

TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using AWS

Project Description:

TravelGo is a full-stack, cloud-based travel booking platform designed to simplify the process of reserving buses, trains, flights, and hotels through a unified interface. Built using Flask as the backend framework, the application is deployed on Amazon EC2 and leverages DynamoDB for efficient storage of user data and bookings. TravelGo allows users to register, log in, search for transportation and accommodation options, and book their travel with ease. Once a booking is confirmed or cancelled, users receive real-time email notifications powered by AWS Simple Notification Service (SNS), keeping them informed throughout their journey.

The platform's user-friendly interface supports dynamic seat selection for buses, hotel filtering based on preferences such as luxury or budget, and provides booking summaries along with centralized cancellation management. By combining cloud scalability, responsive design, and secure session handling, TravelGo delivers a seamless and real-time travel planning experience for users.

Scenario 1: Hassle-Free Multi-Mode Travel Booking Experience

TravelGo offers users a unified platform to search and book buses, trains, flights, and hotels all in one place. For instance, a user planning a trip from Hyderabad to Bangalore can log in, select their preferred mode of transport, choose from available options, and proceed to booking. Flask manages the backend operations such as retrieving travel listings and processing user input in real-time. Hosted on AWS EC2, the platform remains responsive even during high-traffic hours like weekends or holiday seasons, allowing multiple users to browse and book without delay.

Scenario 2: Real-Time Booking Confirmation with AWS SNS

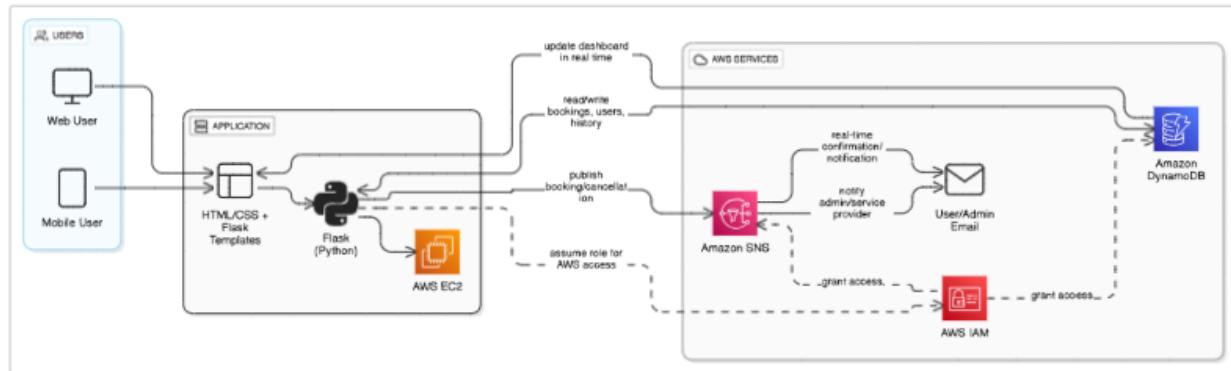
Once a booking is made—whether it's a train ticket or a hotel stay—TravelGo uses AWS SNS to instantly notify the user. For example, after a student books a hotel in Chennai, SNS sends a real-time email notification confirming the booking with all the relevant details. This notification is triggered from the Flask backend after the booking is successfully recorded in DynamoDB. Additionally, SNS can alert admin or service providers, ensuring transparency and real-time updates on every transaction.

Scenario 3: Dynamic Dashboard with Personal Travel History

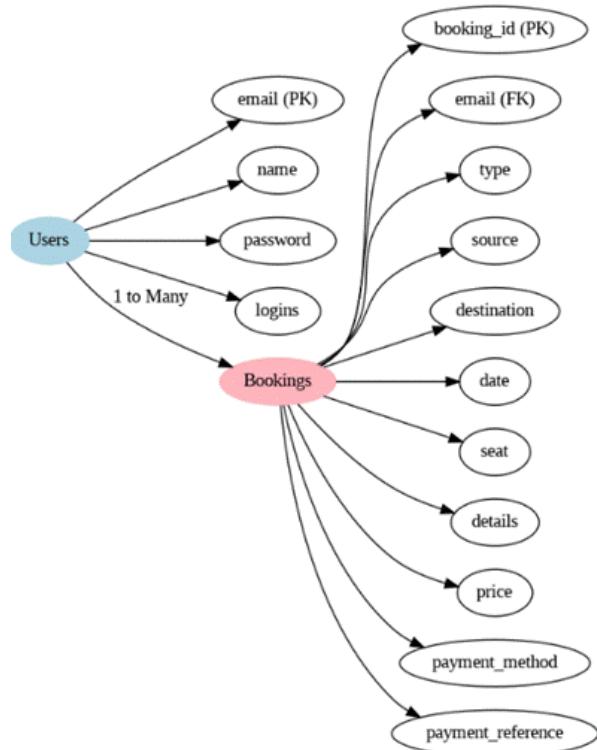
TravelGo features a dynamic user dashboard that displays all past and upcoming bookings for the logged-in user. For example, a user who has booked a flight and a hotel can view these bookings categorized by type, along with dates, price, and cancellation options. Flask fetches this data from AWS DynamoDB, which persistently stores all user bookings. The dashboard UI, powered by responsive HTML/CSS and Flask templates, ensures users can review or manage bookings.

anytime, from any device, with real-time updates and quick cancellation workflows supported.

