**COMP 4968 & COMP 4964 PROJECT**

**Louis Ng A01047428**

**Gar(John) Lee A00911868**

**List of services used**

* Cognito
* Lambda
* Simple Email Service
* API Gateway
* DynamoDB
* S3 Bucket

**Description**

We are going to make a serverless web app for booking appointments. The web app requires the authorization of users. The user pool is handled by Cognito. There will be two dropdowns in the web app. One dropdown is for Book, which has all the free time slots which are available for booking.

When a user books one of the timeslots, the web app calls the Lambda function through the API Gateway, which stores the data into the DynamoDB. The user will then receive an email confirmation for the time slot that they have booked.

The second dropdown on the web app is for Waitlist. Time slots that are booked by someone show up in the Waitlist drop down. When a user waitlists for a time slot, the DynamoDB table is updated with the information and an email confirmation is sent to the user. When someone frees up that timeslot that a user is waitlisted for (changes booking), an email notification is sent to the users that are waitlisted notifying them that their waitlisted time slot is now available for booking.

**Notes:**

We will be doing this project in the Oregon us-west-2 region.

Use gmail accounts for this project.

**Instructions**

**Create a S3 Bucket to host our web application**

1. Create a new s3 bucket. Under bucket settings, deselect Block all public access, and check I acknowledge that the current settings might result in this bucket and the objects within becoming public checkbox.

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

1. Click Create bucket.
2. Open the S3 bucket, go to the permissions tab, select bucket policy.
3. Copy and paste this code, replace Resource with the arn of the bucket

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

"Resource": "<YOUR BUCKET ARN>/\*"

}

]

}

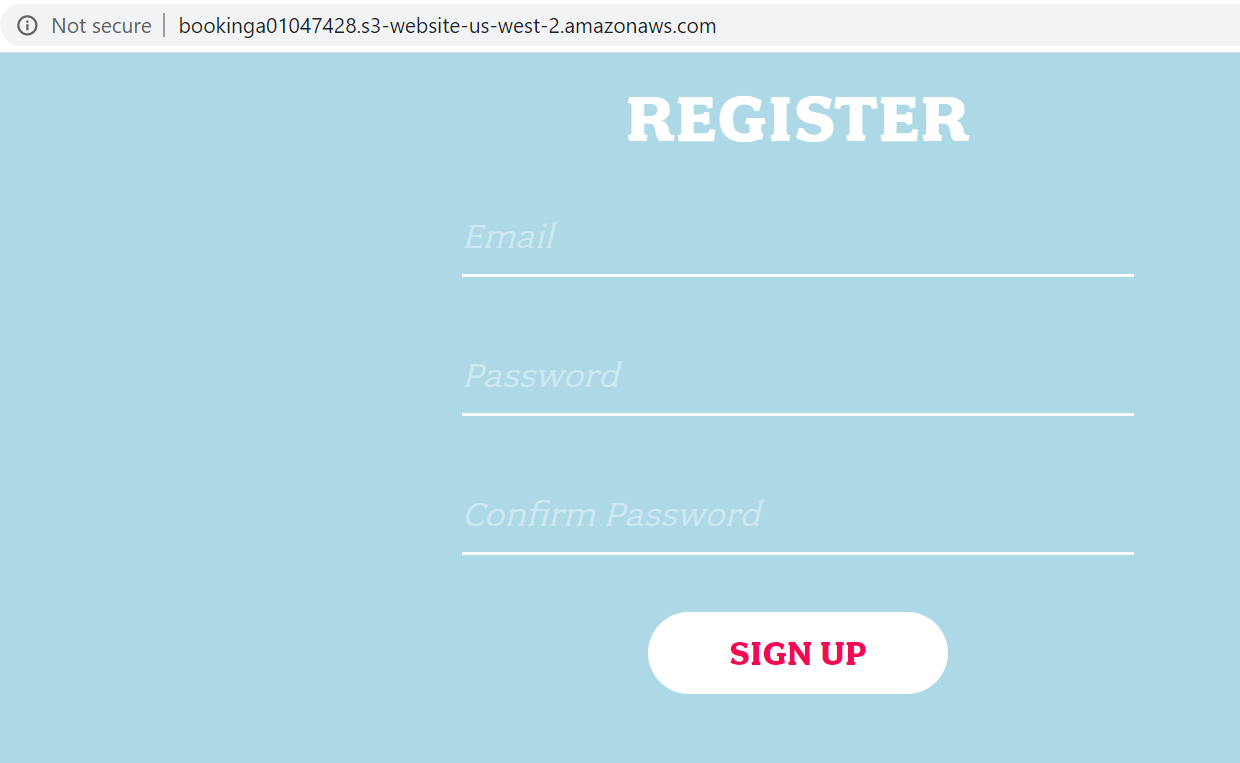
A screenshot of a social media post

Description automatically generated

1. Click save. This policy allows anonymous users read access to your bucket.
2. Go back to the overview tab. From the web application folder provided, select all the files and upload them onto the S3 bucketA screenshot of a cell phone

   Description automatically generated
3. Upload
4. On the S3 bucket, go to the properties tab
5. Click Static Web Hosting
6. Select Use this bucket to host a website
7. Under the index document section enter register.htmlA screenshot of a cell phone

   Description automatically generated
8. Click Save.
9. Click on the Endpoint. It should look something like this. If you don’t then double check the bucket policy.



**Create an Amazon Cognito Pool for our users**

1. Search for the Cognito service
2. Click Manage user pools

A screenshot of a cell phone

Description automatically generated

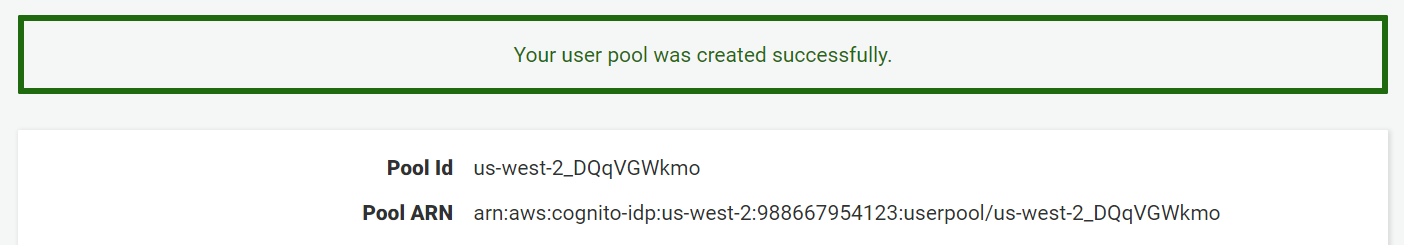
1. Create a new user pool

A screenshot of a cell phone

Description automatically generated

1. Enter a name ex: Appointment Pool.A screenshot of a cell phone

   Description automatically generated
2. Click Review defaults
3. Click Create Pool.
4. You will see a Pool ID, save it for later.



1. Select App clients on the left, add client
2. Add an app client name and Create app client, uncheck secret client option.

A screenshot of a cell phone

Description automatically generated

1. An App client id will appear, save it for later.

A screenshot of a cell phone

Description automatically generated

1. Go back to the S3 bucket. Select the js folder and download the config.js file
2. Open the config.js file on a text editor. We are going to attach the cognito user pool to the web application.
3. Under the cognito json object, replace the userPoolId, userPoolClientId, and region with the values saved earlier. Leave the api (invokeURL) for now.

A screenshot of a cell phone

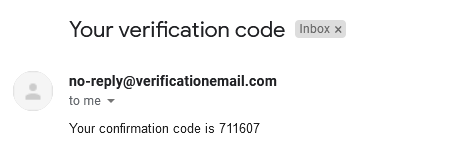
Description automatically generated

1. Reupload the config.js file back into the js folder in the S3 bucket.
2. On the S3 bucket, go to the properties tab, click static website hosting and click the endpoint.A screenshot of a cell phone

   Description automatically generated
3. Register for an account using Gmail. Use a password with greater than 6 characters, an uppercase letter, and a symbol. Eg: Testtest1!

A screenshot of a cell phone

Description automatically generated

1. Click sign up.
2. It should send an email verification to the account. 
3. Verify the email using the verification code. A screenshot of a cell phone

   Description automatically generated
4. Click verify.
5. Check if the user is created in the cognito user pool.
6. Go to cognito in AWS services.
7. Click Manage User Pools, then Appointment Pool.

