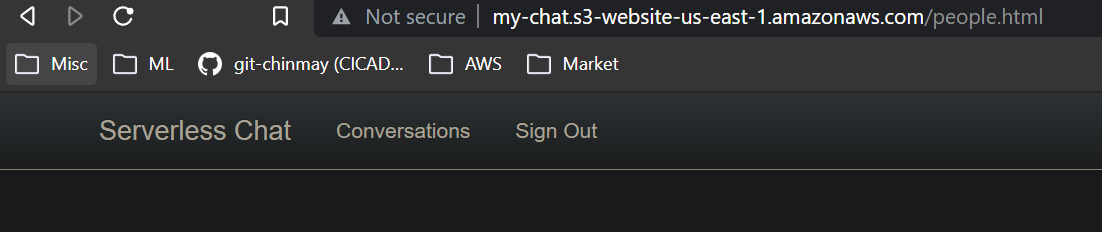


Currently if we click on Start New chat , it will throw error

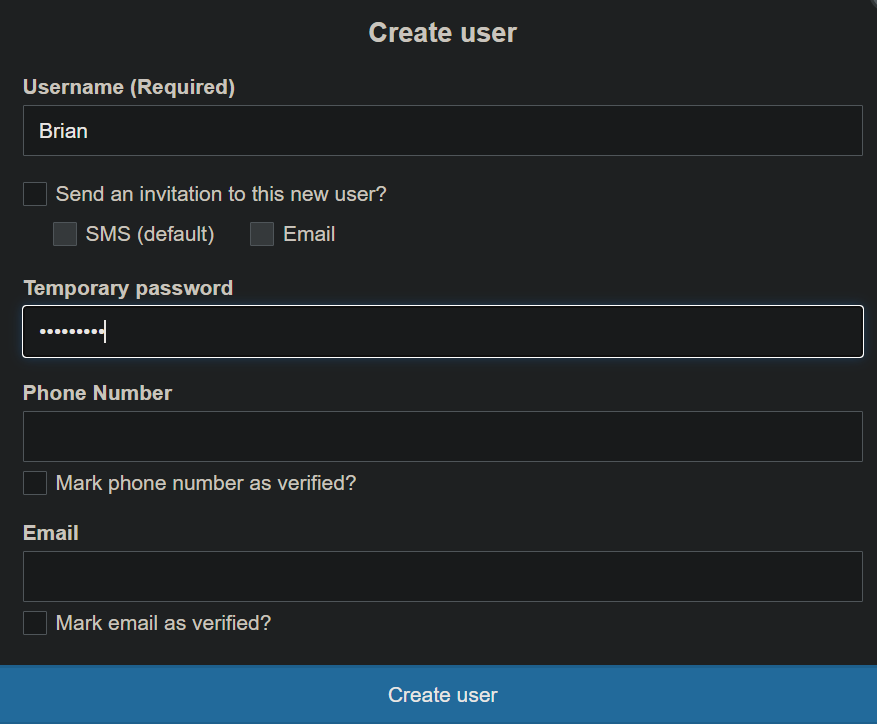


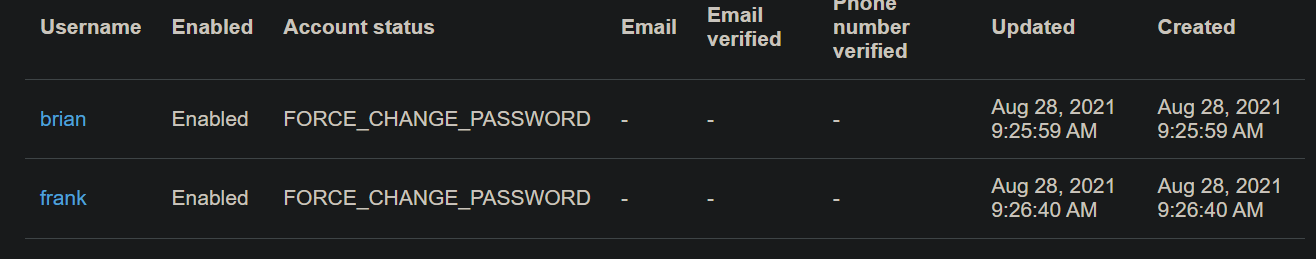
Its looking for a people.html file which is missing. Out lambda will fetch the user details from cognito user pool and send it to the people.html which will show us on browser.

1. Upload the people.html and JS folder from V10/Site directory to S3
2. Now if we click on Start a Conversation it will not throw error but no users will be listed as nothing present as now

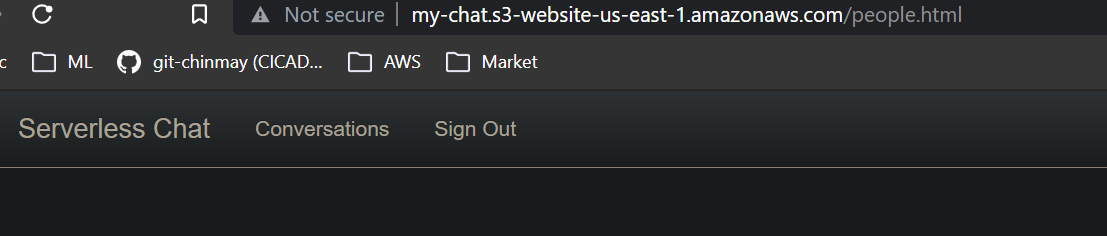


1. Now we will add some users to users pool
   1. AWS Cognito
   2. Manage Userpools
   3. Seletct our userpool
   4. Create user (Brain: Brian@123,, Frank: Frank@123)



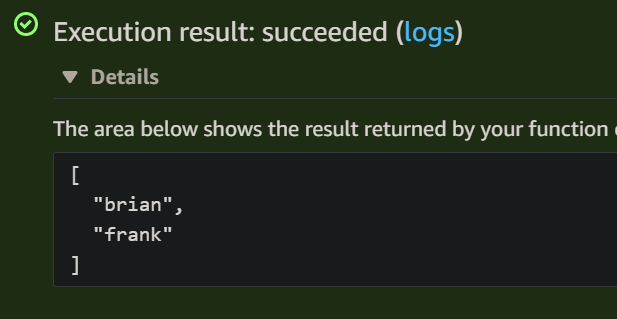


* 1. Now click on Chat app



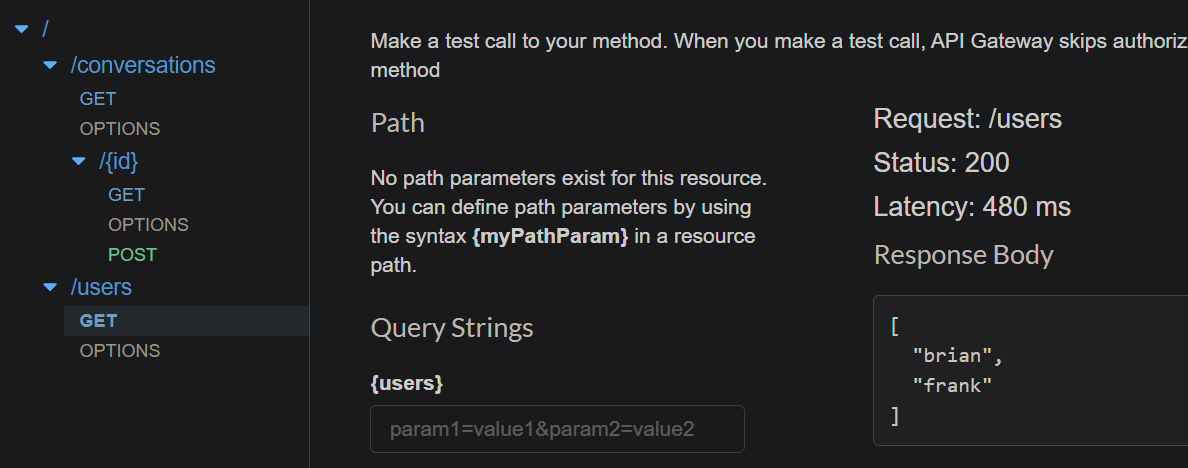
Still nothing coming. So we will debug. We will debug from backend to towards Front end. Our lambda calling Cognito. Cognito already have users so we will start looking into Lambda function.

Create a Test event without any input as we are not passing anything and see logs



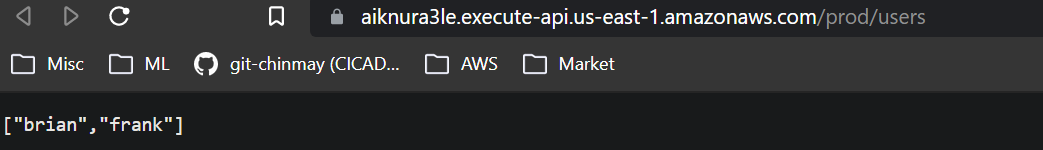
So lambda is fetching the user data from cognito perfectly.So issue is not with Lambda. Now go to next step towards front end which is API Gateway connected to Lambda.

Go to API gateway >> /users resource >> GET Method >> Test



API is working fine. SO the issue is not with the API. Lets try testing the API URL

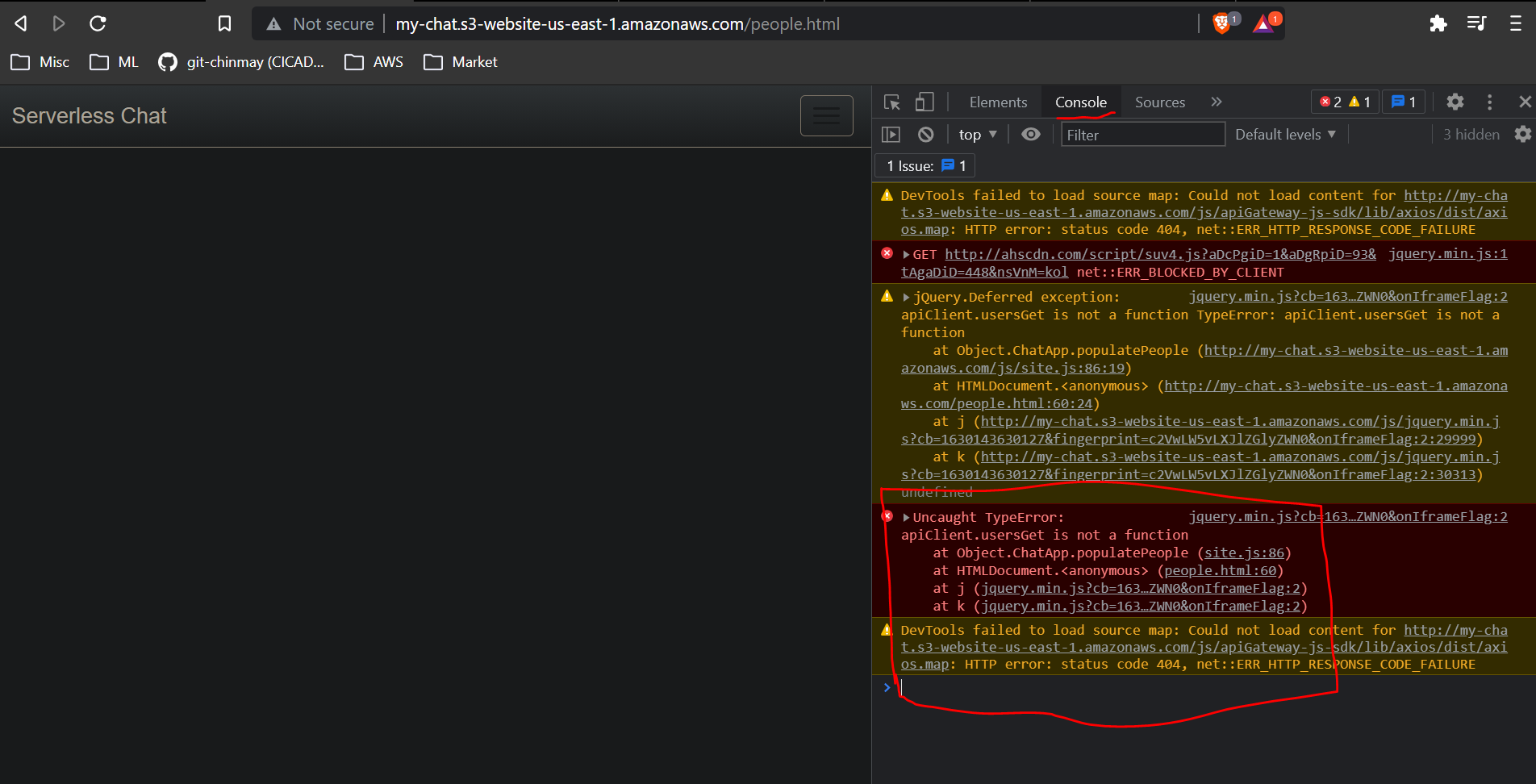
Go to Stage>> Prd>>GET >> Copy the Invoke URL >> Open it in browser



This one also working fine. Now lets check the people.html and Javascript code that loading the data.

