Cloud Based Automatic Mailing And User Authentication System

A PROJECT REPORT

In partial fulfilment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Under the guidance of

Mr. SOUVIK SARKAR

BY

ANUSREE DAS

CHANDRAMA SARKAR

AYASH HOSSAIN

ADRISH BANERJEE

RWITAM PANJA

team FUTURE
Education | Healthcare

FUTURE INSTITUTE OF ENGINEERING AND MANAGEMENT

In association with

ARDENT®
COMPUTECH PVT. LTD.

(ISO 9001:2015)

SDF Building, Module 132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

1. Title of the Project:	**Cloud Based Automatic Mailing And User Authentication System**
2. Project Members:	**ANUSREE DAS** **CHANDRAMA SARKAR** **AYASH HOSSAIN** **ADRISH BANERJEE** **RWITAM PANJA**
3. Name of the guide:	**Mr. SOUVIK SARKAR**
4. Address:	Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified) SDF Building, Module 132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal, 700091

Project Version Control History

Version	Primary Author	Description of Version	Date Completed
Final	ANUSREE DAS CHANDRAMA SARKAR AYASH HOSSAIN ADRISH BANERJEE RWITAM PANJA	Project Report	

Anusree Das
Chandrama Sarkar
Ayash Hossain
Adrish Banerjee
Rwitam Panja

Signature of Team Members

Date: 01/04/2022

Signature of Approver

Date:

Mr. SOUVIK SARKAR

Project Proposal Evaluator

For Office Use Only

Approved	**Not Approved**

DECLARATION

We hereby declare that the project work being presented in the project proposal entitled "**Cloud Based Automatic Mailing Using Authentication System**" in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY at ARDENT COMPUTECH PVT. LTD, SALTLAKE, KOLKATA, WEST BENGAL,** is an authentic work carried out under the guidance of **MR. SOUVIK SARKAR.** The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date: 01/04/2022

ARDENT COMPUTECH PVT. LTD (An ISO 9001:2015 Certified)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

CERTIFICATE

This is to certify that this proposal of minor project entitled

"CLOUD BASED AUTOMATIC MAILING AND USER AUTHENTICATION SYSTEM" is a record of bonafide work, carried out by, **ANUSREE DAS, CHANDRAMA SARKAR, AYASH HOSSAIN, ADRISH BANERJEE** and **RWITAM PANJA** under my guidance at **ARDENT COMPUTECH PVT LTD**. In my opinion, the report in its present form is in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** and as per regulations of the **ARDENT®.** To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

Guide / Supervisor

MR. SOUVIK SARKAR

Technical Head (Core Domain)

ARDENT COMPUTECH PVT. LTD. (An ISO 9001:2015 Certified)

SDF Building, Module 132, Ground Floor, Salt Lake City, GP Block, Sector V,
Kolkata, West Bengal 700091

ACKNOWLEDGEMENT

Success of any project depends largely on the encouragement and guidelines of many others. We take this sincere opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project work.

We would like to show our greatest appreciation to **Mr. SOUVIK SARKAR**, Technical Head (Core domain) at Ardent, Kolkata. We always feel motivated and encouraged every time by his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees, project assistants and other members at Ardent Computech Pvt. Ltd. for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

CONTENTS

TECHNOLOGIES USED

1. HTML
2. CSS
3. FLASK
4. PYTHON3
5. BOTO3
6. CLOUD BASED AWS SERVICES
7. AWS CLI VERSION 2

INTRODUCTION TO CLOUD COMPUTING

Cloud Computing is the provision of computing services such as storage, servers, databases, networking, software, analytics, intelligence, and more, over the Cloud or Internet.

Cloud Computing gives an alternative to the on-premises datacenter. With an on-premises datacenter, we have to manage everything, such as purchasing and installing hardware, virtualization, installing the operating system, and any other required applications, setting up the network, configuring the firewall, and setting up storage for data. After doing all the set-up, we become responsible for maintaining it through its entire lifecycle.

When we choose Cloud Computing, a cloud vendor is responsible for the hardware purchase and maintenance. It provides a wide variety of software and platform as a service. We can take any services that is required by us on rent. The cloud computing services will be charged based on usage.

The cloud environment provides an easily accessible online portal that makes handy for the user to manage the compute, storage, network, and application resources.

HISTORY OF CLOUD COMPUTING

The idea of cloud computing dates back to the 1950s and 1960s. However, the implementation of the cloud started gaining some real traction when IBM launched its virtual machines in the 1970s.

When IBM launched its first VM, users were able to have their own VM's running on hardware that was being maintained and managed by others. This was the beginning of a big breakthrough for cloud computing.

When telecommunication companies made the shift from point-to-point data connections to Virtual Private Networks in the 1990s, cloud computing really started to take off. With the computer boom of the 1980s, a lot of industries were looking for a way to connect all of their computers in-house where they would be able to access each other's shared data and VPNs allowed them to do so. With a decrease in price and an increase in service, you could tell that the cloud was about to take off. And in 1999 Salesforce.com became one of the pioneers in cloud computing by delivering enterprise applications via a simple website. The applications could be accessed by any customer with Internet access and companies were able to purchase the service on a cost-effective on-demand basis.

In 2008 Microsoft launched its cloud application platform Azure. Cloud applications enable people to share files, links, music, and videos on the internet. Because they are in the cloud, they don't necessarily have to consume storage space on the user's computer or device. These applications can be used by anyone with a web browser who can connect to the internet. With Microsoft Azure users are given the ability to manage their data, host websites, run SQL reports as well as much more.

MODERN DAY CLOUD COMPUTING

Over the past decade, cloud computing adoption has seen explosive growth – at both consumer and enterprise levels. Legacy software providers such as Microsoft, Oracle and Adobe have all made huge, concerted efforts to encourage users of their on-premises software offerings to upgrade to their cloud equivalents, which are usually offered on a subscription pay-as-you-go basis.

Over the course of the last ten years or so, cloud computing has evolved from being something that service providers told companies they should be adopting, to the very lifeblood that runs through most modern enterprises. As such, organizations have become increasingly accustomed to the pay-as-you-go cloud billing model, and now look upon IT purchases as a day-to-day expense, rather than a one-off investment that they will be stuck with for the foreseeable future.

Cloud computing has come a long way since those early days of mainframes and dumb terminals – and undoubtedly it has a long way still to go. Moving forward, the organizations they work for will face increasing challenges to not only remain competitive in this ever-changing cloud computing environment, but ensure they stay on the right side of both existing regulations and new ones as and when they emerge. It’s safe to say that the next decade of cloud computing will be just as eventful as the last.