A screenshot of a cell phone

Description automatically generated

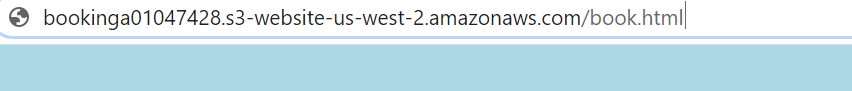
A screenshot of a cell phone

Description automatically generated

1. Click users and groups on the left. The created user should show up. Verify the account status is CONFIRMED.

A screenshot of a social media post

Description automatically generated

1. Go back to the web app.
2. Test if the app prevents non-users from accessing the book.html page. Replace the /signin.html on the URL with /book.html
3. Hit enter, it should redirect back to the signin.html page.
4. Now sign in to the web app and sign in using the account that was created.A screenshot of a cell phone

   Description automatically generated
5. Click sign in.
6. It would then redirect to the /book.html page like this.

A screenshot of a social media post

Description automatically generated

**Build a serverless backend**

1. Go to DynamoDB under services
2. Click create table

A picture containing bird, tree, flower

Description automatically generated

1. Call the table name Users, and Username as the Primary Key

A screenshot of a cell phone

Description automatically generated

1. Click Create.
2. Select the table created, go to the overview tab and save the arn for laterA screenshot of a cell phone

   Description automatically generated
3. Create an IAM role. Go to IAM under services.
4. Go to Roles, create role.

A screenshot of a cell phone

Description automatically generated

1. Choose Lambda, then click Next: Permission.
2. Search for and attach these 5 policies.

* AmazonAPIGatewayInvokeFullAccess
* AmazonDynamoDBFullAccess
* AmazonSESFullAccess
* CloudWatchLogsFullAccess
* AWSLambdaBasicExecutionRole

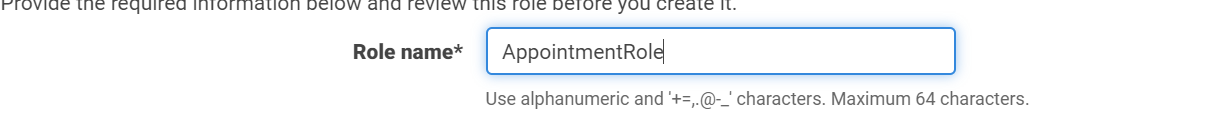
We are going to be using Lambda functions to read/write into our DynamoDB. CloudWatch logs for debugging and troubleshooting in case there is an error.

1. Click Next: Tags and Next: Review
2. Should have these policies attached.

A screenshot of a cell phone

Description automatically generated

1. Enter Role name Eg. AppointmentRole. Click create role.



1. Create the lambda functions for the api gateway.
2. Go to lambda under services.
3. Click functions on the left, then create function
4. Select author from scratch. Enter a function name Eg. home
5. Use an existing execution role.
6. Select the role we created earlier, AppointmentRole

A screenshot of a social media post

Description automatically generated

1. Create function
2. From the lambda folder provided, open the home.js file. Copy and paste the code into the index.js file of the lambda function.

A screenshot of a social media post

Description automatically generated

1. Click Save.
2. This lambda function grabs the data from the dynamo db for the home page. The web application will handle which time slots are available for booking and which time slots are already booked and could be waitlisted.
3. We are going to create a SES service so that our web app can send notifications to users.
4. Go to Simple Email Service in services.
5. Select Email addresses on the left, then click Verify a New Email Address
6. Enter an email address. It can be the same email address as the one used to sign up for the web app.

A screenshot of a social media post

Description automatically generated

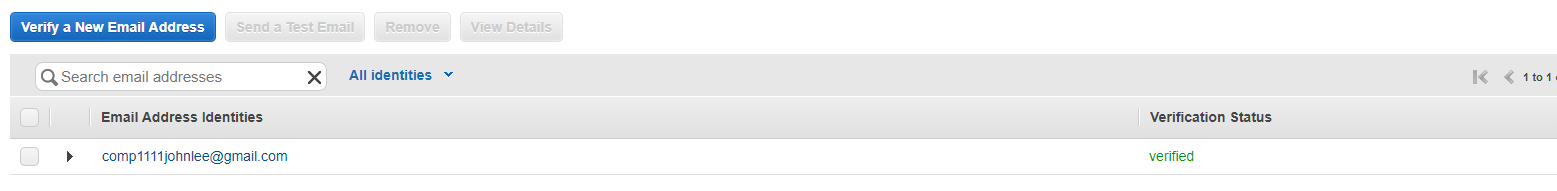
1. Click Verify This Email Address.
2. Click Close.
3. Now go check the email account and verify the email.

A screenshot of a cell phone

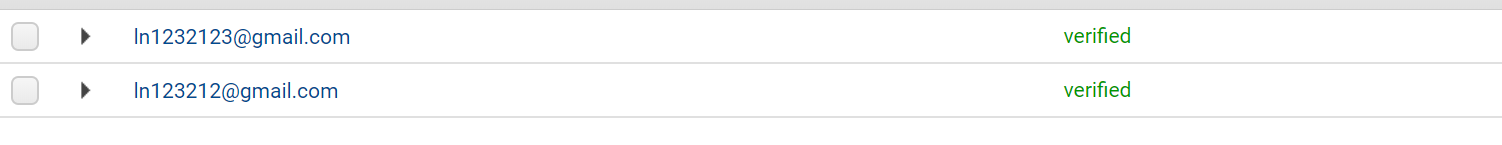
Description automatically generated

1. Take note of the region (us-west-2) Oregon.
2. Go back to the SES and check that the email is verified. Now that email can send and receive emails using the SES.





1. Repeat the steps and add another email.



1. Use the new email and make a second account on the web app at /register.html

A screenshot of a cell phone

Description automatically generated

1. Verify that this account has been created in the cognito user pool
2. Go to cognito in services, manage user pools, Appointment Pool, click users and groups on the left.

A screenshot of a cell phone

Description automatically generated

1. Go to the lambda service, create another lambda function called book. Use an existing role and choose the AppointmentRole. Click create function.

A screenshot of a cell phone

Description automatically generated

1. Open the book.js folder. Copy and paste the code into the index.js folder of the book lambda function.
2. Look at the line of code near the top. For the variable ses, make sure the region is the same as the region that the verified email is (us-west-2).

A screenshot of a cell phone

Description automatically generated

1. Scroll down to the function sendEmail(recipient,Btime). This function uses the SES service to send a confirmation email to the user that booked a time.
2. Replace the source emal with the email that was signed up with SES

A screenshot of a social media post

Description automatically generated

1. Click save.
2. Scroll up to the username variable. This line grabs the current user that is logged in to cognito.



1. Scroll up to the bookAppointment(username, Btime) function
2. This function updates the DynamoDB Users table with the username and their booked time when a user books a time.
3. Create another lambda function. Author from scratch and Call it waitlist. Use an existing role and select AppointmentRole.

A screenshot of a social media post

Description automatically generated

1. Create function
2. Open the waitlist.js file in a text editor in the lambda folder provided. Copy and paste the code into the index.js file of the waitlist lambda function.
3. Make sure the ses region is correct.

A picture containing knife

Description automatically generated

1. Scroll down to sendEmail(recipient, Wtime). This function sends a confirmation email when the user waitslists for a time slot that is already booked. Replace the source with the SES email.

A screenshot of a cell phone

Description automatically generated

1. Scroll up to the username variable. This line grabs the current user that is logged in to cognito.



1. Scroll down to waitList(username, Wtime) function. This function updates the DynamoDB Users table with the time waitlisted when the user signs up for a waitlist.

A screenshot of a cell phone

Description automatically generated

1. Create a new function. Author from scratch and call it notifywatilisted. Use an existing role and choose the role AppointmentRole.