Architecture:



Entity Relationship (ER) Diagram:



Pre-requisites:

- AWS Account Setup :
<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
- AWS IAM (Identity and Access Management) :
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
- AWS EC2 (Elastic Compute Cloud) :
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
- AWS DynamoDB :
<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>
- Amazon SNS :
<https://docs.aws.amazon.com/sns/latest/dg/welcome.html>
- Git Documentation :
<https://git-scm.com/doc>
- VS Code Installation : (download the VS Code using the below link or you can get that in Microsoft store)
<https://code.visualstudio.com/download>

Project WorkFlow:

Milestone 1. Backend Development and Application Setup

- Develop the Backend Using Flask.
- Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login

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

Milestone 3. DynamoDB Database Creation and Setup

- Create a DynamoDB Table.
- Configure Attributes for User Data and Book Requests.

Milestone 4. SNS Notification Setup

- Create SNS topics for book request notifications.
- Subscribe users and library staff to SNS email notifications.

Milestone 5. IAM Role Setup

- Create IAM Role
- Attach Policies

Milestone 6. EC2 Instance Setup

- Launch an EC2 instance to host the Flask application.
- Configure security groups for HTTP, and SSH access.

Milestone 7. Deployment on EC2

- Upload Flask Files
- Run the Flask App

Milestone 8. Testing and Deployment

- Conduct functional testing to verify user registration, login, book requests, and notifications.

Milestone 1. Backend Development and Application Setup

LOCAL DEPLOYMENT

```
✓ TRAVELGO1
  ✓ TravelGo1-main
    > static
    ✓ templates
      ◊ bus.html
      ◊ confirm_bus_details.html
      ◊ confirm_flight_details.html
      ◊ confirm_hotel_details.html
      ◊ confirm_train_details.html
      ◊ dashboard.html
      ◊ flight.html
      ◊ hotel.html
      ◊ index.html
      ◊ layout.html
      ◊ login.html
      ◊ register.html
      ◊ select_bus_seats.html
      ◊ select_flight_seats.html
      ◊ select_train_seats.html
      ◊ train.html
    > venv
    ✓ venv_new
      > Lib
      > Scripts
      ⚙ pyvenv.cfg
      ✎ app.py
      ⓘ README.md
```

Organize the project with HTML templates for each feature (e.g., login, wishlist, quiz, checkout) under the templates folder and manage backend logic in application.py.

Activity 3.1:Develop the Backend Using Flask

Import libraries:

```
from flask import Flask, render_template, request, redirect, url_for, session, jsonify, flash
import boto3
from boto3.dynamodb.conditions import Key, Attr
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
from decimal import Decimal
import uuid
import random
```

Import essential Flask modules for web handling, Boto3 for AWS integration, Werkzeug for password hashing, and datetime for timestamp management

Flask App Initialization:

```
# Initialize the Flask application
application = Flask(__name__)
application.secret_key = 'your_secret_key'
```

Initialize the Flask application and set a secret key to securely manage user sessions and form data.

Database Configuration:

```
# AWS Setup using IAM Role
REGION = 'us-east-1' # Replace with your actual AWS region
dynamodb = boto3.resource('dynamodb', region_name=REGION)
sns_client = boto3.client('sns', region_name=REGION)

users_table = dynamodb.Table('travelgo_users')
trains_table = dynamodb.Table('trains') # Note: This table is declared but not used in the provided routes
bookings_table = dynamodb.Table('bookings')
```

Connect to DynamoDB using Boto3 and define references to the UserTable and WishlistTable for user and wishlist data operations.

- **Routes for Core Functionalities:**

Home and registration routes:

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

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        # Check if user already exists
        # This uses get_item on the primary key 'email', so no GSI needed.
        existing = users_table.get_item(Key={'email': email})
        if 'Item' in existing:
            flash('Email already exists!', 'error')
            return render_template('register.html')

        # Hash password and store user
        hashed_password = generate_password_hash(password)
        users_table.put_item(Item={'email': email, 'password': hashed_password})
        flash('Registration successful! Please log in.', 'success')
        return redirect(url_for('login'))
    return render_template('register.html')
```

Description: Create the home and registration routes, where the registration route securely hashes user passwords and stores user data in DynamoDB upon form submission.

Login routes:

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

        # Retrieve user by email (primary key)
        user = users_table.get_item(Key={'email': email})

        # Authenticate user
        if 'Item' in user and check_password_hash(user['Item']['password'], password):
            session['email'] = email
            flash('Logged in successfully!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid email or password!', 'error')
            return render_template('login.html')
    return render_template('login.html')

@app.route('/logout')
def logout():
    session.pop('email', None)
    flash('You have been logged out.', 'info')
    return redirect(url_for('index'))
```

Description: Implement the user login route to validate credentials using DynamoDB and securely manage session data while updating the user's login count.

