

# OTP Authentication System – Complete Project Documentation

---

## #Connection names

Lambda function name—**otp-auth-handler**

dynamo table name---- **otp\_user** **app\_client**----

[My web app - thh9r8](#) [api\\_name](#)---- [otp-auth-api](#)

## 1. Introduction

Connections Used: Streamlit → API Gateway → AWS Lambda → DynamoDB → Amazon SES

This document presents a detailed explanation of the OTP Authentication System built using AWS serverless services and a Streamlit frontend. The system ensures secure, time-bound authentication using One-Time Passwords (OTP) sent via email.

## 2. Problem Statement

Connections Used: User → Streamlit UI

Traditional password-based authentication systems are vulnerable to brute-force attacks, password leaks, and phishing. This project eliminates password dependency by implementing OTP-based authentication with limited attempts and expiry.

## 3. Overall System Architecture

Connections Used: Streamlit ↔ API Gateway ↔ Lambda ↔ DynamoDB & SES

The system follows a fully serverless architecture. Streamlit serves as the frontend, API Gateway exposes REST endpoints, AWS Lambda executes OTP logic, DynamoDB stores OTP records, and Amazon SES delivers OTP emails.

## 4. Streamlit Frontend

Connections Used: User Browser → Streamlit App → API Gateway

The Streamlit UI collects user email and OTP input. It sends POST requests to API Gateway and displays responses received from the backend. It handles both success and error messages gracefully.

## 5. API Gateway

Connections Used: Streamlit → API Gateway → Lambda

API Gateway acts as the secure entry point for the system. It routes HTTP POST requests to Lambda using proxy integration and handles CORS to allow browser-based access.

## 6. AWS Lambda Function

Connections Used: API Gateway → Lambda → DynamoDB / SES

The Lambda function is the core of the system. It generates OTPs, validates user input, enforces expiry and attempt limits, stores data in DynamoDB, and sends emails through Amazon SES.

## 7. DynamoDB Database

Connections Used: Lambda → DynamoDB

DynamoDB stores OTP details with email as the partition key. Attributes include OTP value, expiry timestamp, attempts count, status, and creation time. TTL automatically deletes expired OTP records.

## 8. DynamoDB TTL (Time To Live)

Connections Used: DynamoDB → Automatic Expiry Engine

TTL is configured on the otp\_expiry attribute. Once the expiry time is reached, DynamoDB automatically removes the item without manual cleanup, improving efficiency and security.

## 9. Amazon SES (Email Service)

Connections Used: Lambda → Amazon SES → User Email Inbox

Amazon SES is used to send OTP emails. In sandbox mode, both sender and receiver emails must be verified. SES ensures reliable email delivery with low latency.

## 10. IAM Roles & Permissions

Connections Used: Lambda Execution Role → DynamoDB & SES

The Lambda execution role grants permissions to DynamoDB (PutItem, GetItem, UpdateItem) and SES (SendEmail). Least-privilege access is followed for security.

## 11. OTP Workflow

Connections Used: User → Streamlit → API Gateway → Lambda → DynamoDB → SES

When a user requests an OTP, Lambda generates and stores it, then sends it via SES. During verification, Lambda validates OTP correctness, expiry, and attempt count before confirming authentication.

## 12. Security Measures

Connections Used: Lambda → DynamoDB TTL → IAM

Security features include OTP expiry, limited verification attempts, server-side validation, IAM access control, and removal of expired records.

## 13. Error Handling & Logging

Connections Used: Lambda → CloudWatch Logs

All errors are handled gracefully. Lambda logs important execution details to CloudWatch, enabling monitoring and debugging.

## 14. Testing Strategy

Connections Used: Streamlit UI / Lambda Test Events / DynamoDB Console

Testing includes sending OTPs, verifying correct and incorrect OTPs, checking expiry behavior, attempt limit enforcement, and validating database records.

## 15. Deployment Process

Connections Used: AWS Console → Lambda → API Gateway → Streamlit

Deployment involves creating DynamoDB tables, configuring SES, deploying Lambda, setting up API Gateway, and running the Streamlit frontend.

## 16. Limitations

Connections Used: SES Sandbox / Email-only OTP

The system currently supports email-based OTP only and is restricted by SES sandbox limitations until production access is granted.

## 17. Future Enhancements

Connections Used: SNS / JWT / MFA Integration

Planned improvements include SMS OTP via SNS, JWT-based session management, multifactor authentication, and admin dashboards.

