SMARTBRIDGE

The Virtual Career Counselor: Harnessing Generative AI and AWS for Personalized Pathways

Project Description:

The old way of career counseling with an increase in demand for personal guidance in a rapidly changing market tends to be less reliable and fruitful due to high costs and limited guidance. My project the “Virtual Career Counselor” address these career related doubts by harnessing Generative AI and use AWS for personalized Pathways. This platform allows students to signup, login and interact by generating career roadmaps and skill gap analysis based on their current skill set and profile. The project uses Flask for backend development, is hosted on AWS EC2 instance and integrated with Amazon DynamoDB for dynamic data management. The Groq Cloud AI API has been used for generating the guidance results regarding the doubts raised by the students related to their career. The AWS SNS sends email notifications from email address name “CareerAI” upon signup as a student or roadmap generation. This in turns increases as well as enhances user engagement with their support. This project not only aims to produce results as mere outputs but also explores various career streams ensuring good, qualitative and fast as well as intelligent access to professional growth for all.

Scenario 1: AI-Powered Career Roadmap Generation

The AWS EC2 provides a reliable infrastructure in the “Virtual Career Counselor” of handling complex AI queries from multiple usersat the same time. For example, a student can log in, navigate to the "setup-goal" page, and input their academic background and professional interests. The system then calls the Groq Cloud AI API to analyze this data and generate a multiple step career roadmap—all in real-time. Flask manages the backend processes, ensuring that the AI’s response is formatted into a clear, actionable plan that helps the user understand exactly which skills or certifications they need next.

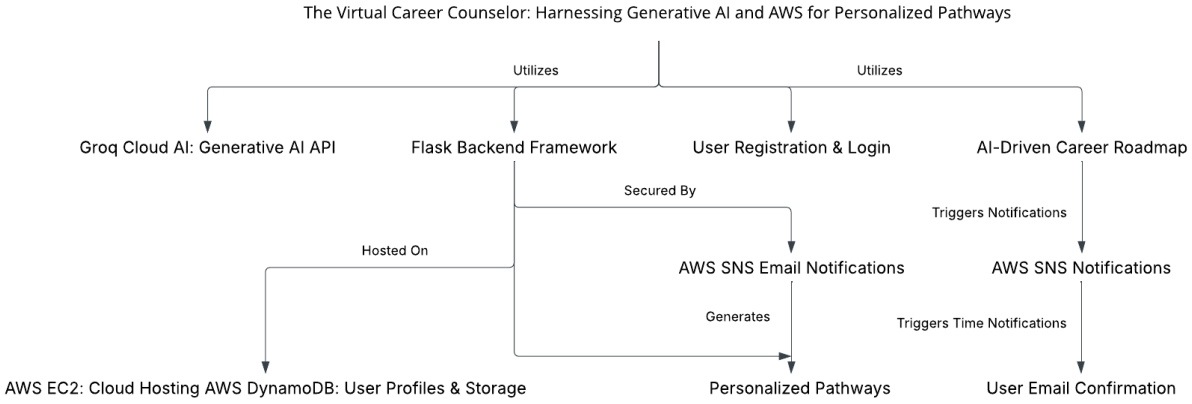
Scenario 2: Seamless Progress and Registration Notifications

When a user registers or saves a new career path, the “Virtual Career Counselor” leverages AWS SNS to send instant email notifications from email address name “CareerAI” to confirm their progress. For instance, once a user completes their profile, Flask processes the data, and SNS sends a customized email with their login credentials or a summary of their newly generated career roadmap. This real-time notification system reduces uncertainty and keeps users motivated, while DynamoDB securely stores their profile and saved roadmaps, allowing them to pick up exactly where they left off during their next session.

SMARTBRIDGE

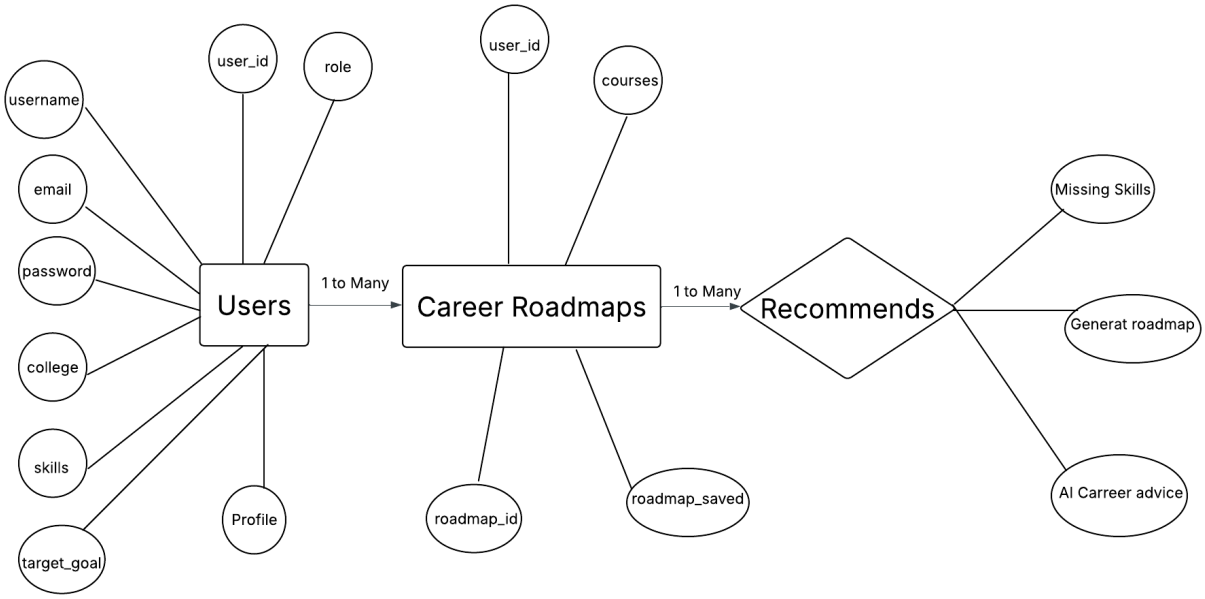
Scenario 3: Scalable Data Access and Profile Management

The “Virtual Career Counselor” offers users a seamless interface to manage their professional profiles and track their growth over time. After logging in, a user can view their saved career goals, skills, and previously generated AI roadmaps. Flask dynamically fetches this information from DynamoDB, ensuring that user data is updated instantly whenever they gain a new skill or change their career focus. Meanwhile, the EC2-hosted application remains stable and responsive even as more students join the platform, providing an uninterrupted and personalized experience that adapts to each individual's unique journey.



AWS Architecture

- Entity Relationship (ER) Diagram:



Pre-requisites:

1. AWS Account Setup: AWS Account Setup
2. Understanding IAM: IAM Overview
3. Amazon EC2 Basics: EC2 Tutorial
4. DynamoDB Basics: DynamoDB Introduction
5. SNS Overview: SNS Documentation
6. Git Version Control: Git Documentation

SMARTBRIDGE

Technical Prerequisites:

1. AWS Infrastructure & Security

- AWS Account Access: An active AWS account to manage cloud resources.
- IAM Role Configuration: A custom IAM Role (e.g., custom_user_role) with AmazonDynamoDBFullAccess and AmazonSNSFullAccess policies attached to your EC2 instance.
- Key Pair Management: An RSA .pem key (e.g., career-key.pem) for secure SSH/SCP access to your server.

2. Development Environment

- Python & Package Manager: Python 3.x installed along with pip3 for managing dependencies.
- Core Libraries: Installation of flask (web framework), boto3 (AWS SDK for Python), and groq (AI API client).
- Code Editor: VS Code for local development and debugging before cloud deployment.

3. Database & Messaging Setup

- Amazon DynamoDB: A NoSQL table named Users created to store student profiles and credentials.
- Amazon SNS: A standard SNS Topic (e.g., CareerNotifications) with an active email subscription for automated alerts.

4. Generative AI Integration

- Groq Cloud API Key: A valid API key to access high-performance LLMs like Llama-3.3-70b-versatile for roadmap generation.

- ## Project Workflow:

1. AWS Account Setup and Login

- Activity 1.1: Set up an AWS account if not already done.
- Activity 1.2: Log in to the AWS Management Console

2. DynamoDB Database Creation and Setup

- Activity 2.1: Create a DynamoDB table named usertable to store user data.
- Activity 2.2: Define and configure attributes in usertable.

3. Backend Development and Application Setup

- Activity 3.1: Develop the Flask backend to handle core application functionalities
- Activity 3.2: Configure API integration with the Groq API to support Generative AI-powered responses.

4. Generative AI Model Integration

- Activity 4.1: Configure Groq API endpoints within the Flask application for three core functionalities:
    - Course Recommendations: Generate a list of personalized courses based on user preferences.
    - Career Path Generation: Generate detailed career paths
    - Job Market Trends: Retrieve up-to-date insights

5. IAM Role Setup

- Activity 5.1: Create IAM roles to control access to AWS resources, ensuring secure connections to DynamoDB and Groq API.
- Activity 5.2: Attach appropriate policies for resource access, ensuring least privilege access for enhanced security.

6. Application Routes and Core Functionalities Development

- Activity 6.1: Establish routes in the Flask app to serve the following functionalities:
    - Home (/): Display the landing page
    - Registration (/register): Implement user registration
    - Login (/login): Authenticate users and initiate sessions for personalized services.

- Logout (/logout): End user sessions and ensure data protection upon logout.
- Counsel (/counsel): Display an interactive page
- Generate Recommendations (/generate_recommendations): Use Groq API to generate a list of recommended courses based on user preferences.
- Career Path Generation (/generate_career_path): Create detailed career paths based on specific career interests.
- Job Market Trends (/job_market_trends): Provide real-time job market insights to help users make informed career decisions.

7. EC2 Instance Setup and Deployment

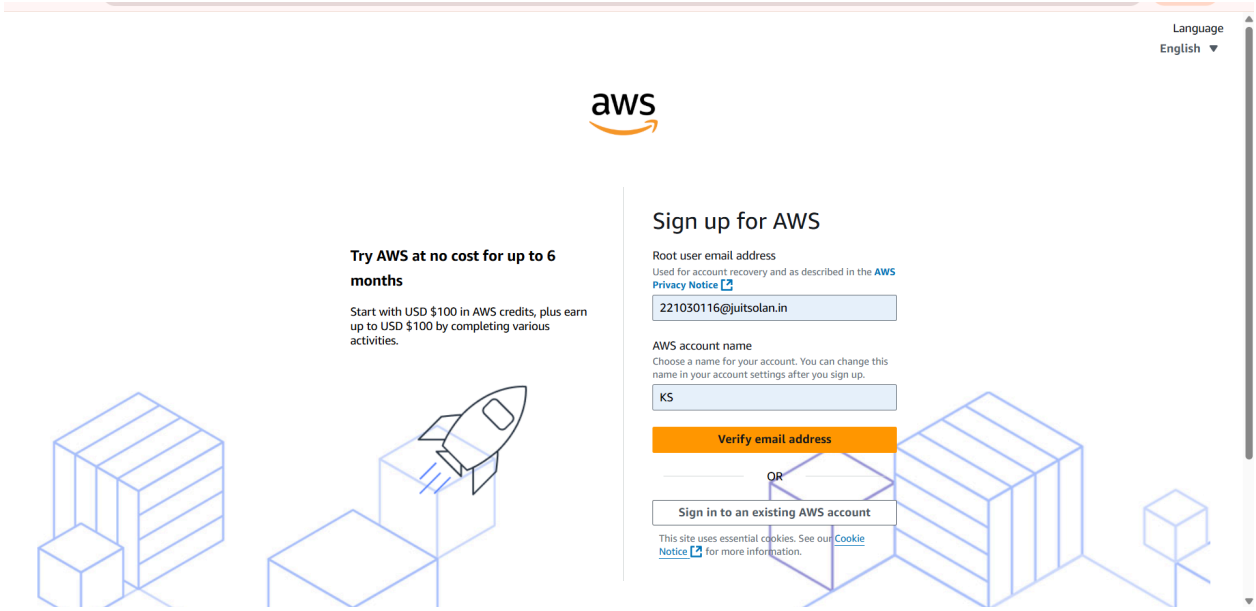
- Activity 7.1: Launch an EC2 instance to host the Flask application, with security configurations for HTTP and SSH access.
- Activity 7.2: Upload Flask application files, templates, and configuration files to the EC2 instance.
- Activity 7.3: Run the Flask app on EC2 and ensure it's accessible to users online.

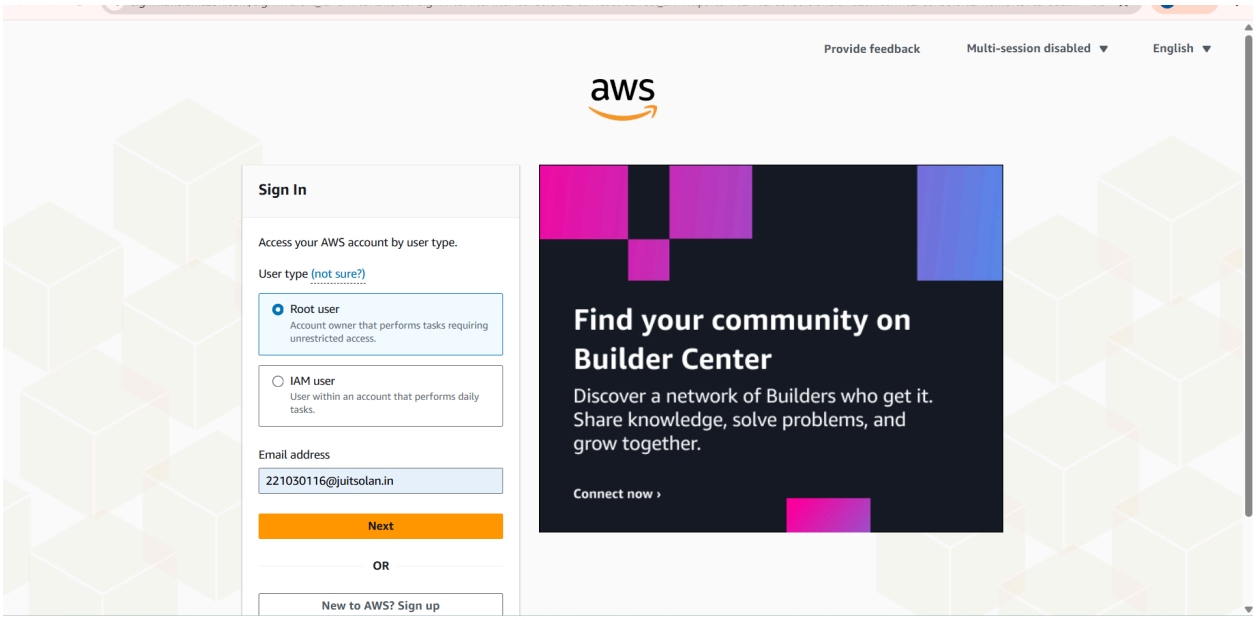
8. Testing and Deployment

Milestone 1: AWS Account Setup and Login

- **Activity 1.1**: Set up an AWS account if not already done.

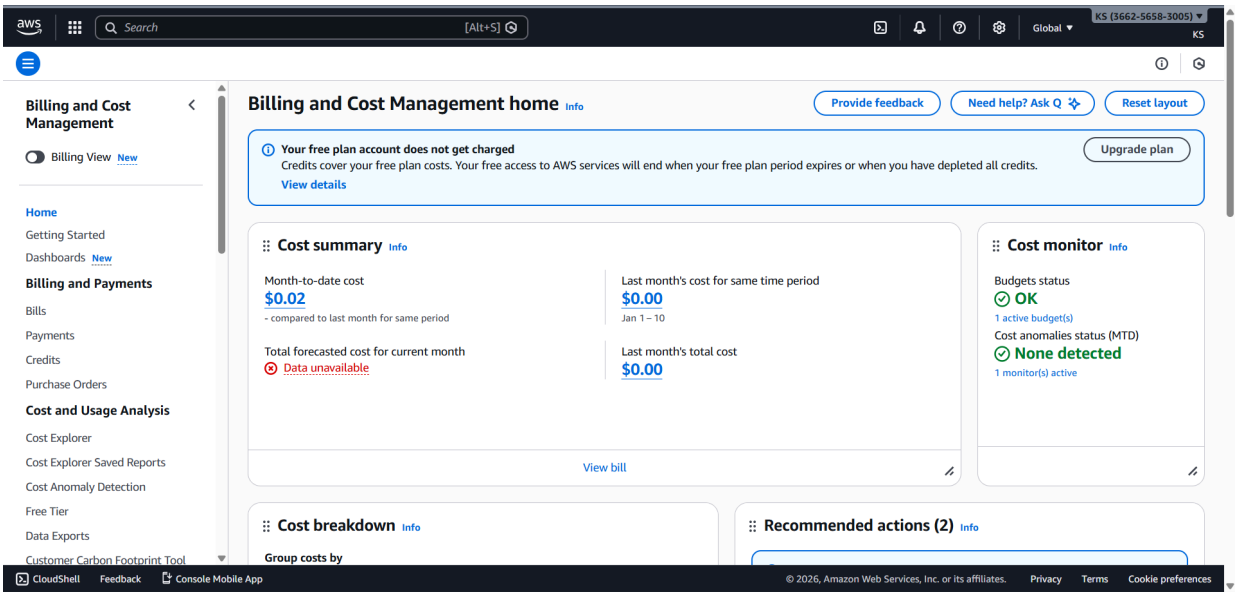
  ➢ Sign up for an AWS account and configure billing settings.