Booking Confirmation Routes:

```
@app.route('/final_confirm_bus_booking', methods=['POST'])
def final_confirm_bus_booking():
    if 'email' not in session:
        return redirect(url_for('login'))

    booking = session.pop('pending_booking', None)
    selected_seats_str = request.form.get('selected_seats') # Get as string

    if not booking or not selected_seats_str:
        flash("Booking failed! Missing data or session expired.", "error")
        return redirect(url_for("bus"))

    selected_seats = selected_seats_str.split(', ')

    # IMPORTANT CHANGE: Re-checking seat availability with a scan to prevent double booking
    response = bookings_table.scan(
        FilterExpression=Attr('item_id').eq(booking['item_id']) &
                        Attr('travel_date').eq(booking['travel_date']) &
                        Attr('booking_type').eq('bus')
    )
    existing_booked_seats = set()
    for b in response.get('Items', []):
        if 'seats_display' in b:
            existing_booked_seats.update(b['seats_display'].split(', '))

    # Check if any of the selected seats are already booked
    if any(s in existing_booked_seats for s in selected_seats):
        flash("One or more selected seats are already booked by another user. Please choose again.", "error")
        # Return to seat selection with current available seats
        booking['seats_display'] = ', '.join(selected_seats) # Keep selected seats for re-display
        session['pending_booking'] = booking # Put back into session
        return redirect(url_for('select_bus_seats',
                               name=booking['name'],
                               source=booking['source'],
                               destination=booking['destination'],
                               time=booking['time'],
```

User dashboard and wishlist routes:

```
# User dashboard route
@app.route('/user_dashboard')
def user_dashboard():
    if 'email' not in session:
        return redirect(url_for('login'))

    return render_template('user_dashboard.html')

# Add to wishlist route
@app.route('/add_to_wishlist', methods=['POST'])
def add_to_wishlist():
    if 'email' not in session:
        return redirect(url_for('login'))

    item_id = request.json['item_id']
    item_name = request.json['item_name']
    item_details = request.json['item_details']

    # Add item to WishlistTable in DynamoDB
    wishlist_table.put_item(
        Item={
            'email': session['email'],
            'item_id': item_id,
            'item_name': item_name,
            'item_details': item_details,
            'added_date': datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S')  # store the current UTC date
        }
    )
    return jsonify({'success': True, 'message': 'Item added to wishlist'})
```

Description: Secure the user dashboard and implement a route to add items to the wishlist, storing them in DynamoDB with item details and a timestamp.

```

# View wishlist route
@app.route('/wishlist')
def wishlist():
    if 'email' not in session:
        return redirect(url_for('login'))

    # Retrieve wishlist items from DynamoDB for the logged-in user
    response = wishlist_table.query(
        KeyConditionExpression=boto3.dynamodb.conditions.Key('email').eq(session['email'])
    )
    wishlist_items = response.get('Items', []) # Ensure all items are passed to the frontend
    -----
    return render_template('wishlist.html')

@app.route('/wishlist_data')
def wishlist_data():
    if 'email' not in session:
        return jsonify({'wishlist': []})

    response = wishlist_table.query(
        KeyConditionExpression=boto3.dynamodb.conditions.Key('email').eq(session['email'])
    )
    wishlist_items = response.get('Items', [])
    return jsonify({'wishlist': wishlist_items})

# Route to remove an item from the wishlist
@app.route('/remove_from_wishlist', methods=['POST'])
def remove_from_wishlist():
    if 'email' not in session:
        return jsonify({'success': False, 'message': 'Not logged in'})

    item_id = request.json.get('item_id')

    if not item_id:
        return jsonify({'success': False, 'message': 'Item ID not provided'})

    # Remove the item from WishlistTable in DynamoDB
    try:
        wishlist_table.delete_item(
            Key={
                'email': session['email'],
                'item_id': item_id
            }
        )
        return jsonify({'success': True, 'message': 'Item removed from wishlist'})
    except Exception as e:
        return jsonify({'success': False, 'message': f'Error: {str(e)}'})

```

Description: Build routes to display, fetch, and delete wishlist items for logged-in users by querying and modifying entries in DynamoDB using the user's email.

Dedicated Search Pages Routes:

```
@app.route('/flights')
def flights_page():
    return render_template('flight.html')

@app.route('/hotels')
def hotels_page():
    return render_template('hotel.html')

@app.route('/trains')
def trains_page():
    return render_template('trains.html')

@app.route('/buses')
def buses_page():
    return render_template('bus.html')
```

Description: Create a route for the virtual exhibition page where users can add detailed jewelry items to their wishlist, with proper validation and storage in DynamoDB.

Quiz Route:

```
# Quiz page and submission
@app.route('/quiz', methods=['GET', 'POST'])
def quiz():
    if request.method == 'POST':
        score = int(request.form.get('score', 0))
        if score >= 12:
            session['won_quiz'] = True # Set the status for quiz win
        else:
            session['won_quiz'] = False
        return redirect(url_for('user_dashboard'))
    return render_template('quiz.html')
```

Description: Create a route to handle quiz submissions, storing the result in the session if the user scores 12 or more, and redirecting to the user dashboard.