## 18. Conclusion

Connections Used: Complete Serverless Authentication System

This project demonstrates a secure, scalable, and cost-effective OTP authentication system using AWS serverless technologies and Streamlit.

#DYNAMODB--IMG

The screenshot displays the AWS DynamoDB console interface. On the left, the navigation menu includes options like Dashboard, Tables, Explore items (selected), PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below this, the DAX section is visible with options for Clusters, Subnet groups, Parameter groups, and Events.

The main content area shows the 'Explore items' view for the 'otp\_users' table. It includes a 'Filter by tag value' dropdown, a search bar for tables, and a list of tables with 'otp\_users' selected. The 'Scan' button is active, and the 'Query' button is disabled. The 'Select a table or index' dropdown is set to 'Table - otp\_users', and the 'Select attribute projection' dropdown is set to 'All attributes'.

Below the selection area, there is a 'Filters - optional' section with a 'Run' button and a 'Reset' button. A green status bar indicates the scan is 'Completed' with 3 items returned, 3 items scanned, 100% efficiency, and 2 RCUs consumed.

The table 'otp\_users' is shown with 2 items returned. The scan started on December 16, 2025, at 16:07:34. The table has columns: email (String), attempts, created\_at, otp, otp\_expiry, and status. The items are as follows:

email (String)	attempts	created_at	otp	otp_expiry	status
deepakmandloi1706...	6	2025-12-16...	958840	1765882101	VERIFIED
adibaahmed842@gm...	6	2025-12-16...	643502	1765879797	VERIFIED