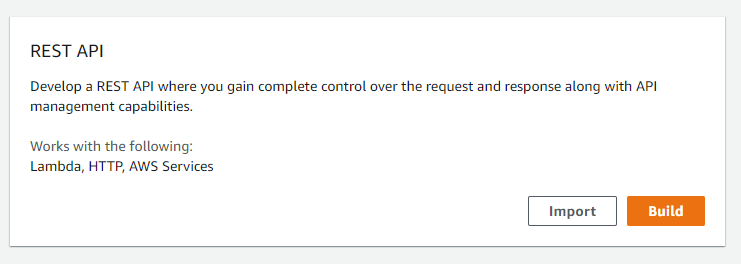
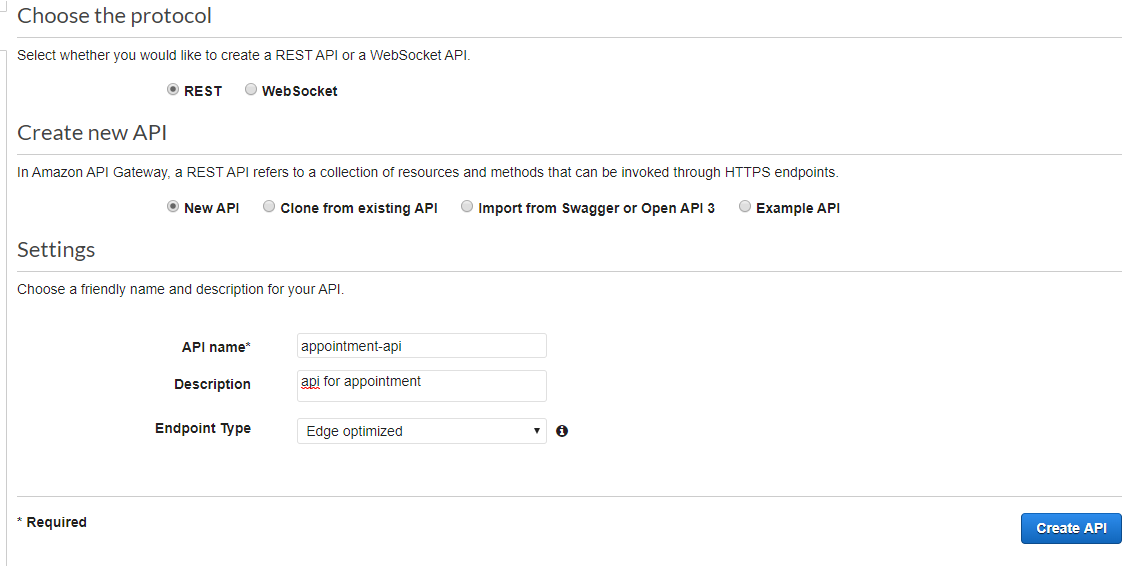
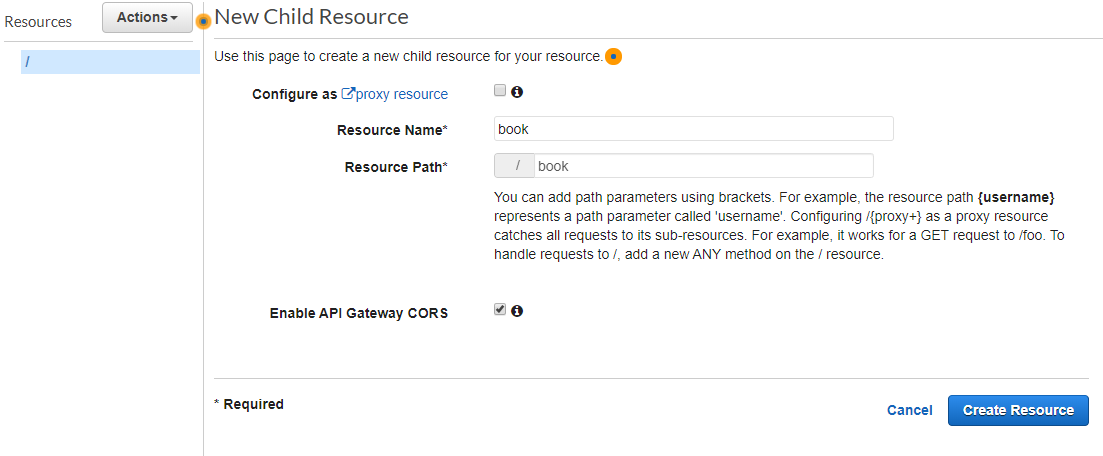
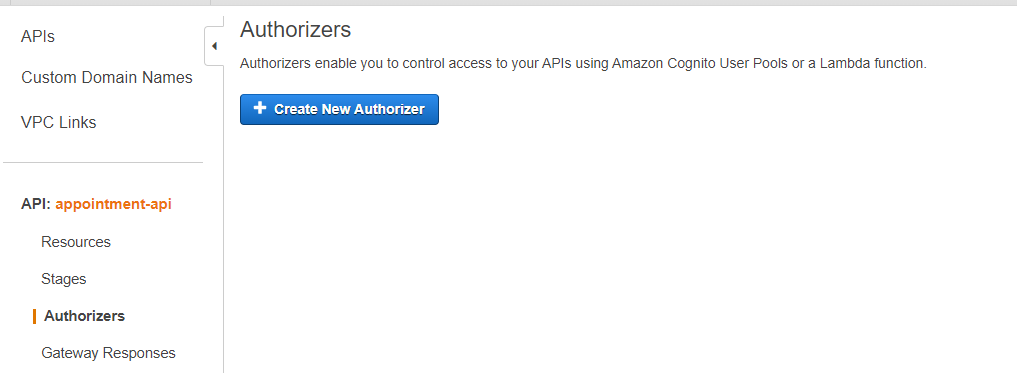
A screenshot of a social media post