Check out route:

```
        elif data['action'] == 'update_quantity':
            # Handle quantity updates
            item_name = data['item_name']
            quantity = int(data['quantity'])
            checkout_items = session.get('checkout_items', [])

            for item in checkout_items:
                if item['name'] == item_name:
                    item['quantity'] = quantity
                    item['total_price'] = int(item['price'].replace(',', '').split(' ')[0]) * quantity
                    break

            subtotal = sum(item['total_price'] for item in checkout_items)
            discount = session.get('discount', 0)
            total_price = subtotal - discount

            session['checkout_items'] = checkout_items
            session['subtotal'] = subtotal
            session['final_price'] = total_price
            session.modified = True

            return jsonify({'success': True, 'total_price': total_price})

        elif data['action'] == 'remove':
            # Handle item removal
            item_name = data['item_name']
            checkout_items = session.get('checkout_items', [])
            updated_checkout_items = [item for item in checkout_items if item['name'] != item_name]

            subtotal = sum(
                int(item['price'].replace(',', '').split(' ')[0]) * item.get('quantity', 1)
                for item in updated_checkout_items
            )
            discount = session.get('discount', 0)
            total_price = subtotal - discount

            session['checkout_items'] = updated_checkout_items
            session['subtotal'] = subtotal
            session['final_price'] = total_price
            session.modified = True

            return jsonify({'success': True, 'message': f"Item {item_name} removed from checkout!", 'total_price': total_price})

        elif data['action'] == 'finalize':
            # Finalize checkout and save order items
            session['order_items'] = session.get('checkout_items', [])
            session.modified = True

            return jsonify({'success': True, 'redirect': url_for('order')})

        return jsonify({'success': False, 'message': 'Invalid action!'})

@app.route('/checkout_items', methods=['GET'])
def checkout_items():
    # Retrieve checkout items for rendering on the frontend
    if 'email' not in session:
        return jsonify({'checkout_items': []})
    return jsonify({'checkout_items': session.get('checkout_items', [])})
```

Description: Implement logic for updating item quantity, removing items, and finalizing orders in the shopping cart, while maintaining session-based checkout data and price calculations.

Add to checkout route:

```
3     @app.route('/add_to_checkout', methods=['POST'])
4     def add_to_checkout():
5         if 'email' not in session:
6             return jsonify({'success': False, 'message': 'Not logged in'})
7
8         item_id = request.json.get('item_id')
9         if not item_id:
10             return jsonify({'success': False, 'message': 'Item ID missing'})
11
12         # Fetch the item from wishlist
13         try:
14             response = wishlist_table.get_item(
15                 Key={'email': session['email'], 'item_id': item_id}
16             )
17             item = response.get('Item')
18             if not item:
19                 return jsonify({'success': False, 'message': 'Item not found in wishlist'})
20
21             # Prepare item for checkout session
22             checkout_item = {
23                 'item_id': item['item_id'],
24                 'item_name': item['item_name'],
25                 'price': item['item_details'].split('Price: ')[-1], # e.g., "3,50,000 INR"
26                 'image': item.get('item_image', ''),
27                 'details': item.get('item_details', ''),
28                 'quantity': 1
29             }
30
31             # Initialize or update checkout session
32             checkout_items = session.get('checkout_items', [])
33             existing = next((i for i in checkout_items if i['item_id'] == item_id), None)
34
35             if not existing:
36                 checkout_items.append(checkout_item)
37                 session['checkout_items'] = checkout_items
38                 session.modified = True
39
40             return jsonify({'success': True, 'message': 'Item added to checkout'})
41         except Exception as e:
42             return jsonify({'success': False, 'message': str(e)})
```

Description: Enable users to add wishlist items to the checkout session by retrieving item details from DynamoDB and updating the session state if the item is not already present.

Cancel Booking Route:

```
@app.route('/cancel_booking', methods=['POST'])
def cancel_booking():
    if 'email' not in session:
        return redirect(url_for('login'))

    booking_id = request.form.get('booking_id')
    user_email = session['email']
    booking_date = request.form.get('booking_date') # This is crucial as it's the sort key

    if not booking_id or not booking_date:
        flash("Error: Booking ID or Booking Date is missing for cancellation.", 'error')
        return redirect(url_for('dashboard'))

    try:
        # Delete item using the primary key (user_email and booking_date)
        # This does not use GSI, so it remains unchanged.
        bookings_table.delete_item(
            Key={'user_email': user_email, 'booking_date': booking_date}
        )
        flash(f"Booking {booking_id} cancelled successfully!", 'success')
    except Exception as e:
        flash(f"Failed to cancel booking {booking_id}: {str(e)}", 'error')

    return redirect(url_for('dashboard'))
```

Description: Handle order placement by collecting customer details, processing the session-based checkout data, and clearing the session after successfully placing the order.

Initiating Flask app:

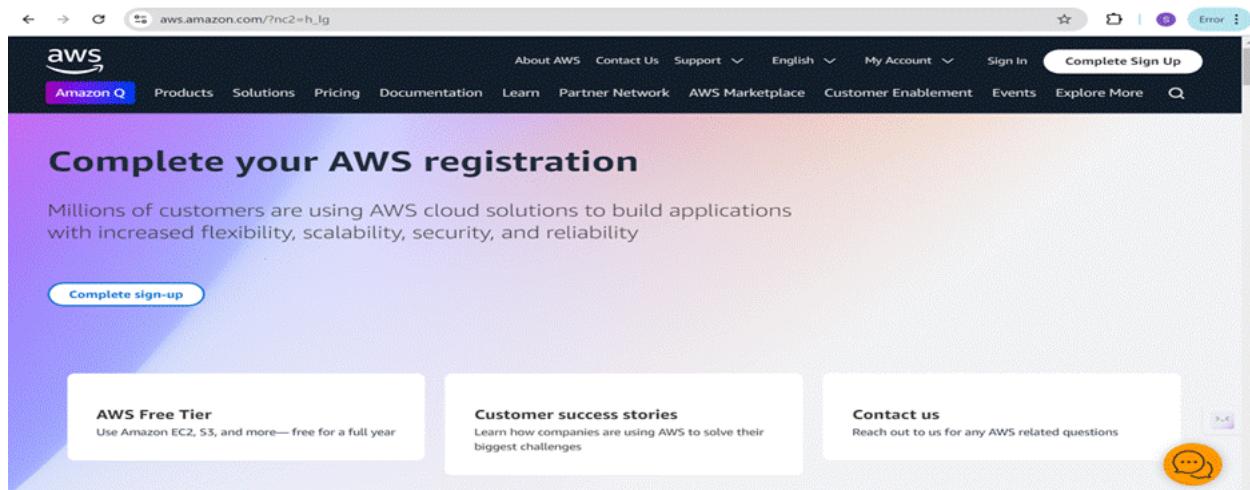
```
# Run the Flask app
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80, debug=True)
```

Description: Start the Flask application on host `0.0.0.0` and port `80` in debug mode to enable live reloads and detailed error logs during development.