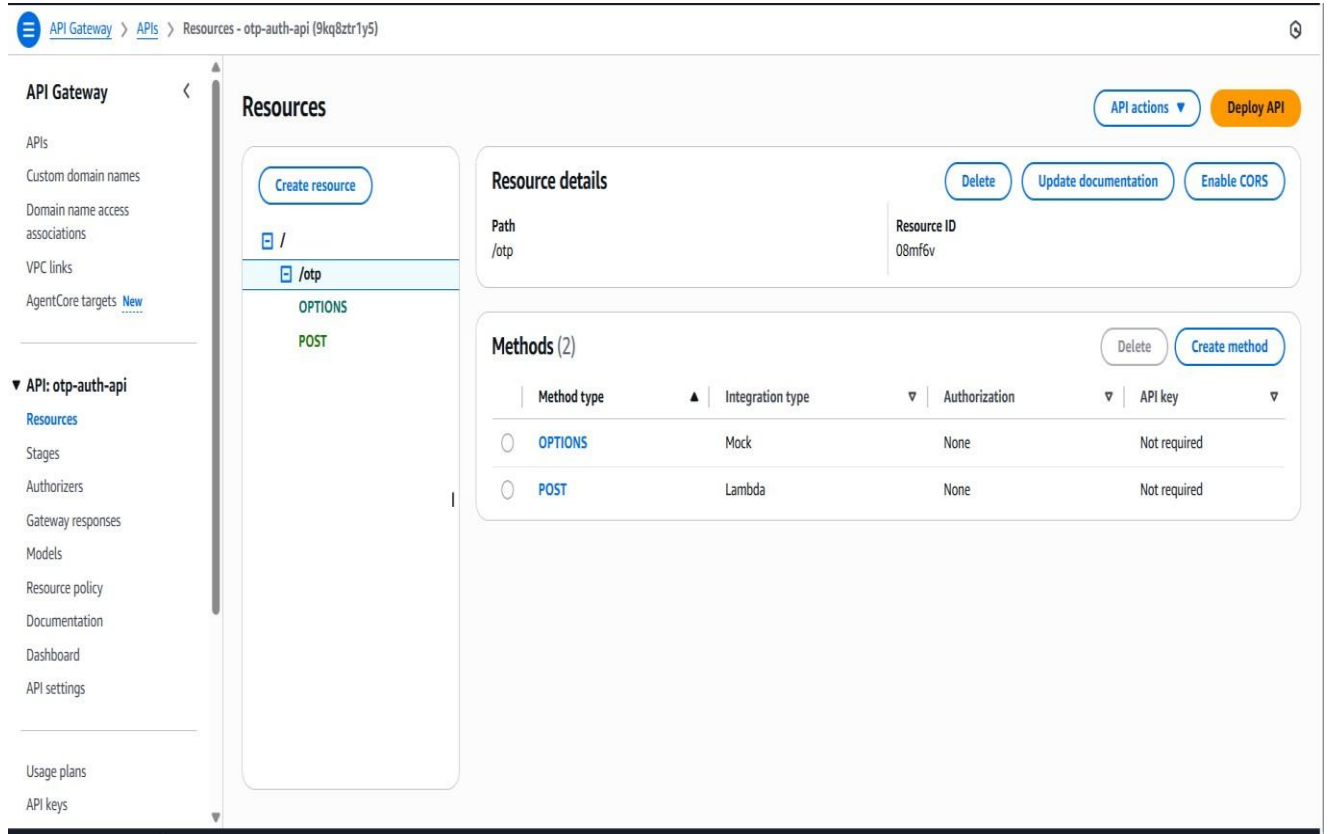
## #COGNITO—IMG

The screenshot displays the Amazon Cognito console interface. The breadcrumb navigation at the top reads: Amazon Cognito > User pools > User pool - thh9r8 > App clients > App client: My web app - thh9r8. On the left sidebar, the 'App clients' link under the 'Applications' section is highlighted. The main content area is titled 'App client: My web app - thh9r8' and includes 'Delete' and 'View login page' buttons. Below the title is the 'App client information' section, which contains the following details:

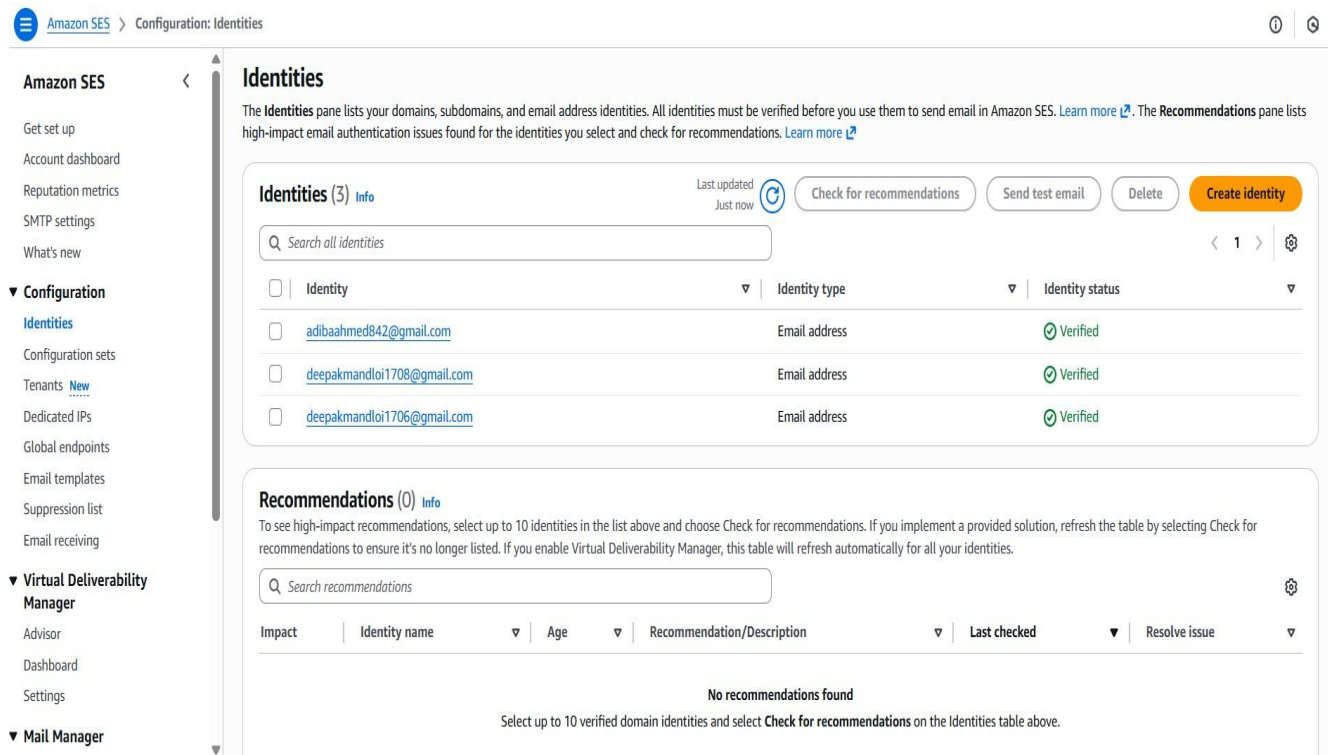
App client information		
<b>App client name</b> My web app - thh9r8	<b>Authentication flow session duration</b> 3 minutes	<b>Created time</b> December 15, 2025 at 17:09 GMT+5:30
<b>Client ID</b> 67amvpsb1j0ugtkm9nq47u5p05	<b>Refresh token expiration</b> 5 day(s)	<b>Last updated time</b> December 15, 2025 at 17:09 GMT+5:30
<b>Client secret</b> ***** <input type="checkbox"/> Show client secret	<b>Access token expiration</b> 60 minutes	
<b>Authentication flows</b> <input checked="" type="checkbox"/> Choice-based sign-in <input checked="" type="checkbox"/> Secure remote password (SRP) <input checked="" type="checkbox"/> Get user tokens from existing authenticated sessions	<b>ID token expiration</b> 60 minutes	
	<b>Advanced authentication settings</b> <input type="checkbox"/> Enable token revocation <input type="checkbox"/> Enable prevent user existence errors	