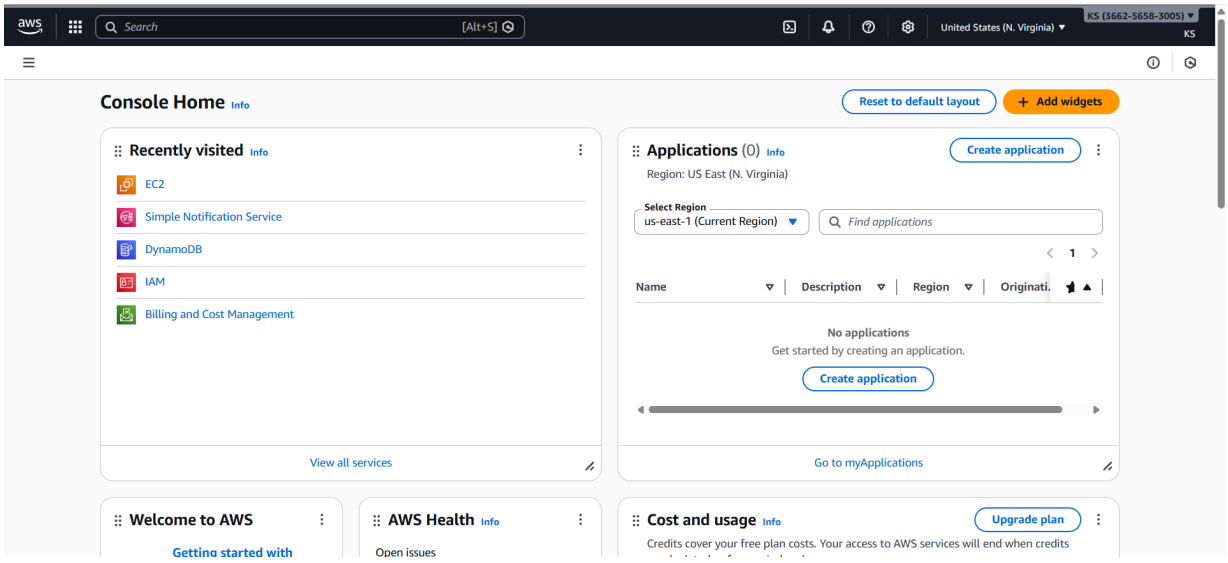
SMARTBRIDGE

- Billing and Cost Management Setup:



- **Activity 1.2**: Log in to the AWS Management Console

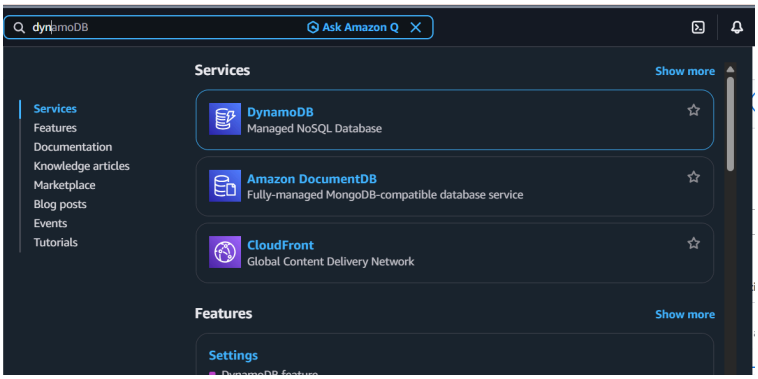
  ➢ Log in to the AWS Management Console

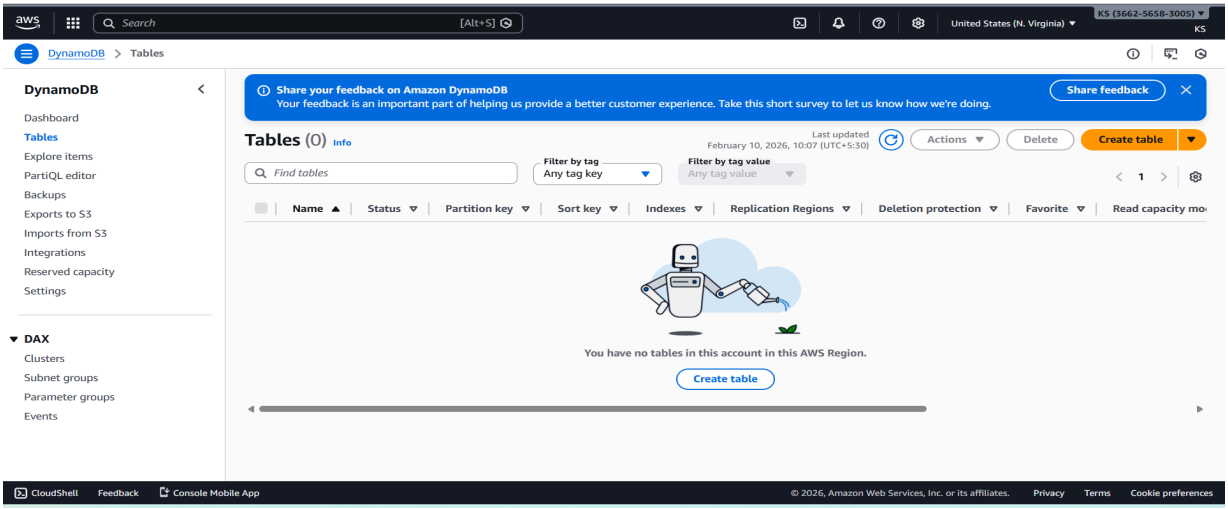


Milestone 2: DynamoDB Database Creation and Setup

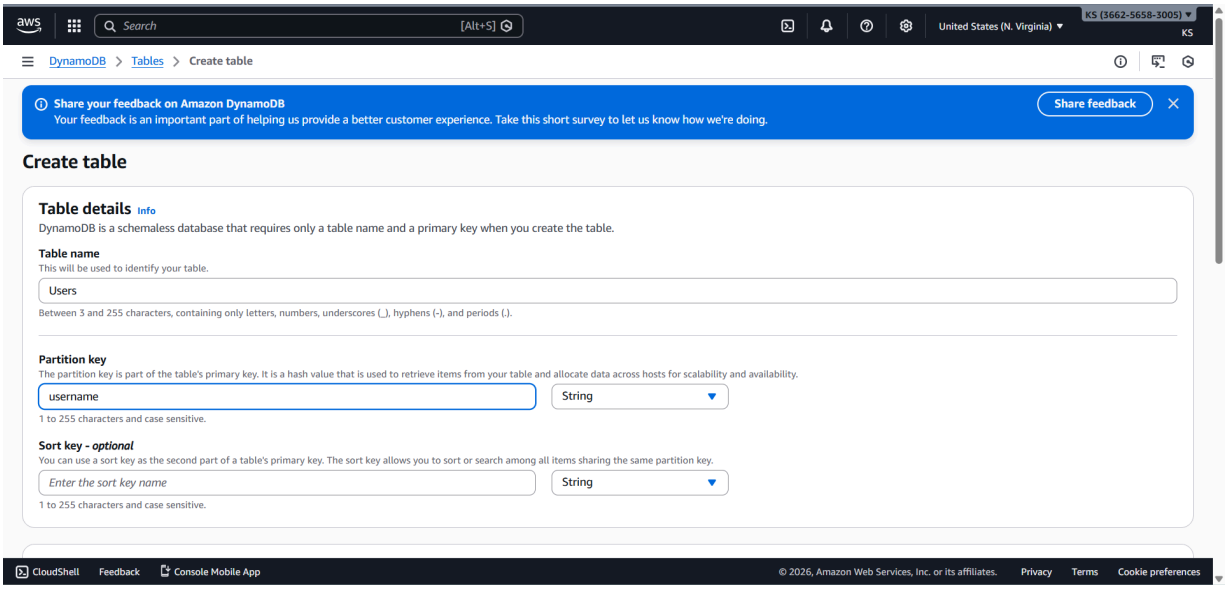
- **Activity 2.1**: Navigate to the DynamoDB

  ➢ In the AWS Console, navigate to DynamoDB and click on create tables.

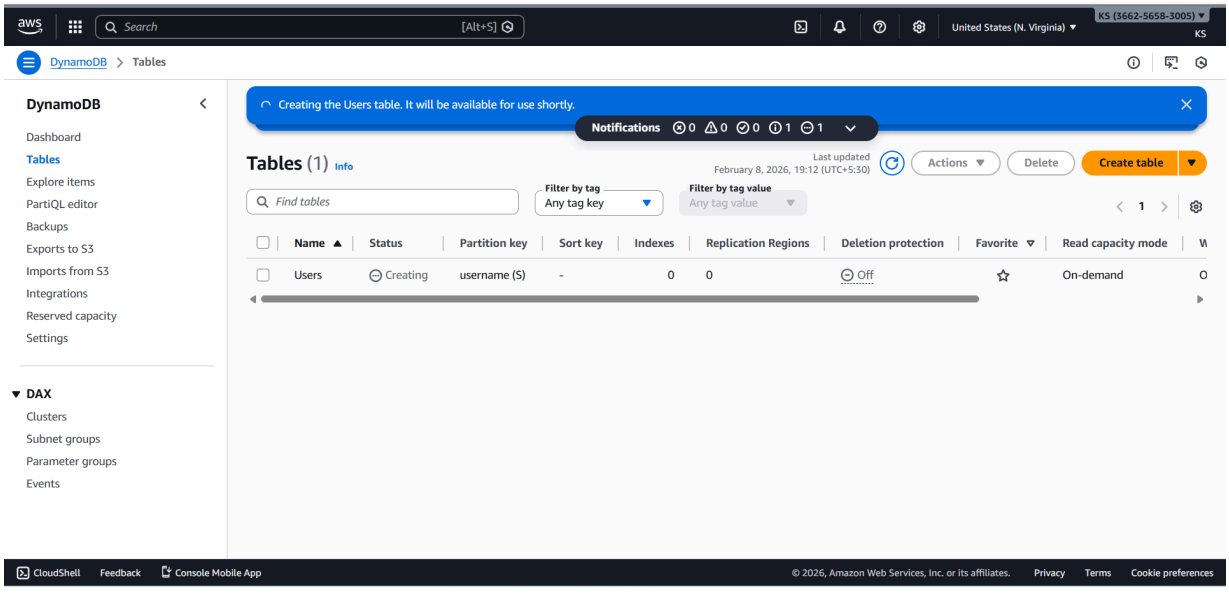




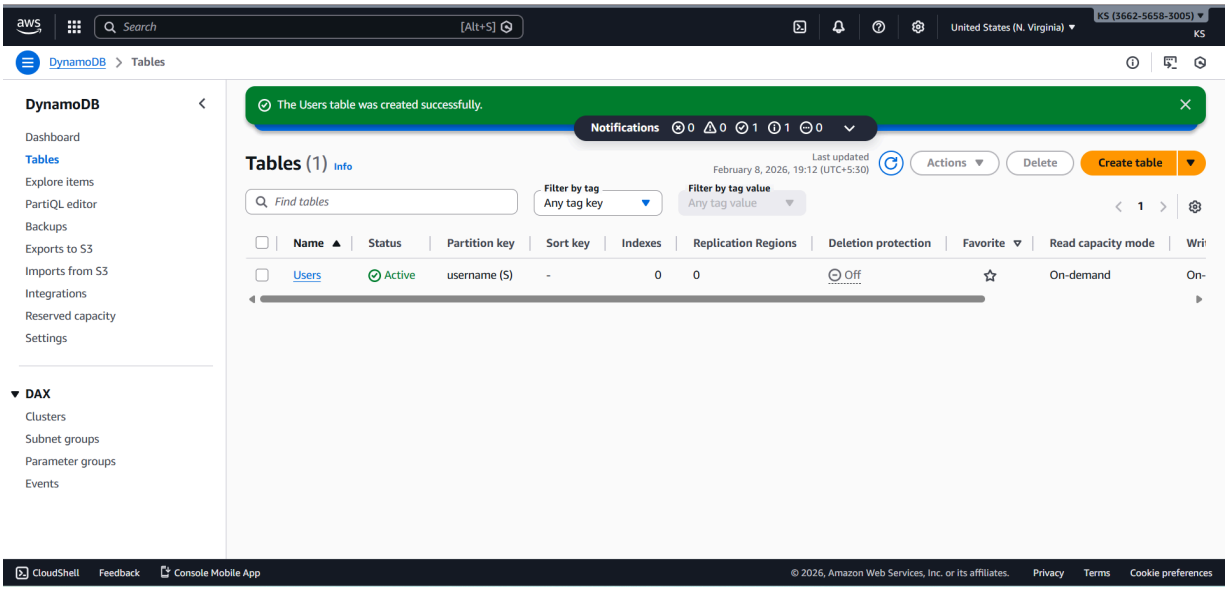
- **Activity 2.2:** Create a DynamoDB table for storing signup details.



➢ Create “Users” table with partition key “username” with type String.



SMARTBRIDGE



SMARTBRIDGE

Milestone 3: Backend Development and Application Setup:

- **Activity 3.1**：Develop the Flask backend to handle core application functionalities, including user registration, login, and session management.
- Description of the code：
- Flask App Initialization

app_aws.py

```python
import boto3
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
from functools import wraps
from groq import Groq

app = Flask(__name__)
app.secret_key = "career_counselor_secret_key"


REGION_NAME = "us-east-1"

dynamodb = boto3.resource('dynamodb', region_name=REGION_NAME)
users_table = dynamodb.Table('Users')

sns_client = boto3.client('sns', region_name=REGION_NAME)
SNS_TOPIC_ARN = "arn:aws:sns:us-east-1:366256583005:CarrerNotifications"

GROQ_CLIENT = Groq(api_key="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")
```

app_aws.py

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        role_selection = request.form.get('role')

        user = get_user_data(username)
        if user and user.get('password') == password:
            if user.get('role') == role_selection:
                session.update({
                    'username': username,
                    'is_admin': user.get('is_admin', False)
                })
                if user.get('is_admin'):
                    return redirect(url_for('admin_dashboard'))
                return redirect(url_for('home'))
            else:
                flash(f"This account is not registered as a {role_selection}.", "danger")
        else:
            flash("Invalid username or password.", "danger")
    return render_template('login.html')

```

SMARTBRIDGE

app_aws.py

```python
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'username' not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function

def admin_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if not session.get('is_admin'):
            flash("Unauthorized Access", "danger")
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function

def get_user_data(username):
    try:
        response = users_table.get_item(Key={'username': username})
        return response.get('Item', {})
    except Exception as e:
        print(f"DynamoDB Error: {e}")
```

app_aws.py

```python
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        role = request.form.get('role')
        try:
            users_table.put_item(
                Item={
                    'username': username,
                    'password': password,
                    'role': role,
                    'is_admin': True if role == 'admin' else False
                },
                ConditionExpression='attribute_not_exists(username)'
            )
            send_sns_notification(username, f"Registered as {role}")
            flash(f"{role.capitalize()} account created!", "success")
            return redirect(url_for('login'))
        except:
            flash("Username already exists.", "danger")
    return render_template('signup.html')
```

app_aws.py

```python
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=False, host='0.0.0.0', port=5000)
```

- **Activity 3.2**: Configure API integration with the Groq API to support Generative AI-powered responses.

➢ This includes setting up API calls to generate career roadmaps, course recommendations, and skill gaps analysis tailored to individual user profiles.

app_aws.py

```python
@app.route('/dashboard')
@login_required
def dashboard():
    user = get_user_data(session['username'])
    target = user.get('target_goal', 'Data Analyst')
    current_skills = user.get('skills', 'Basic Computer Skills')

    prompt = (
        f"Act as a technical career analyst. A student wants to become a {target} "
        f"and currently knows: {current_skills}. "
        "Analyze the industry gap for 2026. Provide exactly 3 skills they already HAVE (Matche
        "and 3 critical skills they are MISSING. "
        "Format your entire response exactly like this: "
        "MATCHED: skill1, skill2, skill3 | MISSING: skill4, skill5, skill6"
    )

    try:
        completion = GROQ_CLIENT.chat.completions.create(
            model="llama-3.3-70b-versatile",
            messages=[{"role": "user", "content": prompt}]
        )
        ai_output = completion.choices[0].message.content
        parts = ai_output.split('|')
        matching_skills = [s.strip() for s in parts[0].replace('MATCHED:', '').split(',')]
        missing_skills = [s.strip() for s in parts[1].replace('MISSING:', '').split(',')]
    except Exception as e:
        matching_skills = ["Check Profile"]
        missing_skills = ["AI Sync Failed"]

    return render_template('dashboard.html',
                           user=user,
                           matching_skills=matching_skills,
                           missing_skills=missing_skills)

```

SMARTBRIDGE

```python
@app.route('/generate-roadmap', methods=['POST'])
@login_required
def generate_roadmap():
    username = session['username']
    u = get_user_data(username)
    prompt = (
        f"Act as an elite Career Architect. Create a 4-phase roadmap for a student at {u.get('
        f"targeting a {u.get('target_goal')} role. Current skills: {u.get('skills')}. "
        "Format your response as exactly 4 sections. Each section must start with 'PHASE X:' "
        "followed by a short title and 2 bullet points for actions."
    )

    try:
        completion = GROQ_CLIENT.chat.completions.create(
            model="llama-3.3-70b-versatile",
            messages=[{"role": "user", "content": prompt}]
        )
        roadmap_content = completion.choices[0].message.content
        users_table.update_item(
            Key={'username': username},
            UpdateExpression="set roadmap_text = :r",
            ExpressionAttributeValues={':r': roadmap_content}
        )
        return jsonify({"success": True, "roadmap": roadmap_content})
    except Exception as e:
        return jsonify({"success": False, "error": str(e)}), 500

```

app_aws.py

```python
@app.route('/api/chat', methods=['POST'])
@login_required
def chat():
    user_query = request.json.get('message')
    u = get_user_data(session['username'])
    system_prompt = f"You are a counselor for a student at {u.get('college')}. Short answers o
    completion = GROQ_CLIENT.chat.completions.create(
        model="llama-3.3-70b-versatile",
        messages=[{"role": "system", "content": system_prompt}, {"role": "user", "content": us
    )
    return jsonify({"response": completion.choices[0].message.content})

```

Milestone 4: Generative AI Model Integration:

- **Activity 4.1**: Configure Groq API endpoints within the Flask application for three core functionalities:
  - ➢ **Course Recommendations**: Generate a list of personalized courses based on user preferences.

app_aws.py

```python
@app.route('/dashboard')
@login_required
def dashboard():
    user = get_user_data(session['username'])
    target = user.get('target_goal', 'Data Analyst')
    current_skills = user.get('skills', 'Basic Computer Skills')

    prompt = (
        f"Act as a technical career analyst. A student wants to become a {target} "
        f"and currently knows: {current_skills}. "
        "Analyze the industry gap for 2026. Provide exactly 3 skills they already HAVE (Matche
        "and 3 critical skills they are MISSING. "
        "Format your entire response exactly like this: "
        "MATCHED: skill1, skill2, skill3 | MISSING: skill4, skill5, skill6"
    )

    try:
        completion = GROQ_CLIENT.chat.completions.create(
            model="llama-3.3-70b-versatile",
            messages=[{"role": "user", "content": prompt}]
        )
        ai_output = completion.choices[0].message.content
        parts = ai_output.split('|')
        matching_skills = [s.strip() for s in parts[0].replace('MATCHED:', '').split(',')]
        missing_skills = [s.strip() for s in parts[1].replace('MISSING:', '').split(',')]
    except Exception as e:
        matching_skills = ["Check Profile"]
        missing_skills = ["AI Sync Failed"]

    return render_template('dashboard.html',
                           user=user,
                           matching_skills=matching_skills,
                           missing_skills=missing_skills)

```

➢ **Career Roadmaps Generation**: Generate detailed career roadmaps.