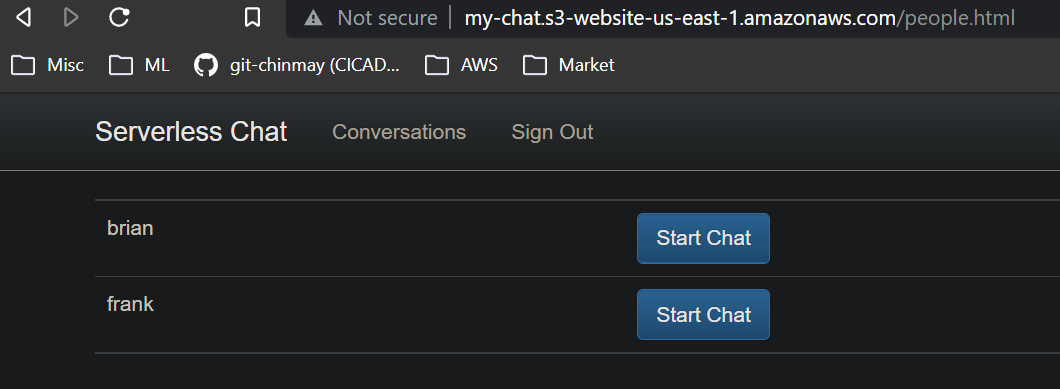
Open the chat app >> Page inspect >>

If you go to the console section we can see the the error apiclient.userGet is not a function type.



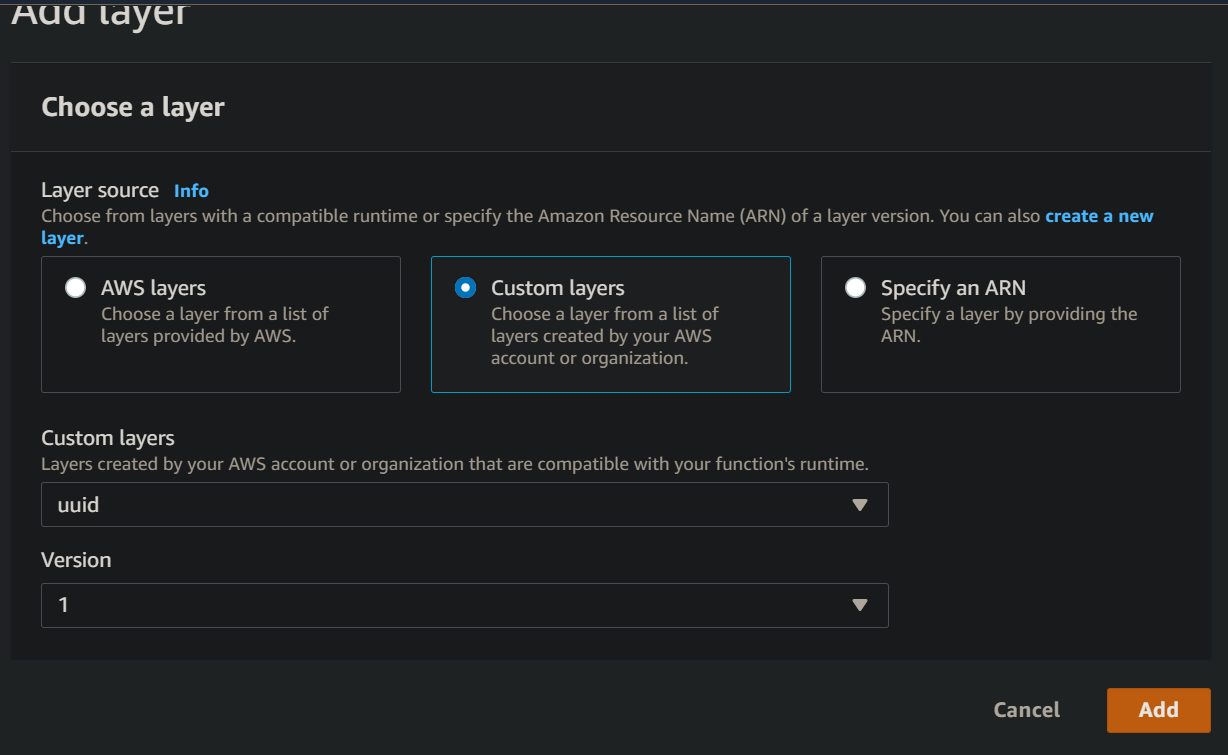
Ok this error is coming because after adding the Cognito lambda to API gateway we have not generated the new SDK ,so we have to generate a new one and upload it to S3 bucket.

Now reload the people.html and we should see the users list



Till Now we don’t have capability to start a new conversations. Lets do it

1. Create a new lambda Chat-Conversation-POST
2. Copy the lambda codes from V11/Lambda & upload it to above function
3. We are using uuid in our lambda and Node.js version 10 onwards provides a dedicated module to handle uuids. We can add this module as a Layer to our lambda. To do that
   1. Create a layer
   2. Uplod the layers zip fiole from resource. (We can also create it manually by doing npm install uuid and then putting entire module inside a zip file)
   3. Once zip file uploaded add runtimes node.js10 11..14 anything over 10
   4. Now go ot the lambda configuration and add the created layer



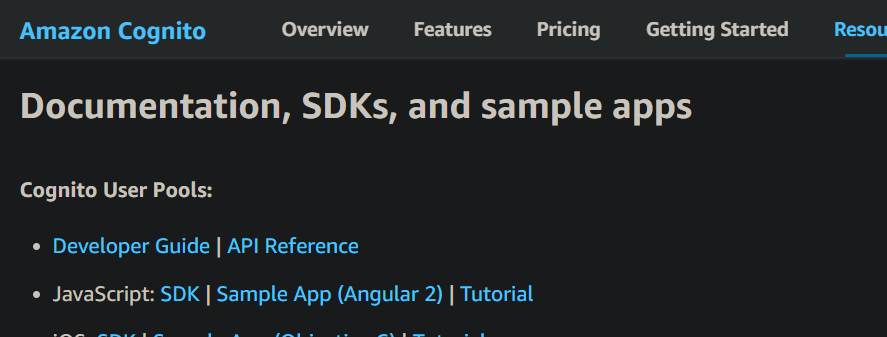
* 1. Now we will create two models at api gateway (Get model schema from V11)
     1. ConversationId
     2. NewConversation
  2. Now we will create a POST method under resource /conversations and linked the new lambda Chat-Conversation-POST to it
  3. Now we will add model NewConversation to the response body of the Method Request and model conversationId to Method Response.
  4. Add a mapping template at integration method of the POST. Got the template from V11
  5. Enable the CORS to the /conversations resource
  6. Deploy the API
  7. Generate the new SDK & upload it to S3 bucket
  8. Now we will upload a new site.js file to js folder from V11 to handle the new chat
  9. Now refresh the chat app page and start a conversation

We can see the uuid

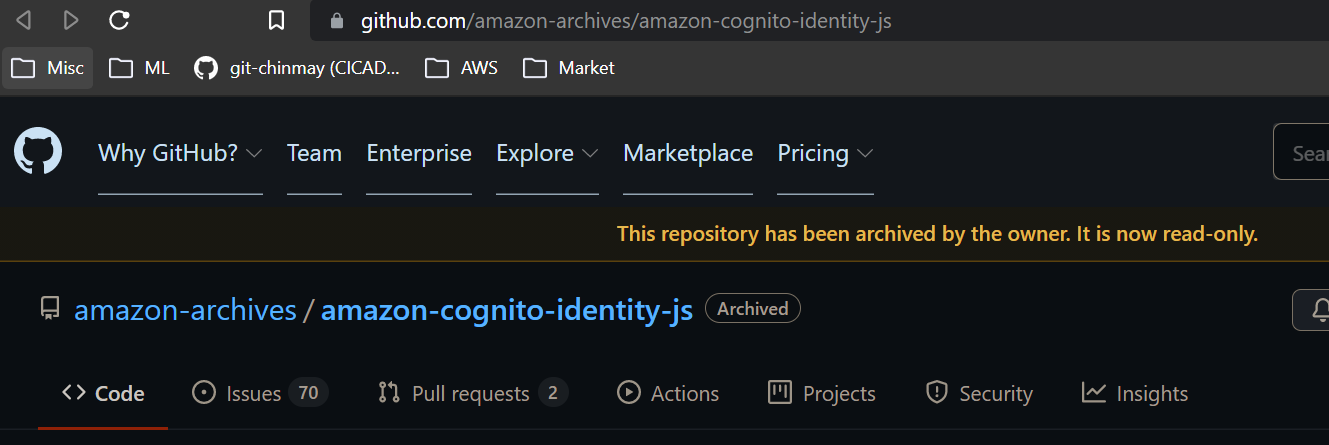


**Create a Sign Up Page:**

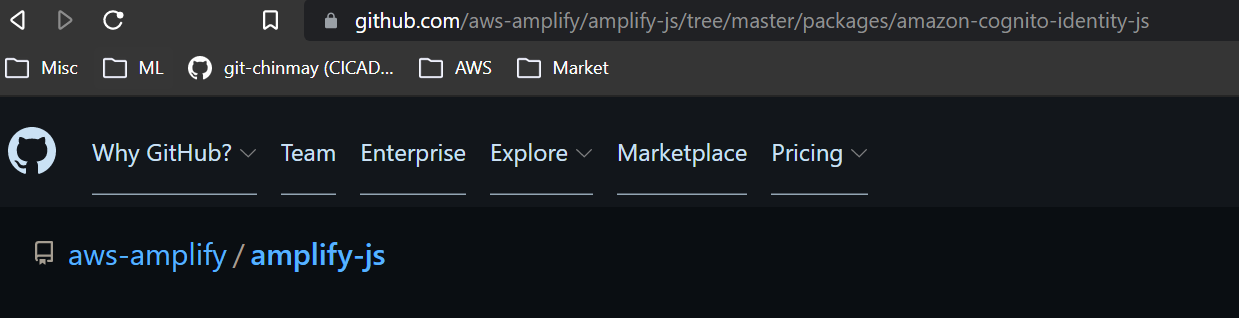
1. Open aws.amazon.com/cognito
2. Click on Resource section
3. Scroll down to Document, SDK sections



1. Click on SDK of the Javascript it will direct you to a GitHub repo

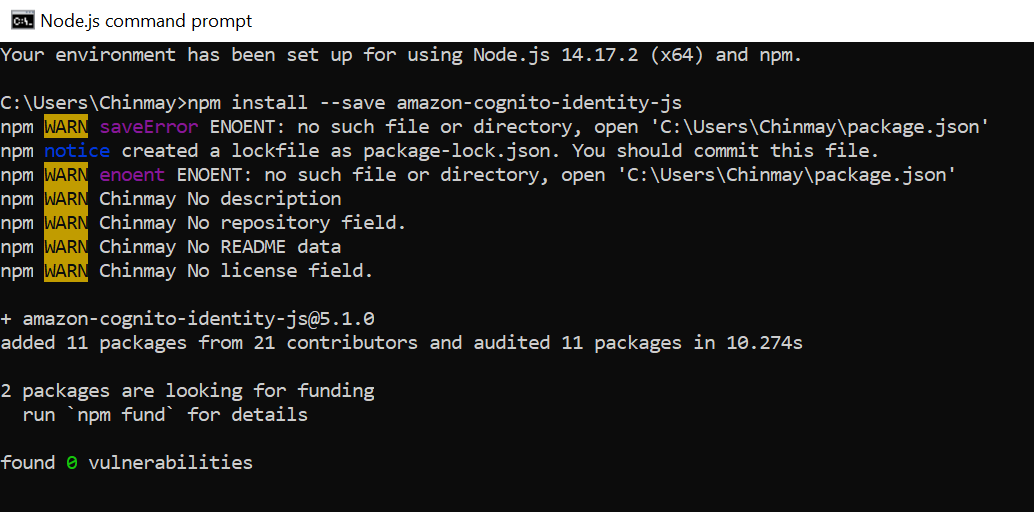


1. Looks like this repo is archived. Go to readme section of the repo. Its saying code moved to Amplify repo so click on that and it will direct you the correct repo

