# Uploading Files to GitHub

**Step-by-Step Guide**

1. **Prepare Files:**

* Create a new folder on your computer.
* Place all the files you want to upload into this folder.

1. **Open Visual Studio Code:**

* Launch Visual Studio Code.
* Use the "Open Folder" option to open the folder you created in step 1.

1. **Initialize Git:**

* Open the integrated terminal in Visual Studio Code.
* Run the following command to initialize Git within the project folder:

**A black screen with a white border

Description automatically generated**

1. **Add Files to Git:**

* In the terminal, navigate to the project folder if not already there.
* Run the following command to add all files to the staging area:

A black screen with white letters

Description automatically generated

1. **Create a New Branch:**

* Decide on a meaningful name for your new branch.
* Run the following command to create and switch to the new branch:

A black background with white text

Description automatically generated

1. **Commit Changes:**

In the terminal, use the following command to commit the staged changes with a descriptive message:



1. **Add Remote Repository:**

* On GitHub, create a new repository if you haven't already.
* Copy the remote repository URL.

A black background with white text

Description automatically generated

* In the terminal, run the following command to add the remote repository URL:

A black background with white text

Description automatically generated

1. **Push Changes to GitHub:**

Use the following command to push your committed changes to GitHub:

A black background with white text

Description automatically generated

**WAREHOUSE PROJECT**

Pavankumar Etta

Renuka Prasanna

Prathyusha Kuppili

Chaya YK

**Table of Contents**

Contents

[Uploading Files to GitHub 1](#_Toc147790239)

[1. AWS RESOURCES AND SETUP 5](#_Toc147790240)

[1.1: Setting up an AWS account 5](#_Toc147790241)

[1.2: Creating an Amazon DynamoDB Table 6](#_Toc147790242)

[1.3: Creating AWS Lambda Functions: 8](#_Toc147790243)

[1.4: Configuring Amazon API Gateway: 11](#_Toc147790244)

[2. SCHEMA DESIGNS FOR THE DATABASE 16](#_Toc147790245)

[2.1: Database systems and the Lifecycle: 16](#_Toc147790246)

[2.2: Requirements Analysis: 18](#_Toc147790247)

[2.3: Entity Relationship (ER) Modelling: 18](#_Toc147790248)

[2.4: Schema Integration Methods: 19](#_Toc147790249)

[2.5: Schema Designs for Project: 20](#_Toc147790250)

[3. SNS (SIMPLE NOTIFICATION SERVICE) 22](#_Toc147790251)

[3.1: Core Concepts: 22](#_Toc147790252)

[3.2: Use Cases: 22](#_Toc147790253)

[3.3: Sending OTPs and Custom Messages Using Amazon SNS: 22](#_Toc147790254)

[4. SES (SIMPLE EMAIL SERVICE) 25](#_Toc147790255)

[4.1: Introduction: 25](#_Toc147790256)

[4.2: Implementation: 25](#_Toc147790257)

[5. GPS INTEGRATION 31](#_Toc147790258)

[5.1: Three steps to fetch the polylines from source to destination: 31](#_Toc147790259)

[6. APPOINTMENT API 33](#_Toc147790260)

[6.1: Verifying Email: 33](#_Toc147790261)

[6.2: Code Part: 33](#_Toc147790262)

[6.3: Testing: 35](#_Toc147790263)

[7. SPLASH SCREEN 36](#_Toc147790264)

[7.1: Introduction: 36](#_Toc147790265)

[7.2: Implementation: 36](#_Toc147790266)

[8. ONBOARDING SCREEN 37](#_Toc147790267)

[8.1: Introduction: 37](#_Toc147790268)

[8.2: Implementation: 37](#_Toc147790269)

# AWS RESOURCES AND SETUP

# 1.1: Setting up an AWS account

Amazon Web Services (AWS) is a comprehensive and widely used cloud computing platform provided by Amazon.com. AWS offers a vast array of cloud services and resources that enable businesses, organizations, and individuals to build and deploy applications, websites, and other services in a flexible, scalable, and cost-effective manner.