```python
@app.route('/generate-roadmap', methods=['POST'])
@login_required
def generate_roadmap():
    username = session['username']
    u = get_user_data(username)
    prompt = (
        f"Act as an elite Career Architect. Create a 4-phase roadmap for a student at {u.get('
        f"targeting a {u.get('target_goal')} role. Current skills: {u.get('skills')}. "
        "Format your response as exactly 4 sections. Each section must start with 'PHASE X:' "
        "followed by a short title and 2 bullet points for actions."
    )

    try:
        completion = GROQ_CLIENT.chat.completions.create(
            model="llama-3.3-70b-versatile",
            messages=[{"role": "user", "content": prompt}]
        )
        roadmap_content = completion.choices[0].message.content
        users_table.update_item(
            Key={'username': username},
            UpdateExpression="set roadmap_text = :r",
            ExpressionAttributeValues={':r': roadmap_content}
        )
        return jsonify({"success": True, "roadmap": roadmap_content})
    except Exception as e:
        return jsonify({"success": False, "error": str(e)}), 500

```

➢ **AI Chat Bot**: Retrieve up-to-date insights on in-demand skills, salary ranges, top hiring companies, and job availability by interacting with AI chatbot powered by GroqAI.

app_aws.py

```python
@app.route('/api/chat', methods=['POST'])
@login_required
def chat():
    user_query = request.json.get('message')
    u = get_user_data(session['username'])
    system_prompt = f"You are a counselor for a student at {u.get('college')}. Short answers c
    completion = GROQ_CLIENT.chat.completions.create(
        model="llama-3.3-70b-versatile",
        messages=[{"role": "system", "content": system_prompt}, {"role": "user", "content": us
    )
    return jsonify({"response": completion.choices[0].message.content})

```

➢ **Configuration Description**:It contains configuration settings for the Virtual Career Counsellor application, including AWS access credentials and the secret key for session management. These settings enable secure access to AWS services like DynamoDB, IAM ,EC2 while ensuring application's overall security.

app_aws.py

```python
import boto3
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
from functools import wraps
from groq import Groq

app = Flask(__name__)
app.secret_key = "career_counselor_secret_key"


REGION_NAME = "us-east-1"

dynamodb = boto3.resource('dynamodb', region_name=REGION_NAME)
users_table = dynamodb.Table('Users')

sns_client = boto3.client('sns', region_name=REGION_NAME)
SNS_TOPIC_ARN = "arn:aws:sns:us-east-1:366256583005:CarrerNotifications"

GROQ_CLIENT = Groq(api_key="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")
```

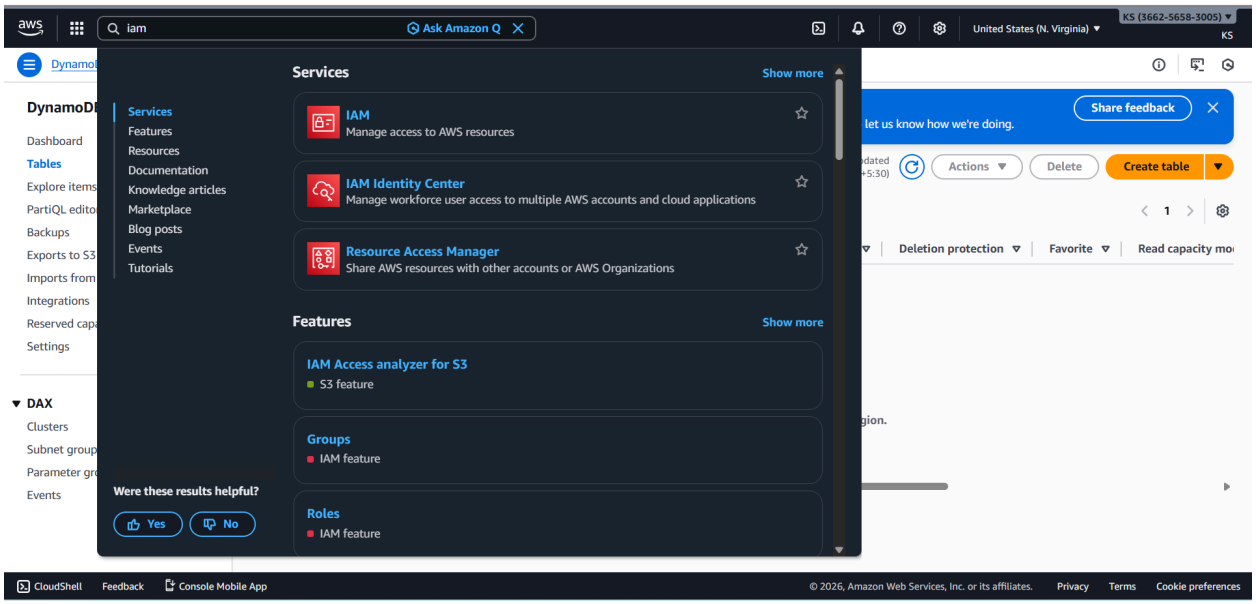
➢ **Secret Description**: This code generates a random secret key for use in applications. The bytes are then converted to a hexadecimal string, ensuring a secure key suitable for cryptographic operations

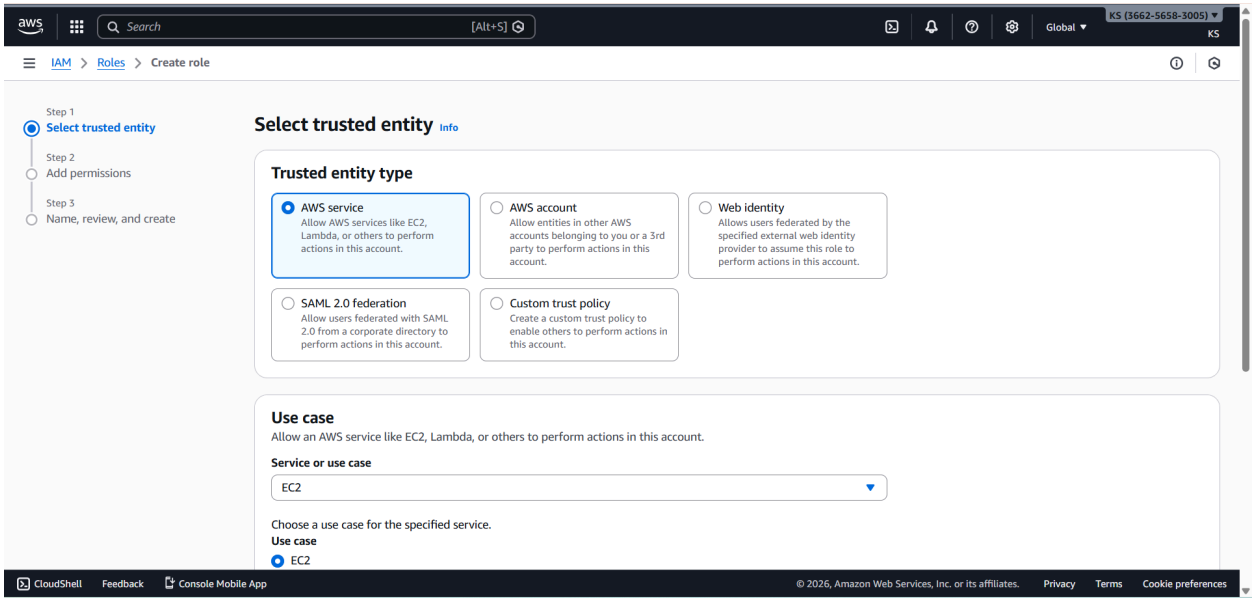
```python
app = Flask(__name__)
app.secret_key = "career_counselor_secret_key"
```

SMARTBRIDGE

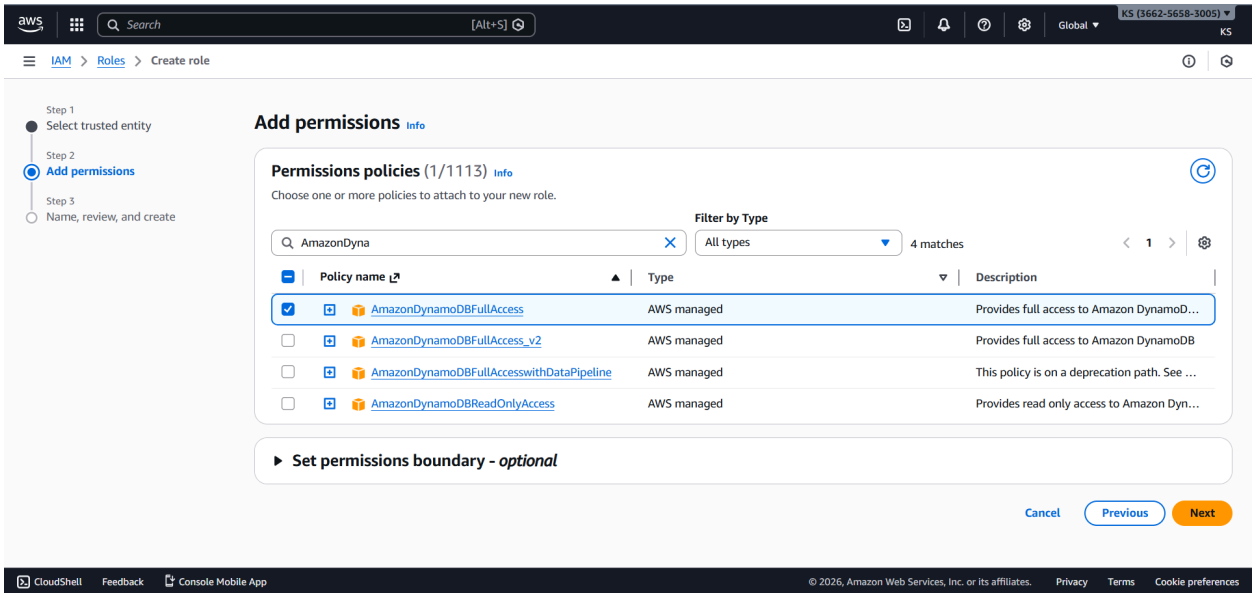
Milestone 5: IAM Role Setup

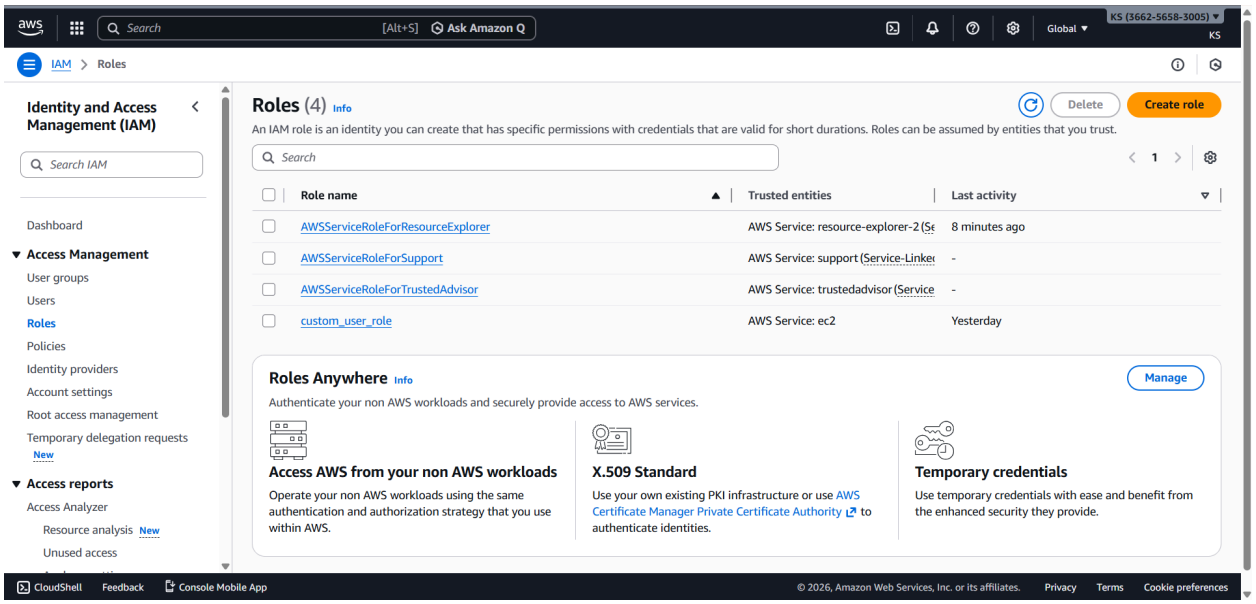
- **Activity 5.1**:Create IAM Role.
  - ➢ In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.



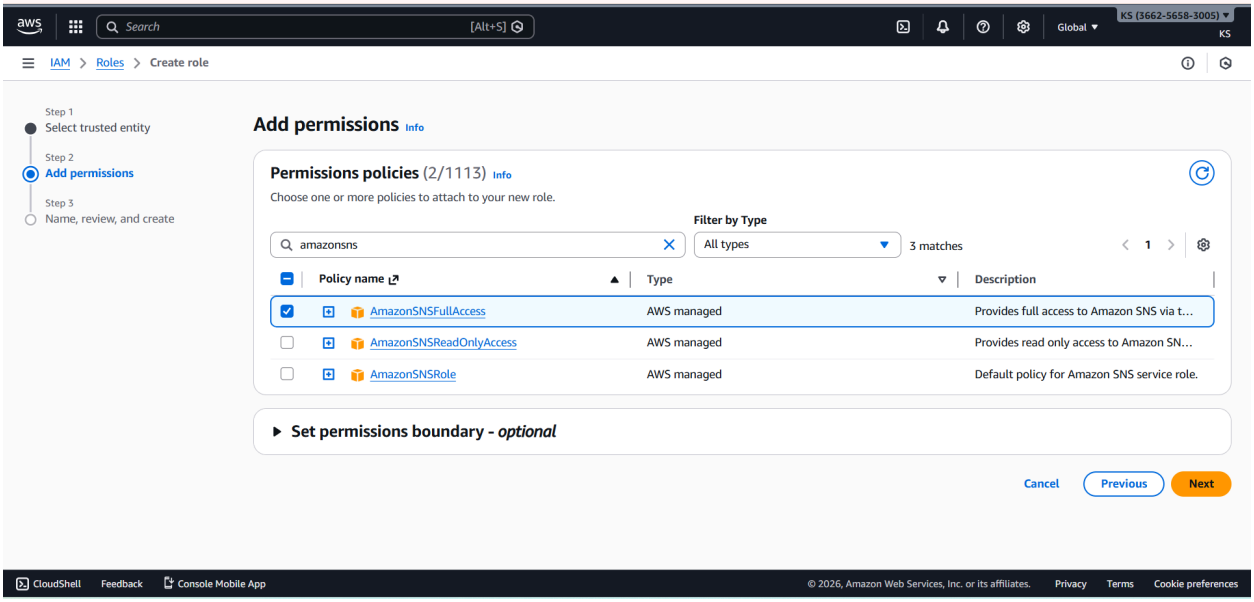


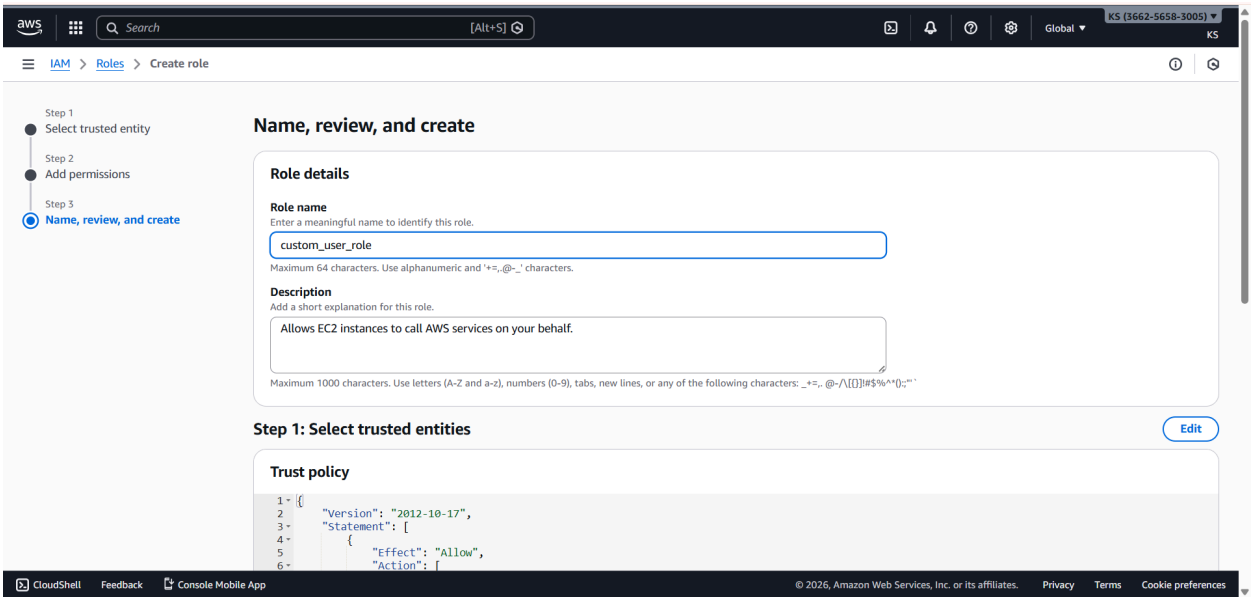
- **Activity 5.2**: Attach Policies.
  - ➢ Attach the following policies to the role:
    - AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.



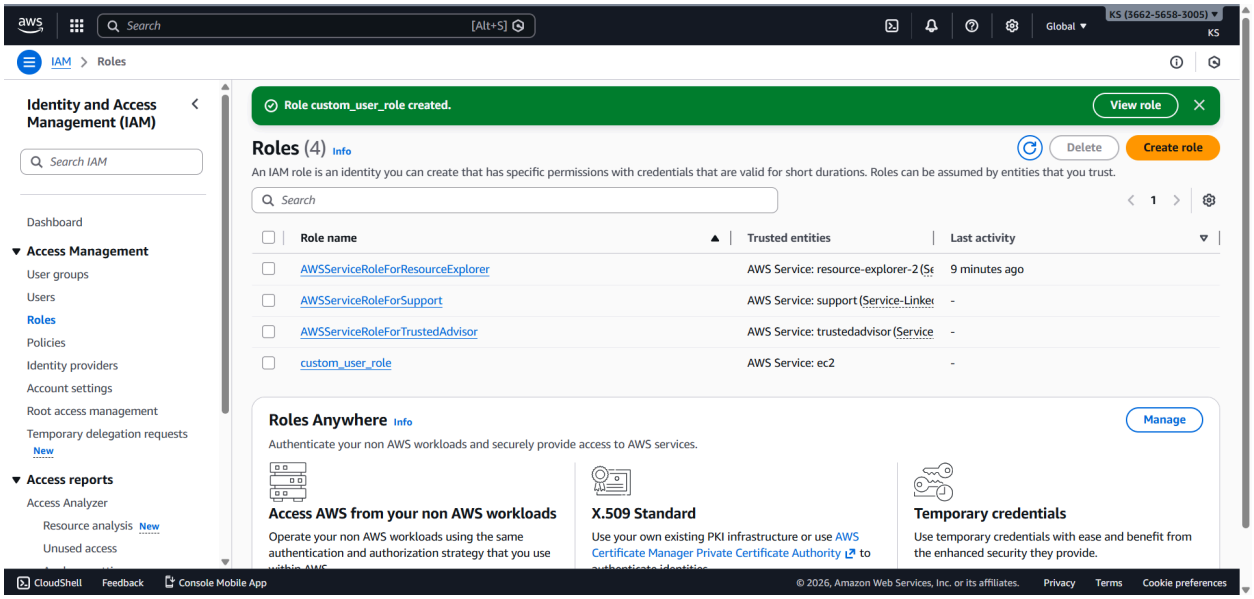


- AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.





SMARTBRIDGE



SMARTBRIDGE

Milestone 6. Application Routes and Core Functionalities

Development

- **Activity 6.1:** Establish routes in the Flask app to serve the following functionalities:

  ➢ **(/):** Display the landing page with an overview of features and links to signup or log in.

app_aws.py