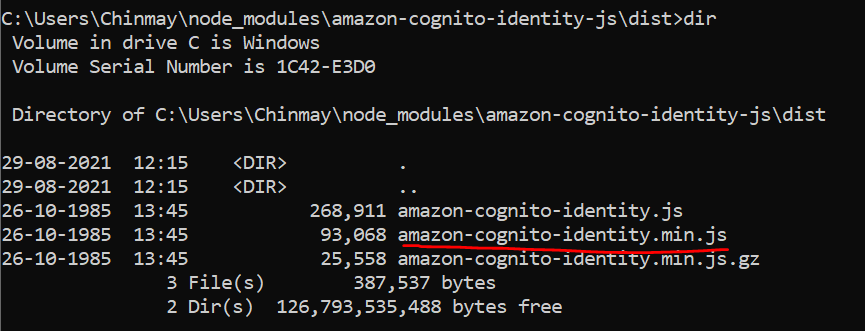


1. Go to Setup section of the Readme file
   1. Open Node.js command prompt
   2. Do the npm install as they mentioned

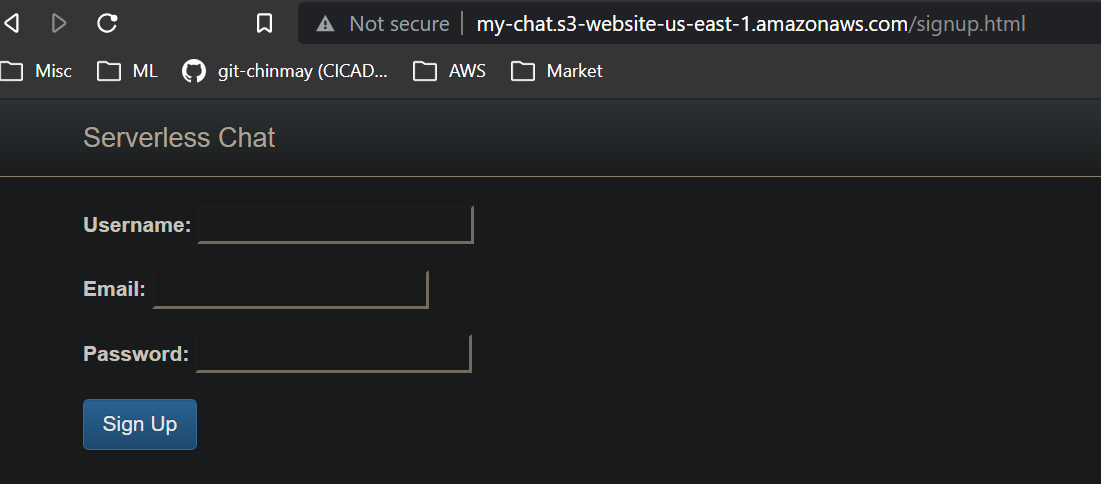
npm install --save amazon-cognito-identity-js



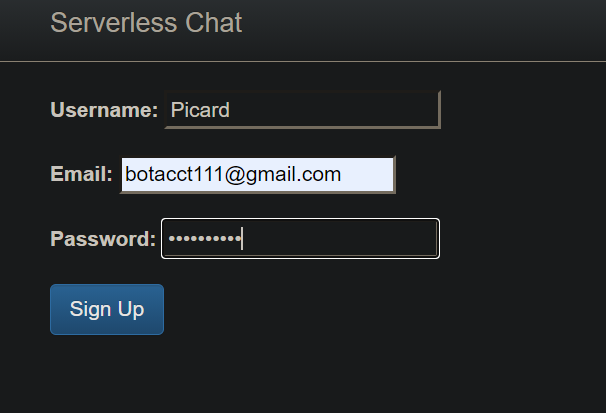
* 1. Go to node\_modules directory
  2. Then cd to amazon-cognito-identity-js
  3. Then cd to dist
  4. Unde dist there is a file amazon-cognito-identity-js-min , we want that file to upload



* 1. Upload the min.js file to JS folder on the S3 bucket
  2. Open Resource/V12/Site/Js/Config.js file and update the Pool id and Client id & save it
  3. Select everything under Site folder of v12 and upload it to S3 bucket
  4. Now test the code by adding signup.html to our chatapp

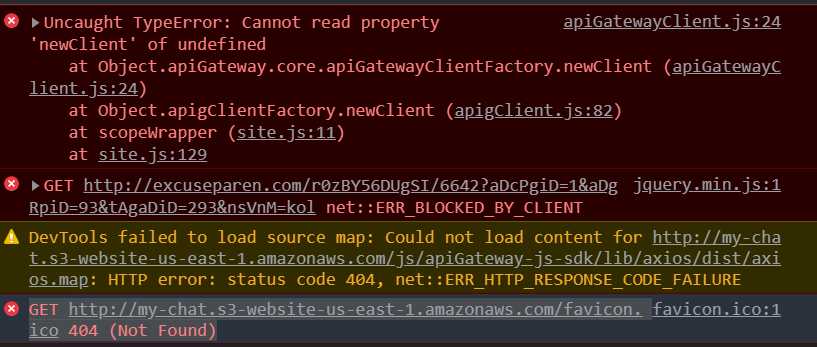


* 1. Lets create a new user. Click on Signup. Nothing happen means it worked bcz our callback function only returns if there any error



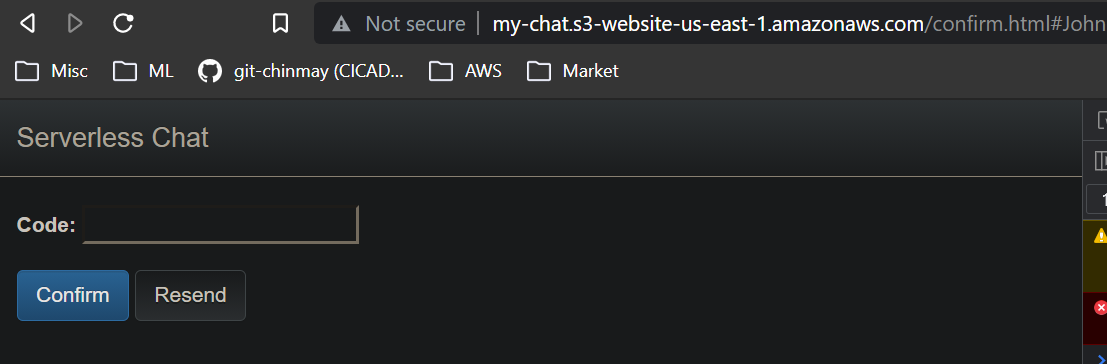
* 1. To verify actually a user got created we can check it in Cognito userpool console

New user is not reflecting in Cognito user pool. After checking the browser page source below error is coming

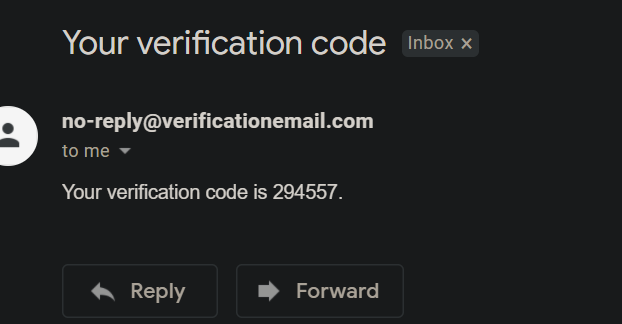


Create a Verification Page:

Add the new site codes from V13/Site . Uplaod the JS folder and confirm.html to S3 bucket.

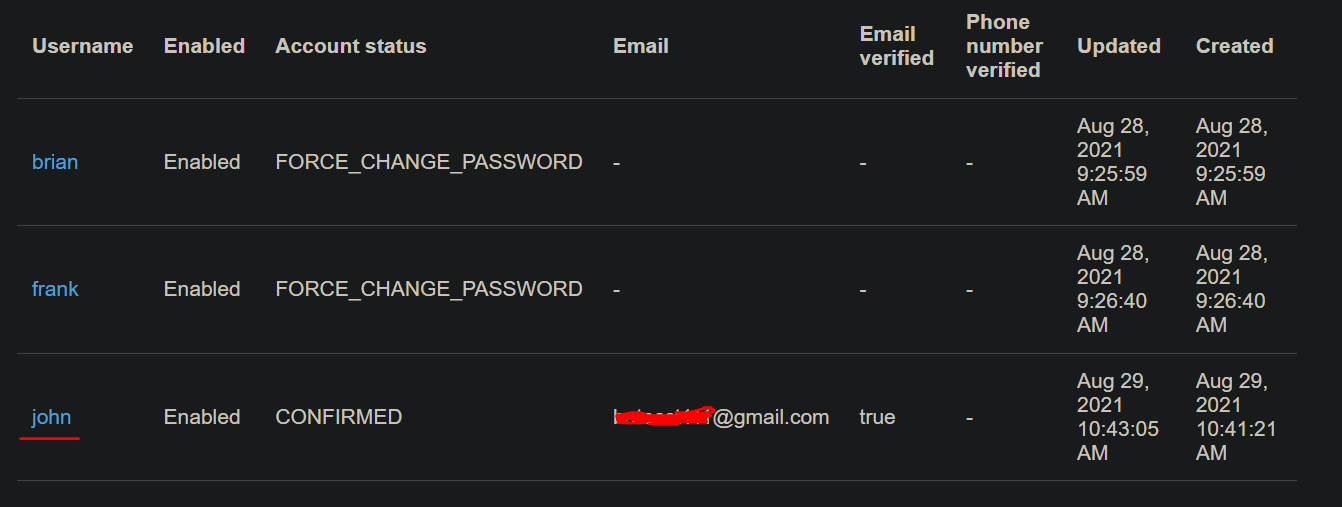


Verification Email Code



Once you confirm the code the page will be redirected to home page which will fail with 404 error which is expected as we have not set the home page yet. The point is its accepting the code and redirecting the site means our confirmation code is working.

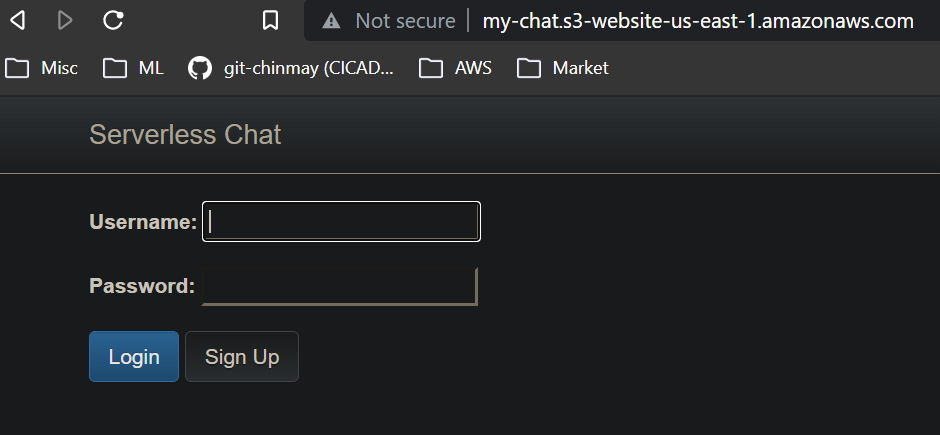
We can also confirm the user created in Cognito user pool



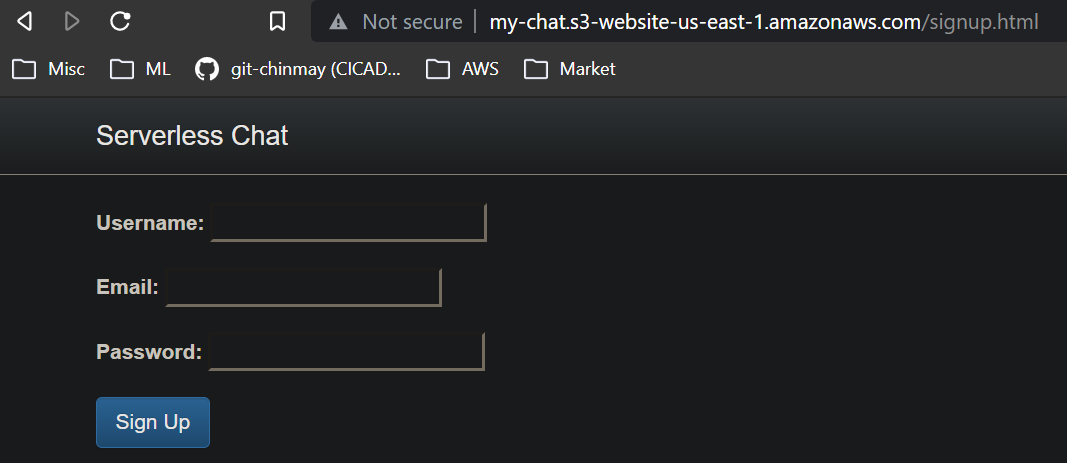
**Add Login and Logout**

Here we will add a home index page and will add the functionality for the app to check whether if an user is logged in or not. Upload the code content from Resource/V14/Site