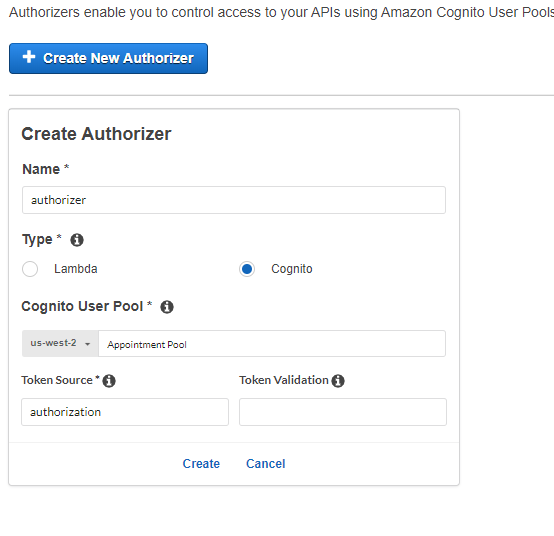
Description automatically generated

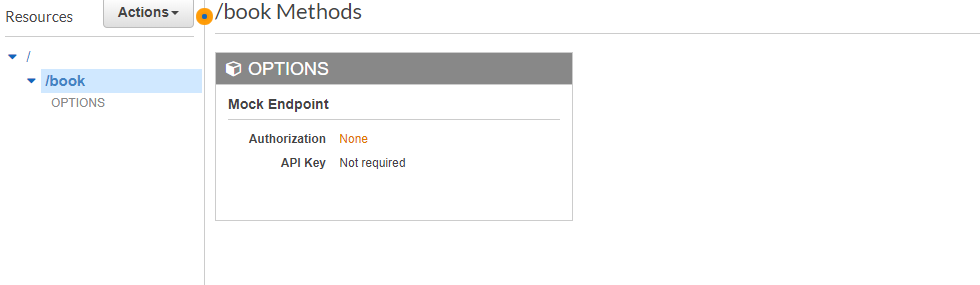
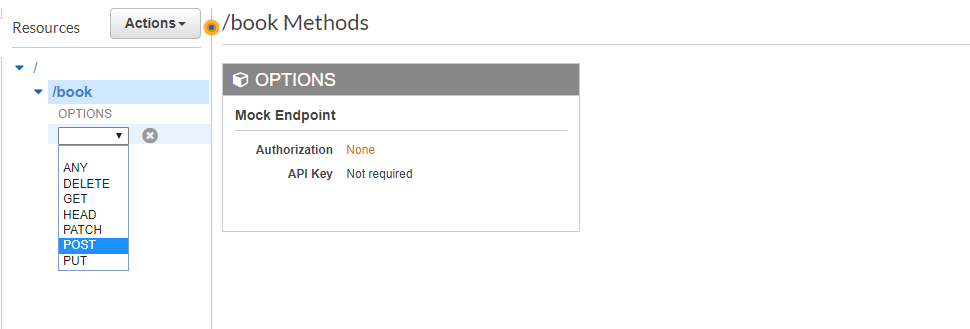
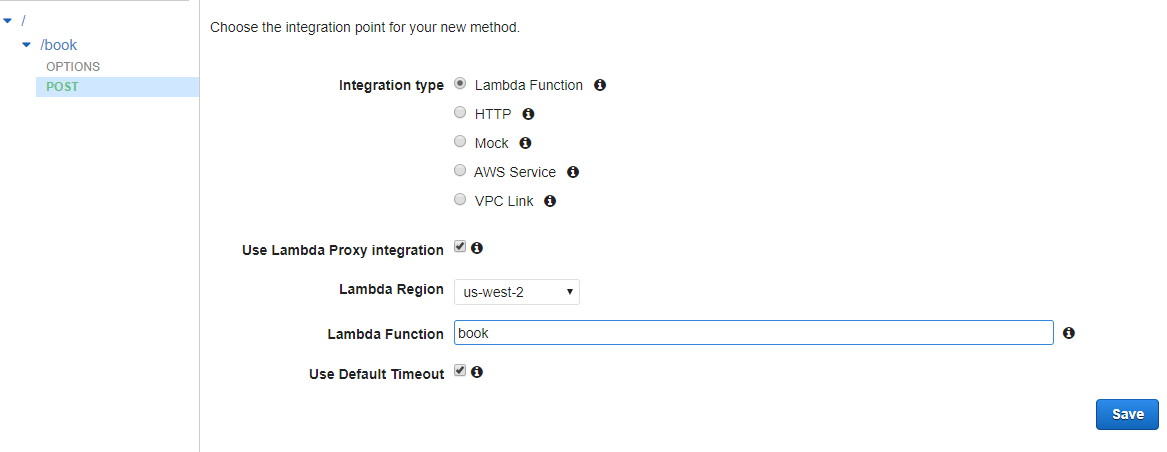
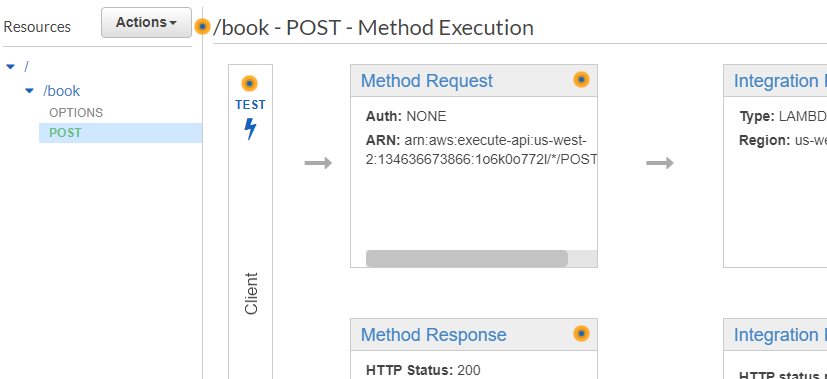
1. Click create function
2. Open the notifywaitlisted.js file to text editor in the lambda folder. Copy and paste the code to the index.js file in the lambda function.
3. Scroll to the sendEmail(recipient, Wtime) function. This function sends an email to the recipient when their waitlist timeslot has been freed.
4. Replace the source with the SES email.
5. Save.

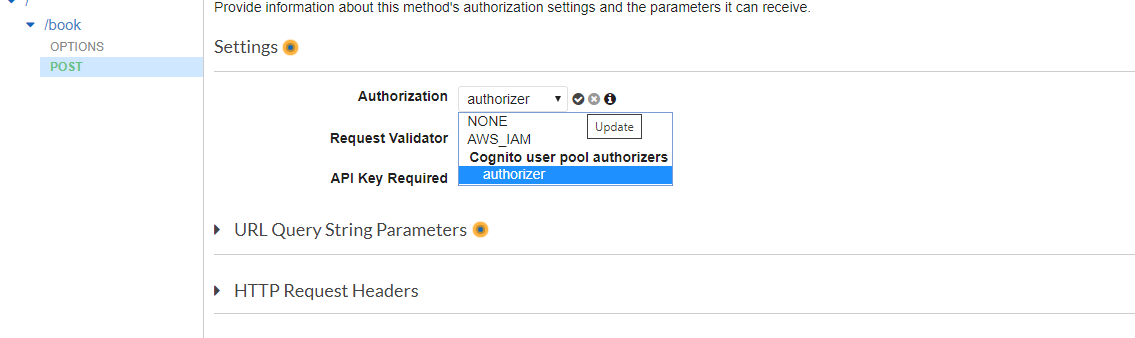
A screenshot of a cell phone

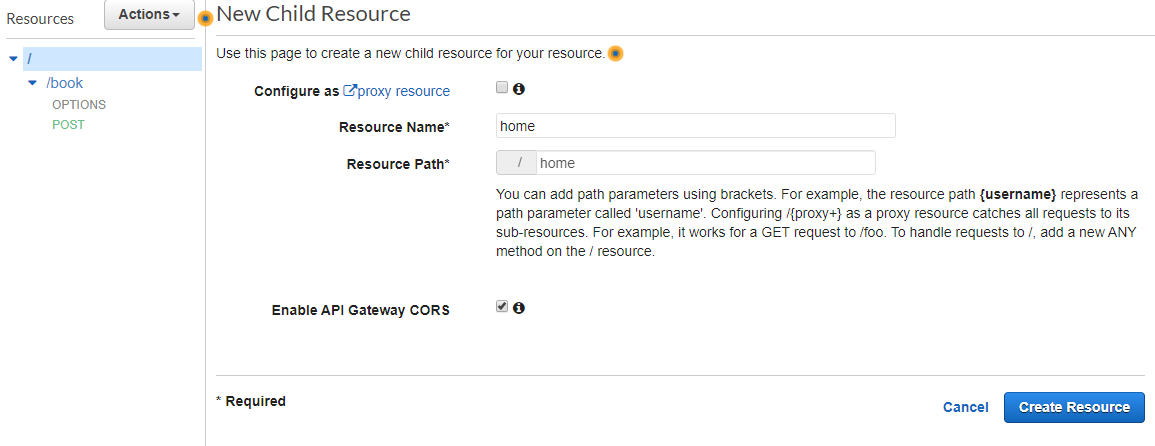
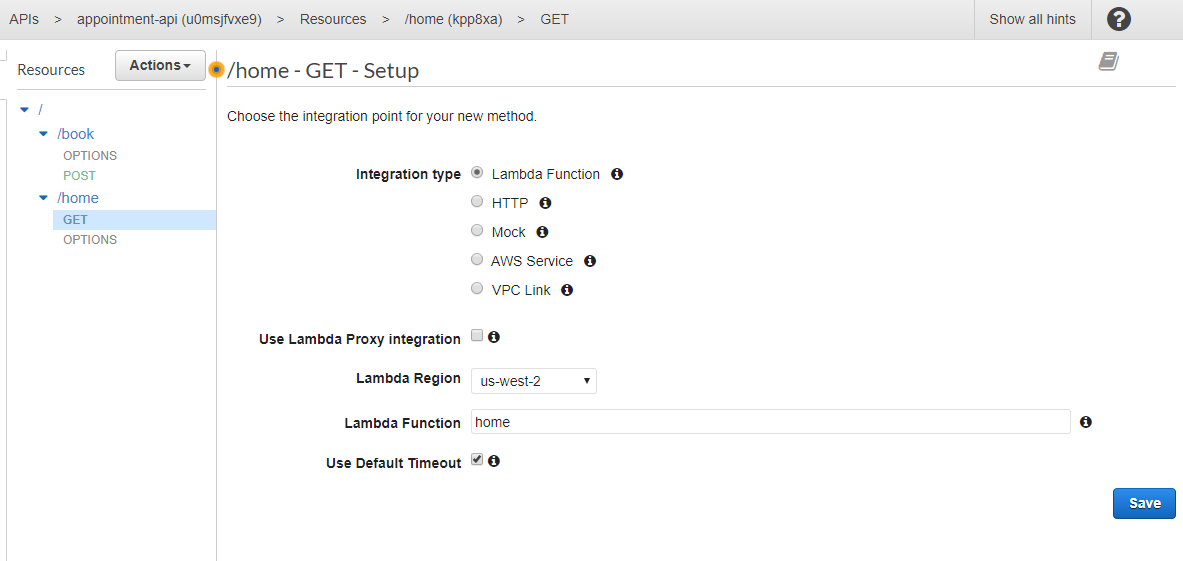
Description automatically generated

1. Next we will be deploying an API Gateway in order to expose our lambda functions that we have just built. Go to the API Gateway section in AWS.
2. We will be developing a REST API so select build on REST API.
3. Give your api a name and a description and select edge optimized under endpoint type and leave the result on defaults 
4. Create a resource and name it book. Enable API Gateway CORS. 
5. We will now make an authorizer for our api gateway so go to the authorizer section on the left and click create new authorizer
6. Select Cognito as the type and select the user pool you created earlier. Type in authorization for the Token Source and leave validation blank. Give it a name, I gave it “authorizer” and hit create.