ADVANTAGES OF CLOUD COMPUTING

1. Because the private cloud is on a remote server, your employees can share files and calendars with ease. This allows for better planning, better time management and an all-round more efficient way to run a business.
2. Cloud computing can be personalized according to the needs of the business. It can prevent freelancing to save worker costs. The cloud computing service provider already offers IT support to the company, so users don’t need to hassle with technical issues.
3. Cloud computing avoids the need for businesses to invest in stand-alone software as well as servers. Companies can avoid additional costs that are otherwise associated with licensing fees, data storage costs, costs for software updates as well as management using utilizing the cloud capabilities. The availability of one-time payments followed by the ability to pay while on the go is among some of the benefits that help save on costs.
4. Cloud computing service providers have maintained that the security layers around their databases are so stringent that hackers would find it difficult to penetrate even just the first tier.
5. Compared to a personal computer, cloud computing can store much more data and can be used for smaller, medium and large companies with an unlimited storage capacity offer.
6. The cloud computing software can be accessed at anytime from anywhere in the world and not only that but also it can be used with any device that has a browser and can have access to Wi-Fi.

FEW CLOUD COMPUTING PLATFORMS

1. MICROSOFT AZURE

   Azure has primarily been considered one of the best cloud services platforms out there. The wide array of included services is enough to fulfill the needs of any business in any industry. With Azure, we can run services on the cloud, or you can combine it with any of our existing infrastructures. Azure was released back in 2010, and since then, it has proven to be a solid option for companies looking to transform digitally.

2. AMAZON WEB SERVICES

   Amazon Web Services (AWS) is one of the most popular cloud computing platforms to build interactive web solutions for a business. It has a vast selection of IaaS and PaaS services, including Elastic Cloud Compute (EC2), Elastic Beanstalk, Simple Storage Service (S3), and Relational Database Service (RDS). AWS's infrastructure is highly customizable, making it possible to trim down costs by utilizing only the services one needs.

3. GOOGLE CLOUD
   Google Cloud is a reliable, user-friendly, security-oriented cloud computing service from one of today’s biggest tech giants. Although Google Cloud’s selection of services isn’t as vast as Azure’s, it is still enough to fulfill all of the IaaS and PaaS needs. User-friendliness and security are two of its headlines.

4. IBM CLOUD
   IBM Cloud is another cloud computing platform that primarily focuses on the following cloud computing services: IaaS (Infrastructure as a Service), SaaS (Software as a Service), and PaaS (Platform as a Service). It is one of the more budget-aligned pricing models in the market, plus it is fully customizable that trims down costs even further. It is super easy to create an account use their APIs.

TYPES OF CLOUD AND CLOUD COMPUTING SERVICES

There are four main types of cloud computing: private clouds, public clouds, hybrid clouds, and multiclouds.

1. PUBLIC CLOUDS

Public clouds are cloud environments typically created from IT infrastructure not owned by the end user. Some of the largest public cloud providers include Alibaba Cloud, Amazon Web Services (AWS), Google Cloud, IBM Cloud, and Microsoft Azure. Traditional public clouds always ran off-premises, but today's public cloud providers have started offering cloud services on clients' on-premise data centers. This has made location and ownership distinctions obsolete.

2. PRIVATE CLOUD

   Private clouds are loosely defined as cloud environments solely dedicated to a single end user or group, where the environment usually runs behind that user or group's firewall. All clouds become private clouds when the underlying IT infrastructure is dedicated to a single customer with completely isolated access.

3. HYBRID CLOUD

   A hybrid cloud is a seemingly single IT environment created from multiple environments connected through local area networks (LANs), wide area networks (WANs), virtual private networks (VPNs), and/or APIs.The characteristics of hybrid clouds are complex and the requirements can differ, depending on whom you ask.

4. MULTI CLOUD

   Multiclouds are a cloud approach made up of more than 1 cloud service, from more than 1 cloud vendor—public or private. All hybrid clouds are multiclouds, but not all multiclouds are hybrid clouds. Multiclouds become hybrid clouds when multiple clouds are connected by some form of integration or orchestration.

There three main types of cloud computing services: Infrastructure-as-a-Service (IaaS), Platforms-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

1. INFRASTRUCTURE-AS-A-SERVICE (IaaS)

IaaS means a cloud service provider manages the infrastructure for the user i.e., the actual servers, network, virtualization, and data storage, through an internet connection. The user has access through an API or dashboard, and essentially rents the infrastructure. The user manages things like the operating system, apps, and middleware while the provider takes care of any hardware, networking, hard drives, data storage, and servers; and has the responsibility of taking care of outages, repairs, and hardware issues. This is the typical deployment model of cloud storage providers.

2. PLATFORM-AS-A-SERVICE (PaaS)

PaaS means the hardware and an application-software platform are provided and managed by an outside cloud service provider, but the user handles the apps running on top of the platform and the data the app relies on. Primarily for developers and programmers, PaaS gives users a shared cloud platform for application development and management (an important DevOps component) without having to build and maintain the infrastructure usually associated with the process.

3. SOFTWARE-AS-A-SERVICE (SaaS)

SaaS is a service that delivers a software application—which the cloud service provider manages—to its users. Typically, SaaS apps are web applications or mobile apps that users can access via a web browser. Software updates, bug fixes, and other general software maintenance are taken care of for the user, and they connect to the cloud applications via a dashboard or API. SaaS also eliminates the need to have an app installed locally on each individual user's computer, allowing greater methods of group or team access to the software.

CLOUD COMPUTING USING AMAZON WEB SERVICES

The full form of AWS is Amazon Web Services. It is a platform that offers flexible, reliable, scalable, easy-to-use and, cost-effective cloud computing solutions.

AWS is a comprehensive, easy to use computing platform offered Amazon. The platform is developed with a combination of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offering.

In 2006, Amazon Web Services (AWS) started to offer IT services to the market in the form of web services, which is nowadays known as cloud computing. With this cloud, we need not plan for servers and other IT infrastructure which takes up much of time in advance. Instead, these services can instantly spin up hundreds or thousands of servers in minutes and deliver results faster. We pay only for what we use with no up-front expenses and no long-term commitments, which makes AWS cost efficient.