Home Page:

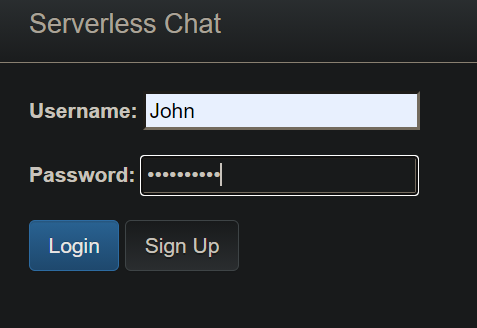


If you click on Signup it will direct to Sign up page

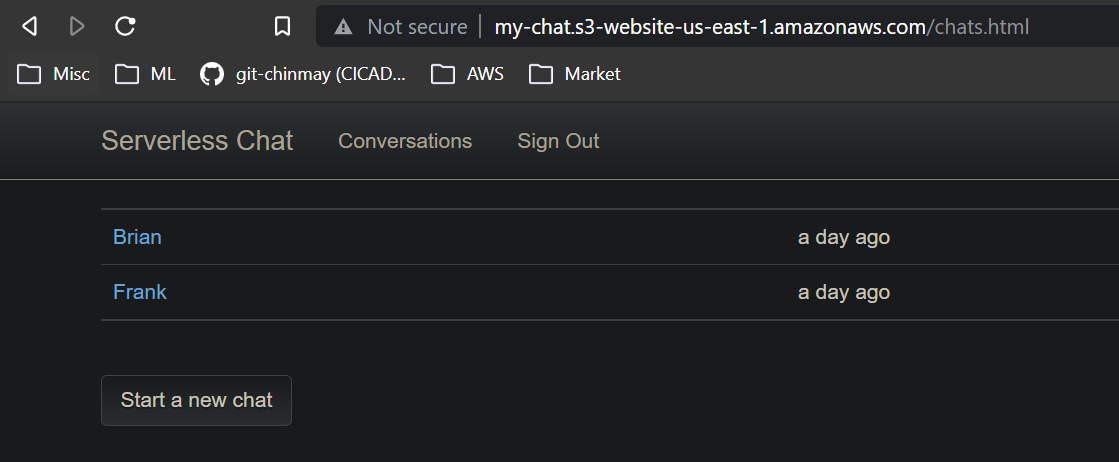


If we try to access the chats.html directly without login it should force direct to home page. It will no longer allow us to see the content.

Lets login with John credentials



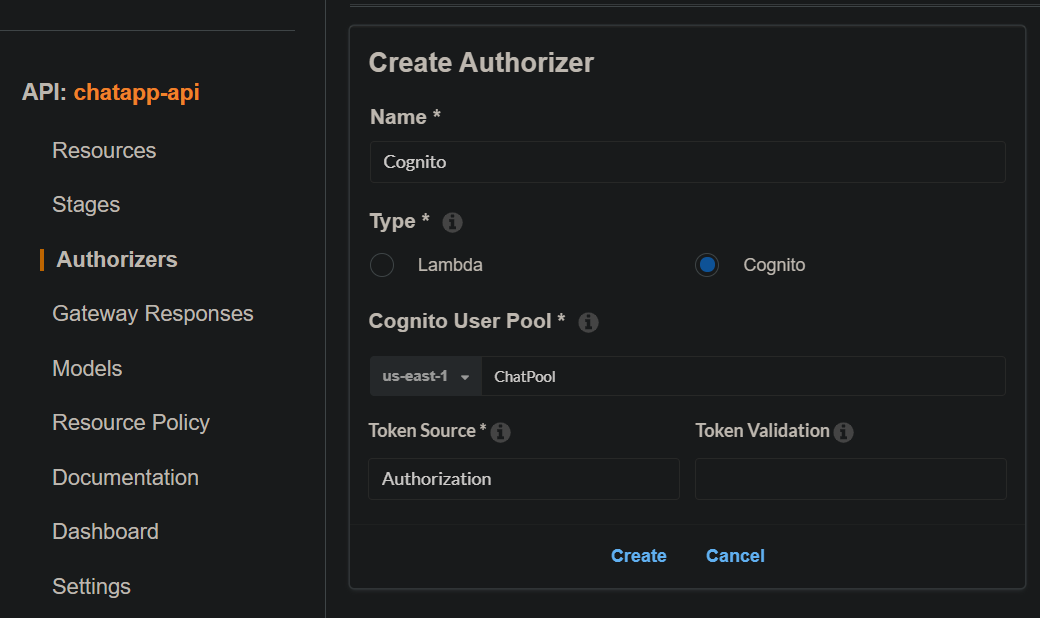
Now we are able to see the chats.html page



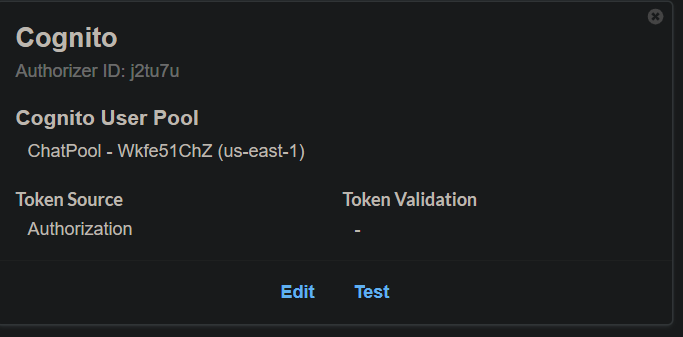
**Add a Cognito Authoriser to API Gateway**

Here John able to see everyones conversations which is wrong. We have to add some security so that John can only see what it means to see.

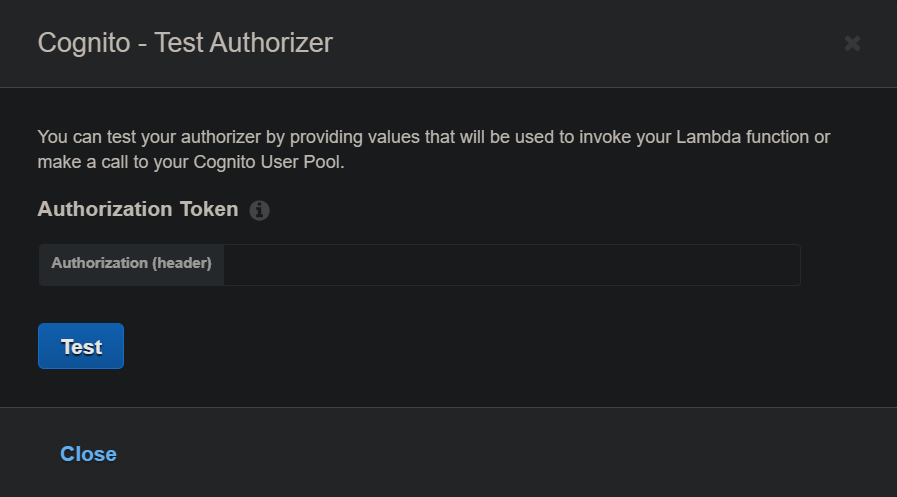
1. Create a API gateway Authorizer



1. Once you clink on Create



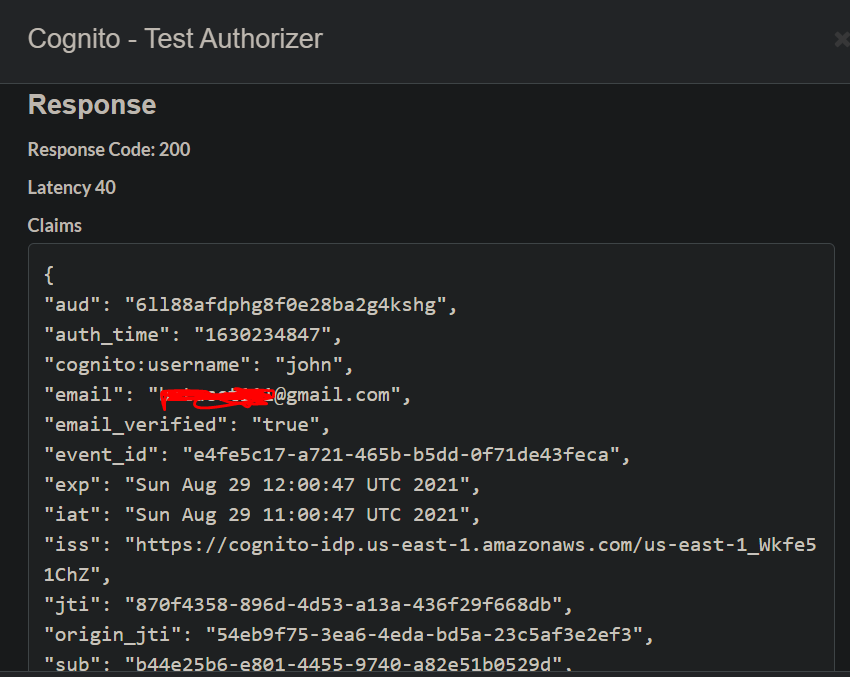
1. Click on Test



1. To get the token for testing we can use the code in V15
   1. Get the code from V15/Console.js
   2. Open the Page Inspect in browser where you logged in to chat app
   3. Paste the code in Console and enter
   4. Copy the token & use it for Authoriser testing



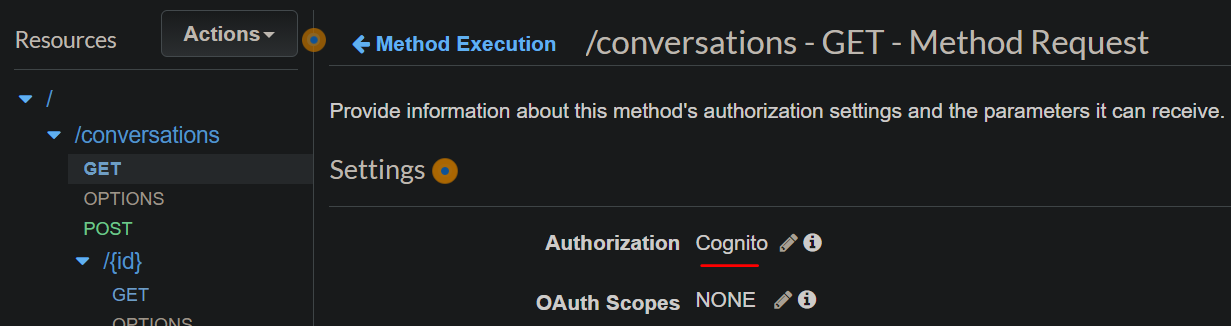
* 1. Test should be passed



Now we have to do two things for all the API resource’s Method

1. Add the newly created Authorizer
2. Replace the hardcoded ‘Student’ from Mapping Templates with Cognito.user

/conversations GET



Integration Request Method Mapping Function



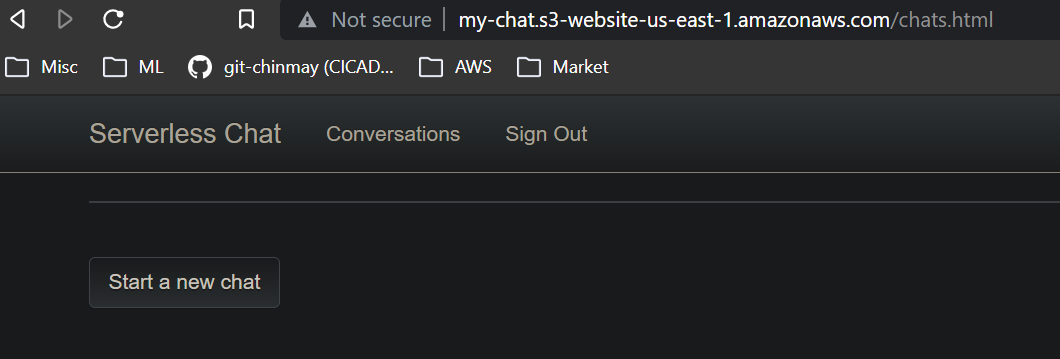
Repeat this process for all the Resources’s methods. For /user resource there will not be any mapping available hence we have to create it with given template. Get all the template files .vtl from Resource/V15/Mappings sections.