Milestone 2 : AWS Account Setup

Set up an AWS account if not already done.

Sign up for an AWS account and configure billing settings.



Log in to the AWS Management Console

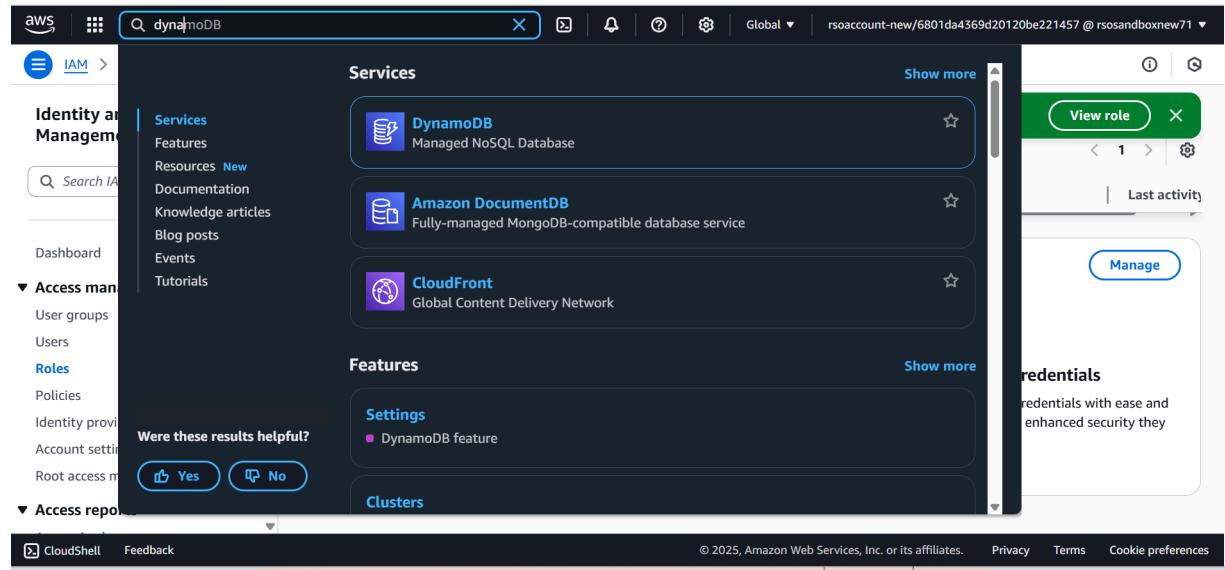
After setting up your account, log in to the [AWS Management Console](#).

A screenshot showing two side-by-side sections. On the left is the AWS sign-in page, which asks for a 'Root user' or 'IAM user' choice, a root user email address (example@example.com), and a 'Next' button. A small note at the bottom states: 'By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our Cookie Notice for more information.' On the right is a dark purple banner for the 'AI Use Case Explorer', featuring the text 'Discover AI use cases, customer success stories, and expert-curated implementation plans' and a 'Explore now >' button.

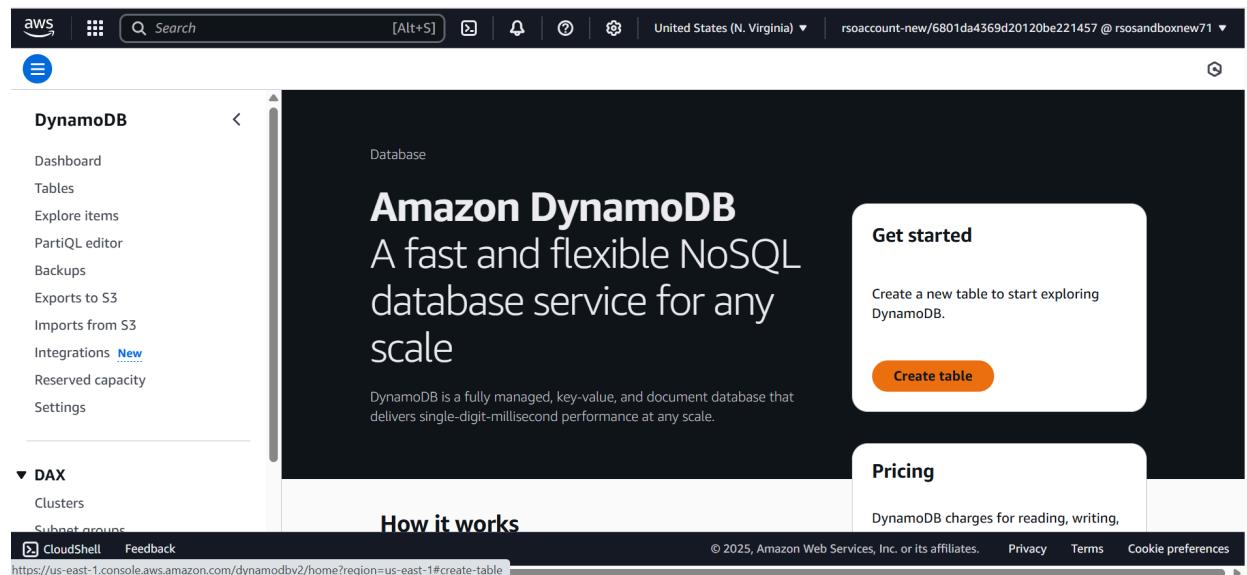
Milestone 3: DynamoDB Database Creation and Setup

