**Rex Angelo Basulgan IT 426 Project**

**System Architecture**



**Rex’s Photoshoot Scheduler**

SA-001

Version 1

October 29, 2024

Content Owner: Rex Angelo Basulgan

**Revisions/Versions**

|  |  |  |
| --- | --- | --- |
| **Document Number** | **Release/Revision** | **Release/Revision Date** |
| SA-001 | **1.0** | **October 29, 2024** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Table of Contents**

Title Page……………………………………………………………………………………. 1

Revisions/Versions.……………………………………………………………………. 2

Table of Contents……………………………………………………………………….. 3

Project Proposal. ………………………………………………………………………. 4

Original Architectural Design…………………………………………………….. 5

Implemented Architectural Design……………………………………………. 6

Challenges …………………………………………………………………………………..7

Guided Lab…………………………………………………………………………………. 8

S3 Bucket …………………………………………………………………… 9

Lambda Function …………………………………………………………… 10

AWS SNS …………………………………………………………………11-13

DynamoDB ………………………………………………………………… 14-16

Time Sheets …………………………………………………………………………. 17-18

Consulting Hours …………………………………………………………………. 19-20

**Project Proposal**

**Project Name**: Rex’s Photoshoot Scheduler

**Deliverable**: A web application where it allows users to schedule photoshoots, select preferred date and times, able to choose the type of photoshoot, and receive a confirmation message. This scheduler will help people who are looking for a specific photoshoot more simpler and user friendly.

**User Stories**:

As a user, I want to fill in a form with my information so that I can book a photo shoot.

As a user, I want to select the preferred date and time of the photo shoot to check availability.

As a user, I want to specify the type of photoshoot so I can get the appropriate session.

As a user, I want to receive a confirmation email after the booking is successfully submitted.

As a user, I want to indicate my contact preference: email or phone, in which I should be notified.

As an administrator, I want a listing of photoshoots scheduled to manage my calendar effectively.

Admin- I want to be informed when the creation of a new photoshoot has taken place for tracking purposes of new bookings.

User- I want to see available time slots to avoid scheduling conflicts.

User- The interface should be very simple and intuitive so that one is easily able to submit a form and book.

User- I want to get a reminder before the photoshoot so that I will not forget my appointment.

**Original Architectural Design**

A diagram of a computer

Description automatically generated

**Explanation:**

The architecture will be composed of S3, hosting a static website that the users will access through their web browsers. The API Gateway acts as a bridge between the front end and the back end, enabling the site to make requests to Lambda functions. These functions will encapsulate the business logic, like fetching or updating of data in DynamoDB. For longer-running tasks, messages can also be set into SQS queues for asynchronous processing. SNS notifies the users or administrators when any particular events occur. AWS Amplify is optionally used to help deal with front-end resources and deployment.

**Implemented Architectural Design**

A diagram of a computer

Description automatically generated

**Explanation:**

The architecture will be composed of S3, hosting a static website that the users will access through their web browsers. The API Gateway acts as a bridge between the front end and the back end, enabling the site to make requests to Lambda functions. These functions will encapsulate the business logic, like fetching or updating data in DynamoDB. queues for asynchronous processing.

SNS notifies the users or administrators when any particular events occur.

**Challenges**

Creating a website from scratch. It was challenging at the beginning because I needed to finalize my idea and make sure that everything worked.

The “Lab role” doesn’t have enough permission for most parts of the lab. Example SQS etc.

I tried SQS, but I did not have enough permission, so I just focused on SNS.

I also had a challenging time with my lambda and lambda code, where I needed to look for examples online and go back to my previous assignments. I also followed the video of Brother Strain on YouTube to make sure I did it right.

**Guided Lab**

Here’s a step-by-steps for hosting your HTML in S3 and integrating it with the form submission to AWS:

1. Create an S3 Bucket:

In the S3 console, create a new bucket with a unique name (photoshootscheduler).