Steps to create AWS Account:

1. **Go to the AWS Website:**

**Open your web browser** and navigate to the AWS website: <https://aws.amazon.com/>.

1. **Create Your AWS Account:**
   1. Click "Sign into the Console."
   2. Choose "**Create an AWS Account**."
   3. Follow the prompts to provide contact information, payment details, and accept AWS terms.
   4. Choose a support plan if needed.
2. **Access the AWS Console:**

After successful setup, log in to the **AWS Management Console** with your account credentials.

You are now ready to start using AWS services for your projects, including setting up resources like Amazon DynamoDB, Lambda functions, and API Gateway.

**A screenshot of a computer

Description automatically generated**

**Fig 1: AWS Management Console**

# 1.2: Creating an Amazon DynamoDB Table

Amazon DynamoDB is a cloud-based database service provided by Amazon Web Services (AWS) that stores and manages your data in a highly scalable and reliable way. Imagine it as a big, fast, and organized digital filing cabinet for your data.

Steps to create DynamoDB Table

1. Navigate to the **DynamoDB service**.

A screenshot of a computer

Description automatically generated

**1**

**Fig 2: DynamoDB service**

1. Click "**Create table**."

A screenshot of a computer

Description automatically generated

**2**

**Fig 3: DynamoDB service**

1. Define the **table name, primary key**, and other optional settings like provisioned capacity or on-demand capacity mode.

A screenshot of a computer

Description automatically generated

**4**

**3**

**Fig 4: Give table name and primary Key.**

1. Review and **create the table**.

A screen shot of a computer

Description automatically generated

**5**

**Fig 5: Click on Create table.**

**For our project,** we have established two tables:

* **'WH\_Consumers’** with 'Consumer\_id' as the primary key, which stores consumer demographic details
* **'WH\_Products**,’ with 'Product\_id' as the primary key, which stores product-related information.
* **'WH\_Trip’** with ‘Trip\_id as the Primary key, which stores Trip-related information.

A screenshot of a computer

Description automatically generated

**Fig 6: Created 3 tables.**

# 1.3: Creating AWS Lambda Functions:

AWS Lambda is a serverless compute service offered by Amazon Web Services (AWS) that allows you to run code without provisioning or managing servers. Lambda is designed to help developers build applications in a scalable, cost-effective, and low maintenance way.

Steps to create a Lambda Function:

* Navigate to the **AWS Lambda service** in the AWS Management Console.

A screenshot of a computer

Description automatically generated

**1**

**Fig 7: Click on Lambda service.**

* Click "**Create function**."

**2**

A screenshot of a computer

Description automatically generated

**Fig 8: Click on Create function.**

* Choose the "**Author from scratch**" option.

A screen shot of a computer

Description automatically generated

**3**

**Fig 9: Choose Author from scratch.**

* Configure the function as follows:
  + **Function name**: Give your Lambda function a descriptive name.
  + Runtime: Choose the runtime compatible with your **programming language** (e.g., Node.js, Python, Java).

Note: For Project we used “Python3.8”

* + Execution role: Create a new **role** with basic Lambda permissions and DynamoDB access. This role allows the Lambda function to interact with DynamoDB.

Note: For Project we used “userrole”

* Click "**Create function**."

**A screenshot of a computer program

Description automatically generated**

**5**

**4**

**Fig 10: Give Function name and Runtime.**

**A screenshot of a computer

Description automatically generated**

**7**

**6**

**Fig 11: Choose Execution role.**

**A screenshot of a computer

Description automatically generated**

**8**

**Fig 12: Click on Create function.**

1. In the function **code editor**, write the code for your Lambda function to perform the desired CRUD operation. For example, if it's the function for creating a document, write code to insert data into DynamoDB.
2. **Deploy** your Lambda function.

**A screenshot of a computer

Description automatically generated**

**10**

**9**

**Fig 13: Write Lambda Function and Deploy.**

For our project, we have developed **CRUD Lambda functions** for 'WH\_Consumers' ,'WH\_Products' and WH\_Trip tables.

**A screenshot of a computer

Description automatically generated** **A screenshot of a computer

Description automatically generated**

**Fig 14: Created CRUD Lambda functions separately for tables.**

**GitHub Link:**

# 1.4: Configuring Amazon API Gateway:

1. Navigate to the Amazon **API Gateway** service in the AWS Management Console.

A screenshot of a computer

Description automatically generated

**1**

**Fig 15: Click on API Gateway**

1. Click "**Create API**."

A screenshot of a computer

Description automatically generated

**2**

**Fig 16: Click on Create API**

* Choose the "**REST API**" option.