```python
@app.route('/')
def index():
    return redirect(url_for('login'))

```

  ➢ **Home (/home)**: Display the home page with an overview of features and links to signup or log in.

```python
@app.route('/home')
@login_required
def home():
    user = get_user_data(session['username'])
    return render_template('home.html', user=user)

```

Description: A route is defined for the home page ('/') that renders the home.html template for all users, regardless of their authentication status. This route serves as the main landing page for the application.

➢ **Signup (/signup)**: Implement user signup, securing passwords and saving details to DynamoDB.

```python
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        role = request.form.get('role')
        try:
            users_table.put_item(
                Item={
                    'username': username,
                    'password': password,
                    'role': role,
                    'is_admin': True if role == 'admin' else False
                },
                ConditionExpression='attribute_not_exists(username)'
            )
            # Notify admin of new registration
            send_sns_notification(username, f"Registered as {role}")
            flash(f"{role.capitalize()} account created!", "success")
            return redirect(url_for('login'))
        except:
            flash("Username already exists.", "danger")
    return render_template('signup.html')
```

➢ **Description**: The /signup route displays the signup page and handles user signup through both GET and POST methods. With a GET request, it renders the signup.html template for user input, while a POST request processes the form data and saves user details to DynamoDB. After successful signup, the user is redirected to the home page.

SMARTBRIDGE

1. **Login (/login)**: Authenticate users and initiate sessions for personalized services.

app_aws.py

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        role_selection = request.form.get('role')

        user = get_user_data(username)
        if user and user.get('password') == password:
            if user.get('role') == role_selection:
                session.update({
                    'username': username,
                    'is_admin': user.get('is_admin', False)
                })
                if user.get('is_admin'):
                    return redirect(url_for('admin_dashboard'))
                return redirect(url_for('home'))
            else:
                flash(f"This account is not registered as a {role_selection}.", "danger")
        else:
            flash("Invalid username or password.", "danger")
    return render_template('login.html')

```

Description: The /login route displays the login page with a form for user credentials. On form submission (POST), it retrieves user data from DynamoDB, and redirects authenticated users to the home page. If authentication fails, the user is redirected to the signup page.

2. **Logout (/logout)**: End user sessions and ensure data protection upon logout.

app_aws.py

```python
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))

```

Description: The /logout route handles user logout by clearing session data to end the user session. Once logged out, the user is redirected to the home page.

3. **About (/about)**: It contains information about the Career AI website..

app_aws.py

```python
@app.route('/about')
def about():
    return render_template('about.html')

```

4. **Setup-goal (/setup-goal)**: It directs user to the setup career profile.

app_aws.py

```python
@app.route('/setup-goal', methods=['GET', 'POST'])
@login_required
def setup_goal():
    username = session['username']
    if request.method == 'POST':
        data = {
            'college': request.form.get('college'),
            'education': request.form.get('education'),
            'cgpa': request.form.get('cgpa'),
            'skills': request.form.get('skills'),
            'target_goal': request.form.get('target_goal')
        }
        users_table.update_item(
            Key={'username': username},
            UpdateExpression="set college=:c, education=:e, cgpa=:cg, skills=:s, target_goal=:
            ExpressionAttributeValues={
                ':c': data['college'], ':e': data['education'], ':cg': data['cgpa'],
                ':s': data['skills'], ':t': data['target_goal']
            }
        )
        send_sns_notification(username, f"Updated target to {data['target_goal']}")
        return redirect(url_for('dashboard'))
    return render_template('setup_goal.html', user=get_user_data(username))
```

SMARTBRIDGE

5. **Dashboard (/dashboard)**: It directs you to Career roadmap page..

app_aws.py

```python
@app.route('/dashboard')
@login_required
def dashboard():
    user = get_user_data(session['username'])
    target = user.get('target_goal', 'Data Analyst')
    current_skills = user.get('skills', 'Basic Computer Skills')

    prompt = (
        f"Act as a technical career analyst. A student wants to become a {target} "
        f"and currently knows: {current_skills}. "
        "Analyze the industry gap for 2026. Provide exactly 3 skills they already HAVE (Matche
        "and 3 critical skills they are MISSING. "
        "Format your entire response exactly like this: "
        "MATCHED: skill1, skill2, skill3 | MISSING: skill4, skill5, skill6"
    )

    try:
        completion = GROQ_CLIENT.chat.completions.create(
            model="llama-3.3-70b-versatile",
            messages=[{"role": "user", "content": prompt}]
        )
        ai_output = completion.choices[0].message.content
        parts = ai_output.split('|')
        matching_skills = [s.strip() for s in parts[0].replace('MATCHED:', '').split(',')]
        missing_skills = [s.strip() for s in parts[1].replace('MISSING:', '').split(',')]
    except Exception as e:
        matching_skills = ["Check Profile"]
        missing_skills = ["AI Sync Failed"]

    return render_template('dashboard.html',
                           user=user,
                           matching_skills=matching_skills,
                           missing_skills=missing_skills)

```

6. **Generate-Roadmap (/generate-roadmap)**:It generates the career roadmap bsed upon your target goal..

```python
@app.route('/generate-roadmap', methods=['POST'])
@login_required
def generate_roadmap():
    username = session['username']
    u = get_user_data(username)
    prompt = (
        f"Act as an elite Career Architect. Create a 4-phase roadmap for a student at {u.get('
        f"targeting a {u.get('target_goal')} role. Current skills: {u.get('skills')}. "
        "Format your response as exactly 4 sections. Each section must start with 'PHASE X:' "
        "followed by a short title and 2 bullet points for actions."
    )

    try:
        completion = GROQ_CLIENT.chat.completions.create(
            model="llama-3.3-70b-versatile",
            messages=[{"role": "user", "content": prompt}]
        )
        roadmap_content = completion.choices[0].message.content
        users_table.update_item(
            Key={'username': username},
            UpdateExpression="set roadmap_text = :r",
            ExpressionAttributeValues={':r': roadmap_content}
        )
        return jsonify({"success": True, "roadmap": roadmap_content})
    except Exception as e:
        return jsonify({"success": False, "error": str(e)}), 500

```

7. **Api-Chat(/api/chat)**: It helps you to chat with AI bott.

app_aws.py

```python
@app.route('/api/chat', methods=['POST'])
@login_required
def chat():
    user_query = request.json.get('message')
    u = get_user_data(session['username'])
    system_prompt = f"You are a counselor for a student at {u.get('college')}. Short answers c
    completion = GROQ_CLIENT.chat.completions.create(
        model="llama-3.3-70b-versatile",
        messages=[{"role": "system", "content": system_prompt}, {"role": "user", "content": us
    )
    return jsonify({"response": completion.choices[0].message.content})

```

8. **Admin-Dashboard(/admin-dashboard)**:It contains details about no. of students enrolled with their username and email.

```python
@app.route('/admin-dashboard')
@admin_required
def admin_dashboard():
    response = users_table.scan()
    all_users = response.get('Items', [])
    return render_template('admin_dashboard.html',
                           all_users=all_users,
                           user_count=len(all_users))
```

9. **Deployment Code:**

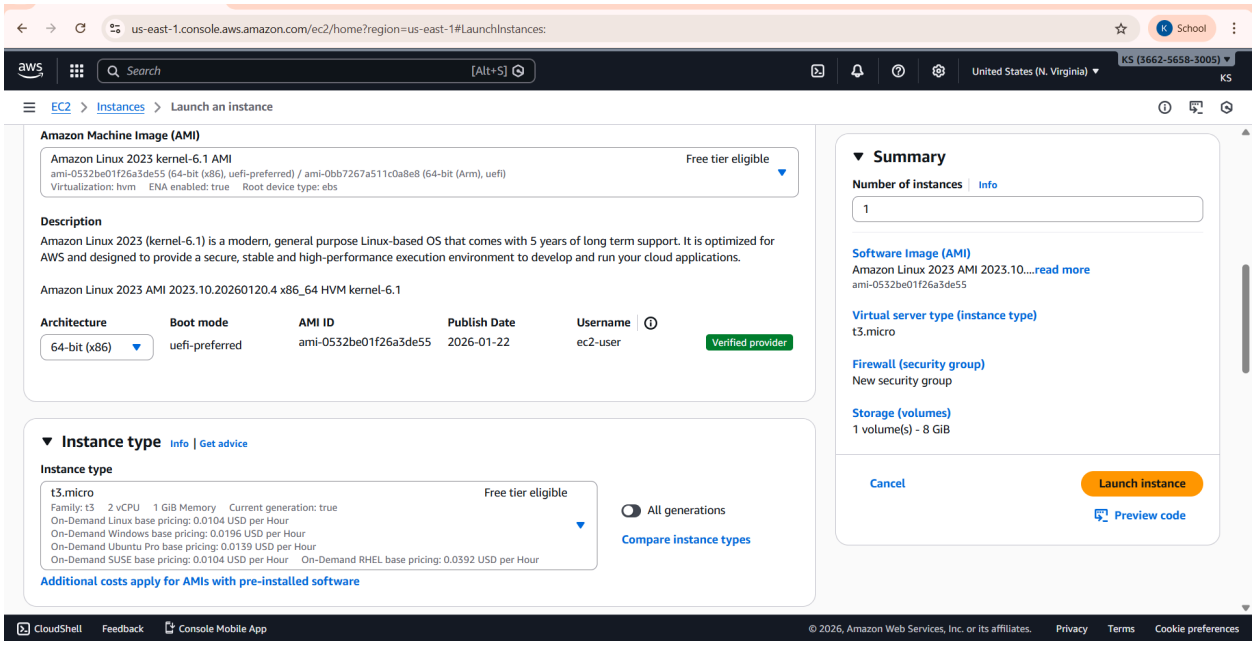
```python

if __name__ == '__main__':
    app.run(debug=False, host='0.0.0.0', port=5000)
```

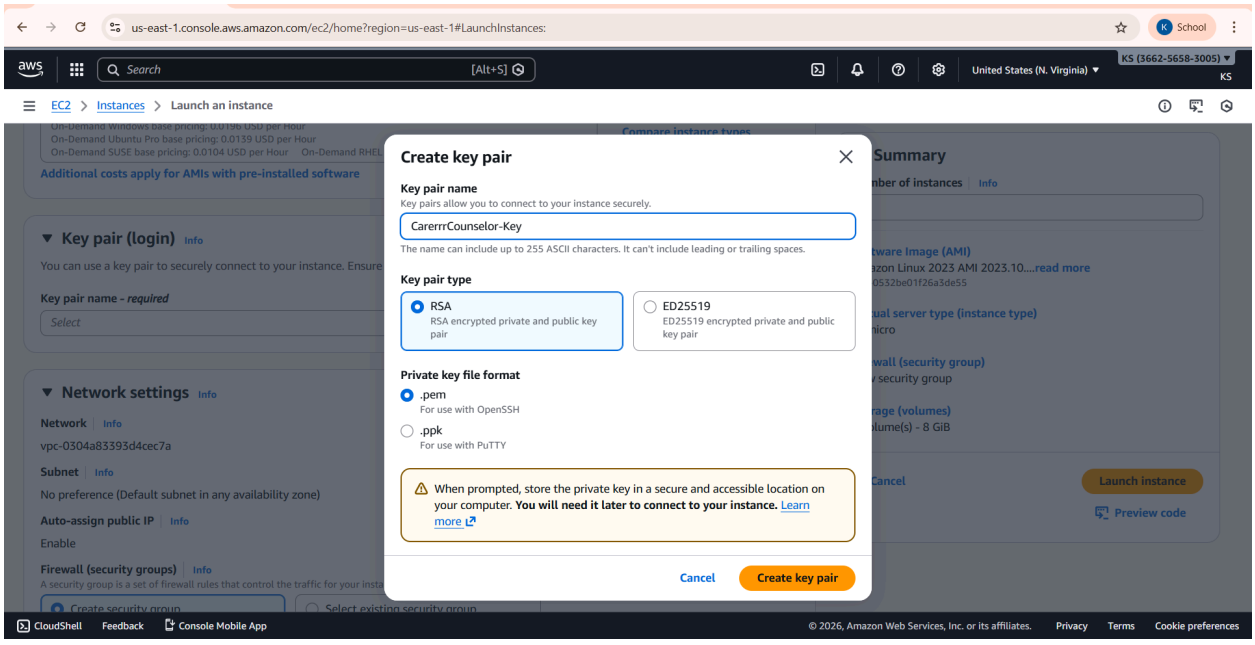
Description: This code snippet serves as the main entry point for the Flask application. When the script is executed directly, it starts the Flask development server in debug mode, allowing for live reloading and detailed error messages, which is useful for development and testing.

- **Milestone 7: EC2 Instance Setup and Deployment:**

- **Activity 7.1**: Launch EC2 Instance
  - ➢ In the AWS Console, navigate to EC2 and launch a new instance.