1. Make sure /book is selected now and create a new method under the actions dropdown menu
2. Select POST, click the check mark.
3. On the next page for lambda function select the corresponding lambda function so in our case it would be book, enable lambda proxy integration and hit save and ok to add permission. 
4. Click on method request 
5. Select your authorizer that you created earlier and select the check mark to update



1. Select the “/” at the top again to get back to the base path and create another resource and call it “home”. It should show /home in the path not /book/home. Enable API Gateway CORS and hit create resource. 
2. Create a get method for home, click the checkmark and select your home lambda function. Hit save and ok.
3. You can test this api by going to your Dynamo DB, under Users table, and creating a new item, selecting text in the dropdown where it says “tree” and using this

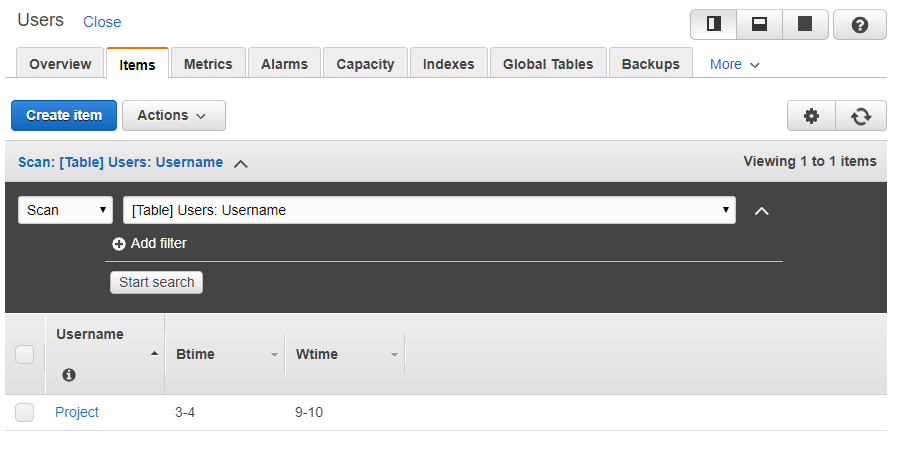
{

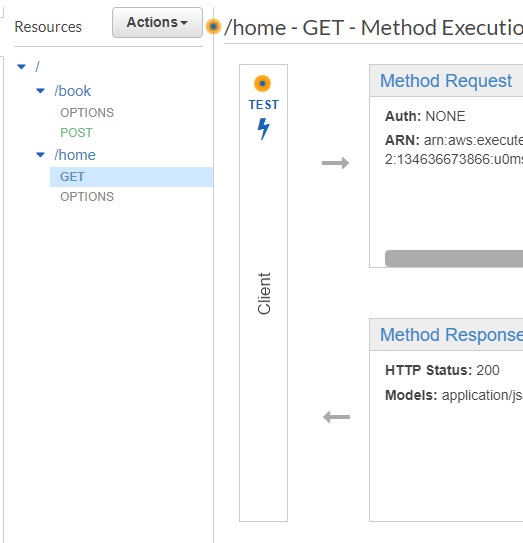
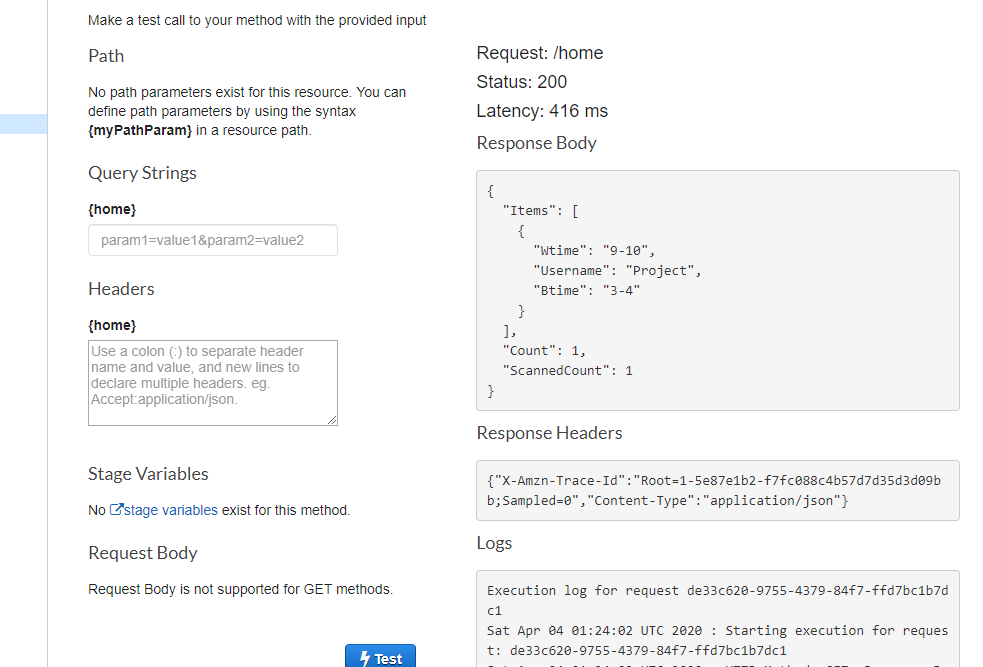
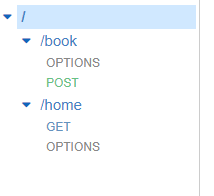
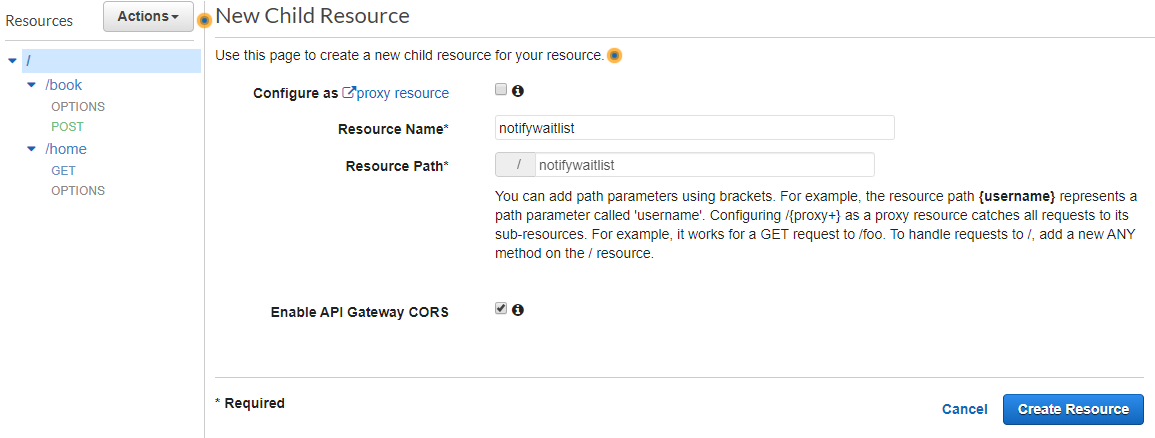
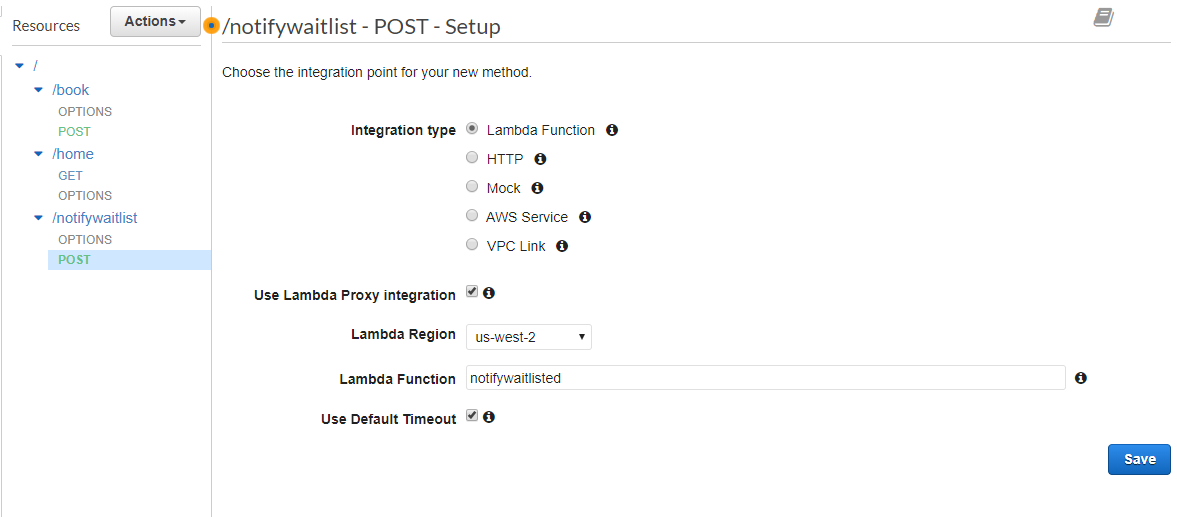
"Username": "Project",