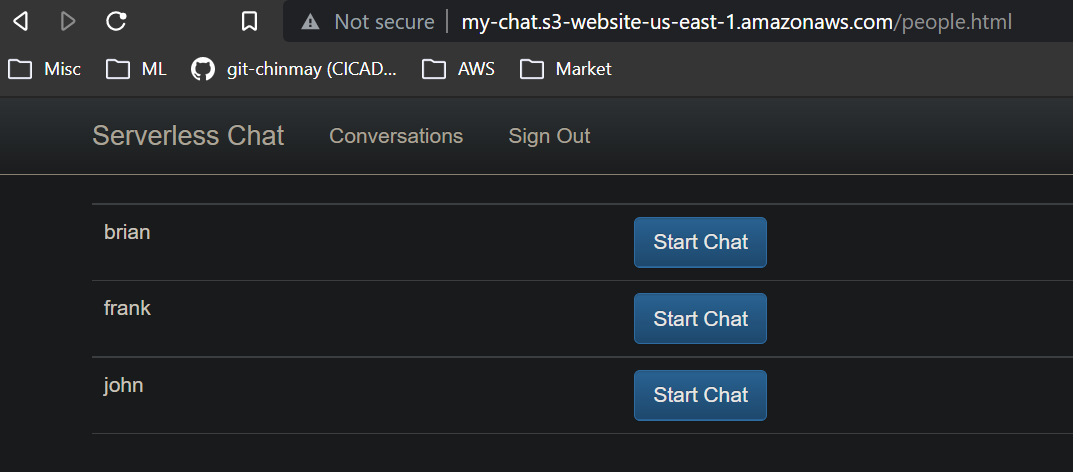
Now deploy the API

At this point we have Application sign in done and we also have setup the API authoriser now we have to tie both. For token we ran the code in console and take the token and test the authoriser. We have to do this programmatically by adding a callback function.

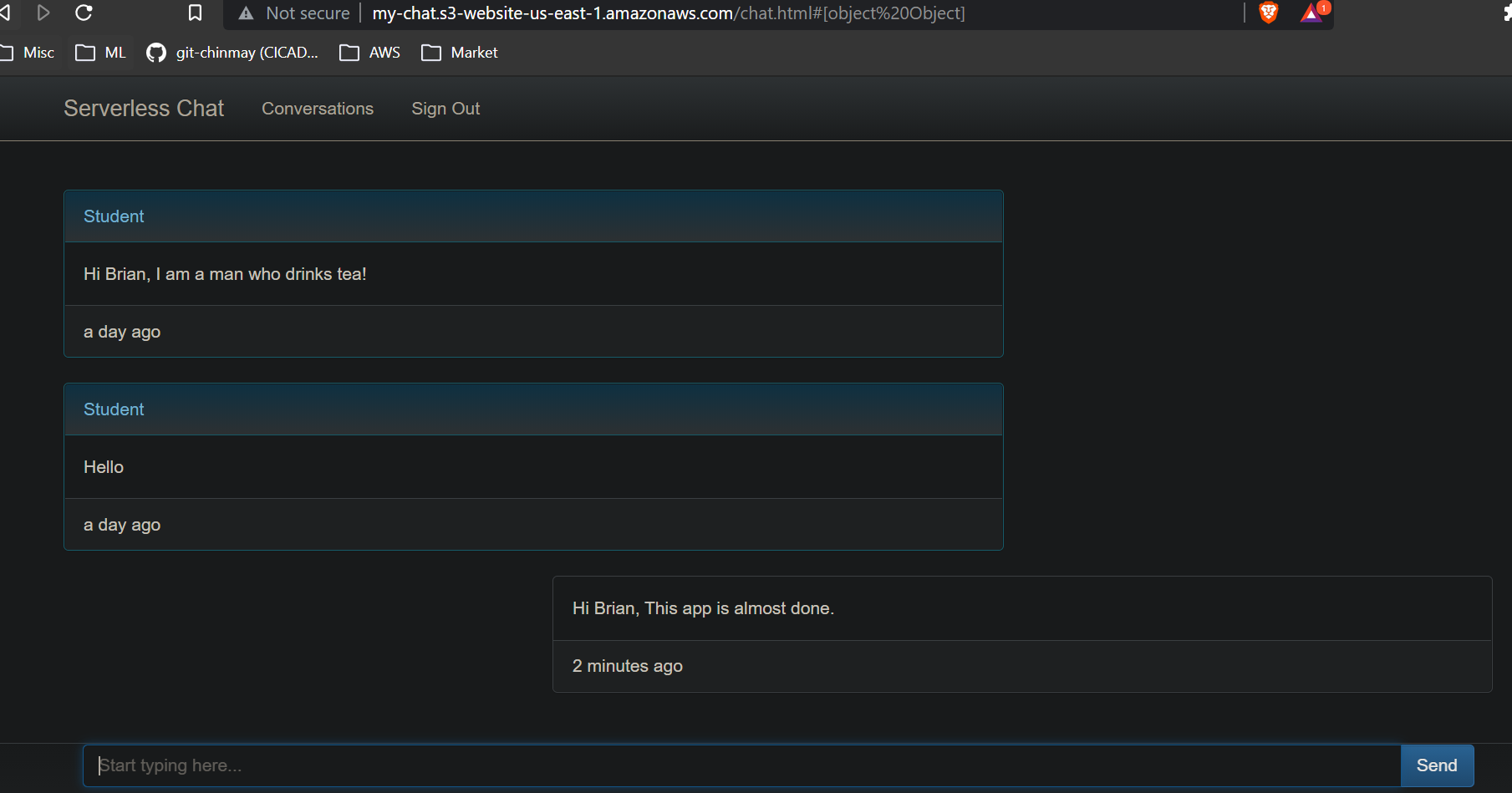
Upload the updated site.js file to S3 bucket/JS folder from V15/Site section

Now if we click on Start a New Chat we should see the list of users to start conversations





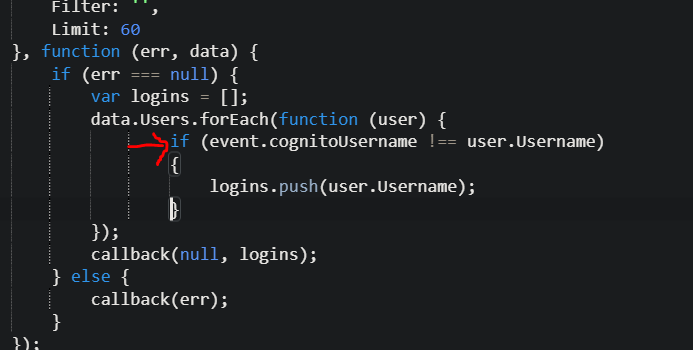
Lets start a chat with Brian



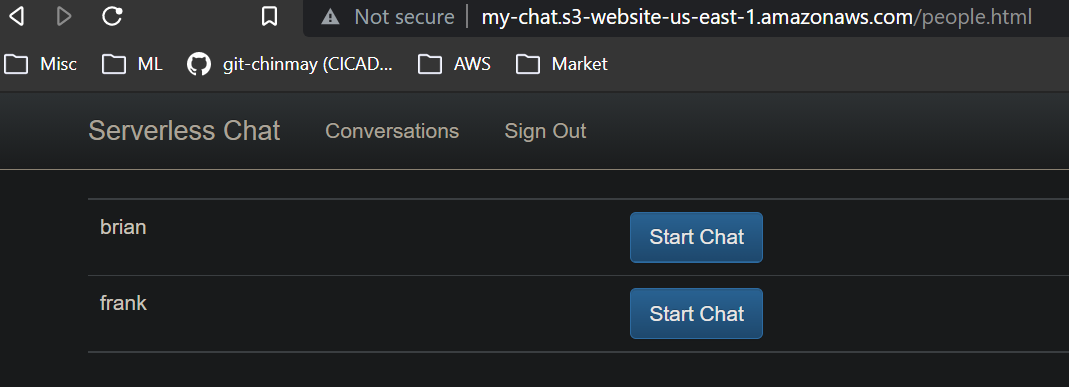
We need some refinement.

We cant chat with ourself so we have to remove the our name when start a new chart

1. Go to the Chat-Users-GET lambda and add a condition to the code

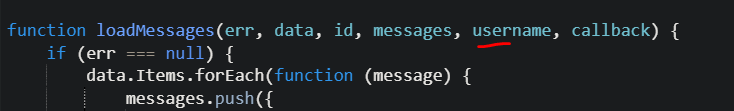


1. Now deploy the code, refresh the app page and now we should not see the John name in chat list as we logged in as John



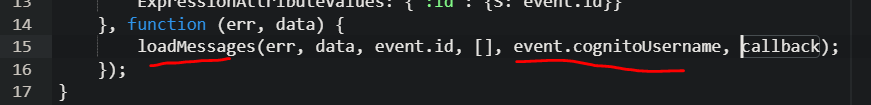
**Prevent the people not to involve in the chat which they are not part by hacking the url**

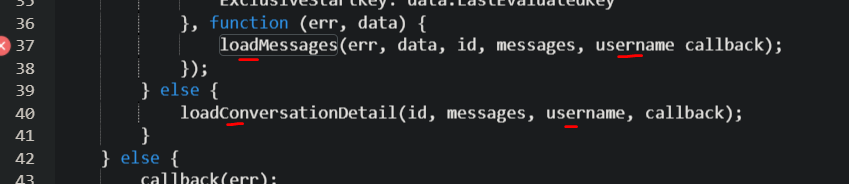
1. Open Chat-Messages-GET lambda function
2. Add a new function parameter username to functions loadmessages and loadconversationdetails



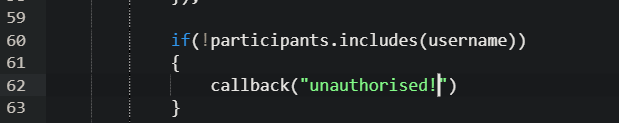


1. Pass the same argument wherever you are calling these two functions





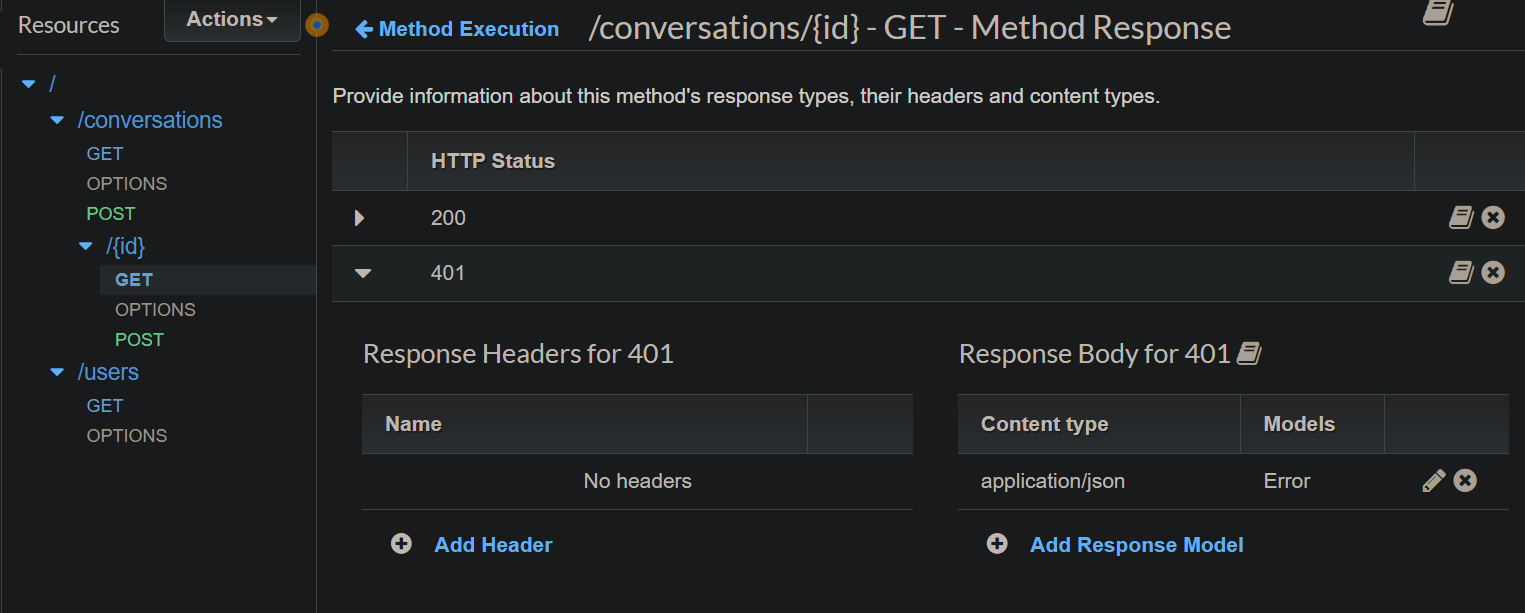
Will add this condition if the participant is not an user then callback error