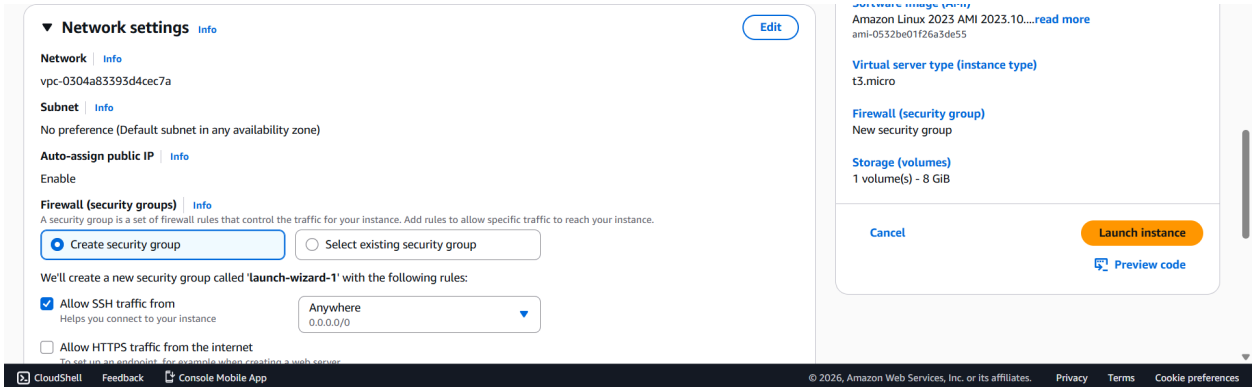
  - ➢ Click on Launch instance to launch EC2 instance

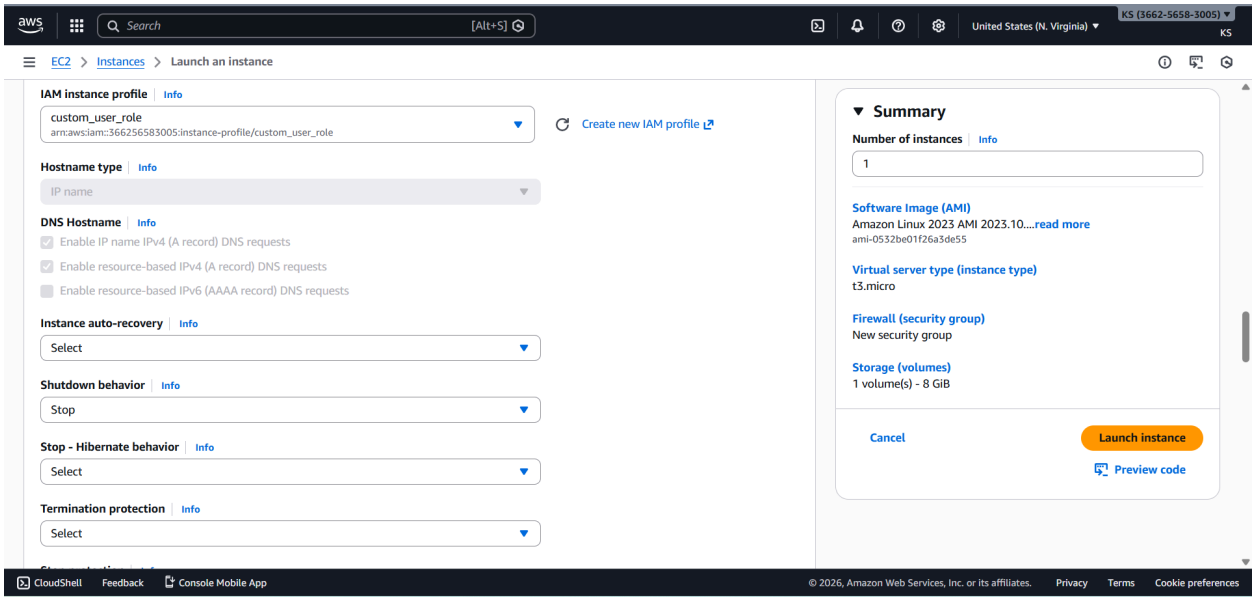




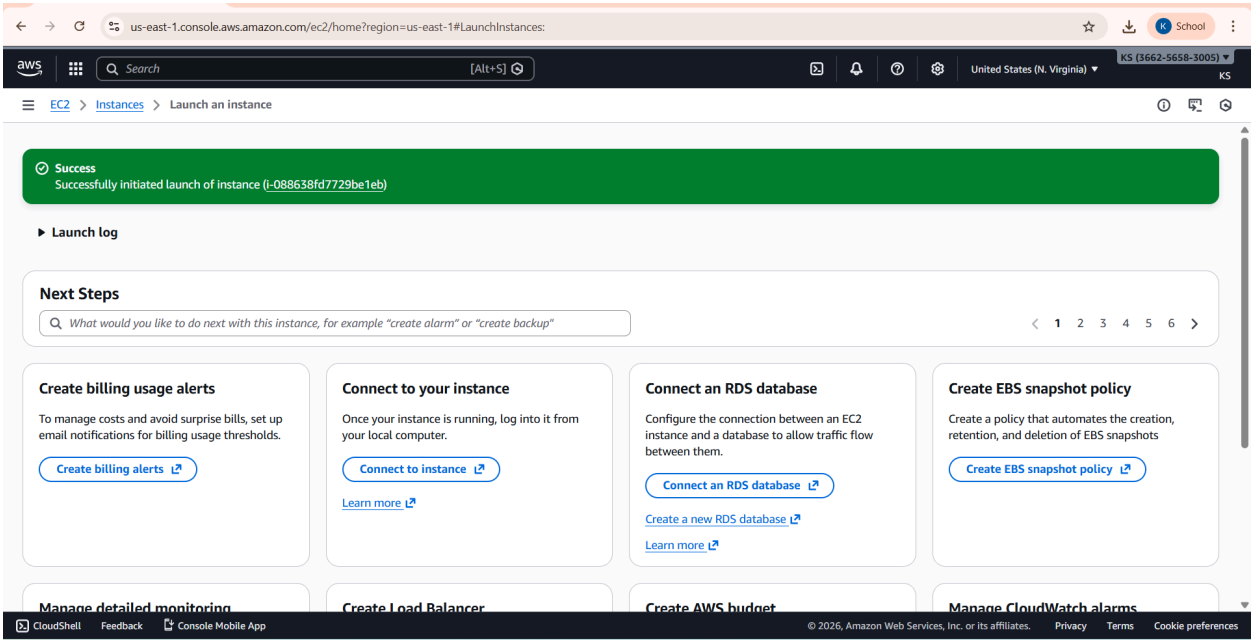
SMARTBRIDGE

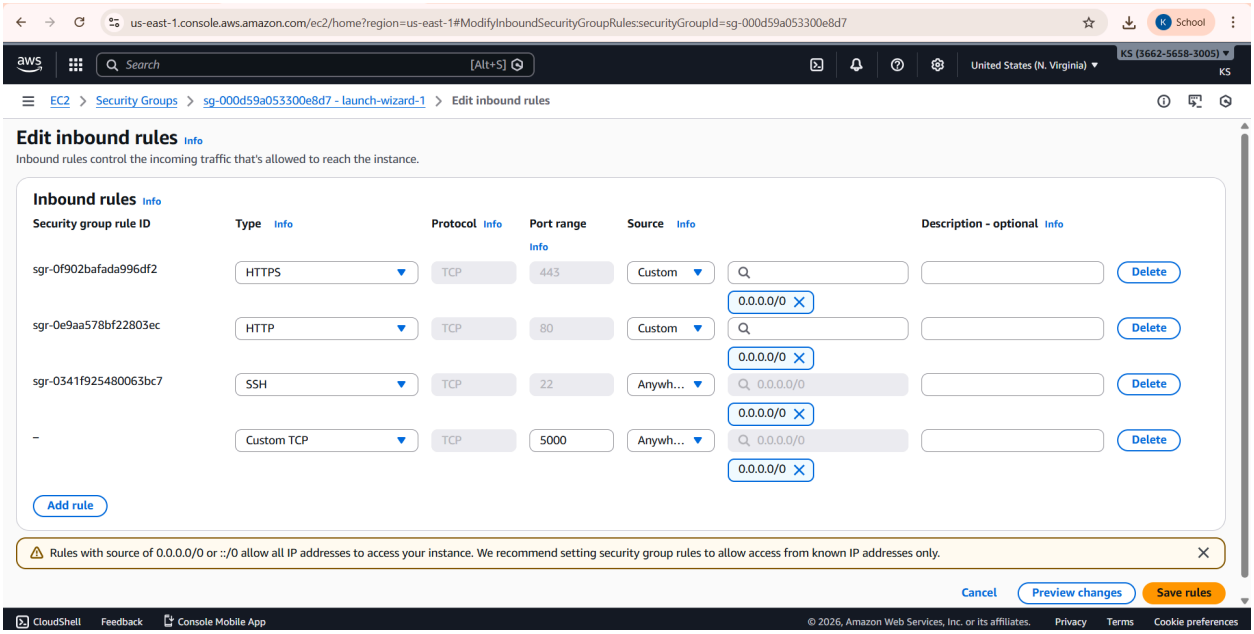
- **Activity 7.2**: Configure security groups for HTTP, and SSH access.