Choose the region closest to your audience for better performance. (N. Virginia)

Click Create Bucket

1. Upload HTML, CSS, and JavaScript Files:

Here’s the link:   
WEBSITE: <https://photoshootscheduler.s3.us-east-1.amazonaws.com/index.html>

HTML and CSS: <https://drive.google.com/drive/folders/10KltJdh2mslUA7UIHXUPR9ltcv0Ur_uO?usp=sharing>

Go to the Objects tab and upload all your website files (index.html, css/styles.css, and any images or assets).

1. Check the URL

Go to your bucket, copy and paste your Object URL to a new tab or browser.

You notice that the website says access denied. Let’s move to the next step.

1. Enable static website hosting

To enable static website hosting, go to the Properties tab of your bucket. Look for Static website hosting, then click edit. Inside the static website hosting, click enable. In the index document, type index.html. Leave everything in default, then save changes.

1. Set Permissions for Public Access:

Go to the Permissions tab of your bucket, and edit the Bucket policy to allow public read access to your files.

Use this example policy, replacing YOUR\_BUCKET\_NAME with your actual bucket name:

JSON:

{

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

"Statement": [

{

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::YOUR\_BUCKET\_NAME/\*"

}

]

}

After enabling public access, you can view your site using the S3 Static Website URL found in the Static Website hosting section of your S3 bucket properties.

Create Lambda Function

**Steps to Set Up HTTP API and Deploy**

**Step 1: Create and Configure the HTTP API**

1. **Go to API Gateway**:
   * Navigate to **API Gateway** in the AWS Management Console.
   * Click **Create API**, then select **HTTP API**.
2. **Create an HTTP API**:
   * Name your API (e.g., PhotoshootAPI).
   * Under **Configure routes**, click **Add route** to create a route for the /schedule endpoint.
3. **Set the Method and Integration**:
   * For the **Method**, select POST.
   * Under **Integration**, choose **Lambda Function**.
   * In the **Lambda Function** field, choose the Lambda function that processes the form data (e.g., process\_schedule\_data).
   * Click **Create** to save this route.

**Step 2: Deploy the HTTP API**

1. **Create a Stage** (if not already created):
   * After creating the HTTP API, go to **Deployments** in the left sidebar.
   * If no deployment stage exists, create a new stage by clicking **Create stage**.
   * Name the stage (e.g., prod) and click **Create**.
2. **Deploy the API**:
   * In the **Deployments** section, click **Deploy** to deploy your API.
   * Choose the stage (e.g., prod) where you want to deploy the API.

**Step 3: Obtain the Invoke URL**

1. **Find the Invoke URL**:
   * After deployment, go to the **Stages** section for your API.
   * The **Invoke URL** for the deployed stage (e.g., prod) will appear at the top of the page.
   * Copy this URL.
2. **Update Your Form's Action**:
   * In your HTML form, update the action attribute to point to the **Invoke URL** of your HTTP API, appending /schedule (or the correct endpoint path):

<form action="https://your-api-id.execute-api.your-region.amazonaws.com/prod/schedule" method="POST">

TEST THE SETUP

1. **Submit the Form**:
   * Go to your webpage and submit the form.
2. **Verify Lambda Execution**:
   * Check the **CloudWatch Logs** to verify that the Lambda function was triggered successfully.
3. **Check for Data**:
   * If you added logic to store the form data (e.g., in DynamoDB), verify that the data was saved as expected.
   * If you added notifications (e.g., via SNS), verify that the notifications were sent.

**Step 1: Create an SNS Topic**

1. **Log in to AWS Management Console** and navigate to **Amazon SNS**.
2. Click on **Topics** in the left menu.
3. Click **Create Topic**.
   * **Topic type**: Standard (for most cases; FIFO is for strict ordering and deduplication).
   * **Name**: PhotoshootSchedulerTopic.
4. Click **Create Topic**.

**Step 2: Add Subscriptions**

1. Go to the topic you just created (PhotoshootSchedulerTopic).
2. Click on the **Create subscription** button.
3. For **Protocol**, choose:
   * **Email**: If you want to send email notifications.
   * **Lambda**: To trigger your Lambda function.
   * **HTTP/S**: To send the data to a webhook or another system.
4. Enter the **endpoint** based on the protocol you selected (e.g., email address, Lambda ARN, or webhook URL).
5. Click **Create subscription**.
6. If using **email**, check your inbox for a confirmation email and confirm the subscription.