A close-up of a computer screen

Description automatically generated

**3**

**Fig 17: Click on Build**

1. Configure your API, including the **API name** and default stage name and Click "**Create API**."

A screenshot of a computer

Description automatically generated

**7**

**6**

**5**

**4**

**Fig 18: Configure API**

1. Create **resources** and **methods** for each CRUD operation.

**Create Resources**

Resources represent the endpoints of your API. They are organized hierarchically to match the structure of your API.

* In the "**Actions**" menu, click "**Create Resource**”.
* Enter a **name** for the resource (e.g., "/users")."
* Enable API gateway CORS.
* Click "**Create Resource**."

A screenshot of a web page

Description automatically generated

**2**

**1**

**Fig 19: Creating Resource**

A screenshot of a computer

Description automatically generated

**4**

**5**

**3**

**Fig 20: Creating Resource**

**Create Methods**

Methods define how API Gateway handles incoming requests for a resource. Common methods are GET, POST, PUT, DELETE, etc.

* Select the resource you just created.
* Click "**Create Method**."
* Choose the **HTTP method** (e.g., GET, POST) you want to associate with the resource.

A screenshot of a computer

Description automatically generated

**1**

**Fig 21: Creating Methods**

**A screenshot of a method type

Description automatically generated**

**2**

**Fig 22: Choose Method Type**

1. Configure each method to use Lambda integration. Choose the Lambda function created earlier as the integration target for each method.

**3**

**A screenshot of a computer

Description automatically generated**

**4**

**Fig 23: Choose Lambda function to integrate.**

1. Deploy the API to create an accessible endpoint.

**A screenshot of a computer

Description automatically generated**

**1**

**Fig 24: Click on Deploy API.**

**A screenshot of a computer

Description automatically generated**

**3**

**2**

**Fig 25: Select stage and deploy.**

1. After deployment, you can test your API using the provided Invoke URL

**A close-up of a computer screen

Description automatically generated**



**Fig 26:Test API using URL .**

# SCHEMA DESIGNS FOR THE DATABASE

# 2.1: Database systems and the Lifecycle:

1. **data**—a fact, something upon which an inference is based (information or knowledge has value, data has cost)
2. **data item**—smallest named unit of data that has meaning in the real world (examples: last name, address, SSN, political party)
3. **data aggregate (or group)** -- a collection of related data items that form a whole concept; a simple group is a fixed collection, e.g., date (month, day, year); a repeating group is a variable length collection, e.g., a set of aliases.
4. **record**—group of related data items treated as a unit by an application program (examples: presidents, elections, congresses)
5. **file**—collection of records of a single type (examples: president, election)
6. **database**—computerized collection of interrelated stored data that serves the needs of multiple users within one or more organizations, i.e., interrelated collections of records of potentially many types.

A diagram of a product

Description automatically generated

A screenshot of a computer

Description automatically generated

A diagram of a data distribution

Description automatically generated

**Fig 19: Database Life Cycle**

# 2.2: Requirements Analysis:

Example: order entry clerk

Function: Take customer orders and either fill them or adjust.

|  |  |  |
| --- | --- | --- |
| Task Def | Volume | Data Elements |
| 1. Create order | 2000 | A, B, E, H |
| 1. Validate order. | 2000 | A, B, G, H, J |
| 1. Fill out error form. | 25 | A, C |
| 1. Reserve item/price | 6000 | A, D, H |
| 1. Request alternate items. | 75 | A, E, I, K, M |

Frequency: daily

# 2.3: Entity Relationship (ER) Modelling:

**Concepts:**

1. **Entity** - a class of real-world objects having common characteristics and properties about which we wish to record information.
2. **Relationship** - an association among two or more entities

* **occurrence** - instance of a relationship is the collective instances of the related entities.
* **degree** - number of entities associated in the relationship (binary, ternary, other n-ary)
* **connectivity** - one-to-one, one-to-many, many-to-many
* **existence dependency** (constraint) - optional/mandatory

1. **Attribute** - a characteristic of an entity or relationship

A diagram of a work flow

Description automatically generated

**Fig 20: ER Modelling**

# 2.4: Schema Integration Methods:

1. Goal in schema integration- to create a non-redundant unified (global) conceptual schema.

Completeness - all components must appear in the global schema.