Today, AWS provides a highly reliable, scalable, low-cost infrastructure platform in the cloud that powers multitude of businesses in 190 countries around the world.
Amazon Web Services offers a wide range of different business purpose global cloud-based products. The products include storage, databases, analytics, networking, mobile, development tools, enterprise applications, with a pay-as-you-go pricing model.

AWS SERVICES USED: A BRIEF INTRODUCTION

AWS IAM ROLES

AWS Identity and Access Management (IAM) provides fine-grained access control across all of AWS. With IAM, we can specify who can access which services and resources, and under which conditions. With IAM policies, one can manage permissions to one's workforce and systems to ensure least-privilege permissions.
With IAM, we can manage AWS permissions for workforce users and workloads.

AWS SES

Amazon Simple Email Service (SES) is a cost-effective, flexible, and scalable email service that enables developers to send mail from within any application. We can configure Amazon SES quickly to support several emails uses cases, including transactional, marketing, or mass email communications. Amazon SES's flexible IP deployment and email authentication options help drive higher deliverability and protect sender reputation, while sending analytics measure the impact of each email. With Amazon SES, we can send email securely, globally, and at scale.

API GATEWAY

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services. Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, CORS support, authorization and access control, throttling, monitoring, and API version management. API Gateway has no minimum fees or startup costs. We pay for the API calls we receive and the amount of data transferred out and, with the API Gateway tiered pricing model, you can reduce your cost as your API usage scales.

AWS DYNAMODB

Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. DynamoDB offers built-in security, continuous backups, automated multi-Region replication, in-memory caching, and data export tools.

AWS LAMBDA SERVICES

AWS Lambda is a serverless, event-driven compute service that lets us run code for virtually any type of application or backend service without provisioning or managing servers. We can trigger Lambda from over 200 AWS services and software as a service (SaaS) application, and only pay for what you use. We use Amazon Simple Storage Service (Amazon S3) to trigger AWS Lambda data processing in real time after an upload, or connect to an existing Amazon EFS file system to enable massively parallel shared access for large-scale file processing.

BOTO3

Boto3 is the name of the Python SDK for AWS. It allows us to directly create, update, and delete AWS resources from your Python scripts. The AWS SDK for Python (Boto3) provides a Python API for AWS infrastructure services. Using the SDK for Python, we can build applications on top of Amazon S3, Amazon EC2, Amazon DynamoDB, and more.

CLOUD BASED AUTOMATIC MAILING AND USER AUTHENTICATION SYSTEM

PROJECT SUMMARY

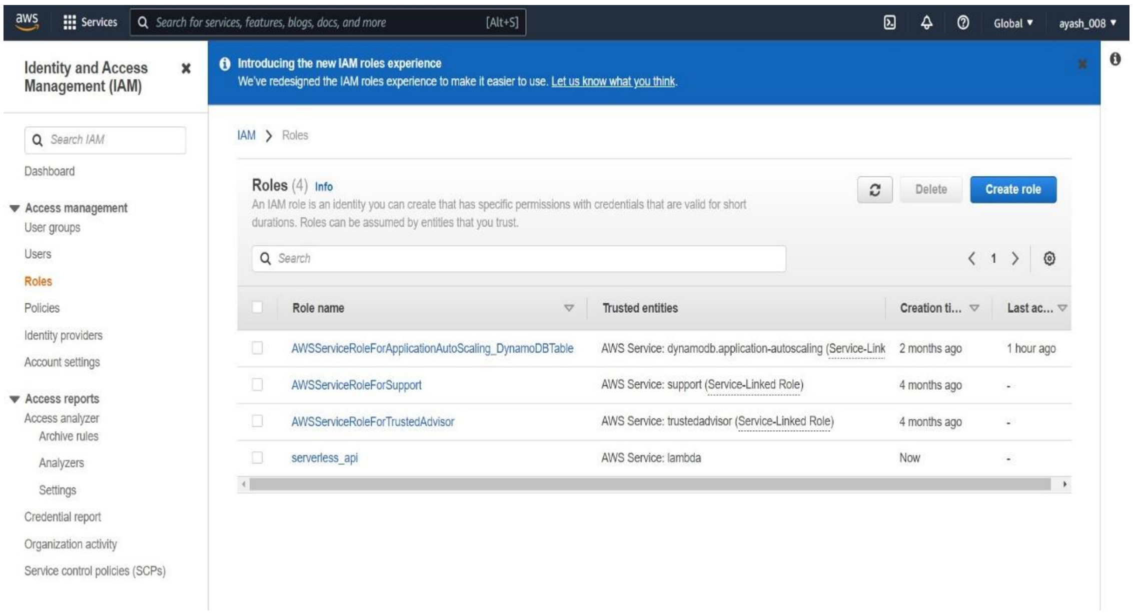
The project mainly focuses on making a user login and authentication system portal using the different AWS cloud computing services available to us.

A user creates an account in the website and gets registered. The registered user is sent an automated mail to his registered email ID using AWS lambda, SES and Boto3 services. The mail contains the user’s username and password. Once the registration process is completed, the user is allowed to sign in the site using the given credentials after successful authentication of the given credentials. The credentials are stored in DynamoDB. There is an option to reset password after successfully entering an autogenerated OTP, in case the user forgets his/her password. The new password is updated in the DynamoDB table.

WORKING

1. **Creating AWS IAM roles:**
   - Sign into the AWS Management Console as an administrator of the main Account.
   - Navigate to the IAM console.
   - In the navigation pane, choose roles.
   - Choose **create new Role.**
   - Type the names of the roles that are required.
   - Choose **Next Step** to attach policy that sets the permissions for those roles.
   - Under **Attach Polices**, choose the following roles,
     - AWSLambdaDynamoDBExecutionRole
     - AmazonSESFullAccess
   - Choose **Next Step** and review the roles.

- Click on **Create Role**.



Introducing the new IAM roles experience
We've redesigned the IAM roles experience to make it easier to use. Let us know what you think.