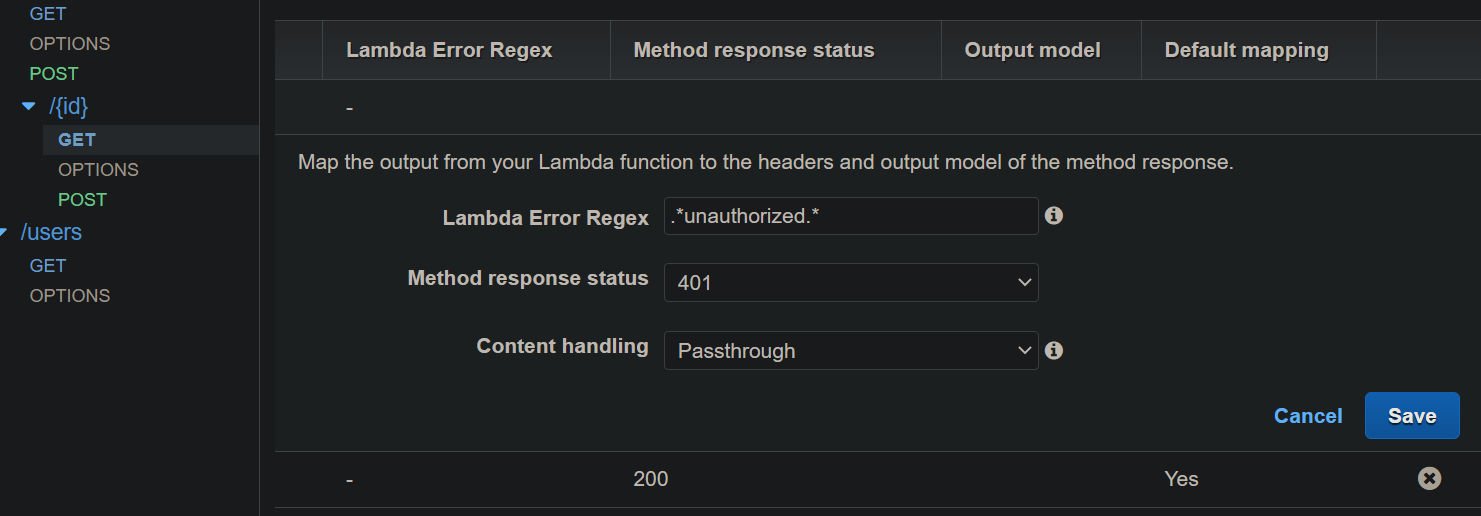
If we directly go to chats.html now we will not see anything



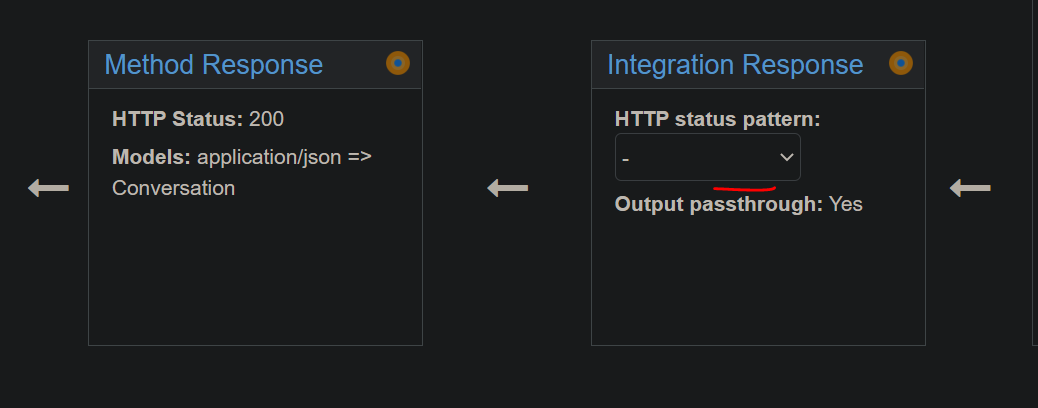
We should get a 401 error, so go to the API Gateway add a 401 Resoponse Method



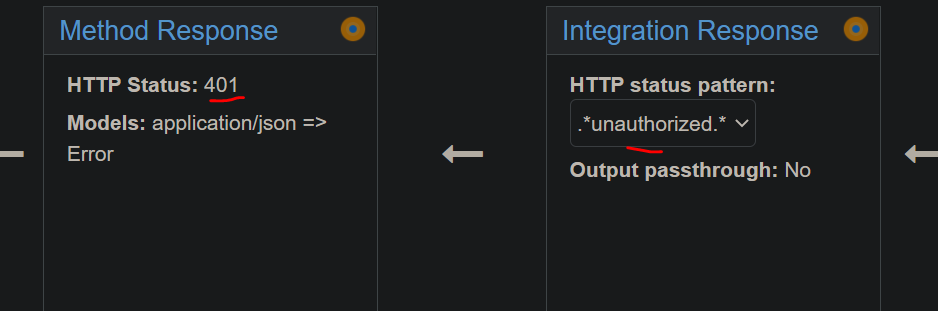
Then go to the Integration response add the response



Now you will see a drop down at Integration response and based on its selection the Response method will change



Changing the HTTP status pattern >>



Deploy the API

Now refresh the page and we will see the 401 error in console

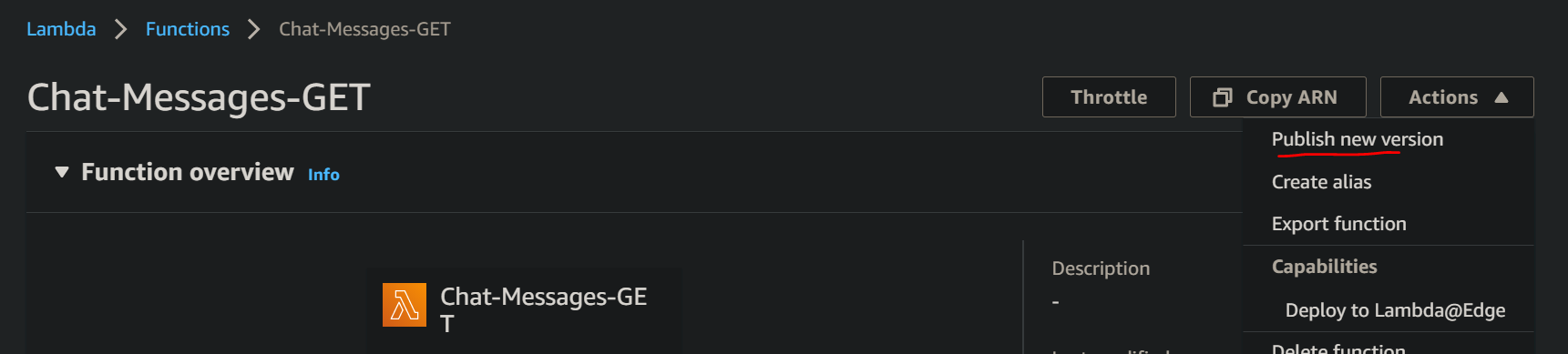


And here with it we have completed a Serverless chat application.

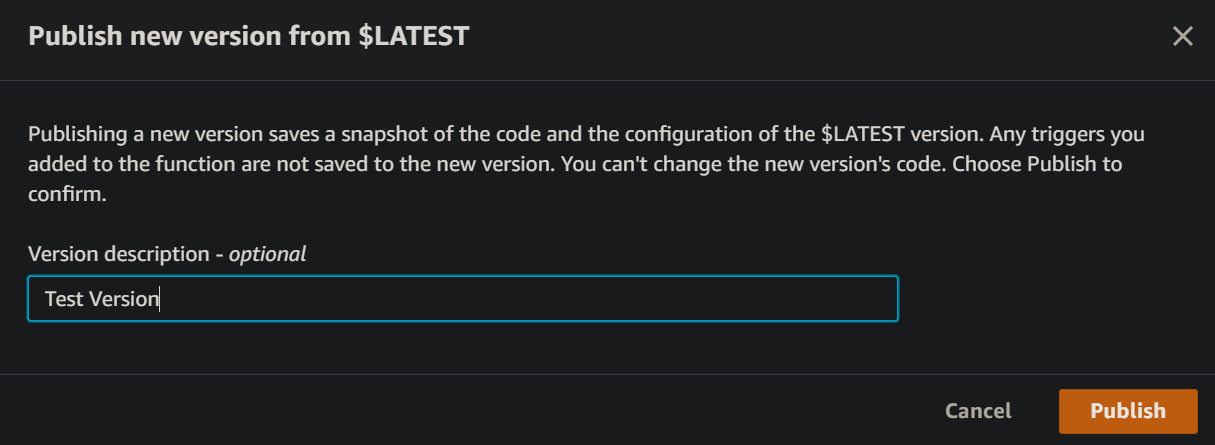
**Optimization & Preparation for Production:**

**Lambda Versioning:**

1. When you create a Lambda it created with a default version called $LATEST
2. Lets create a new version for lambda Chat-Messages-GET
   1. Go to Publish a new version



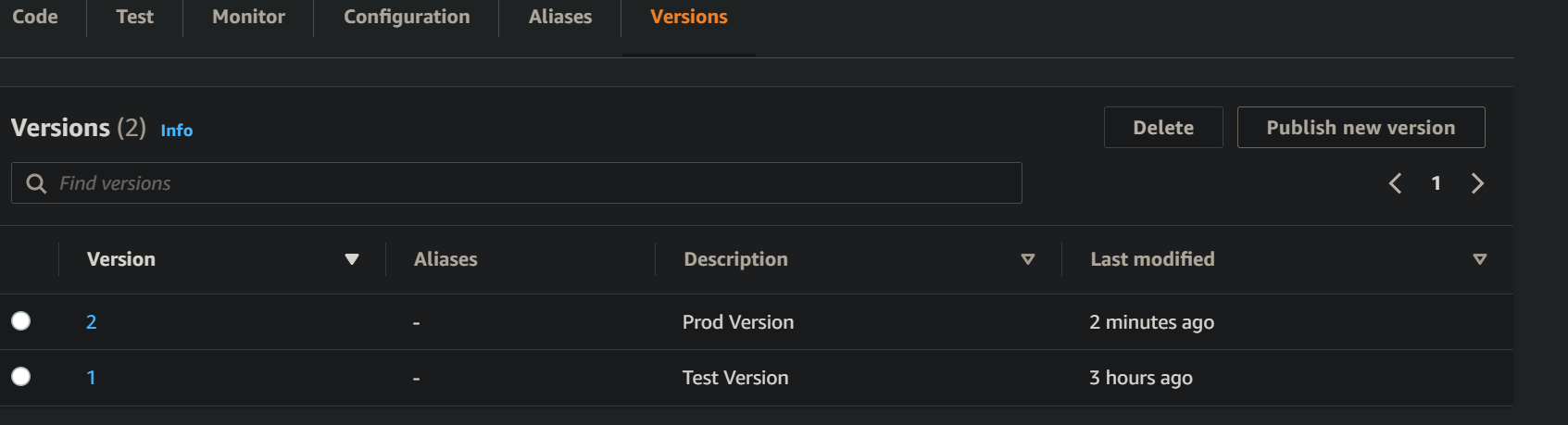
* 1. Give a description. AWS will take version no automatically