Minimality - remove redundant concepts in the global schema.

Understandability - does global schema make sense?

1. **Comparing of schema:**

* **Naming conflicts**- same name for different concepts synonyms - different names for the same concept
* **Structural conflicts** - different modelling construct for the same concept
* **Type conflicts** - different modelling construct for the same concept (e. g. “order” as an entity, attribute, relationship)
* **Dependency conflicts** - connectivity is different for different views (e.g., job-title vs. job-title-history)
* **key conflicts** - same concept but different keys are assigned (e.g., ID-no vs. SSN)
* **Behavioral conflicts** - different integrity constraints (e.g., null rules for optional/mandatory: insert/delete rules)

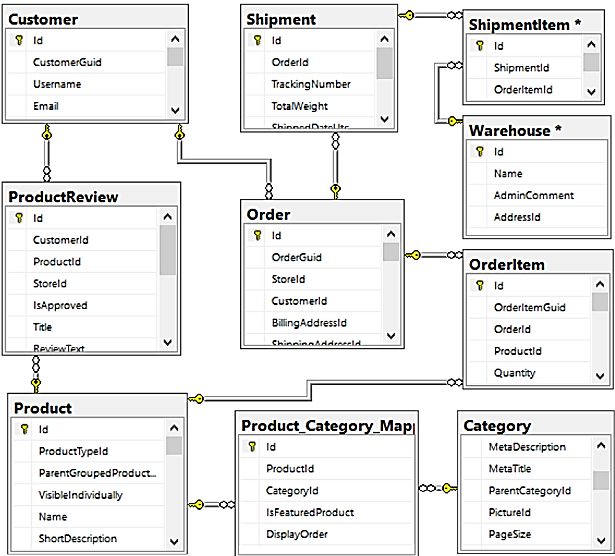
1. **Conforming of schema**

* Resolve conflicts (often user interaction is required)
* Conform or align schema to make compatible for integration.
* Transform the schema via
* Renaming (homonyms, synonyms, key conflicts)
* Type transformations (type or dependency conflicts)
* Modify assertions (behavioral conflicts)

1. **Merging and restructuring**

* superimpose entities.
* restructure result of superimposition

# 2.5: Schema Designs for Project:



A diagram of a customer order

Description automatically generated

# SNS (SIMPLE NOTIFICATION SERVICE)

Amazon Simple Notification Service (SNS) is a fully managed messaging service that enables the delivery of messages, notifications, and alerts to many recipients or subscribers. It follows a publish-subscribe model, allowing decoupled communication between different parts of an application or system.

SNS serves as a fundamental building block for scalable and event-driven architectures in the cloud. It offers benefits such as real-time communication, flexible message types, security controls, and seamless integration with other AWS services.

# 3.1: Core Concepts:

* **Publish-Subscribe Model**

At its core, SNS operates on a publish-subscribe messaging model. Publishers send messages to topics, and subscribers receive messages from the topics they are interested in.

* **Topics**

Topics act as communication channels or categories for messages. They enable the categorization of messages, making it easy to target specific audiences.

* **Subscriptions**

Subscriptions represent the endpoints or clients that want to receive messages from SNS topics. Subscribers can be Amazon SQS queues, AWS Lambda functions, HTTP/S endpoints, email addresses, SMS endpoints, and more.

* **Protocols and Endpoints**

SNS supports various protocols for message delivery, including HTTP/S, email, SMS, and application-specific protocols. Subscribers specify their preferred protocol and endpoint for message delivery.

* **Message Attributes**

Messages sent to SNS topics can include custom message attributes that provide additional metadata about the message content. Subscribers can use these attributes for filtering and processing.

* **Message Filtering**

SNS allows subscribers to specify filter policies to receive only the messages that match specific criteria, reducing the volume of messages received.