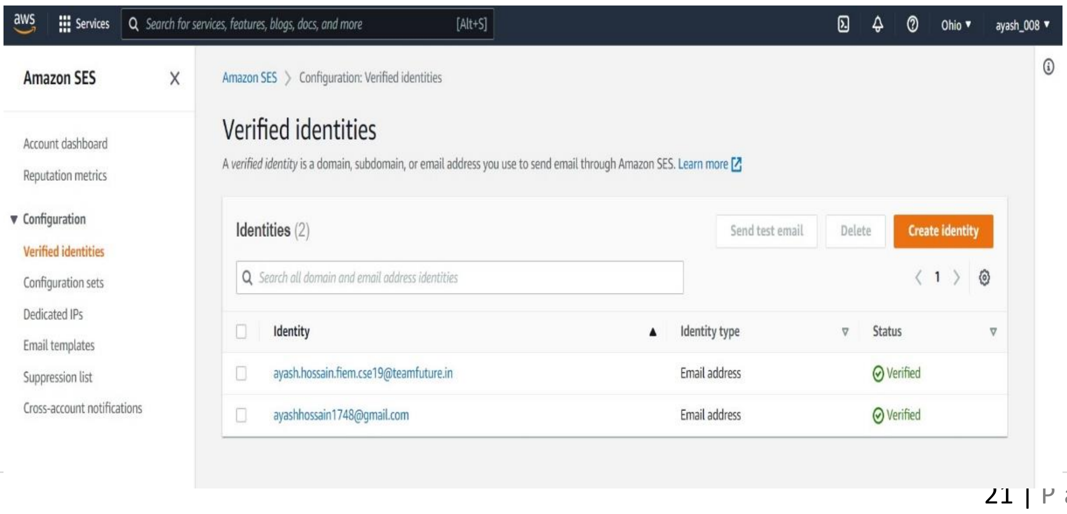
IAM > Roles

Roles (4) Info
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Creation ti...	Last ac...
AWSServiceRoleForApplicationAutoScaling_DynamoDBTable	AWS Service: dynamodb.application-autoscaling (Service-Link	2 months ago	1 hour ago
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	4 months ago	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	4 months ago	-
serverless_api	AWS Service: lambda	Now	-

2. Identity Verification using AWS SES:

- Sign In to the AWS Management Console and open the Amazon SES control.
- In the navigation pane, under Configuration, choose **Verified Identities.**
- Choose **Create Identity**.
- Under Identity details, choose **Email Address** as the identity type you want to create.
- Enter the email address you want to verify.
- Choose **Create Identity**. After it is successfully created, we receive a verification email within five-six minutes. Verify the email address by following the mentioned instructions.



Amazon SES > Configuration: Verified identities

Verified identities

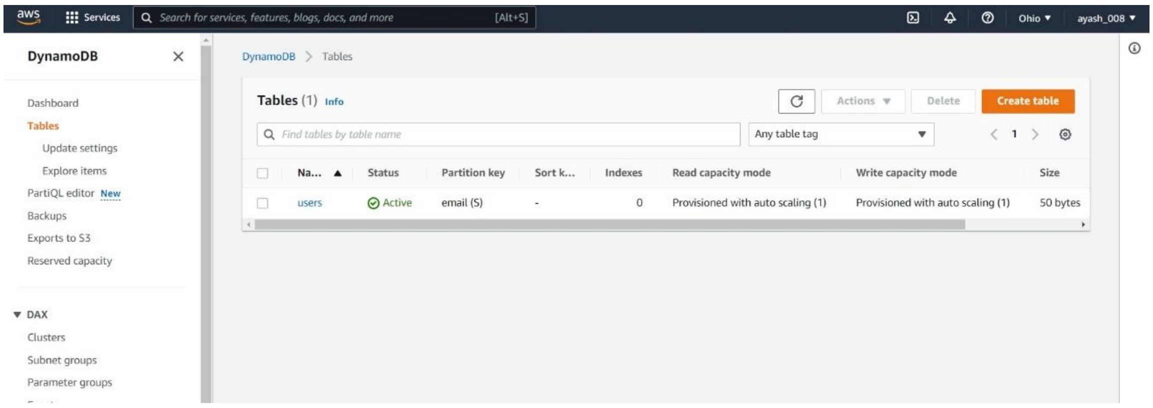
A verified identity is a domain, subdomain, or email address you use to send email through Amazon SES. Learn more

Identities (2)

Identity	Identity type	Status
ayash.hossain.fiem.cse19@teamfuture.in	Email address	Verified
ayashhossain1748@gmail.com	Email address	Verified

3. Creating an empty table in DynamoDB.

- Open the DynamoDB console.
- In the Create DynamoDB table screen, Enter the table name.
- Adjust the necessary settings and click in create table.


DynamoDB > Tables

Tables (1) Info

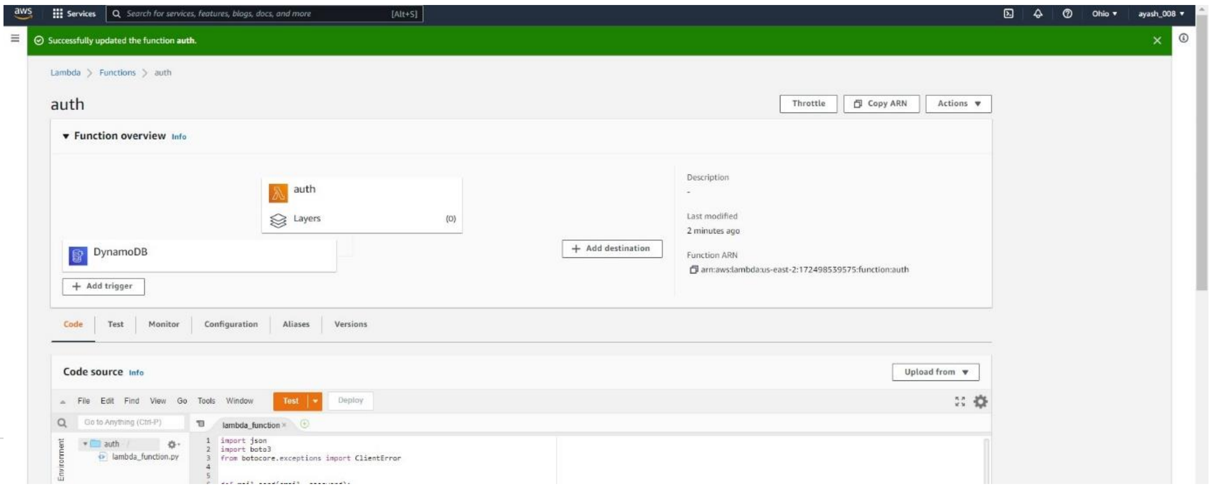
	Na...	Status	Partition key	Sort k...	Indexes	Read capacity mode	Write capacity mode	Size
☐	users	Active	email (S)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	50 bytes

4. Configuring AWS Lambda for SES:

- Open the Lambda console from the service navigation panel.
- Choose Create Function.
- Choose function name.
- For Runtime, confirm that Python3 is selected.
- Choose Create Function.
- Click on Add Trigger and choose the appropriate DynamoDB table.