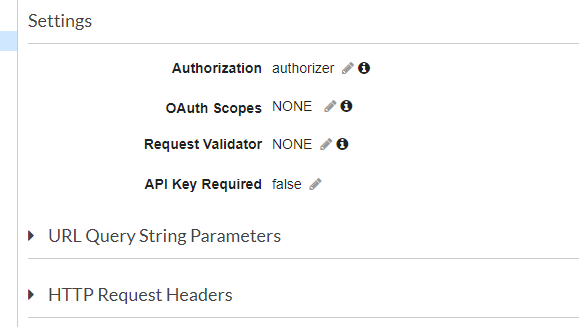
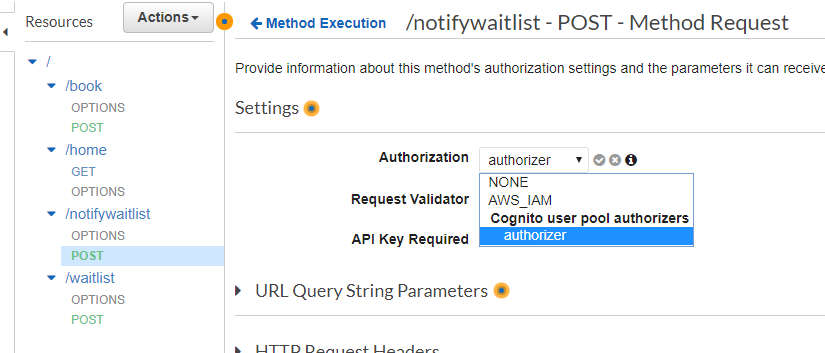
"Wtime": "9-10",

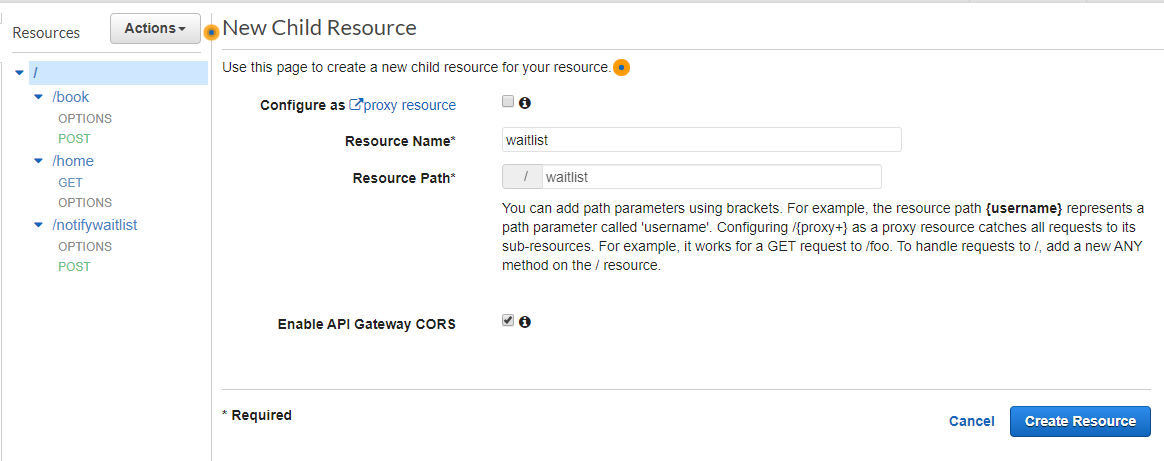
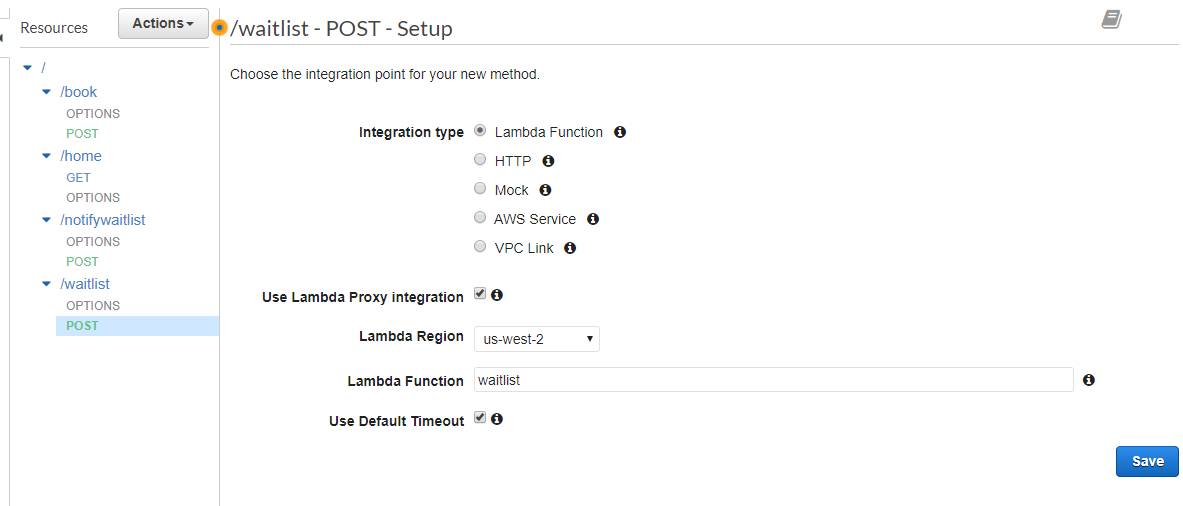
"Btime":"3-4"