# 3.2: Use Cases:

Amazon SNS can be used in various scenarios, including:

**Real-time Notifications**: Sending notifications to users or systems in real time.

**Event-Driven Architecture**: Building event-driven and decoupled architectures.

**Scalable Application Communication**: Enabling communication between microservices and components.

# 3.3: Sending OTPs and Custom Messages Using Amazon SNS:

Using Amazon SNS to send OTPs and custom messages to employees and Consumers through Lambda functions and API Gateway, with payload including country codes and phone numbers, is a robust approach for secure communication. Below is a detailed guide on setting up this workflow:

**1. Create an SNS Topic:**

* Login to the AWS Management Console.
* Navigate to the Amazon SNS dashboard.
* Click on "Topics" and then "Create topic."
* Name the topic, e.g., "OTP-Notifications."

**2. Create Subscriptions:**

* Add subscribers to the topic. Subscribers can be employees' email addresses, phone numbers (for SMS), and any other relevant contact points.
* You can create multiple subscriptions for different types of recipients (e.g., one for employees and another for users).

**3. Lambda Function for OTP Generation:**

* Create a Lambda function to generate OTPs, and Custom messages.
* The function should accept the country code and phone number as input parameters.
* It generates the OTP and publishes it to the SNS topic.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

**Fig 21: Lambda Function to send OTP.**

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

**Fig 22: Lambda Function to send Custom messages.**

**4. API Gateway Setup:**

* Create an API Gateway API to act as the endpoint for OTP requests.
* Define the API resource, e.g., /send-otp.
* Create a POST method for this resource.

# SES (SIMPLE EMAIL SERVICE)

# 4.1: Introduction:

Amazon Simple Email Service (SES) is an email platform that provides an easy, cost-effective way for you to send and receive email using your own email addresses and domains. There are a few steps to setting up SES, such as verifying your domain, verifying your email address, and setting up MX records.

# 4.2: Implementation:

1. **Create S3 bucket:**

A screenshot of a computer

Description automatically generated

**1**

**Fig 23: Select S3 Service**

1. **Fill the required fields to create the buckets.**

A screenshot of a computer

Description automatically generated

**2**

**Fig 24: Fill the required fields.**

1. **Click on create bucket with these fields.**

A screenshot of a computer

Description automatically generated

**3**

**Fig 25: click on create.**

1. **Bucket will be created and select the bucket.**

**A screenshot of a computer

Description automatically generated**

**4**

**Fig 26: Bucket Created**

1. **Now create IAM role and policies.**

A screenshot of a computer

Description automatically generated

**5**

**Fig 27: Select IAM Service**

1. **Create role with the name and specified fields.**

**6**

A screenshot of a computer

Description automatically generated

**Fig 28: Create Role**

1. **Create lambda function with existing role.**

A screenshot of a computer

Description automatically generated

**7**

A screenshot of a computer

Description automatically generated

**8**

**Fig 29: Create Lambda Function**

1. A screenshot of a computer

   Description automatically generated**Write the code for the lambda function to create the email services.**

**9**

**Fig 29: Code for Lambda Function**

1. **Now go to Amazon SES within the services.**

A screenshot of a computer

Description automatically generated

**10**

**Fig 30: Select SES service.**

1. **Create the identity with Email Address.**

A screenshot of a computer

Description automatically generated

**11**

A screenshot of a computer

Description automatically generated

**12**

A screenshot of a computer

Description automatically generated

**13**

**Fig 31: Create Identity with Email Address**

1. **Now you will get list of identities with verifying the email address.**

A screenshot of a computer

Description automatically generated

**14**

**Fig 32: List of Identites**

1. **Now create SMTP credentials to create login details to SMTP account under Amazon SES.**

A screenshot of a mail transfer

Description automatically generated

**Fig 33: Create SMTP Credentials**

1. **Now, after creating the credentials we can get our API and based on the API mails will be sent.**

# GPS INTEGRATION

# 5.1: Three steps to fetch the polylines from source to destination:

**1. Set Up Google Maps API:**

* Go to the **Google Cloud Console**.
* Create a new project or use an existing one.
* **Enable the necessary APIs** for your project:
* Google Maps JavaScript API: For displaying maps and obtaining directions.
* Google Places API: For locating places of interest.
* Google Directions API: For obtaining directions between source and destination.
* Create **API keys** for each of these APIs.