Here’s the example screenshot:

A screenshot of a email

Description automatically generated

**Step 3: Integrate SNS with Your Lambda Function**

Update your Lambda function to publish messages to the SNS topic when a form is submitted:

python

Copy code

UPDATE LAMBDA function:   
  
import json

def lambda\_handler(event, context):

try:

# Validate and parse the body

if 'body' not in event or not event['body']:

raise ValueError("Request body is missing or empty.")

body = json.loads(event['body'])

# Validate required fields

required\_keys = ['name', 'email', 'phone', 'date', 'time', 'type']

for key in required\_keys:

if key not in body:

raise KeyError(f"Missing required field: {key}")

# Extract data

name = body['name']

email = body['email']

phone = body['phone']

date = body['date']

time = body['time']

photoshoot\_type = body['type']

# Return a successful response

return {

'statusCode': 200,

'headers': {

'Content-Type': 'application/json',

'Access-Control-Allow-Origin': '\*'

},

'body': json.dumps({

'message': 'Your photoshoot has been scheduled successfully!',

'data': {

'name': name,

'email': email,

'phone': phone,

'date': date,

'time': time,

'type': photoshoot\_type

}

})

}

except (ValueError, KeyError, json.JSONDecodeError) as e:

# Handle errors gracefully

return {

'statusCode': 400,

'headers': {

'Content-Type': 'application/json',

'Access-Control-Allow-Origin': '\*'

},

'body': json.dumps({

'error': 'Failed to schedule photoshoot',

'details': str(e)

})

}

**Step 4: Update Lambda Environment Variables**

1. Open the Lambda function in the AWS Console.
2. Under **Configuration**, go to **Environment variables**.
3. Add a new key-value pair:
   * **Key**: SNS\_TOPIC\_ARN
   * **Value**: The ARN of your SNS topic (arn:aws:sns:us-east-1:123456789012:PhotoshootSchedulerTopic).
4. Save the changes.

**Step 5: Test the Integration**

1. Deploy your Lambda function.
2. Submit the form from your web app.
3. Verify:
   * Email subscribers receive a notification with form details.
   * HTTP or Lambda subscribers process the message correctly.

**Steps to Deploy This Lambda Function**:

1. **Set Up the Lambda Function**:
   * Go to **AWS Lambda** and create a new function.
   * Add the above code, and in the environment variables, set QUEUE\_URL to your SQS queue URL.
2. **Connect Lambda to API Gateway**:
   * Create an API Gateway REST API.
   * Add a POST method to an endpoint like /schedule.
   * Link it to your Lambda function.

Create DYNAMODB

**1. Create a DynamoDB Table**

1. Go to the [DynamoDB Console](https://console.aws.amazon.com/dynamodb).
2. Click **"Create table"**.
3. Configure the table:
   * **Table Name**: PhotoshootSchedule (or your desired name).
   * **Partition Key**: id (String).
4. Click **"Create"**.

**2. Update IAM Role**

Ensure your Lambda function's IAM role has permissions to interact with DynamoDB.

**Attach Policy:**

1. Go to **IAM** in the AWS Console.
2. Find the role associated with your Lambda function.
3. Attach the AmazonDynamoDBFullAccess policy.

Update LAMBDA code:   
  
import json

def lambda\_handler(event, context):

try:

# Retrieve form data from the POST request

body = json.loads(event['body'])

name = body.get('name')

email = body.get('email')

phone = body.get('phone')

date = body.get('date')

time = body.get('time')

photoshoot\_type = body.get('type')

# Optional: Process or save the data (e.g., store it in a database or send a confirmation email)

# For now, we're just returning the received data as a response

# Log the received data (optional but useful for debugging)

print(f"Received data: {body}")