* 1. A new version(Non editable) lambda created

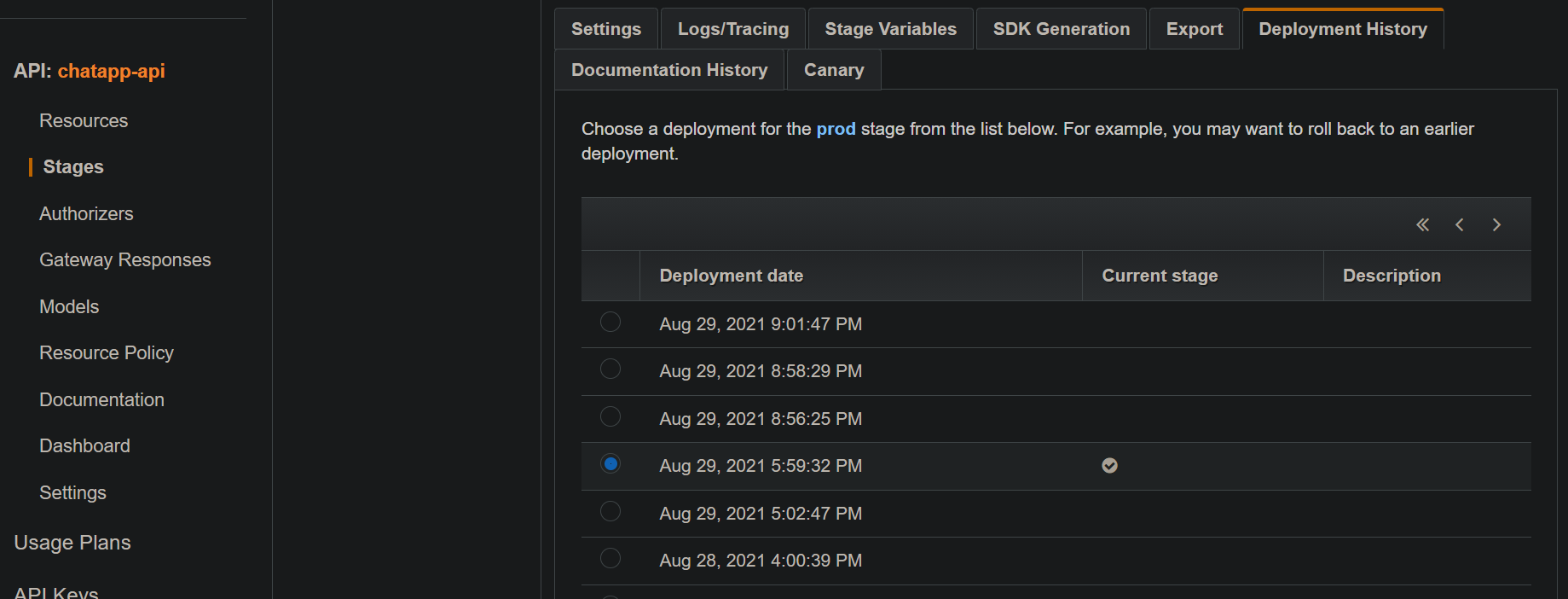


* 1. We can create as amny versions as we wantand and can be seen at version tab



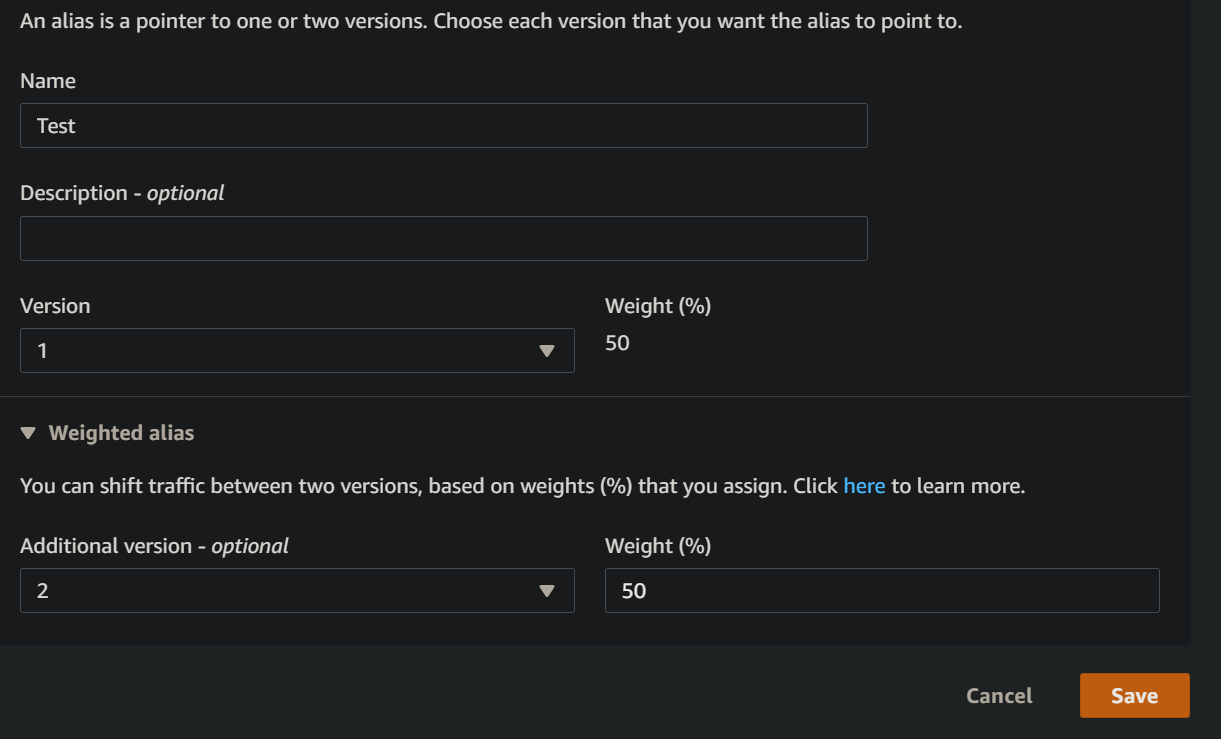
* 1. We also need to add the version no to API gateway to use it otherwise it will always take $LATEST one

We can deploy the changes back and forth from deployment history



By enabling Canary we can send traffic 50-50 between two traffic

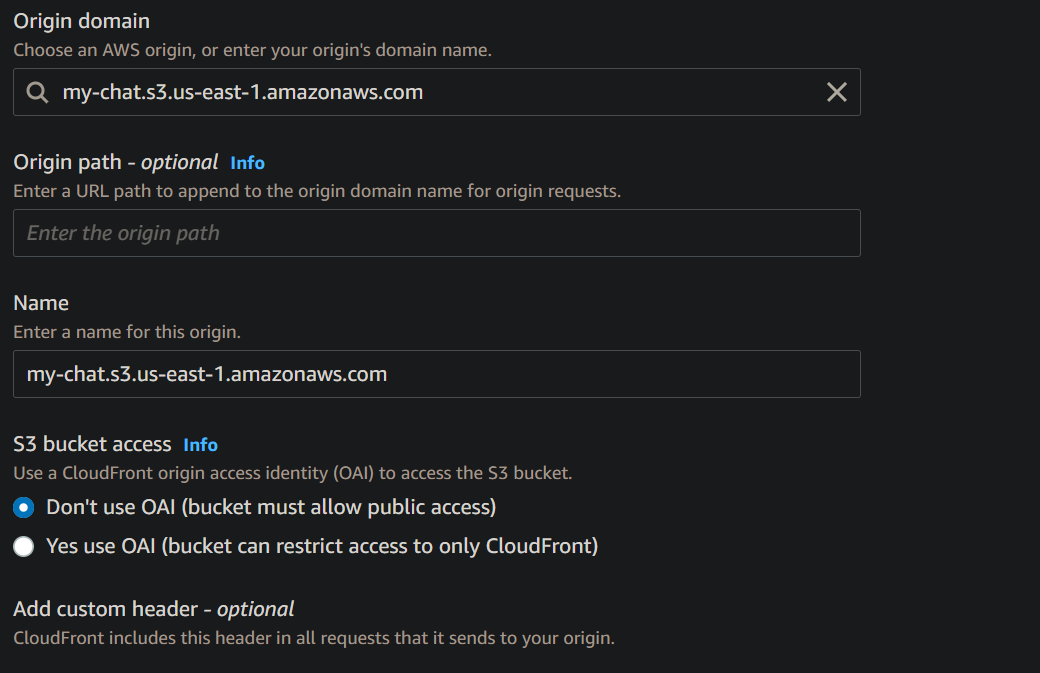
We can also do the same by creating two different aliases of the Lambda and adding a weightage to each alias.

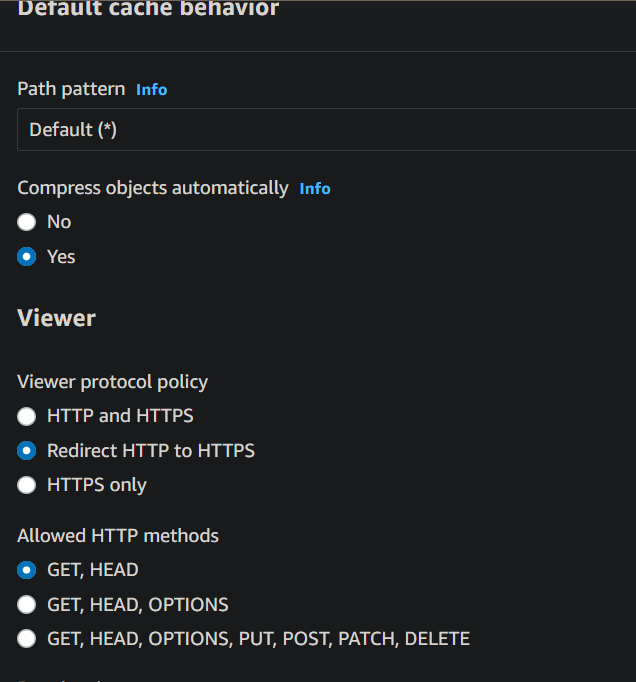


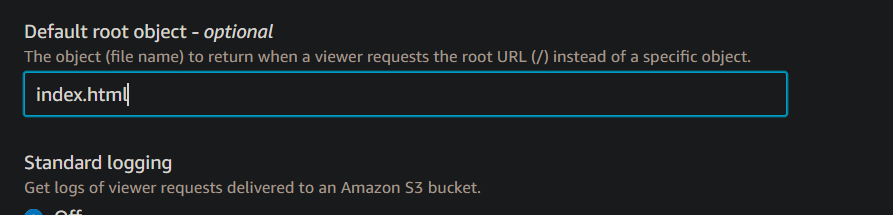
**Canary need two stages while Alias needs two versions of the Lambda.**

**Setting up the CloudFront:-**

1. Go to AWS CloudFront
2. Create a CF distribution

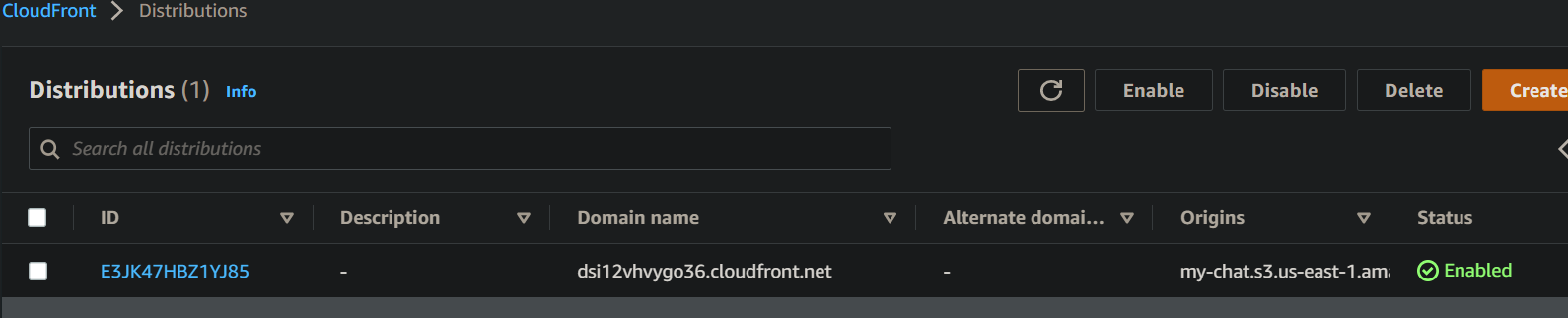




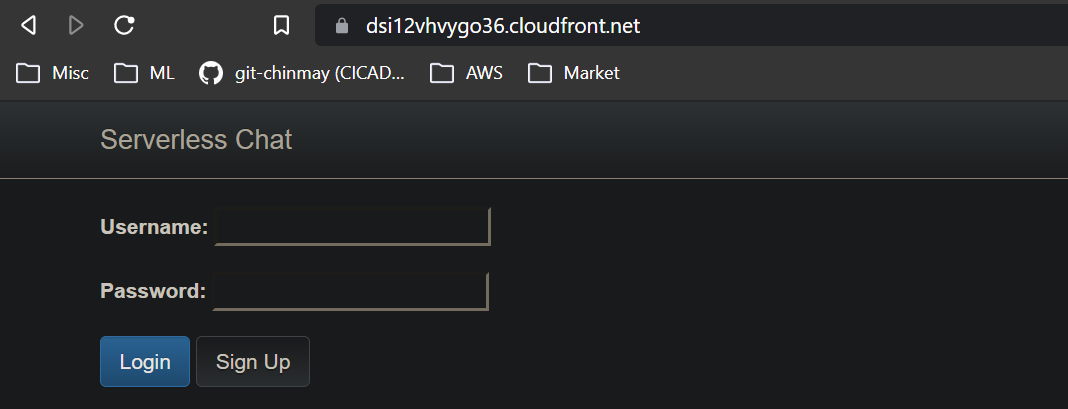


Keep everything else as it is.