**2. Frontend Integration:**

* Embed Google Maps in your web application using the Google Maps JavaScript API.
* Use the Directions Service from the Google Maps API to obtain directions and polylines between source and destination.
* Use the Places Service from the Google Maps API to search for places of interest near the source or destination.
* Build an apk for the GPS tracking from customer location to the destination**.**

A screenshot of a phone

Description automatically generated

**Fig 34: GPS tracking from customer location to the destination**

* The starting and ending point to locate the map.

A screenshot of a phone

Description automatically generated A screenshot of a phone

Description automatically generated

**Fig 35: Starting and Ending point to locate the map.**

* Final map route integration in the app

A map with a route

Description automatically generated

**Fig 36: Final Map Route**

**3. Backend Integration:**

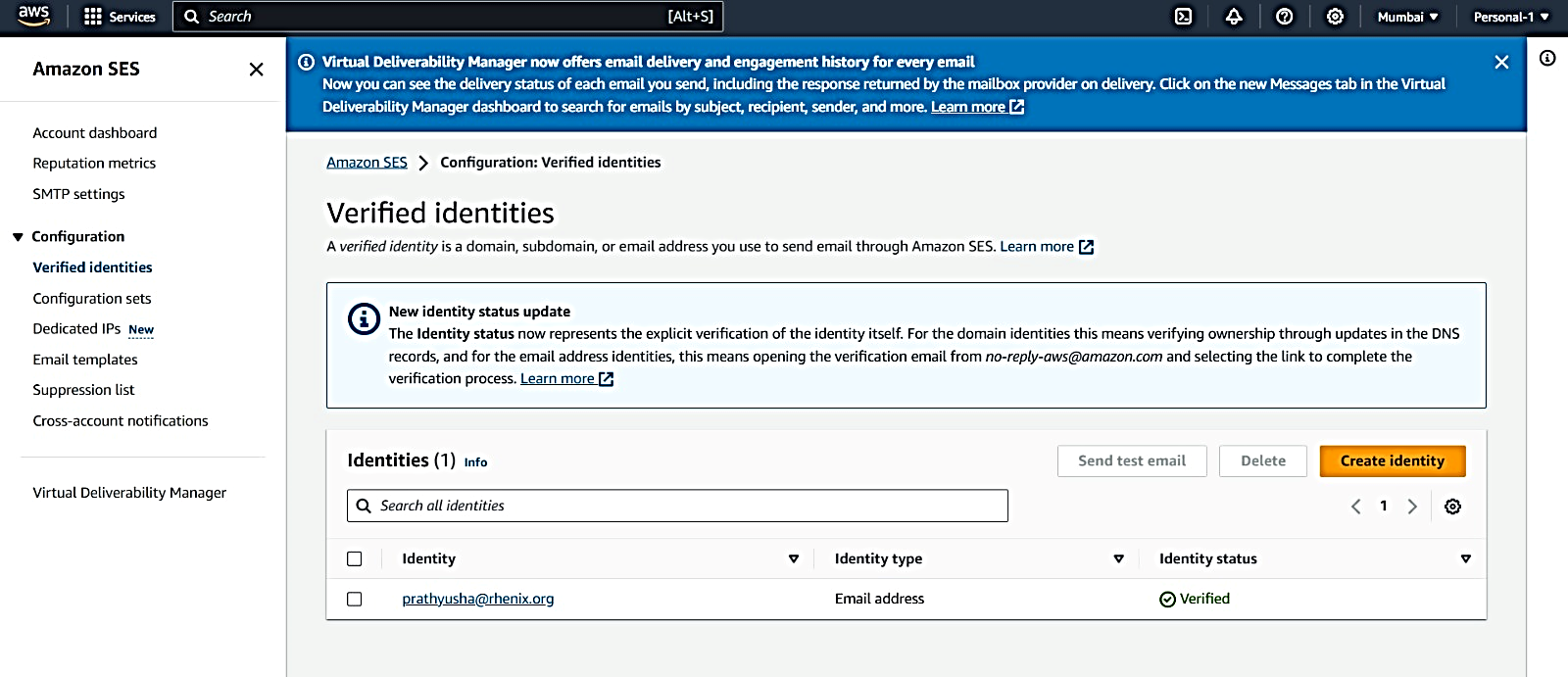
* You may also want to implement a backend server to store and manage API keys securely, especially if you have usage restrictions.
* You can make API requests from your server to Google Maps APIs, passing the results to the frontend.
* Note: Remember to replace Source Address, Destination Address, and other parameters with your actual values.