# Return a successful response

return {

'statusCode': 200,

'headers': {

'Content-Type': 'application/json',

'Access-Control-Allow-Origin': '\*' # CORS header for cross-origin requests

},

'body': json.dumps({

'message': 'Your photoshoot has been scheduled successfully!',

'data': {

'name': name,

'email': email,

'phone': phone,

'date': date,

'time': time,

'type': photoshoot\_type

}

})

}

except Exception as e:

# Log the error for debugging

print(f"Error occurred: {str(e)}")

# Return an error response

return {

'statusCode': 400,

'headers': {

'Content-Type': 'application/json',

'Access-Control-Allow-Origin': '\*' # CORS header

},

'body': json.dumps({

'error': 'Failed to schedule photoshoot',

'details': str(e)

})

}

**4. Test the Integration**

1. Deploy the updated Lambda function.
2. Use the **API Gateway URL** and test the form submission.
3. Verify the data is being stored in the DynamoDB table:
   * Go to DynamoDB in the AWS Console.
   * Select the PhotoshootSchedule table.
   * Check the **Items** tab for the submitted data.

### **Expected DynamoDB Table Structure**

After submission, each item should look like this in the DynamoDB table:

| **id** | **name** | **email** | **phone** | **date** | **time** | **type** |
| --- | --- | --- | --- | --- | --- | --- |
| Random numbers like 1234556gndkjdlgfgf | John Doe | johndoe@example.com | 1234567890 | 2024-12-10 | 10:00 | Portrait |

**Time Sheets**

|  |  |  |  |
| --- | --- | --- | --- |
| **DATE** | **ITEM** | **TIME SPENT** | **RUNNING TOTAL** |
| Oct 29, 2024 | Brainstorming and asking for ideas in class with my groupmate | 1 hour | 1 hour |
| November 2, 2024 | Finalize my project and look for ideas online. | 2 hours | 3 hours |
| November 7, 2024 | Create a website to use for this project and upload in s3 bucket | 2 hours | 5 hours |
| November 8, 2024 | Create lambda function and API | 4 hours | 9 Hours |
| November 10, 2024 | Updated the website and the guided lab. | 2 hours | 11 Hours |
| November 12, 2024 | Started creating Architectural Design | 2 hours | 13 Hours |
| November 19, 2024 | Starts with SQS / Lambda function | 3 hours | 16 Hours |
| November 26, 2024 | SNS | 1 hour | 17 hours |
| December 3, 2024 | SNS | 2 hours | 19 hours |
| December | Dynamo DB | 2 hours | 21 hours |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Consulting Hours**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DATE** | **GIVEN OR**  **RECEIVED** | **WHO** | **ITEM** | **TIME SPENT** |
| 11/19/2024 | Received | Michael Palopalo | IAM roles – He explained to me that the role that we use “Labrole” doesn’t provide enough access to SQS, SES, and SNS. | 5 mins. |
| 11/19/2024 | Given | Domingo Gallibu | Lambda Function – Explained how to use lambda functions and suggest some videos to follow. | 10 mins. |
| 11/19/2024 | Received | Jonathan Crisanto | Jonathan mentioned that it plays crucial role in decoupling my frontend to my backend system. He showed me some examples too. | 15 minutes |
| 11/19/2024 | Given | Jonathan Crisanto | I showed some examples of lambda functions. I also mentioned that he needs to code price scraper in Lambda to gather price information from product URL. | 15 minutes |
| 11/26/2024 | Received / Given | Maybelle | We talked about our plans for our project because we have same problem about the SQS. | 5 minutes |
| 11/26/2024 | Received | Nico Gonzales | Asked about SNS process. | 5 Minutes |
| 11/26/2024 | Given | Domingo Gallibu | Helped how to create SNS | 5 minutes |
| 11/26/2024 | Given | Jonathan Crisanto | Jonathan needs help on how to connect api gateway. I told him that by creating a gateway, he could either use HTTP or REST for API type and what resource he's going to use like Lambda or HTTP endpoint. | 5 minutes |

**Demo Video**

**Youtube Link:** [**https://youtu.be/qUjQVDs8RtA**](https://youtu.be/qUjQVDs8RtA)