Navigate to the DynamoDB

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



The screenshot shows the AWS IAM (Identity and Access Management) service page. The top navigation bar includes the AWS logo, a search bar containing "dynamo", and various global settings like notifications and regions. The left sidebar is titled "IAM" and contains sections for Identity and Access Management, Access management, and Access reporting. The main content area displays a list of services under "Services", with "DynamoDB" highlighted as a "Managed NoSQL Database". Below this is a section for "Amazon DocumentDB" and "CloudFront". Under "Features", there's a "Settings" section with a note about "DynamoDB feature". A feedback poll at the bottom asks if the results were helpful, with "Yes" and "No" buttons. The footer includes links for CloudShell, Feedback, and legal notices.



The screenshot shows the Amazon DynamoDB home page. The top navigation bar includes the AWS logo, a search bar, and regional information for "United States (N. Virginia)". The left sidebar has sections for "DynamoDB" (Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, Settings), "DAX" (Clusters, Subnet groups), and CloudShell/Feedback. The main content area features a large "Amazon DynamoDB" heading with the subtext "A fast and flexible NoSQL database service for any scale". It describes DynamoDB as a fully managed, key-value, and document database. A "Get started" box with a "Create table" button is present, along with a "Pricing" section. The footer includes copyright and legal links.

Create a DynamoDB table for storing Data.

Create a travel-Users table for storing registered user details with partition key “Email” with type

String and click on create tables.

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The 'Table details' step is active. A table named 'travelgo_users' is being created with 'email' as the partition key (String type) and no sort key defined. The 'CloudShell' and 'Feedback' buttons are at the bottom left, and copyright information is at the bottom right.

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The 'Dynamodb features' step is active. It shows 'Resource-based policy' selected for 'Resource protection'. A note indicates that auto scaling will be deactivated. The 'Cancel' and 'Create table' buttons are at the bottom right.

- Follow the same steps to create a Bookings for storing booking records with email as the partition key and booking_date as the sort key.

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.
 bookings
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
 user_email String
1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 booking_date String
1 to 255 characters and case sensitive.

Table settings

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Resource protection Off On

Resource-based policy Not active Yes

Tags
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.
No tags are associated with the resource.
 Add new tag
You can add 50 more tags.

Note This table will be created with auto scaling deactivated. You do not have permissions to turn on auto scaling.

Cancel Create table

- Follow the same steps to create a trains for storing train number with date as the partition key and train_number as the sort key.

aws | Search [Alt+S] | United States (N. Virginia) | rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71

DynamoDB > Tables > Create table

Create table

Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
 String
1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 String
1 to 255 characters and case sensitive.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

aws | Search [Alt+S] | United States (N. Virginia) | rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71

DynamoDB > Tables > Create table

Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.
No tags are associated with the resource.
[Add new tag](#)
You can add 50 more tags.

Info This table will be created with auto scaling deactivated. You do not have permissions to turn on auto scaling.

[Cancel](#) [Create table](#)

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the AWS DynamoDB console. On the left, there's a navigation sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a section for DAX with Clusters and Subnet groups. The main area is titled 'Tables (3) Info' and shows three tables: 'bookings', 'trains', and 'travelgo_users'. A success message at the top says 'The trains table was created successfully.' There are buttons for Actions, Delete, and Create table.

Milestone 4: SNS Notification Setup

- In the AWS Console, search for SNS and navigate to the SNS Dashboard.

The screenshot shows the AWS SNS dashboard. The left sidebar includes links for Services, Features, Resources, Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. A feedback section asks if the results were helpful, with 'Yes' and 'No' buttons. The main content area is titled 'Services' and lists 'Simple Notification Service' (SNS managed message topics for Pub/Sub), 'Route 53 Resolver' (Resolve DNS queries in your Amazon VPC and on-premises network), and 'Route 53' (Scalable DNS and Domain Name Registration). Below this is a 'Features' section with 'Events' (ElastiCache feature) and 'SMS'.

- Click on Create Topic and choose a name for the topic.

The screenshot shows the AWS SNS Topics page. On the left, there's a sidebar with links for Dashboard, Topics (which is selected), Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and CloudShell. The main area has a blue banner at the top stating "New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more". Below it, a table header for "Topics (0)" includes columns for Name, Type, and ARN, with sorting arrows. A search bar and buttons for Edit, Delete, Publish message, and Create topic are at the top right. A message says "No topics. To get started, create a topic." with a "Create topic" button. At the bottom, there are links for CloudShell, Feedback, and copyright information.

Choose Standard type for general notification use cases and name as Travelgo1 and Click on Create Topic.