# APPOINTMENT API

The Appointments API allows you to create, read, update and delete appointments from a schedule. Note that it doesn’t support service schedules and while you can create new appointments in a capacity schedule, it is not possible to create empty slots through the API.

# 6.1: Verifying Email:

To create a new appointment, you need to send an HTTP POST request to /api/bookings. Json (or .xml). The request should either contain a JSON or an XML document describing the new user or have the fields as URI encoded parameters. See the table below for an explanation of the fields.



**Fig 37: Verifying Email**

# 6.2: Code Part:

Creating an appointment API using Flutter typically involves building a mobile app that communicates with a backend API.

import logging

import boto3

from botocore.exceptions import NoCredentialsError

# Initialize logger and set log level

logger = logging.getLogger()

logger.setLevel(logging.INFO)

# Initialize SES client for the appropriate region

ses\_client = boto3.client('ses', region\_name="ap-south-1")

# Initialize SNS client for the appropriate region

sns\_client = boto3.client('sns', region\_name="ap-south-1")

def send\_email(subject, body, recipient\_email):

try:

response = ses\_client.send\_email(

Source="admin@gmail.com", # Sender's email address

Destination={'ToAddresses': [recipient\_email]},

Message={

'Subject': {'Data': subject},

'Body': {'Text': {'Data': body}}

}

)

return response

except NoCredentialsError:

logger.error("No AWS credentials found.")

raise

def send\_sms(phone\_number, message):

try:

response = sns\_client.publish(

PhoneNumber=phone\_number,

Message=message,

MessageAttributes={

'AWS.SNS.SMS.SenderID': {

'DataType': 'String',

'StringValue': 'SENDERID'

},

'AWS.SNS.SMS.SMSType': {

'DataType': 'String',

'StringValue': 'Transactional'

}

}

)

return response

except Exception as e:

logger.error(f"Failed to send SMS: {str(e)}")

raise

def lambda\_handler(event, context):

# Extract appointment details from the event

appointment\_date = event.get("appointment\_date", "unknown")

appointment\_time = event.get("appointment\_time", "unknown")

customer\_name = event.get("customer\_name", "Customer")

recipient\_email = event.get("recipient\_email", "admin@gmail.com")

# Compose the email subject and body

email\_subject = "Appointment Confirmation"

email\_body = f"Hi {customer\_name},\n\nYour appointment for {appointment\_date} at {appointment\_time} has been booked successfully."

# Compose the SMS message

sms\_message = f"Hi {customer\_name}, your appointment for {appointment\_date} at {appointment\_time} has been booked successfully."

# Send email

email\_response = send\_email(email\_subject, email\_body, recipient\_email)

# Send SMS

sms\_response = send\_sms(event["countrycode"] + event["phone\_number"], sms\_message)

logger.info("Email response: %s", email\_response)

logger.info("SMS response: %s", sms\_response)

return {

'statusCode': 200,

'body': 'Appointment confirmation email and SMS sent successfully'