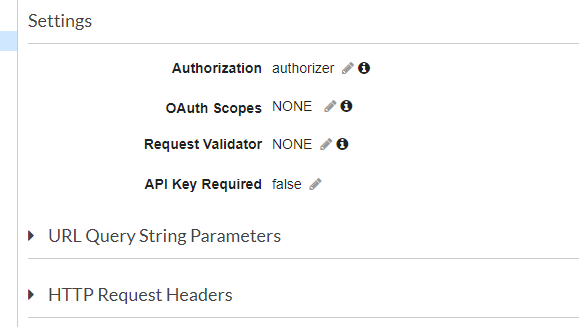
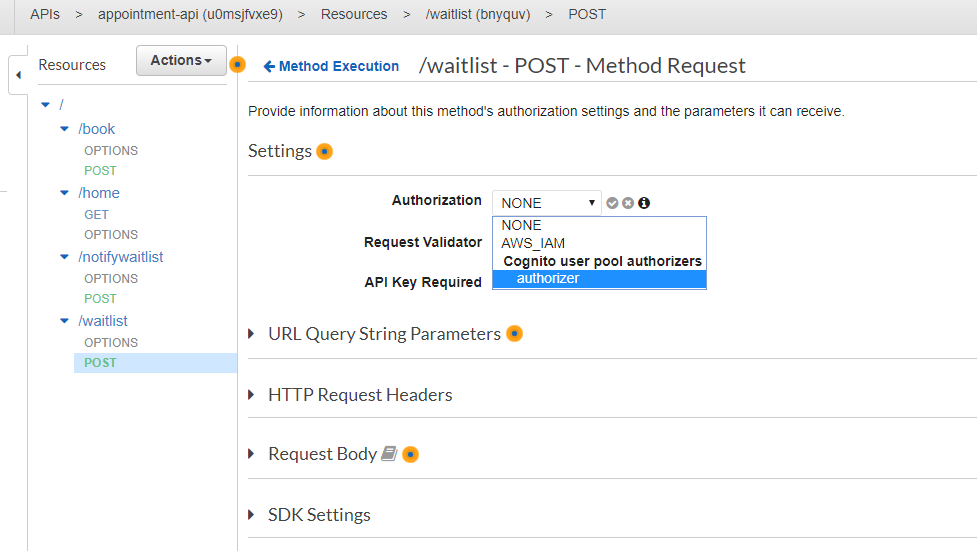
}



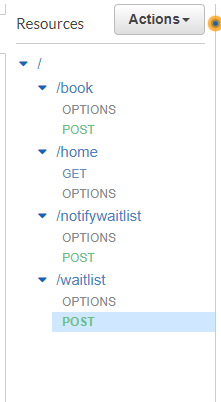
1. Go back to your API gateway get method and select test on the left 
2. Hit test and the one line you entered into the dynamodb should appear. 
3. Go back to your DynamoDB, select the row you just made and click the actions dropdown to hit delete.
4. We will now be making the rest of the methods. Select the root path again 
5. Create a resource called notifywaitlist, enable API Gateway CORS and hit create resource. 
6. Create a method for this resource and make it a POST. Click the checkmark. Select use Lambda Proxy Integration and select the Lambda function notifywaitlisted and save.
7. Select the method request for this post and select the authorizer you created earlier for the authorization.



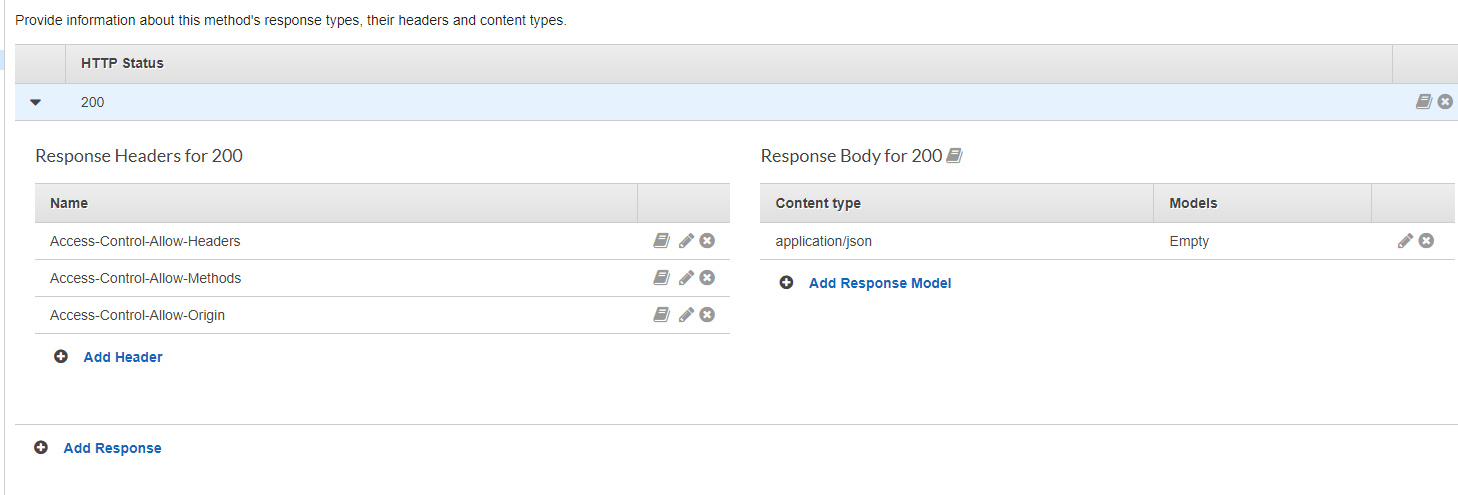
1. Select the root path again and create another resource called waitlist and enable CORS. 
2. Create a POST method under waitlist. Click the checkmark. Enable proxy integration again and select the waitlist lambda function
3. Select the method request for the post and select the authorizer from earlier for authorization



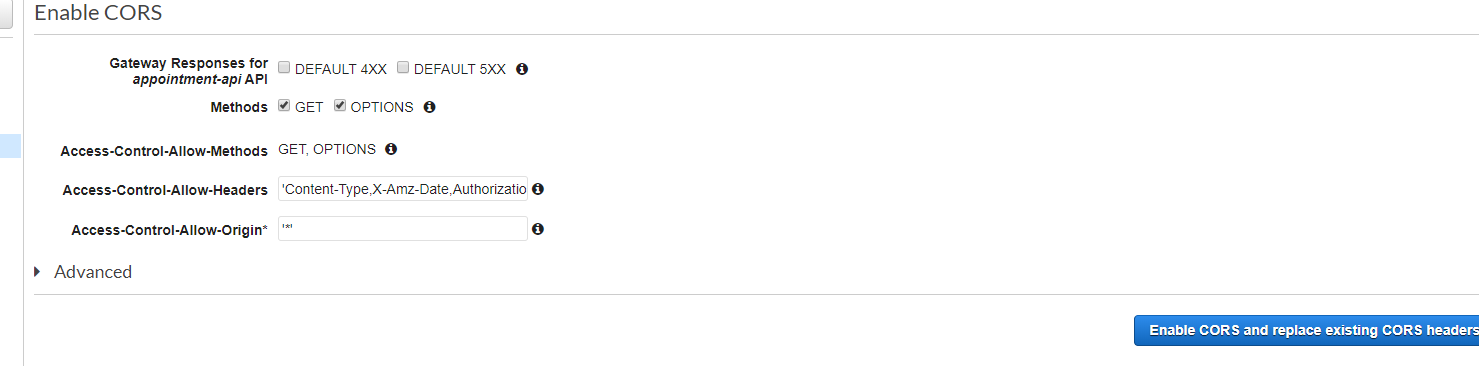
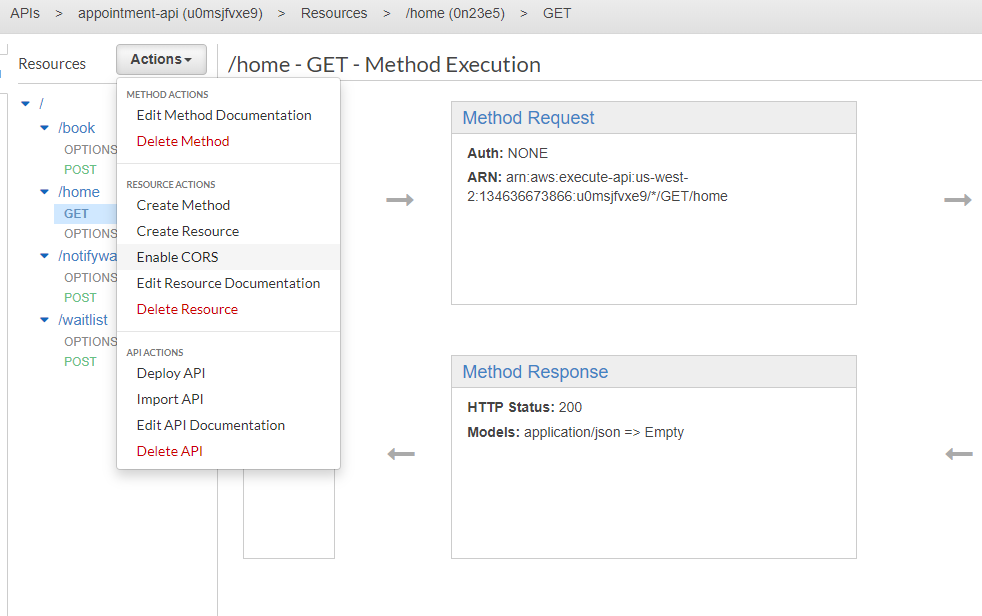
1. Your api gateway should now look like this