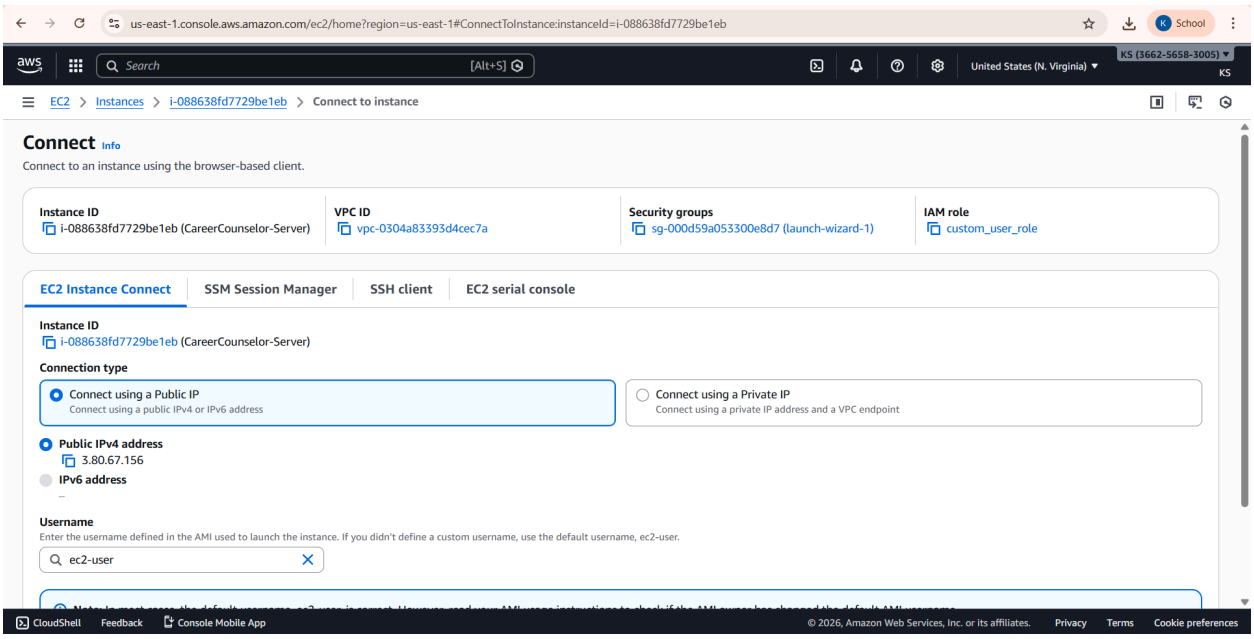


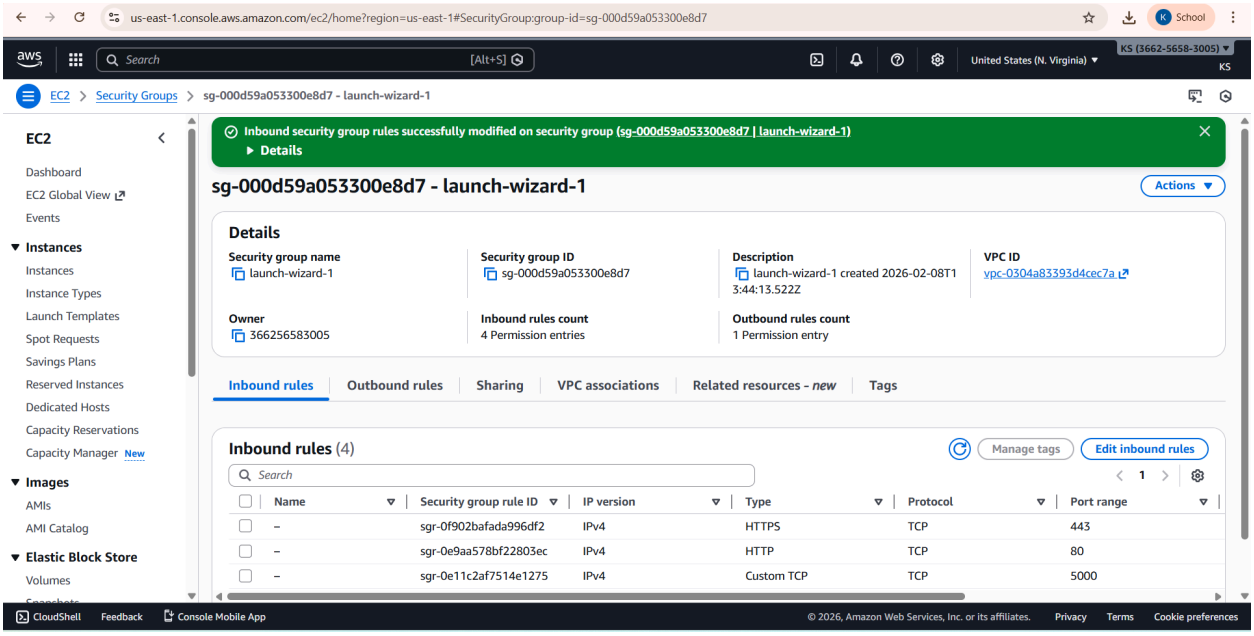
SMARTBRIDGE



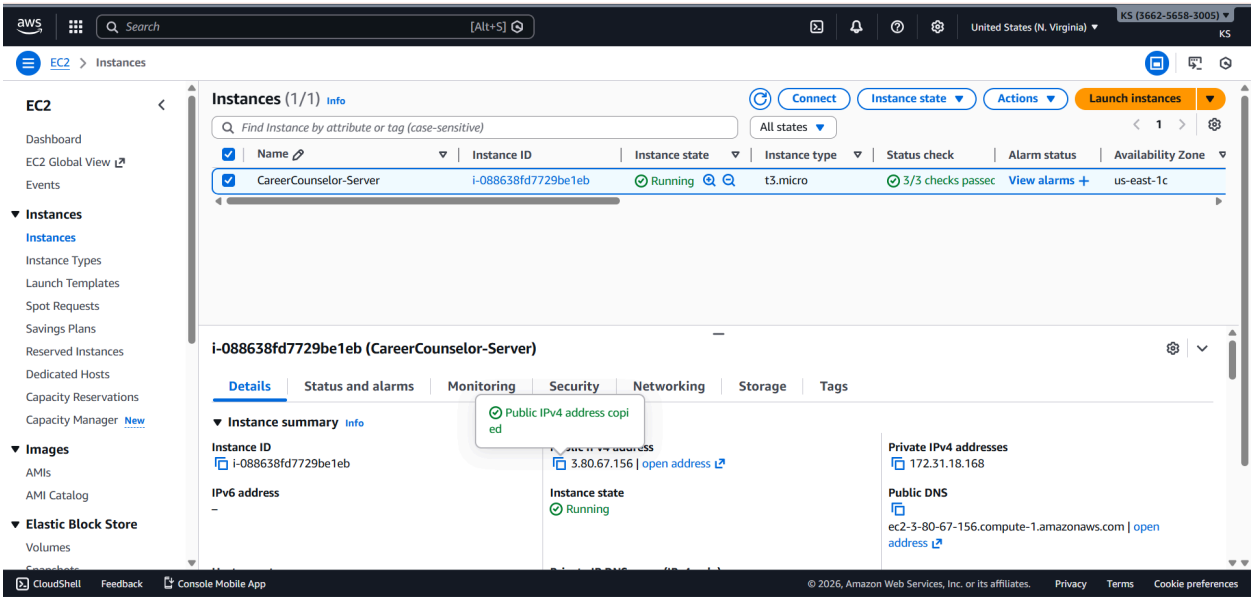


SMARTBRIDGE

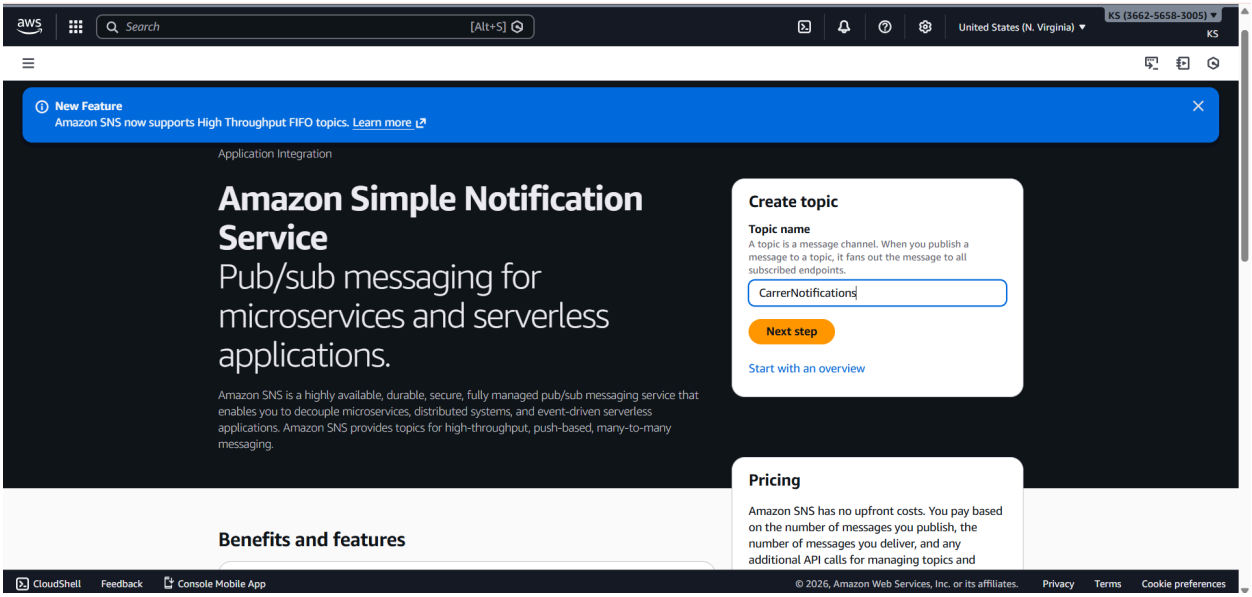




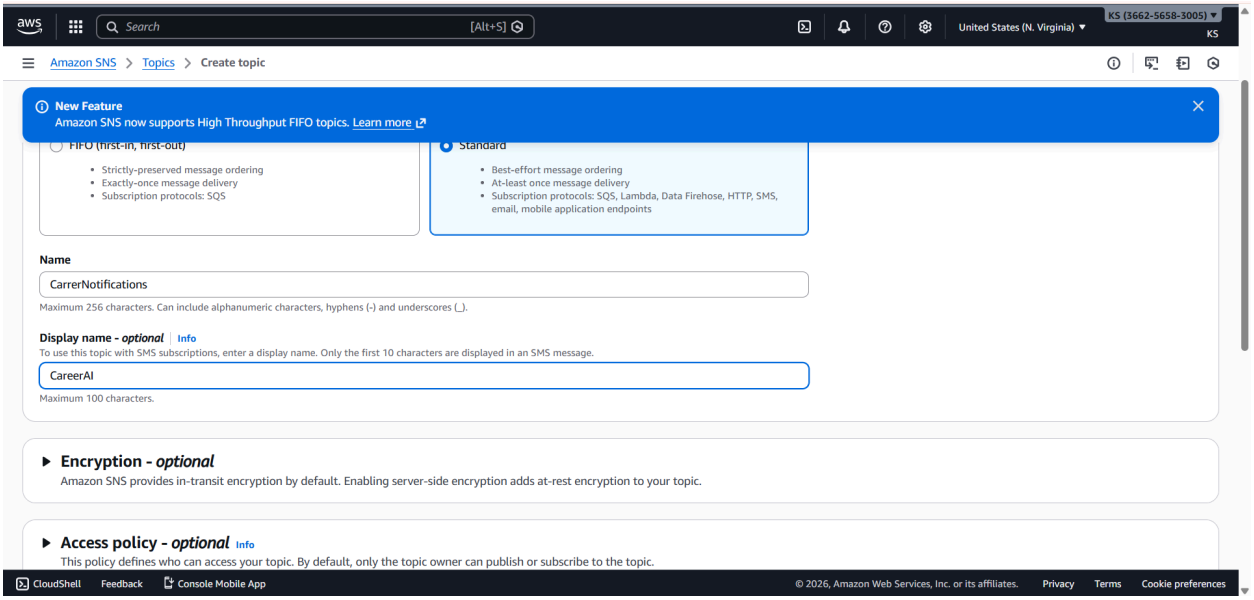
- **SNS Setup:**

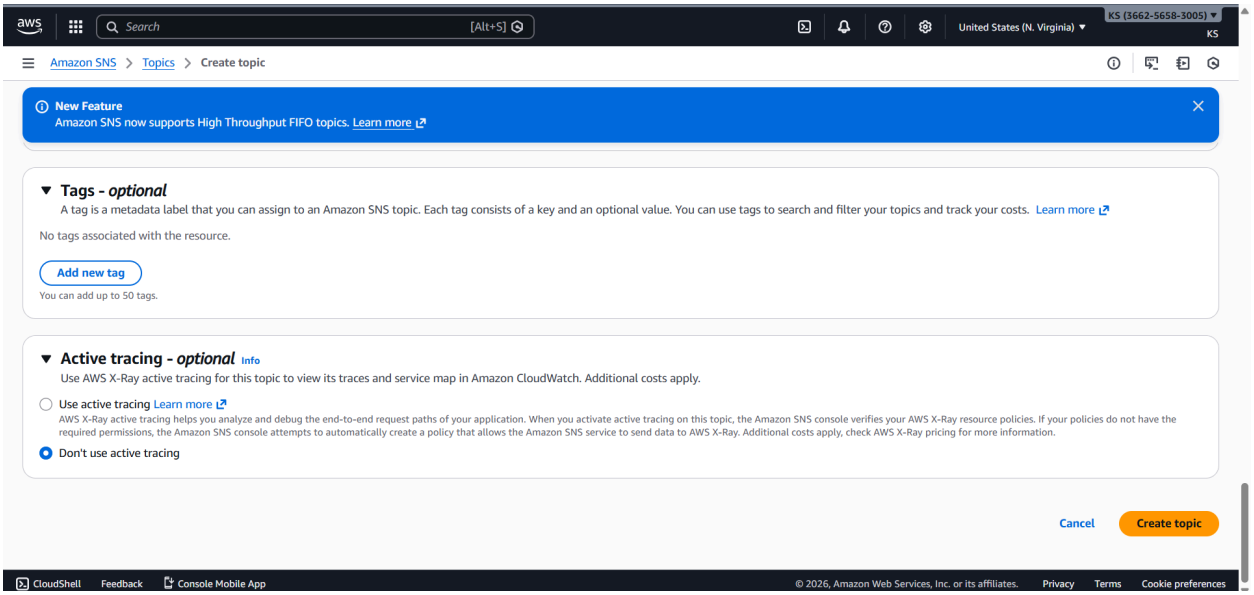


➢ Creating SNS Topics:

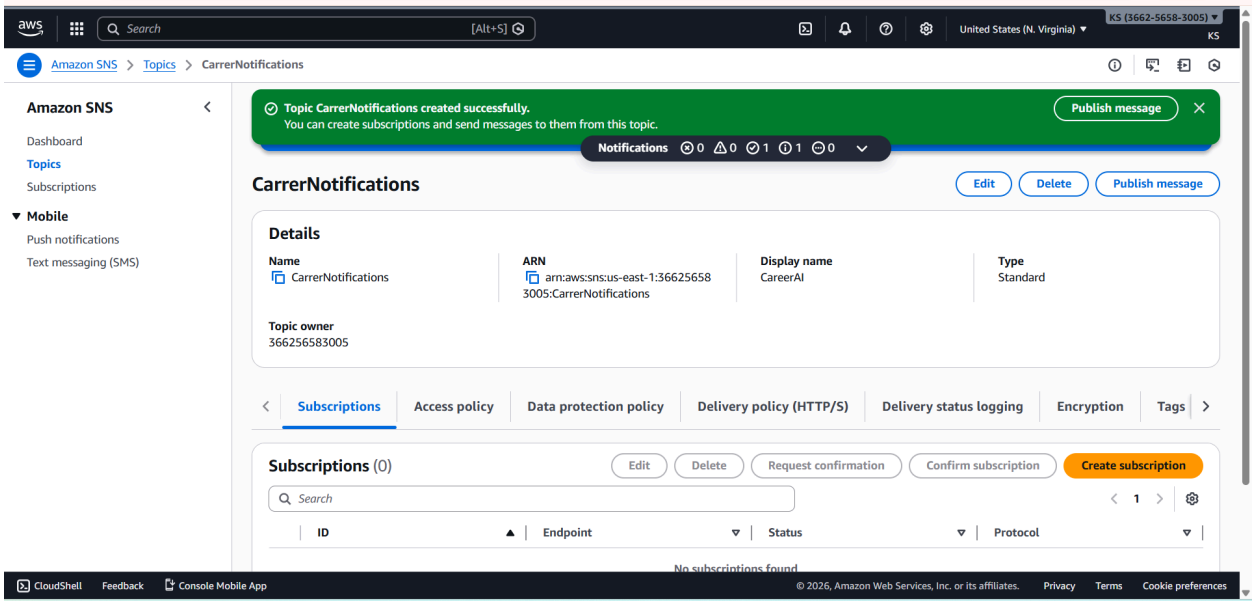


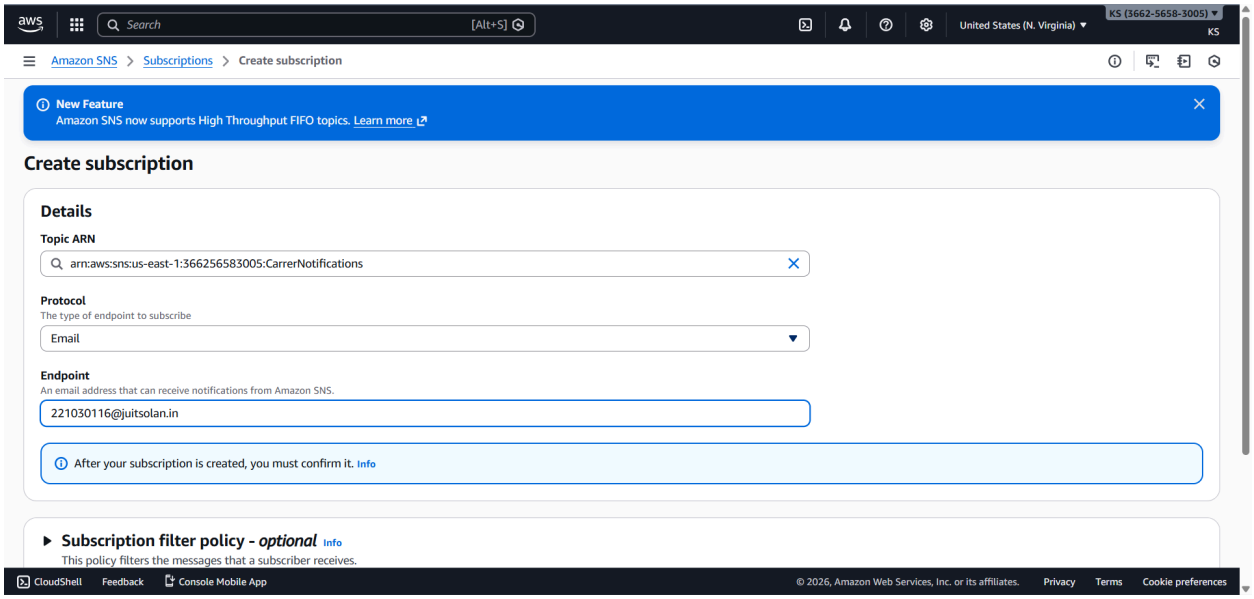
SMARTBRIDGE



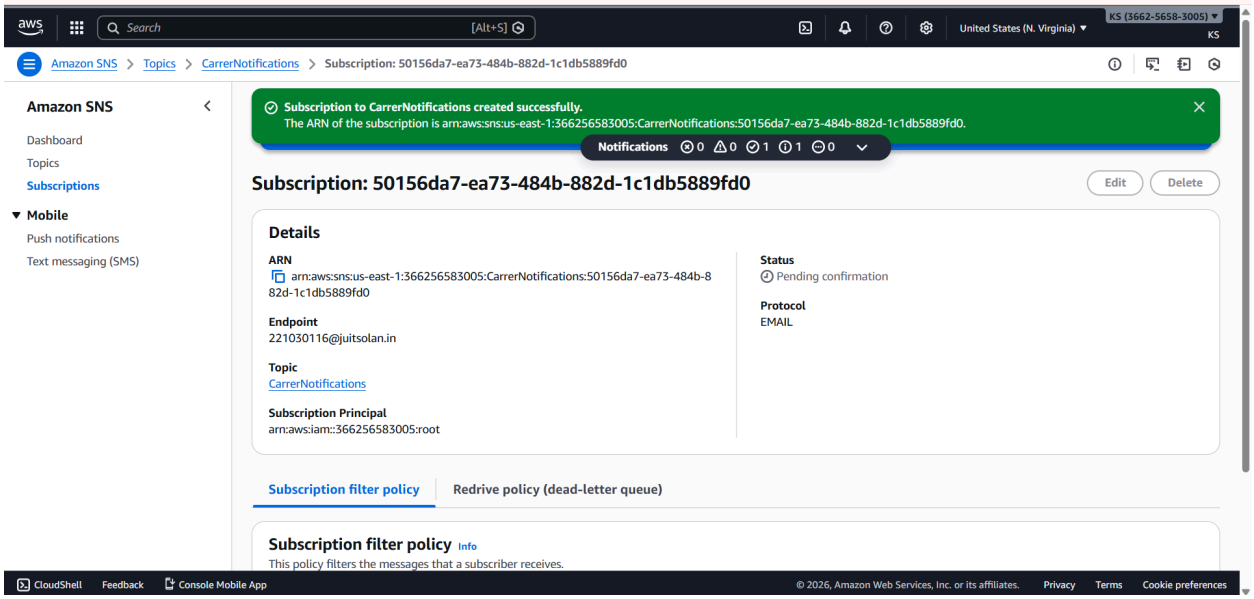


➢ Creating SNS Subscriptions:





SMARTBRIDGE



- ## Milestone 8: Testing and Deployment

  • **Activity 8.1**:Deploy to EC2

1. Connect EC2 terminal.
2. Set up any necessary environment variables, including database connection strings.
3. Configure the web server to serve your application.
4. Start your application and ensure it's accessible via the EC2 instance's public IP or domain.
5. Run the below commands on ec2 terminal
6. sudo yum update -y
7. sudo yum install python3 -y
8. sudo pip3 install virtualenv
9. python3 -m venv venv
10. source venv/bin/activate
11. pip install flask

SMARTBRIDGE

```
[ec2-user@ip-172-31-18-168 ~]$ sudo dnf update -y
Amazon Linux 2023 Kernel Livepatch repository                                                    233 kB/s |  31 kB     00:00
================================================================================================================================
WARNING:
  A newer release of "Amazon Linux" is available.

  Available Versions:

  Version 2023.10.20260202:
    Run the following command to upgrade to 2023.10.20260202:

      dnf upgrade --releasever=2023.10.20260202

    Release notes:
     https://docs.aws.amazon.com/linux/al2023/release-notes/relnotes-2023.10.20260202.html

================================================================================================================================
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-18-168 ~]$
```

i-088638fd7729be1eb (CareerCounselor-Server)

PublicIPs: 3.80.67.156 PrivateIPs: 172.31.18.168

CloudShell Feedback Console Mobile App © 2026, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

```
Complete!
[ec2-user@ip-172-31-18-168 ~]$ sudo dnf install python3-pip -y
```

i-088638fd7729be1eb (CareerCounselor-Server)

PublicIPs: 3.80.67.156 PrivateIPs: 172.31.18.168

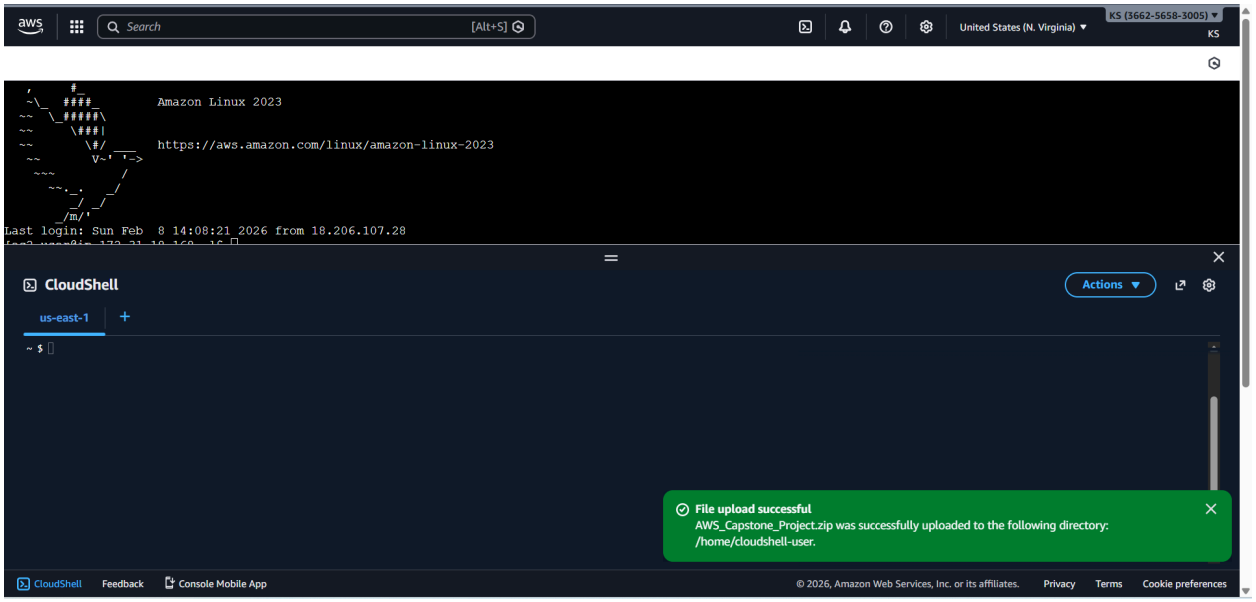
CloudShell Feedback Console Mobile App © 2026, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