The screenshot shows the "Create topic" wizard. It starts with a note that "Topic type cannot be modified after topic is created". Two options are shown: "FIFO (first-in, first-out)" and "Standard". "Standard" is selected, with a description: "Best-effort message ordering", "At-least once message delivery", and "Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints". Below this, the "Name" field is filled with "TravelGo1", and the "Display name - optional" field contains "My Topic". At the bottom, there's a section for "Encryption - optional" with a note about default in-transit encryption. The footer includes CloudShell, Feedback, and copyright information.

Access policy - optional Info

This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

Data protection policy - optional Info

This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

Delivery policy (HTTP/S) - optional Info

The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

Delivery status logging - optional Info

These settings configure the logging of message delivery status to CloudWatch Logs.

Tags - optional

A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)

Active tracing - optional Info

Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

[Cancel](#)

[Create topic](#)

Configure the SNS topic and note down the Topic ARN.

The screenshot shows the AWS SNS Topics page. On the left, there's a sidebar with 'Amazon SNS' and links for 'Dashboard', 'Topics' (which is selected), and 'Subscriptions'. Below that is a 'Mobile' section with 'Push notifications' and 'Text messaging (SMS)'. The main content area has a blue banner at the top stating 'New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more'. Below this, a green banner says 'Topic Travelgo created successfully. You can create subscriptions and send messages to them from this topic.' with 'Publish message' and 'Edit' buttons. The 'Travelgo' topic card itself has a 'Details' section with fields: 'Name' (TravelGo1), 'ARN' (arn:aws:sns:us-east-1:047719615036:TravelGo1), 'Type' (Standard), 'Display name' (empty), and 'Topic owner' (047719615036). At the bottom of the page are links for 'CloudShell', 'Feedback', '© 2025, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

Subscribe users

- Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.

The screenshot shows the AWS SNS Subscriptions page for the 'Travelgo' topic. The top navigation bar includes the AWS logo, search bar, and various icons. The left sidebar has links for Dashboard, Topics (selected), Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and CloudShell/Feedback. The main content area shows a table header for 'Subscriptions (0)' with columns for ID, Endpoint, Status, and Protocol. Below the table, a message says 'No subscriptions found' and 'You don't have any subscriptions to this topic.' A 'Create subscription' button is at the bottom. The top right shows 'United States (N. Virginia)', 'rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71', and a gear icon.

- After subscription request for the mail confirmation

The screenshot shows the 'Create subscription' wizard. Step 1: Enter Endpoint. It asks for an email address to receive notifications from Amazon SNS. The input field contains 'udayarakesh2004@gmail.com'. A note below says 'After your subscription is created, you must confirm it.' A 'Subscription filter policy - optional' section is shown with a note about filtering messages. A 'Redrive policy (dead-letter queue) - optional' section is also shown with a note about sending undeliverable messages to a dead-letter queue. At the bottom are 'Cancel' and 'Create subscription' buttons. The footer includes CloudShell/Feedback, © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.

The screenshot shows a Gmail inbox with the following details:

- Title:** AWS Notification - Subscription Confirmation
- To:** AWS Notifications (to me)
- Date:** Wed 2 Jul, 15:08 (3 days ago)
- Content Preview:** You have chosen to subscribe to the topic: arn:aws:sns:us-east-1:047719615036:TravelGo1
To confirm this subscription, click or visit the link below (If this was in error no action is necessary): [Confirm subscription](#)
Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)
- Buttons:** Reply, Forward, Delete

A notification bar at the bottom says "Enable desktop notifications for Gmail. OK No, thanks".

The screenshot shows an AWS Simple Notification Service (SNS) confirmation email with the following details:

- Subject:** Subscription confirmed!
- Text Content:** You have successfully subscribed.
Your subscription's id is:
arn:aws:sns:us-east-1:863518417312:Travelgo:4d0ab424-e011-467f-a4e3-64a598a296ea
If it was not your intention to subscribe, [click here to unsubscribe](#).

- Successfully done with the SNS mail subscription and setup, now store the ARN link.

The screenshot shows the AWS SNS console. In the top navigation bar, the path is: Amazon SNS > Topics > TravelGo1 > Subscription: 2888f19c-04ee-4400-96cc-ec1576d79fb4. A blue banner at the top right says "New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more". The main area displays the subscription details for the topic "TravelGo1". The ARN is listed as arn:aws:sns:us-east-1:047719615036:TravelGo1:2888f19c-04ee-4400-96cc-ec1576d79fb4. The endpoint is udayarakesh2004@gmail.com. The status is "Confirmed" and the protocol is "EMAIL". Below this, there are tabs for "Subscription filter policy" (which is selected) and "Redrive policy (dead-letter queue)". A note below the tabs states: "No filter policy configured for this subscription. To apply a filter policy, edit this subscription." At the bottom of the page, there are links for CloudShell, Feedback, and a footer with copyright information and links for Privacy, Terms, and Cookie preferences.

Milestone 5 : IAM Role Setup

Create IAM Role.

- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.

The screenshot shows the AWS IAM service page. The search bar at the top contains the text "iam". The main content area is titled "Services" and lists three services: "IAM" (Manage access to AWS resources), "IAM Identity Center" (Manage workforce user access to multiple AWS accounts and cloud applications), and "Resource Access Manager" (Share AWS resources with other accounts or AWS Organizations). Below the services, there is a section titled "Features" which includes a "Groups" item under the "IAM feature" category. On the left side, there is a sidebar titled "Console" with recent activity items: EC2, Simple Storage Service (S3), and IAM. A red box highlights the "IAM" item in the sidebar. At the bottom of the page, there is a feedback section asking "Were these results helpful?" with "Yes" and "No" buttons, and a footer with links for CloudShell, Feedback, and standard AWS footer links.