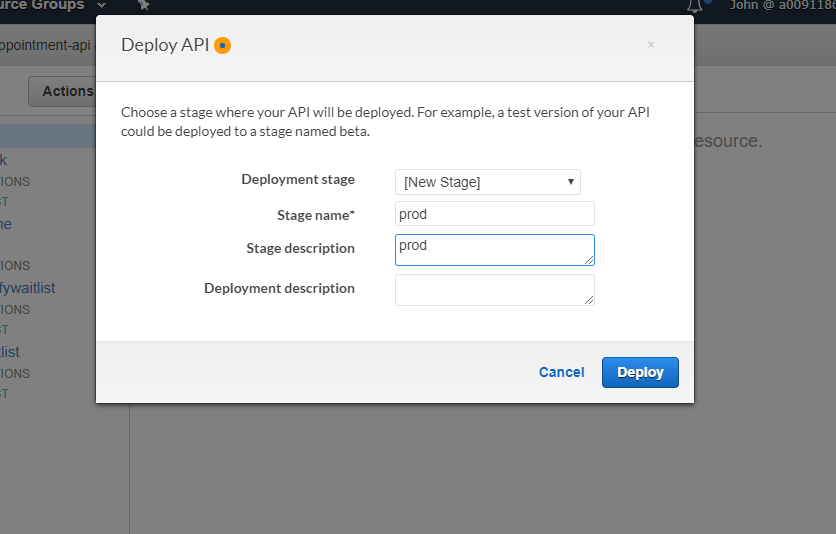


1. On each POST and GET, select method response and add these response headers for 200 – Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin



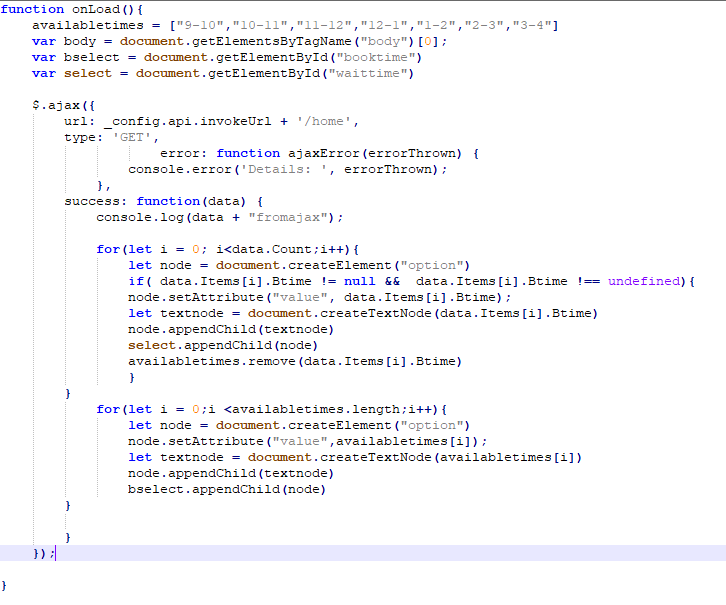
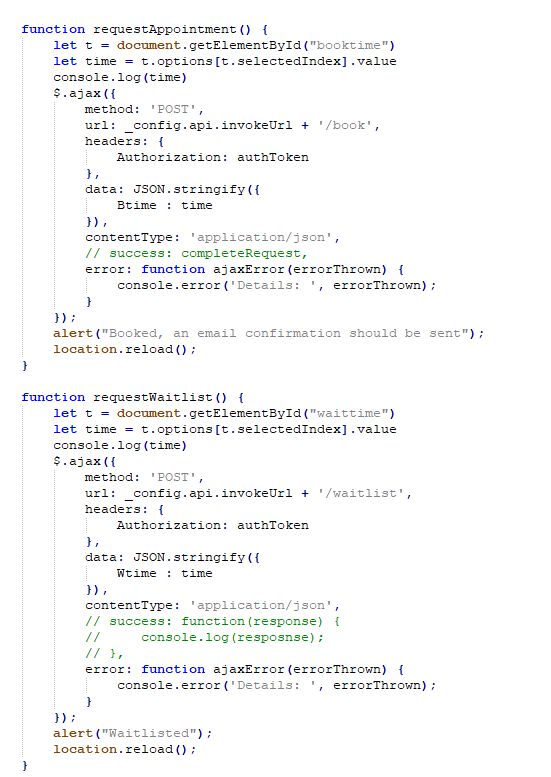
1. On the get method for /home select enable cors and accept the defaults

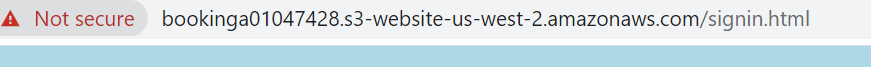


1. Click the actions dropdown on your API and deploy your api. Type in “prod” for the stage name
2. Grab the invoke URL. We will use this later.
3. Download the config.js file from your s3 bucket in the js folder.
4. Change the invoke url to the one you got from your API gateway in step 85

A screenshot of a cell phone

Description automatically generated

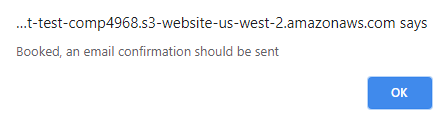
1. Reupload the new config.js file onto your s3 bucket js folder.
2. Take a look at the Book.js file, the onload method helps generate the initial list of available times and by removing the booked times from the full list of available times. It grabs the booked times from the /home path from the API gateway we connected with the invoke url put in earlier. 
3. The checkBooking function grabs from the same api but takes the wait time instead and if it matches up an available time with the wait time for a user then the user will be emailed an alert with /notifywaitlist. 
4. requestAppointment and requestWaitlist both use the API to post a book time and a wait time to the DynamoDB. 
5. Go back to the static web app from the s3 bucket.
6. Add /book.html to the end of the URL and hit enter.
7. If it redirects to /signin.html, login again with your account.



1. Login to a user, select a booking time, click book, you should receive and alert saying you will receive an email confirmation. Clicking book should call the /book POST method from the api gateway which goes to the book lambda function.

A screenshot of a cell phone

Description automatically generated



1. Check your dynamo DB table for that booking time

A screenshot of a cell phone

Description automatically generated

1. Check your email for a confirmation

A screenshot of a cell phone

Description automatically generated

1. Keep this account logged in, now on a different browser, log on to your second account that was created previously
2. Once you get to the booking part it should look like this now
3. Notice how 9-10 is on the waitlist now because your other account has it booked already.
4. Using your second account, click on waitlist for the 9-10 slot. This should call /waitlist POST from the api gateway which points to the waitlist lambda function. The email with that signed up for the waitlist should see a confirmation email

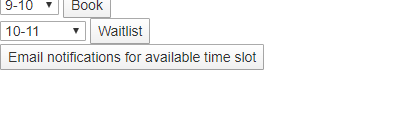
A picture containing bird, flower, tree

Description automatically generated

1. Confirm that one user has a booked time and one user has a waitlisted time A screenshot of a cell phone

   Description automatically generated
2. Now go back to your first account and select a different time, I did 10-11 and click on book. Confirm the DynamoDB entry changed. A screenshot of a cell phone

   Description automatically generated
3. Go back to the booking website and click on the email notifactions for available time slot button, this should send an email to your second account since it was waitlisted by now that slot is available. That should call /notifywaitlist from the api gateway which points to the notifywaitlist lambda function.



1. Confirm that the second account received an email notification. It should say 

**References**

North, Freddie. “Getting Started.” *Amazon*, NTC Pub. Group, 1998, aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/.

Web App code structure was modified and based off an amazon example repo

https://github.com/awslabs/aws-serverless-workshops/archive/master.zip