  Lambda creates a Python3 function and an execution role that grants the function permission to upload logs. The Lambda function assumes the execution role when we invoke our function, and uses the execution role to create credentials for the AWS SDK and to read data from event sources.

  (We use boto3 to send Email using SES and lambda).


Successfully updated the function auth.

Lambda > Functions > auth

auth

Function overview Info

auth
Layers (0)
DynamoDB

Description
-

Last modified
2 minutes ago

Function ARN
arn:aws:lambda:us-east-2:172498539575:function:auth

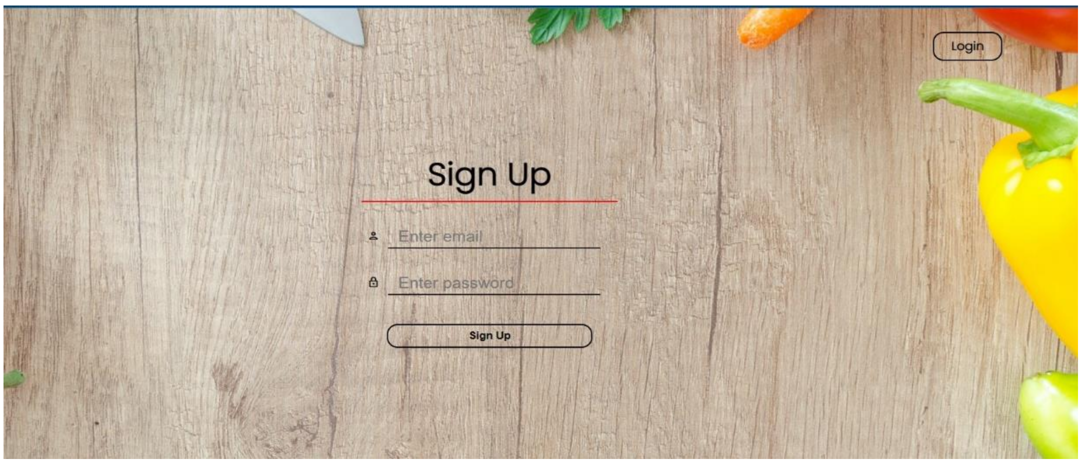
Code source Info

```
import json
import boto3
from botocore.exceptions import ClientError
```

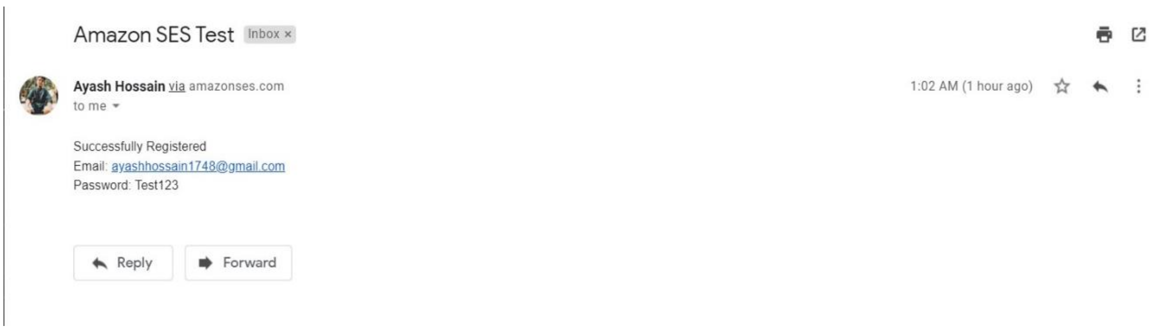
- Write the below Boto3 Source Code in the “Code source” section.

```
import json
import boto3
from botocore.exceptions import ClientError


def mail_send(email, password):
    print("mail send")
    SENDER = "Ayash Hossain <ayash.hossain.fiem.cse19@teamfuture.in>"
    RECIPIENT = email

    AWS_REGION = "us-east-2"
    SUBJECT = "Amazon SES Test"
    BODY_TEXT = ("Amazon SES Test\r\n"
                 "This email was sent from our website "
                )

    BODY_HTML = """
                <html>
                <head></head>
                <body>
                    <p>Successfully Registered
                    <br>
                    Email: """ + str(email) + """ <br>
                    Password: """ + str(password) + """
                    </p>
                </body>
                </html>
                """

    CHARSET = "UTF-8"
    client = boto3.client('ses', region_name=AWS_REGION)
    try:
        response = client.send_email(
            Destination={
                'ToAddresses': [
                    RECIPIENT,
                ],
            },
            Message={
                'Body': {
                    'Html': {
                        'Charset': CHARSET,
                        'Data': BODY_HTML,
                    },
                    'Text': {
                        'Charset': CHARSET,
                        'Data': BODY_TEXT,
                    },
                },
                'Subject': {
                    'Charset': CHARSET,
                    'Data': SUBJECT,
                },
            },
            Source=SENDER,
        )
    except ClientError as e:
        print(e.response['Error']['Message'])
    else:
        print("Email sent! Message ID:"),
        print(response['MessageId'])


def lambda_handler(event, context):
       try:
               for record in event['Records']:

                      if record['eventName'] == 'INSERT':
                              handle_insert(record)

               return "Success!"
       except Exception as e:
               print(e)
               return "Error"


def handle_insert(record):
       print("Handling INSERT Event")
       newImage = record['dynamodb']['NewImage']
       email = newImage['email']['S']
       password = newImage['password']['S']
    print("before mail send")
       mail_send(email, password)
```

5. **User Registration**
   - The user enters the Email Id and sets the password and clicks on Sign Up.



   - An automatic mail is sent to the user which has the credential details required for future login to the site.