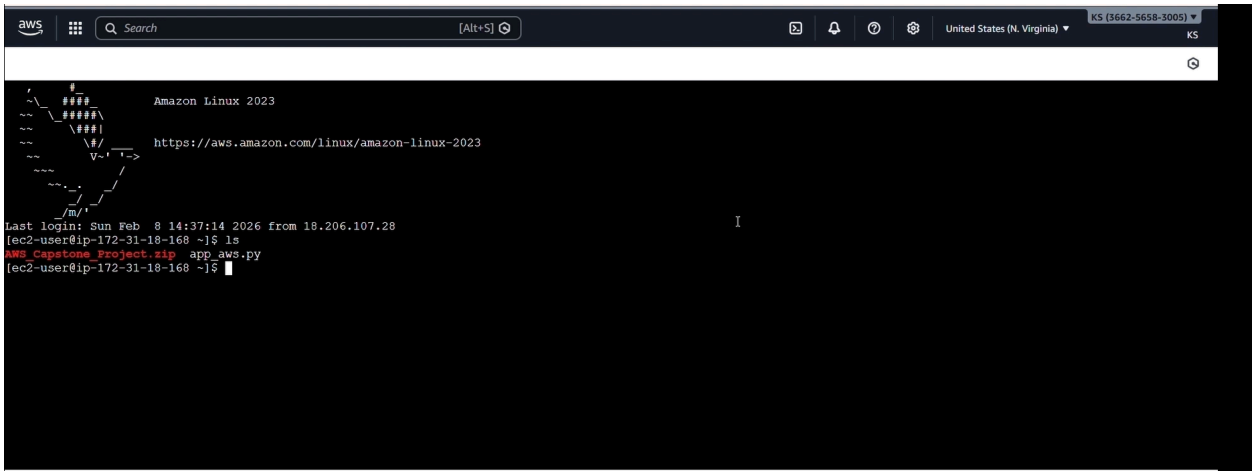
6°C Clear Search ENG IN 19:43 08-02-2026

aws Search [Alt+S] United States (N. Virginia) KS (3662-5658-3005) KS

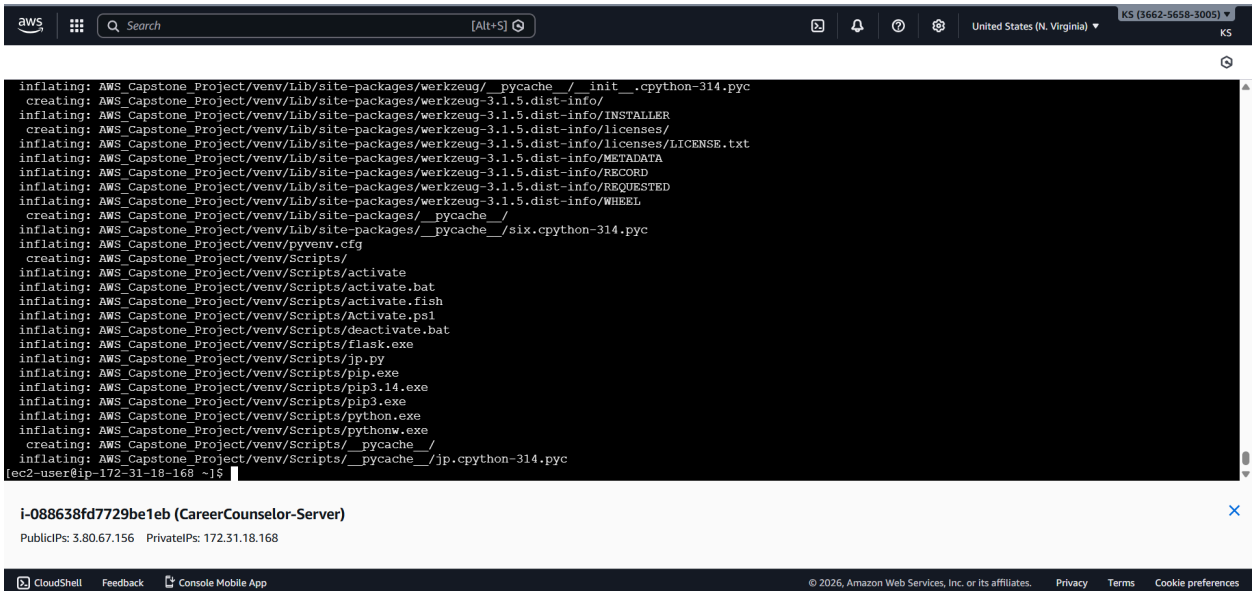
```
================================================================================================================================
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-18-168 ~]$ sudo dnf install python3-pip -y
Last metadata expiration check: 0:00:31 ago on Sun Feb  8 14:13:06 2026.
Dependencies resolved.
================================================================================================================================
 Package                         Architecture            Version                          Repository                    Size
================================================================================================================================
Installing:
 python3-pip                     noarch                  21.3.1-2.amzn2023.0.15           amazonlinux                  1.8 M
Installing weak dependencies:
 libxcrypt-compat                x86_64                  4.4.33-7.amzn2023                amazonlinux                   92 k

Transaction Summary
================================================================================================================================
Install  2 Packages

Total download size: 1.9 M
Installed size: 11 M
Downloading Packages:
(1/2): libxcrypt-compat-4.4.33-7.amzn2023.x86_64.rpm                             2.2 MB/s |  92 kB     00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.15.noarch.rpm                              30 MB/s | 1.8 MB     00:00
--------------------------------------------------------------------------------------------------------------------------------
Total                                                                             19 MB/s | 1.9 MB     00:00
Running transaction check
Transaction check succeeded.
```

➢ Uploading zip file of AWS_Capstone_Project to cloudshell

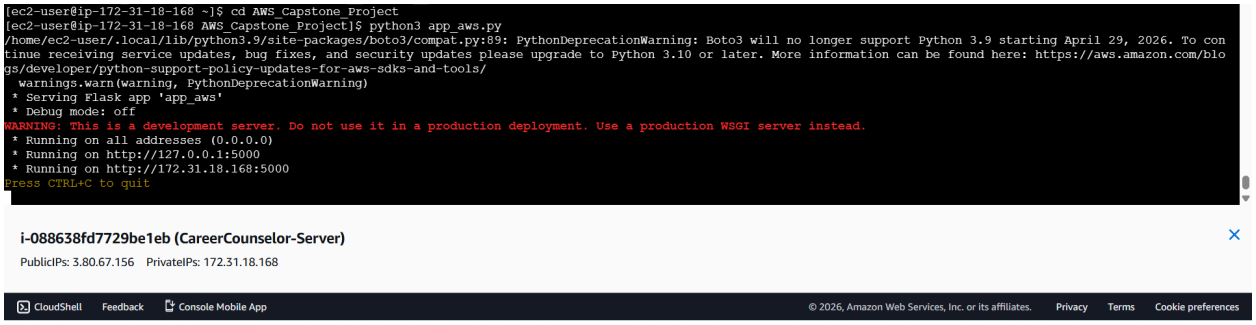




➢ Unzipping AWS_Capstone_Project folder



- **Functional Testing**
  - ➢ Test the app.py application for functionality, including database interactions and frontend features.
  - ➢ Run the Flask app **python3 app.py**
  - ➢ It will give you the link
  - ➢ Access the website through:



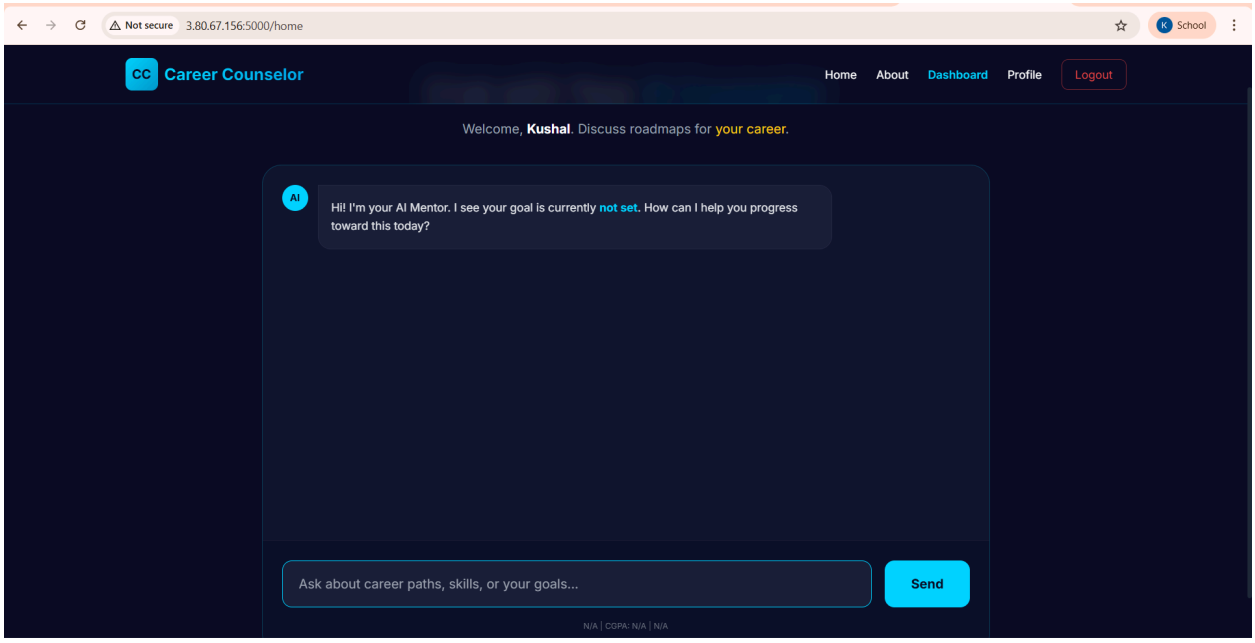
➢ PublicIPs**: http:3.80.67.156.5000** (Now no longer available)

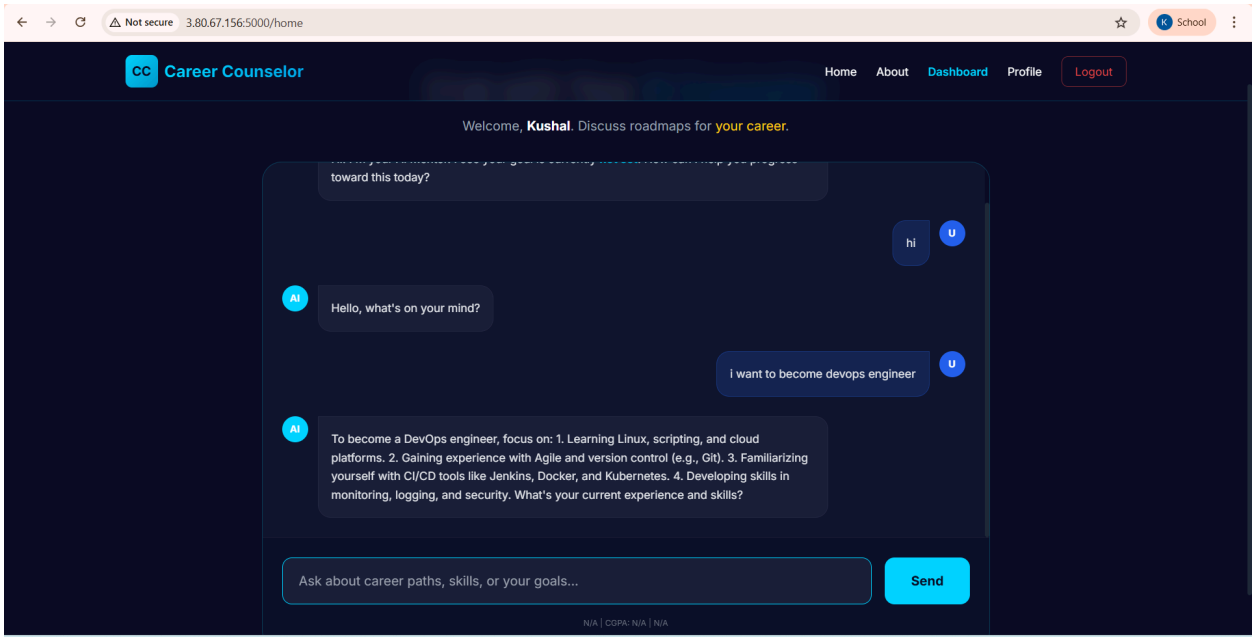
SMARTBRIDGE

Activity-2:

- ➢ Deploy the application in a production environment, ensuring high availability and performance
- ➢ Click on the link above and it will take you to the webpage:

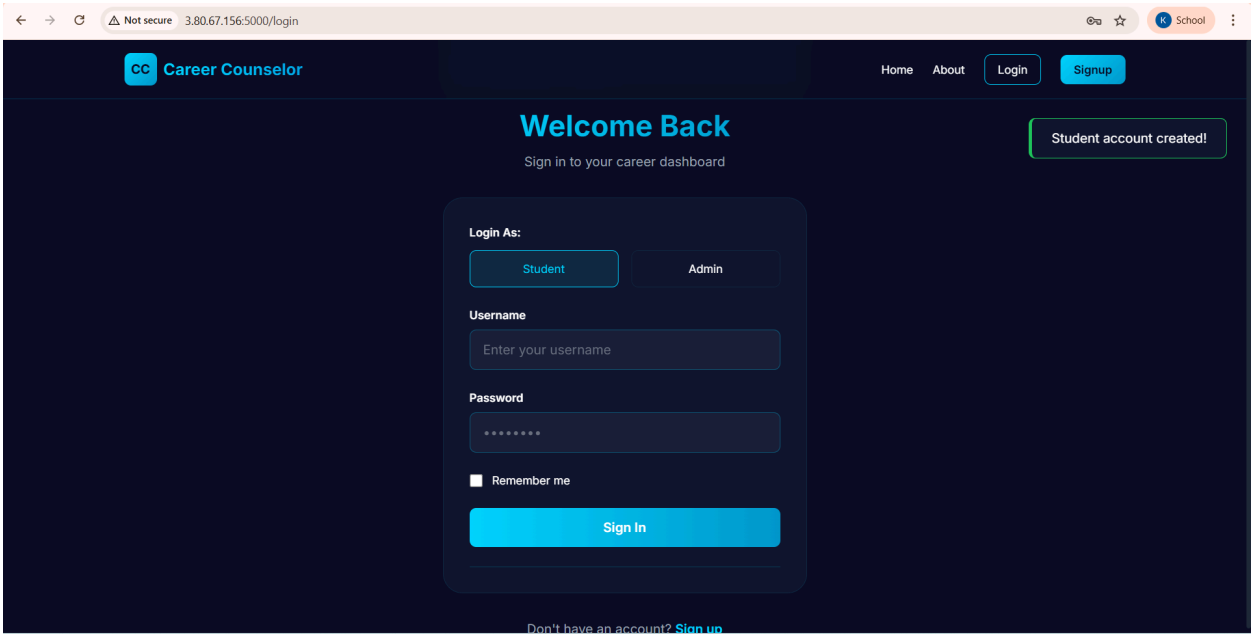
- **Home Page:**



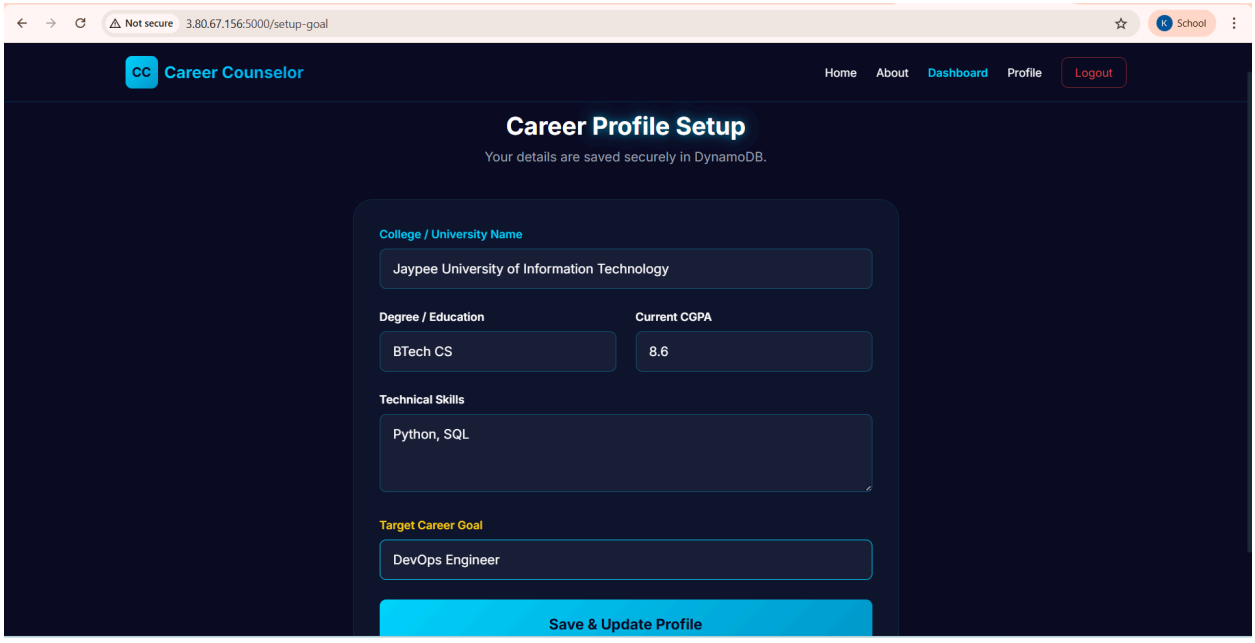


SMARTBRIDGE

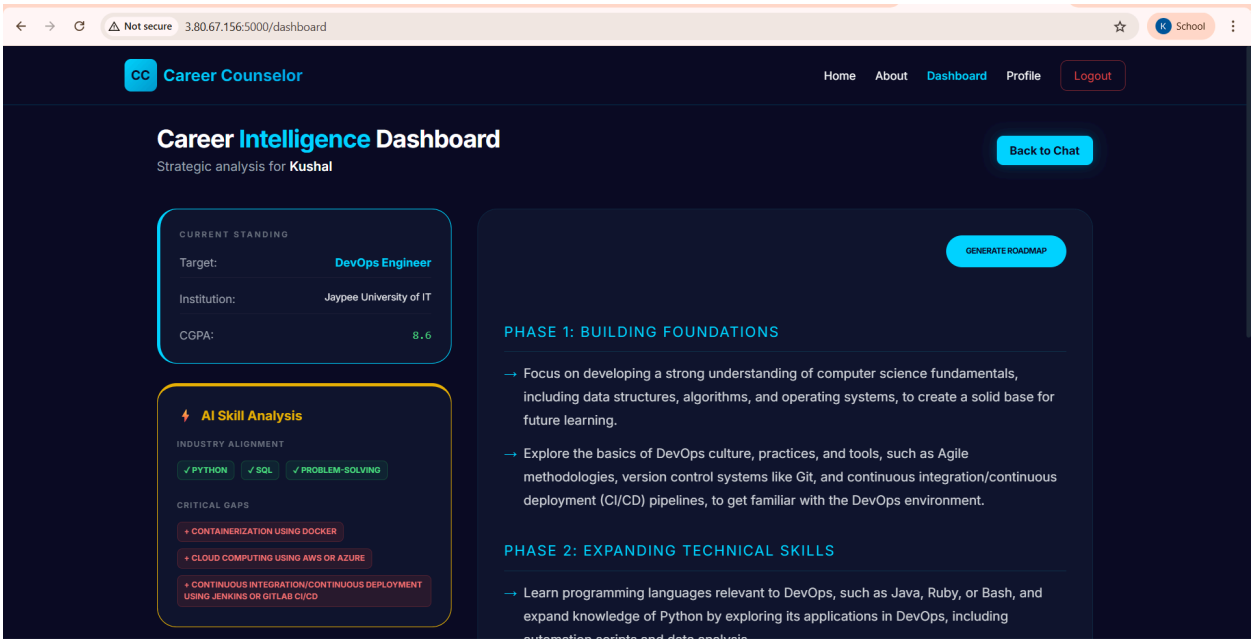
- **Login Page:**

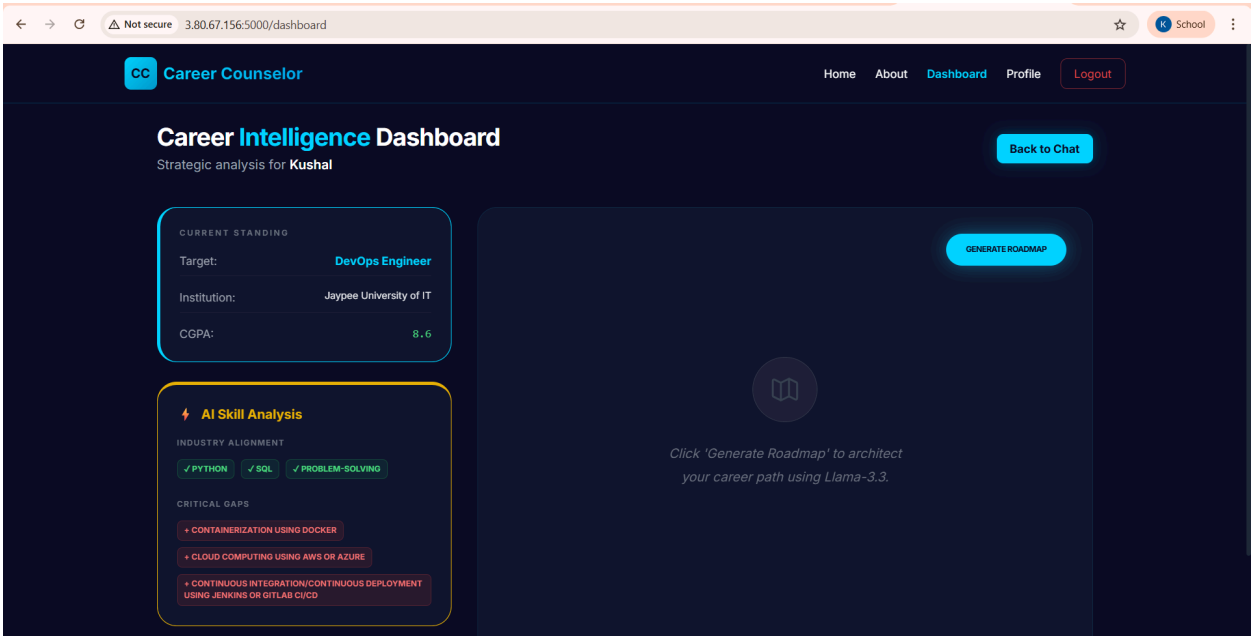


- **Setup-Goal Page:**



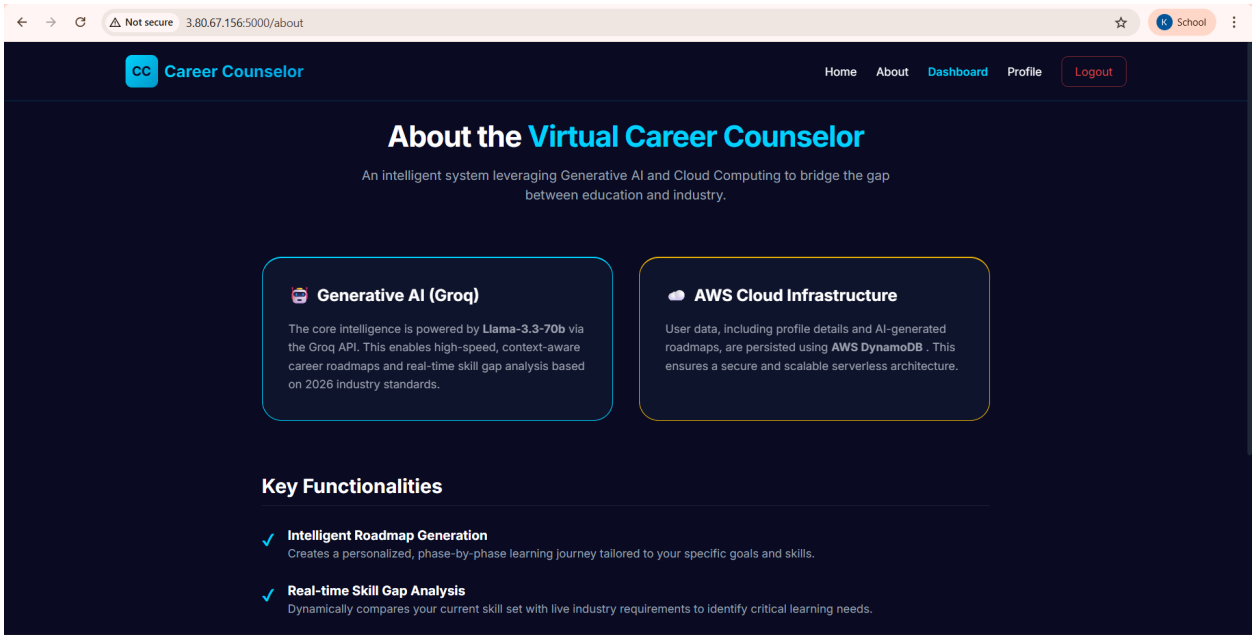
- **Dashboard Page:**



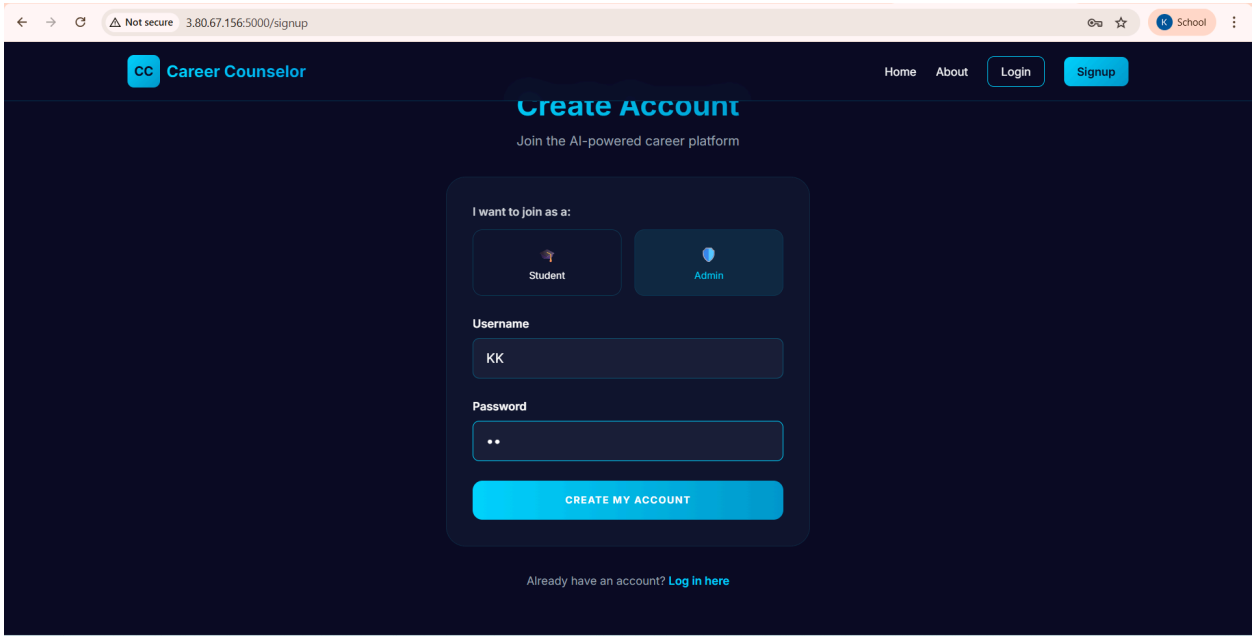


SMARTBRIDGE

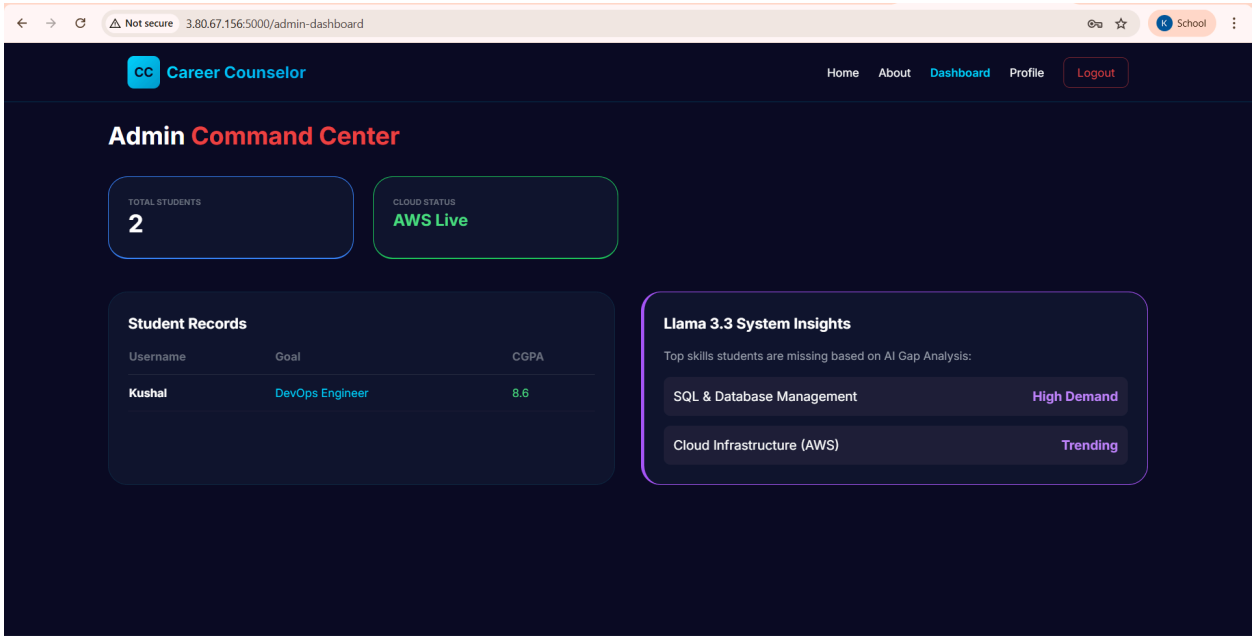
- **About Page:**



- **Signup Page:**

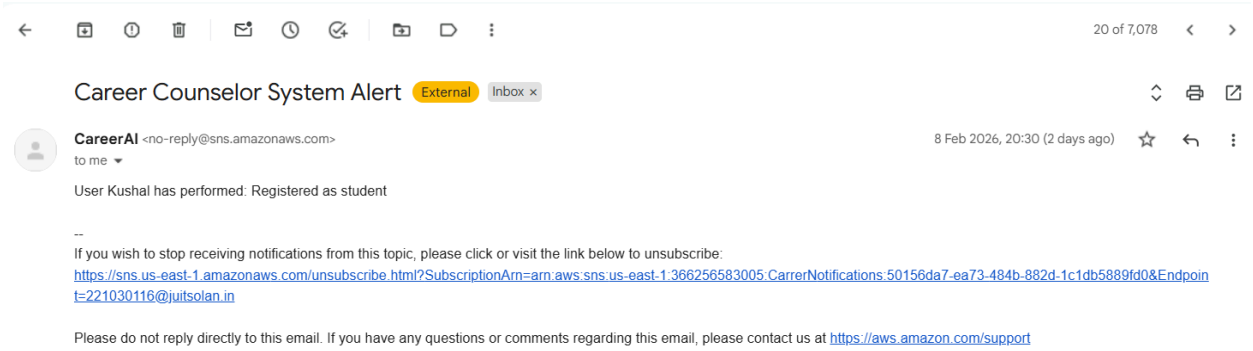


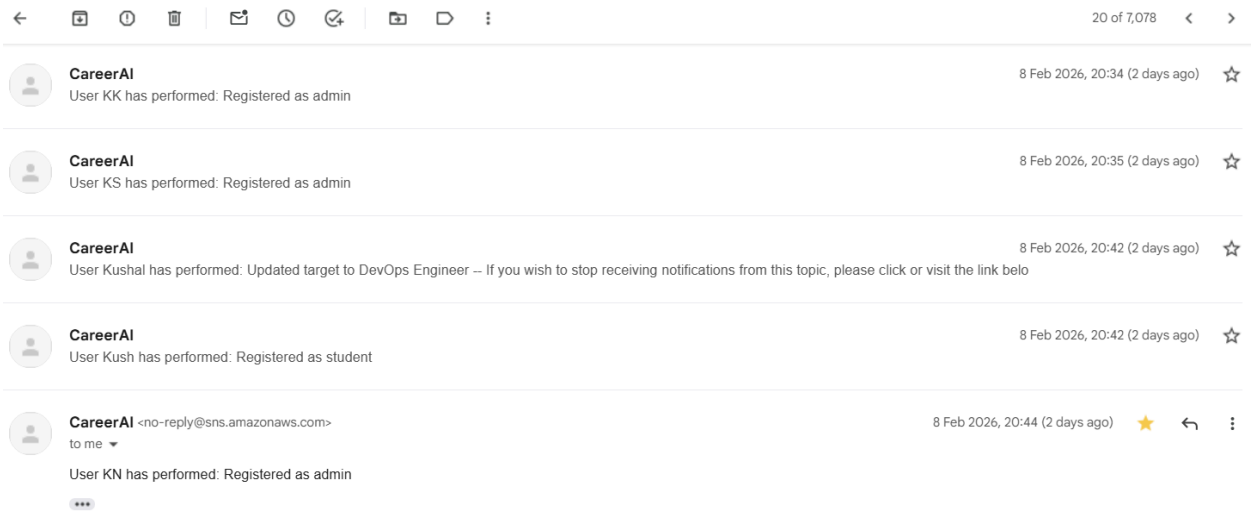
- **Admin-Dashboard Page:**



SMARTBRIDGE

- **CarrerAI Notifications:**





- **Conclusion:**

  The Virtual Career Counselor has been successfully developed as a cloud-based application, leveraging AWS services and Groq's Generative AI capabilities to provide users with personalized career guidance. Key AWS components like DynamoDB and IAM enable secure and scalable data storage and access management, ensuring a robust framework that can support a growing user base. With a Flask backend managing core functionalities, the application allows users to register, log in, and explore personalized career insights, enhancing their experience through easy navigation and secure data handling. Comprehensive testing confirmed seamless performance across features, including user authentication, data security, and API-based recommendations. The integration with Groq's API powers the platform's core offering—providing tailored career paths, course recommendations, and job market trend analyses. This AI-driven approach enables the Virtual Career Counselor to deliver dynamic and relevant insights based on individual user profiles, helping users make informed decisions about their career development. Ultimately, the Virtual Career Counselor showcases how AI and cloud services can together drive innovation in personalized career guidance, making the path to career exploration more intuitive and accessible.