Create the distribution.

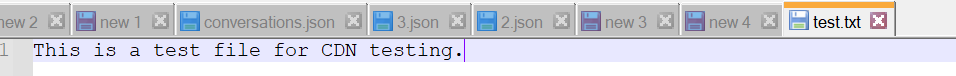


If we open the domain name <https://dsi12vhvygo36.cloudfront.net/>

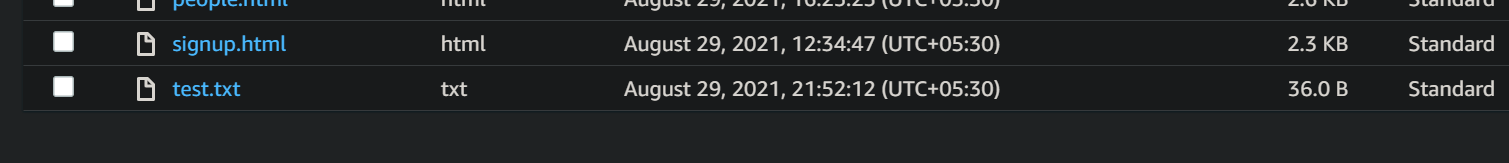


Lets test actually CDN is working or not

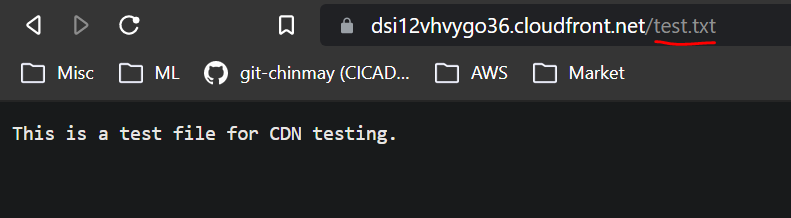
1. Create a test file and upload it to our S3 bucket



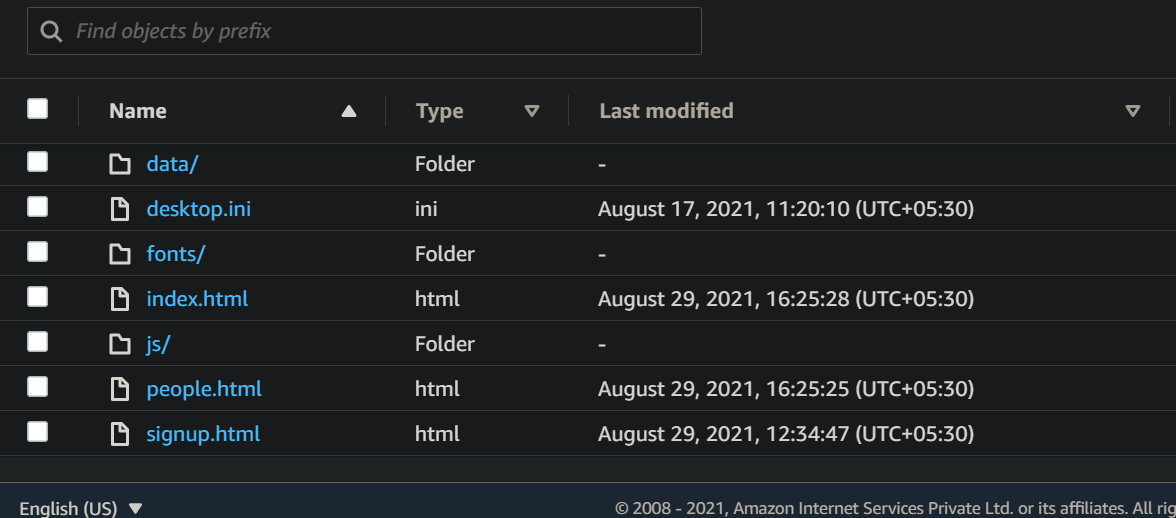
1. File uploaded to S3



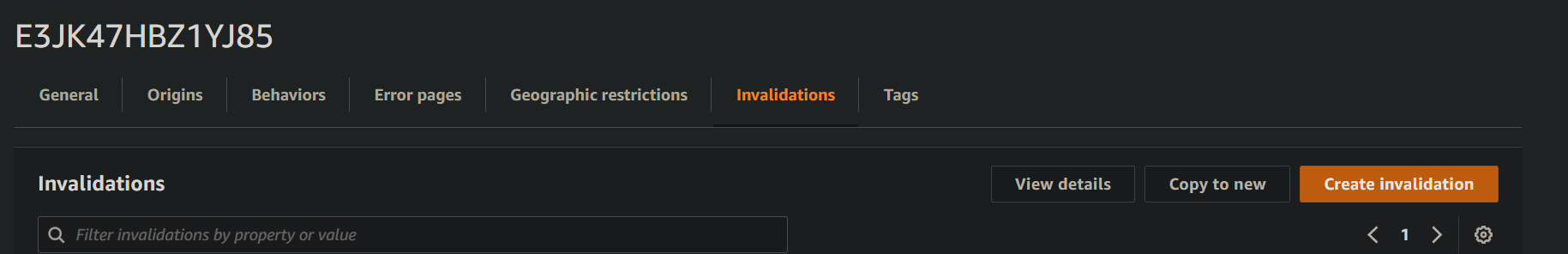
1. Lets access this file in CDN.



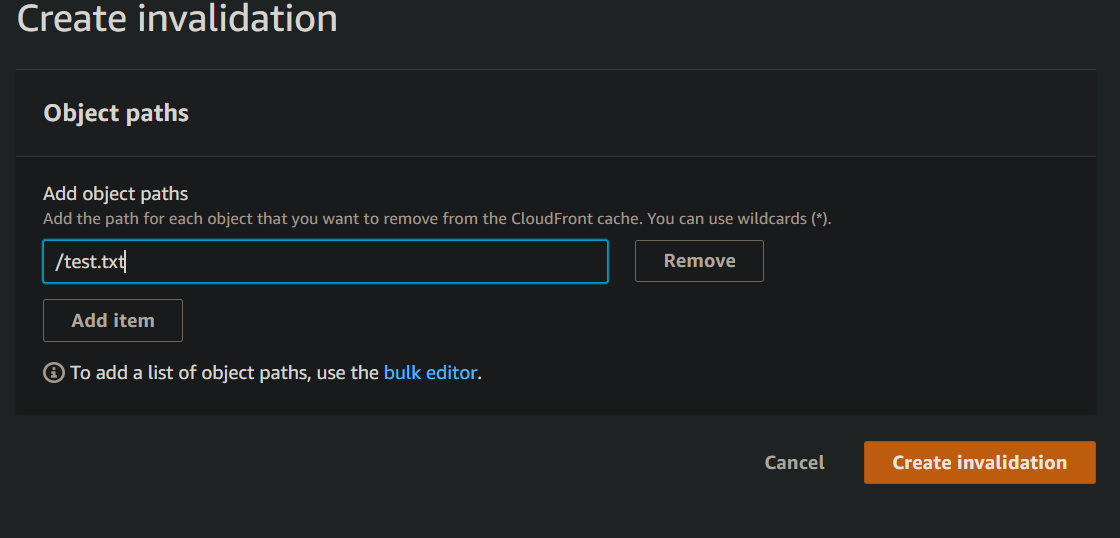
1. Now delete the file from S3 bucket

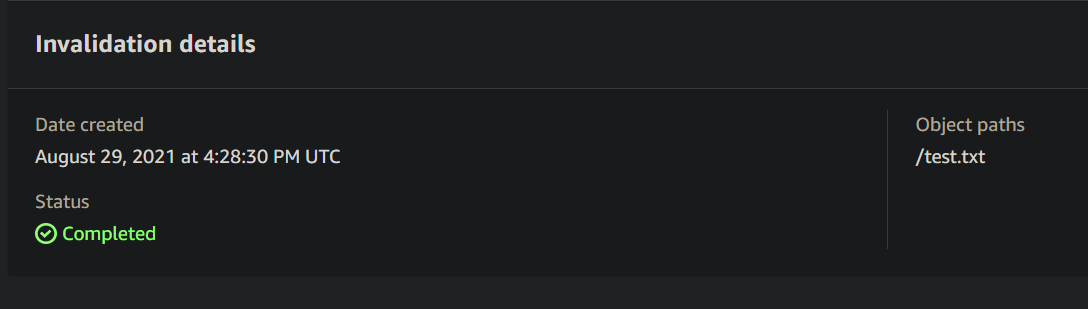


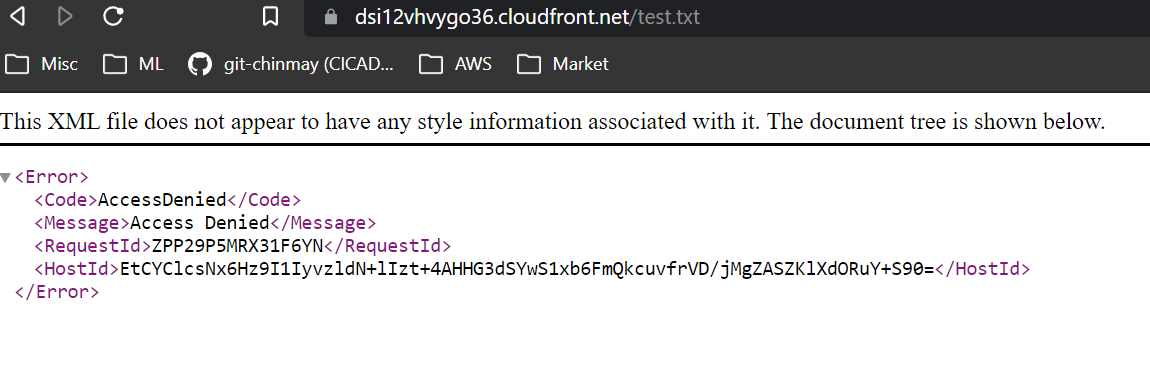
1. Now again access the file from CDN and it will be there. It will only remove if we add it to the CloudFront invalidation list.
2. Go to cloudfront invalidation



1. Create invalidation

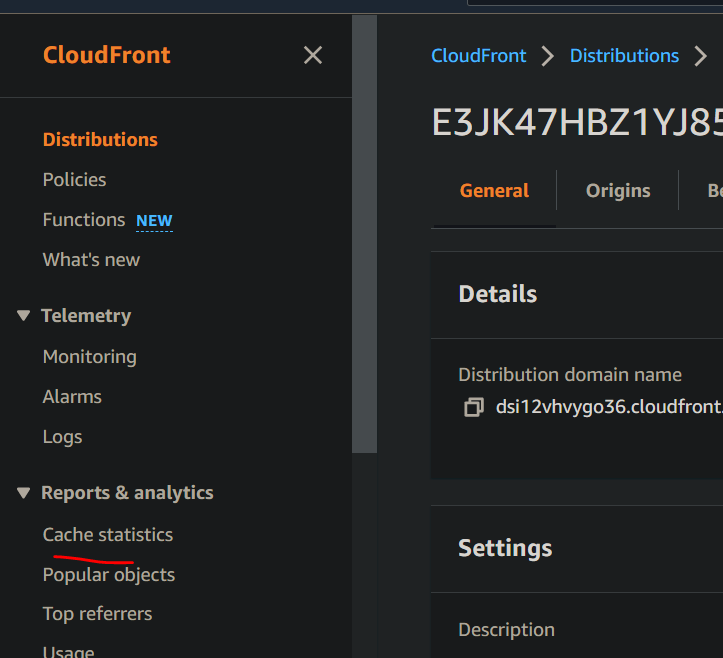


1. It will take some time to get created.
2. 
3. Once created now try to access to test page at CDN

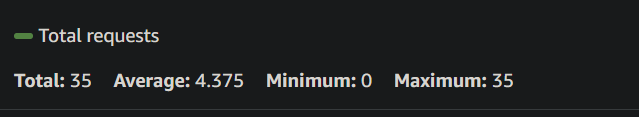


**Monitoring :**

We can see the analysis from CloudFront Report & Analysis



Total 35 requests till now to our CDN cacahe



We can see the Popular object people are accessing in our app from CloudFront Popular object



100% requests are coming from desktop