Amazon SES Test

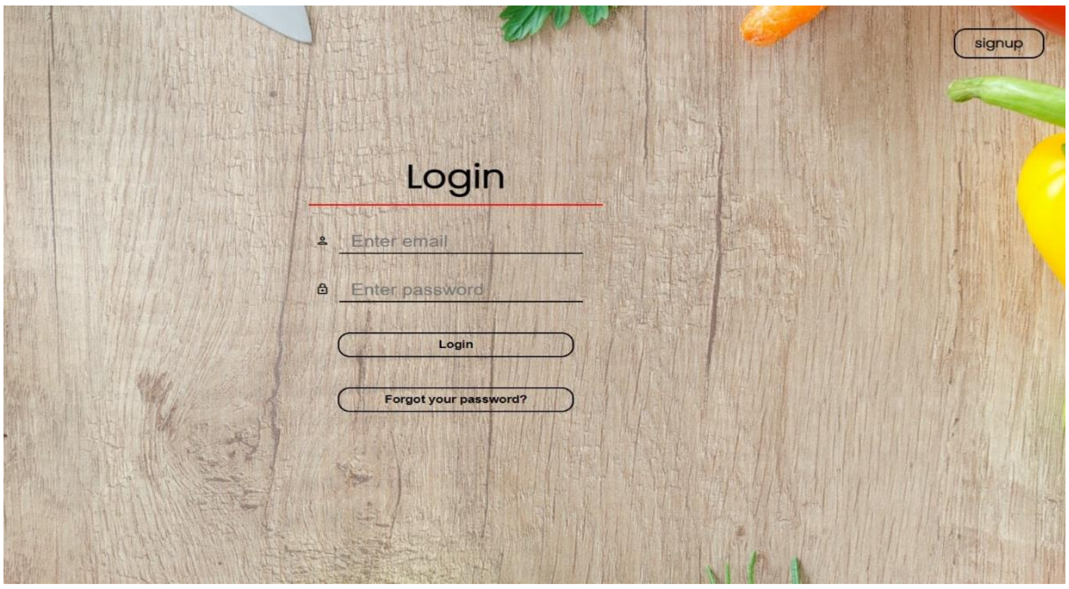
Ayash Hossain via amazonses.com
to me

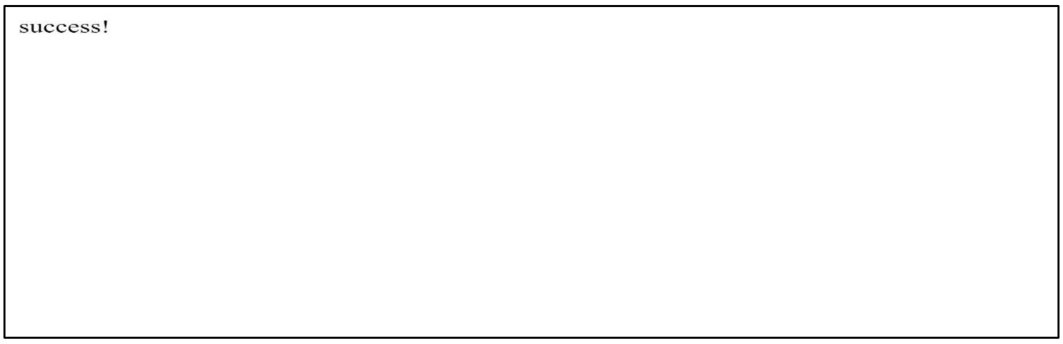
1:02 AM (1 hour ago)

Successfully Registered
Email: ayashhossain1748@gmail.com
Password: Test123

6. User Login

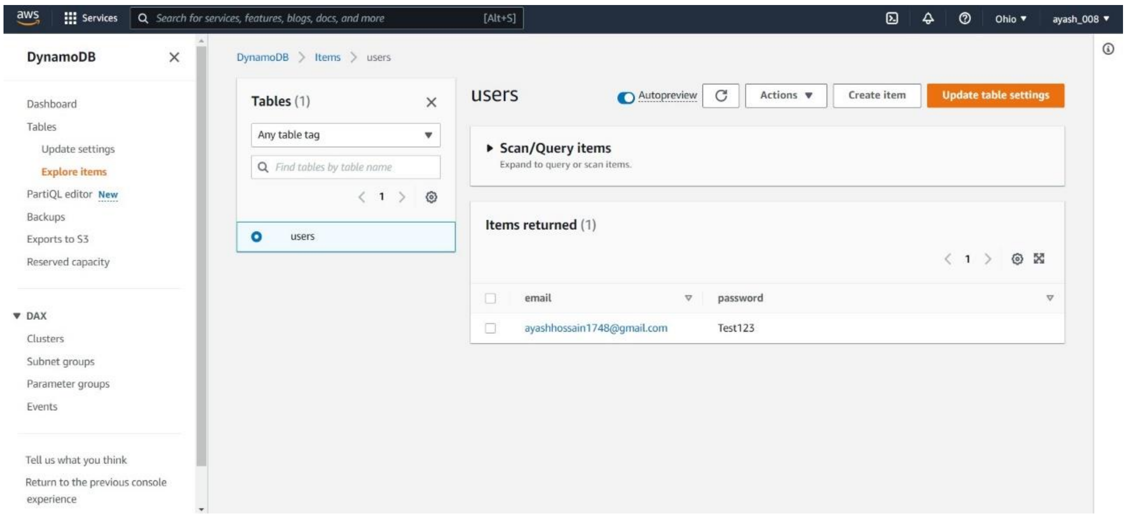
- The user can now successfully login to the site using the given credentials.