Below the information section are tabs for 'Quick setup guide', 'Attribute permissions', 'Login pages', 'Threat protection', and 'Analytics'. The 'Quick setup guide' tab is active, showing a section titled 'What's the development platform for your web application?' with four radio button options.

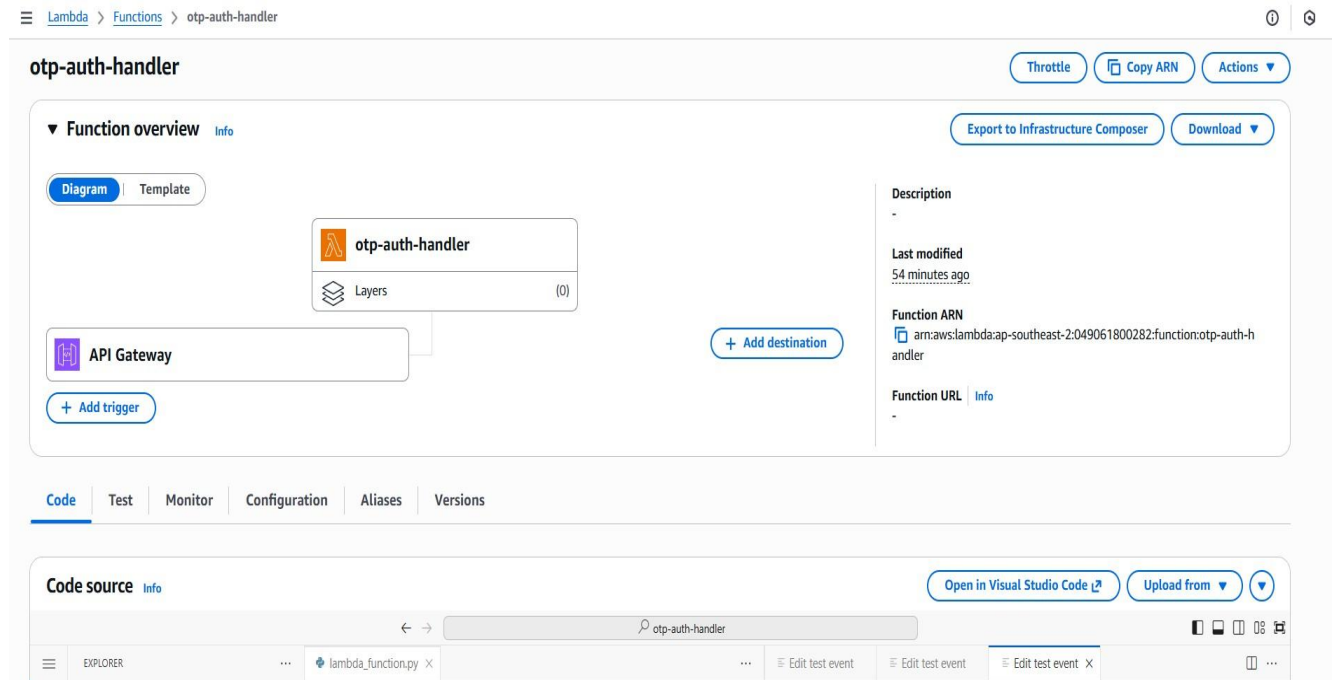
## #API\_GATEWAY---IMG



#SES--IMG



#LAMBDA---IMG



```
#APP.py
```

```
import streamlit as st
```

```
import requests
```

```
import json
```

```
API_URL = "https://9kq8ztr1y5.execute-api.ap-southeast-2.amazonaws.com/prod/otp"
```

```
st.set_page_config(page_title="OTP Authentication", layout="centered") st.title("OTP Authentication System")
```

```
email = st.text_input("Enter your email")
```

```

def show_response(r):
    try:
        data = r.json()    body =
    json.loads(data["body"])    msg =
    body.get("message", "No message")    except
    Exception:
        msg = r.text or "Invalid server response"

    if r.status_code == 200:
        st.success(msg)
    else:
        st.error(msg)
    # SEND OTP if st.button("Send OTP",
    disabled=not email):
        r = requests.post(API_URL, json={
            "action": "send_otp",
            "email": email
        })
        show_response(r)

    st.divider()

    otp = st.text_input("Enter OTP")

```



```
# VERIFY OTP if st.button("Verify OTP", disabled=not
(email and otp)):
    r = requests.post(API_URL, json={
        "action": "verify_otp",
        "email": email,
        "otp": otp
    })
    show_response(r)
```

---

## #LAMBDA FUNCTION

```
import json import boto3
import time import random
import re from datetime
import datetime

dynamodb = boto3.resource("dynamodb") table =
dynamodb.Table("otp_users") ses = boto3.client("ses",
region_name="ap-southeast-2")

SENDER_EMAIL = "deepakmandloi1708@gmail.com"
OTP_VALIDITY_SECONDS = 15 * 60
MAX_ATTEMPTS = 6
EMAIL_REGEX = r"^[^@]+@[^@]+\.[^@]+"
```

```

def generate_otp():
    return str(random.randint(100000, 999999))

def send_otp_email(email, otp):
    ses.send_email(
        Source=SENDER_EMAIL,
        Destination={"ToAddresses": [email]},
        Message={
            "Subject": {"Data": "Your OTP Code"},
            "Body": {"Text": {"Data": f"Your OTP is {otp}. Valid for 15 minutes."}}
        }
    )

def response(code, msg):
    return {
        "statusCode": code,
        "headers": {
            "Content-Type": "application/json",
            "Access-Control-Allow-Origin": "*"
        },
        "body": json.dumps({"message": msg})
    }

def lambda_handler(event, context):

```

```

    body = {}    if "body" in event and
event["body"]:

    try:

        body = json.loads(event["body"])
except Exception:

    return response(400, "Invalid JSON body")
else:

    body = event # fallback
    action = body.get("action") email
= body.get("email")    otp_input =
body.get("otp")

    if not action or not email:

        return response(400, "Action and email are required")

    if not re.match(EMAIL_REGEX, email):

        return response(400, "Invalid email format")

    if action == "send_otp":        otp = generate_otp()
expiry = int(time.time()) + OTP_VALIDITY_SECONDS

    table.put_item(

        Item={

            "email": email,

```

```

        "otp": otp,
        "otp_expiry": expiry, # TTL
        "attempts": 0,
        "status": "PENDING",
        "created_at": datetime.utcnow().isoformat()
    }
)
send_otp_email(email, otp) return
response(200, "OTP sent successfully")

if action == "verify_otp":

    if not otp_input:
        return response(400, "OTP is required")

    res = table.get_item(Key={"email": email})
    user = res.get("Item")

    if not user:
        return response(404, "OTP not found")

    if user["attempts"] >= MAX_ATTEMPTS:
        return response(403, "OTP attempts exceeded (6)")

```

```
    if int(time.time()) > user["otp_expiry"]:
return response(403, "OTP expired")
```

```
    if otp_input != user["otp"]:
table.update_item(
    Key={"email": email},
    UpdateExpression="SET attempts = attempts + :inc",
    ExpressionAttributeValues={":inc": 1}
)
return response(401, "Invalid OTP")
```

```
table.update_item(
    Key={"email": email},
    UpdateExpression="SET #s = :v",
    ExpressionAttributeNames={"#s": "status"},
    ExpressionAttributeValues={":v": "VERIFIED"}
)
```

```
return response(200, "OTP verified successfully")
```

```
return response(400, "Invalid action")
```

---

#JSON TEST EVENTS

```
test3--- {  
  "action": "send_otp",  
  "email": "deepakmandloi1708@gmail.com"  
}
```

```
Test4----  
{  
  "action": "send_otp",  
  "email": "deepakmandloi1706@gmail.com"  
}
```

```
Test5----  
{  
  "body":  
  "{ \"action\": \"send_otp\", \"email\": \"deeoakmandloi1708@gmail.com\" }"  
}
```