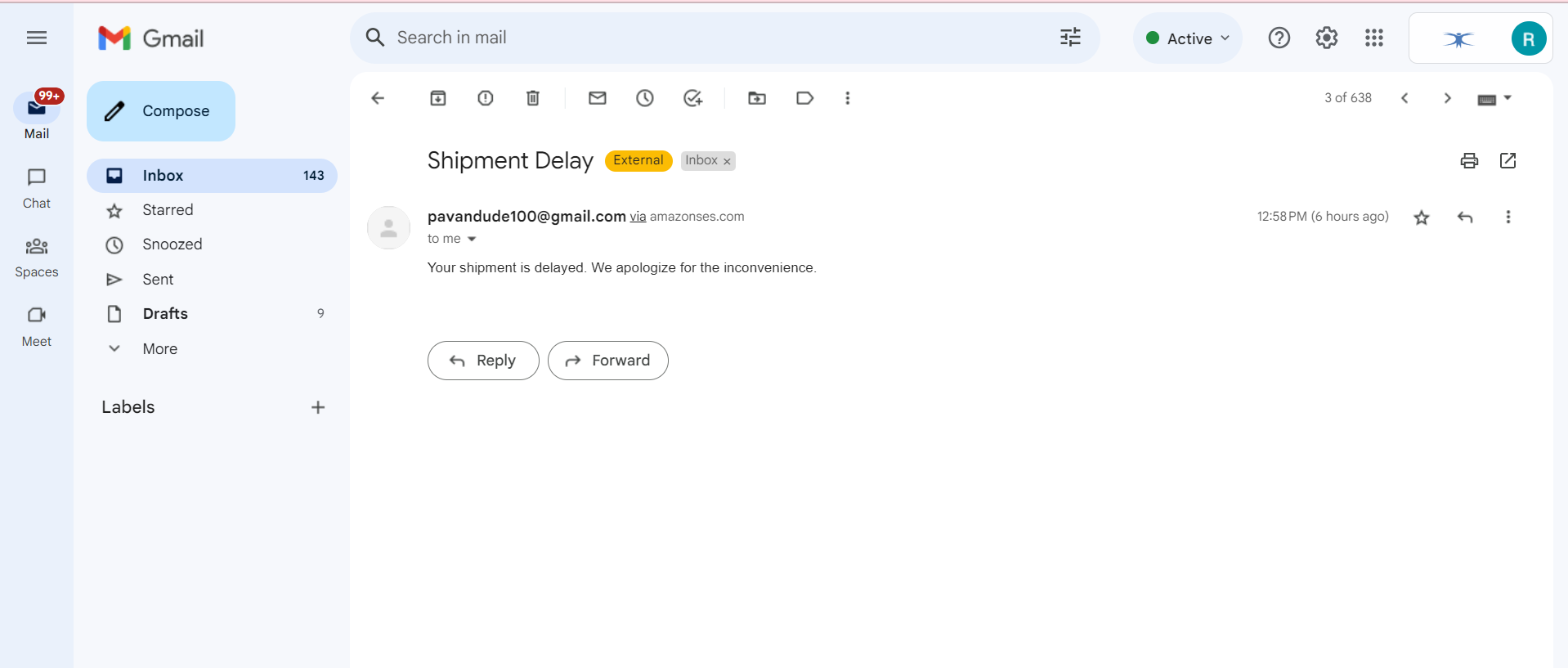
}

# 6.3: Testing:

A screenshot of a computer

Description automatically generated

**Fig 38: Testing with Postman**



**Fig 39: Mail received.**

# SPLASH SCREEN

# 7.1: Introduction:

* A splash screen is an initial screen that gets displayed right when the user launches the app before the main page loads.
* They also help let your users know when there's a loading delay due to a network issue or other error.

# 7.2: Implementation:

1. Creating Flutter Application**.**

A black screen with a black background

Description automatically generated

1. Add flutter lottie package in your terminal.

A screen shot of a computer

Description automatically generated

1. And now create a splash screen widget.

A computer screen shot of text

Description automatically generated

1. After creating the widget write the code for storing the lottie files.

A black background with white text

Description automatically generated

1. When the animation is complete navigate to the next page.

A screen shot of a computer

Description automatically generated

# ONBOARDING SCREEN

# 8.1: Introduction:

* An onboarding screen is a sequence of screens or tutorials that appear when a user opens a new application or website for the first time.
* The purpose of onboarding screens is to introduce the user to the features and functionality of the application, as well as to provide guidance on how to use it effectively. Onboarding screens can include welcome messages, product tours, feature highlights, and user account setup.

# 8.2: Implementation:

1. Add onboarding dependency

A black background with white text

Description automatically generated

2. Add path to the pubspec.yaml

A screenshot of a computer program

Description automatically generated

3. Create a widget of onboarding screen

A computer screen with text

Description automatically generated

4. Create Page Controllers

A computer screen with colorful text

Description automatically generated

5.Return the container with the images

A screen shot of a computer program

Description automatically generated