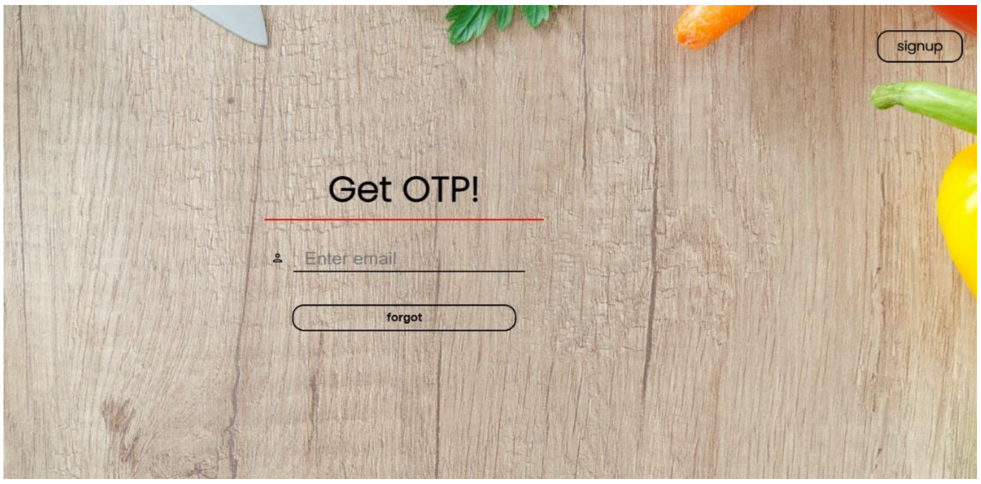
```
success!
```

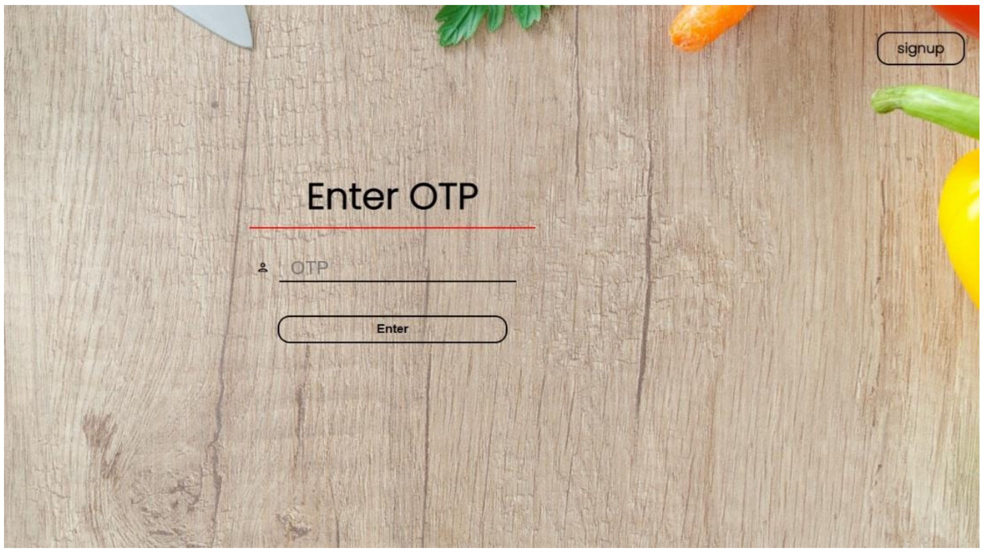
- The data is successfully stored in the DynamoDB table.

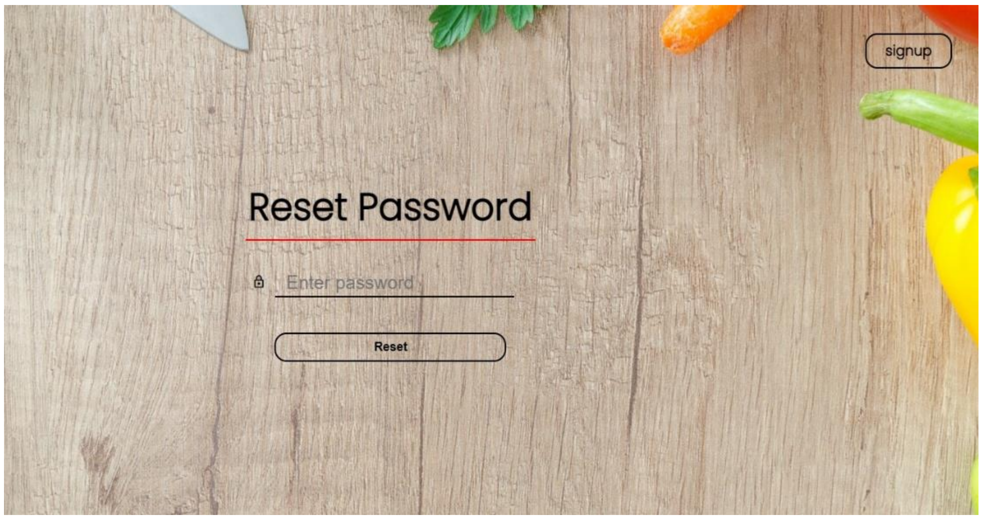


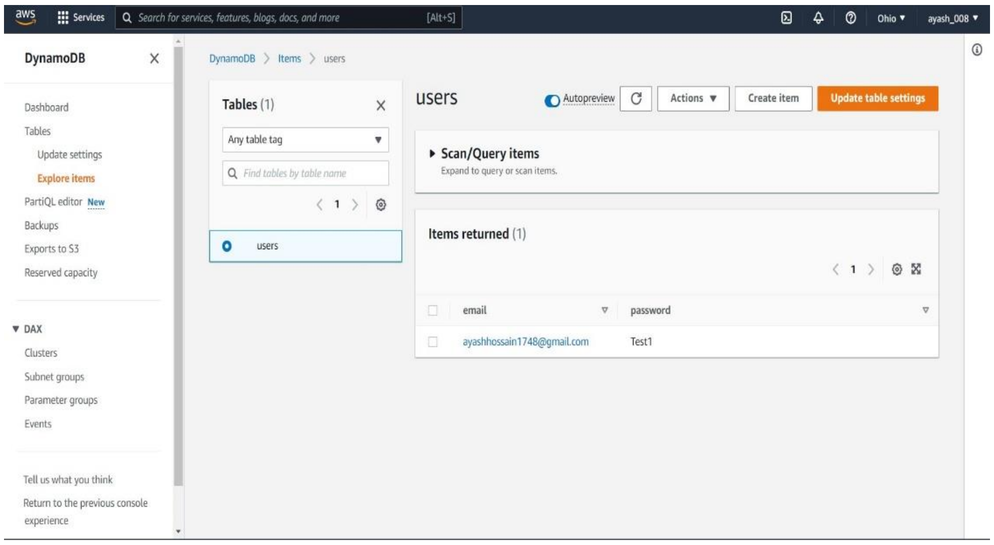
email	password
ayashhossain1748@gmail.com	Test123

- When the user clicks on “Forgot your password?”, an automatic mail is again sent to the registered Email Id which contains an OTP. The user enters the OTP in the form and is given permission to reset the password. The new password is then updated in the DynamoDB table.









users

Items returned (1)

email	password
ayashhossain1748@gmail.com	Test1

CODES

application.py

```
from flask import Flask, render_template, request
import key_config as keys
import boto3
import random
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError
OTP = 0
email_glo = ""
application = app = Flask(__name__)


dynamodb = boto3.resource('dynamodb',
                          aws_access_key_id=keys.ACCESS_KEY_ID,
                          aws_secret_access_key=keys.ACCESS_SECRET_KEY)

# for update_item in dynamodb table
dynamodb_client = boto3.client('dynamodb')


@app.route('/')
def index():
    return render_template('index1.html')


@app.route('/signup', methods=['post'])
def signup():
    if request.method == 'POST':
        email = request.form['mail']
        password = request.form['pass']
        try:
            if("@" in email and password):

                table = dynamodb.Table('users')

                table.put_item(
                    Item={
                        'email': email,
                        'password': password
                    },
                    ConditionExpression='attribute_not_exists(email)'
                )
                return render_template('login1.html')
        except:
            return render_template('index1.html')
    return render_template('index1.html')


@app.route('/login', methods=['post'])
def login():
    return render_template('login1.html')
```

```
@app.route('/check', methods=['post'])
def check():
    if request.method == 'POST':
        table = dynamodb.Table('users')
        email = request.form['mail']
        password = request.form['pass']
        try:
            response = table.query(
                KeyConditionExpression=Key('email').eq(email)
            )
            items = response['Items']
            if password == items[0]['password']:
                return "success!"
            return render_template("login1.html")
        except:
            return render_template("login1.html")


@app.route('/forgot', methods=['post'])
def forgot():
    return render_template("forgot.html")


@app.route('/checkforgot', methods=['post'])
def checkforgot():
    if request.method == 'POST':
        table = dynamodb.Table('users')
        email = request.form['mail']
        global email_glo
        email_glo = email
        try:
            response = table.query(
                KeyConditionExpression=Key('email').eq(email)
            )
            global OTP
            OTP = random.randint(1000, 9999)
            mail_send(email, OTP)
            return render_template("enterotp.html")
        except:
            return render_template("forgot.html")


@app.route('/otpcheck', methods=['post'])
def otpcheck():
    if request.method == 'POST':
        try:
            user_otp = request.form['otp']
            print(user_otp)
            print(type(user_otp))
            print(OTP)
            if(OTP == int(user_otp)):
                print("yes")
                return render_template('resetpass.html')
        except:
            return render_template('enterotp.html')

    return render_template('enterotp.html')
```

```
@app.route('/resetpassword', methods=['post'])
def resetpassword():
    if request.method == 'POST':
        email = email_glo
        password = request.form['pass']
        if(password):
            key = {
                'email': {'S': email}
            }

            response = dynamodb_client.update_item(
                TableName='users',
                Key=key,
                ExpressionAttributeNames={
                    '#ps': 'password'
                },
                UpdateExpression="set #ps = :pass",
                ExpressionAttributeValues={
                    ':pass': {
                        'S': password
                    }
                }
            )
            return render_template('login1.html')
    return render_template('resetpass.html')


def mail_send(email, OTP):

    SENDER = "Ayash Hossain <ayash.hossain.fiem.cse19@teamfuture.in>"
    RECIPIENT = email

    AWS_REGION = "us-east-2"
    SUBJECT = "Amazon SES Test"
    BODY_TEXT = ("Amazon SES Test\r\n"
                 "This email was sent from our website "
                )

    BODY_HTML = """
                <html>
                <head></head>
                <body>
                    <p>
                    OTP: """ + str(OTP) + """
                    </p>
                </body>
                </html>
                """

    CHARSET = "UTF-8"

    client = boto3.client('ses', region_name=AWS_REGION)
    try:
        response = client.send_email(
            Destination={
                'ToAddresses': [
                    RECIPIENT,
                ],
            },
            Message={
                'Body': {
                    'Html': {
                        'Charset': CHARSET,
                        'Data': BODY_HTML,
                    },
                    'Text': {
                        'Charset': CHARSET,
                        'Data': BODY_TEXT,
                    },
                },
                'Subject': {
                    'Charset': CHARSET,
                    'Data': SUBJECT,
                },
            },
            Source=SENDER,
        )
    except ClientError as e:
        print(e.response['Error']['Message'])
    else:
        print("Email sent! Message ID:"),
        # print(response['MessageId'])


if __name__ == "__main__":

    app.run(debug=True)
```

key_config.py

```
ACCESS_KEY_ID = 'AKIASQKNW2Q3TW55K2MU'

ACCESS_SECRET_KEY = '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'
```

Boto3 Lambda Function

```
import json
import boto3
from botocore.exceptions import ClientError


def mail_send(email, password):
    print("mail send")
    SENDER = "Ayash Hossain <ayash.hossain.fiem.cse19@teamfuture.in>"
    RECIPIENT = email

    AWS_REGION = "us-east-2"
    SUBJECT = "Amazon SES Test"
    BODY_TEXT = ("Amazon SES Test\r\n"
                 "This email was sent from our website "
                 )

    BODY_HTML = """
                <html>
                <head></head>
                <body>
                    <p>Successfully Registered
                    <br>
                    Email: """ + str(email) + """ <br>
                    Password: """ + str(password) + """
                    </p>
                </body>
                </html>
                """

    CHARSET = "UTF-8"

    client = boto3.client('ses', region_name=AWS_REGION)
    try:
        response = client.send_email(
            Destination={
                'ToAddresses': [
                    RECIPIENT,
                ],
            },
            Message={
                'Body': {
                    'Html': {
                        'Charset': CHARSET,
                        'Data': BODY_HTML,
                    },
                    'Text': {
                        'Charset': CHARSET,
                        'Data': BODY_TEXT,
                    },
                },
                'Subject': {
                    'Charset': CHARSET,
                    'Data': SUBJECT,
                },
            },
            Source=SENDER,
        )
    except ClientError as e:
        print(e.response['Error']['Message'])
    else:
        print("Email sent! Message ID:"),
        print(response['MessageId'])


def lambda_handler(event, context):
        try:
                for record in event['Records']:

                        if record['eventName'] == 'INSERT':
                                handle_insert(record)

                return "Success!"
        except Exception as e:
                print(e)
                return "Error"


def handle_insert(record):
        print("Handling INSERT Event")
        newImage = record['dynamodb']['NewImage']
        email = newImage['email']['S']
        password = newImage['password']['S']
    print("before mail send")
        mail_send(email, password)
```

CONCLUSION

The project has been appreciated by all the users in the organization. It is easy to use, since it uses the AWS provided in the user dialog. User friendly screens are provided. The usage of software increases the efficiency, decreases the effort. It has been efficiently employed as a Site Management, Registration and Authentication mechanism. It has been thoroughly tested and implemented.

The software collects the information of the user to provide a secure gateway for the user's identity. The software provides a reliable platform for keeping all sensitive information.

REFERENCES

1. https://www.tutorialspoint.com
2. https://aws.amazon.com
3. https://www.youtube.com
4. https://javatpoint.com