Identity and Access Management (IAM)

Roles (10) 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	Last activity
AWSServiceRoleForAccessAnalyzer	AWS Service: access-analyzer (Service-Linked Role)	124 days ago
AWSServiceRoleForAmazonEKS	AWS Service: eks (Service-Linked Role)	145 days ago
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service-Linked Role)	-
AWSServiceRoleForECS	AWS Service: ecs (Service-Linked Role)	134 days ago
AWSServiceRoleForOrganizations	AWS Service: organizations (Service-Linked Role)	209 days ago
AWSServiceRoleForSSO	AWS Service: sso (Service-Linked Role)	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-

Create role

Select trusted entity

Trusted entity type

- AWS service**
Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account**
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity**
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- SAML 2.0 federation**
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- Custom trust policy**
Create a custom trust policy to enable others to perform actions in this account.

The screenshot shows the 'Create role' wizard in the AWS IAM console. The current step is 'Set role type'. A sidebar on the left lists three steps: 'Step 1 Select trusted entity' (selected), 'Step 2 Add permissions' (selected), and 'Step 3 Name, review, and create'. The main content area shows a list of role types under 'EC2 role':

- EC2**
Allows EC2 instances to call AWS services on your behalf.
- EC2 Role for AWS Systems Manager**
Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.
- EC2 Spot Fleet Role**
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.
- EC2 - Spot Fleet Auto Scaling**
Allows Auto Scaling to access and update EC2 spot fleets on your behalf.
- EC2 - Spot Fleet Tagging**
Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.
- EC2 - Spot Instances**
Allows EC2 Spot Instances to launch and manage spot instances on your behalf.
- EC2 - Spot Fleet**
Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.
- EC2 - Scheduled Instances**
Allows EC2 Scheduled Instances to manage instances on your behalf.

At the bottom right are 'Cancel' and 'Next' buttons. The footer includes links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

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.
- **AmazonEc2FullAccess**:

The screenshot shows the 'Create role' wizard in the AWS IAM console, Step 2: 'Add permissions'. The sidebar shows 'Step 2 Add permissions' is selected. The main content area is titled 'Add permissions' and shows a search result for 'Permissions policies (1/1059)'. The result for 'AmazonEc2FullAccess' is highlighted with a blue border and labeled 'AWS managed' and 'Provides full access to Am'.

Permissions policies (1/1059) [Info](#)

Choose one or more policies to attach to your new role.

Filter by Type

Policy name	Type	Description
AmazonEc2FullAccess	AWS managed	Provides full access to Am

▶ Set permissions boundary - optional

Cancel Previous Next

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS IAM 'Create role' wizard, Step 2: Add permissions.

The search bar shows 'dynamodbfull'. The results table lists three policies:

Policy name	Type	Description
AmazonDynamoDBFullAccess	AWS managed	Provides full access to Am
AmazonDynamoDBFullAccess_v2	AWS managed	Provides full access to Am
AmazonDynamoDBFullAccesswithDataPipeline	AWS managed	This policy is on a depreca

Screenshot of the AWS IAM 'Create role' wizard, Step 2: Add permissions.

The search bar shows 'sns'. The results table lists five policies:

Policy name	Type	Description
AmazonSNSFullAccess	AWS managed	Provides full access to Am
AmazonSNSReadOnlyAccess	AWS managed	Provides read only access
AmazonSNSRole	AWS managed	Default policy for Amazon
AWSElasticBeanstalkRoleSNS	AWS managed	(Elastic Beanstalk operati
AWSIoTDeviceDefenderPublishFindingsToSNS...	AWS managed	Provides messages publish

Screenshot of the AWS IAM 'Create role' wizard, Step 3: Name, review, and create.

Role details

Role name: Studentuser

Description: Allows EC2 instances to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy:

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {
```

Step 3: Add tags

Add tags - optional

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

Add new tag

You can add up to 50 more tags.

Buttons: Cancel, Previous, Create role.

The screenshot shows the AWS IAM Roles page. At the top, there is a green success message box that says "Role Studentuser created." Below this, there is a search bar and a table with columns for "Role name", "Trusted entities", and "Last activity". The table has one row, which is collapsed. On the left side, there is a sidebar with sections for "Access management" (User groups, Users, Roles, Policies, Identity providers, Account settings, Root access management) and "Access reports". At the bottom, there is a footer with links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

Milestone 6 : EC2 Instance Setup

Load your Project Files to Github

- Launch EC2 Instance
- In the AWS Console, navigate to EC2 and launch a new instance.

The screenshot shows the AWS EC2 Instances page. The search bar at the top contains "ec2". The main content area features a card for the "EC2" service, which is described as "Virtual Servers in the Cloud". Below this, there are cards for "EC2 Image Builder" and "EC2 Global View". On the left, there is a sidebar with sections for "Amazon S" (Dashboard, Topics, Subscriptions), "Mobile" (Push notifications, Text messaging), and a "Were these results helpful?" poll with "Yes" and "No" buttons. At the bottom, there is a footer with links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

- Click on Launch instance to launch EC2 instance

The screenshot shows the AWS EC2 home page. On the left, there's a navigation sidebar with sections like Dashboard, EC2 Global View, Events, Instances (with sub-options like Instances, Instance Types, Launch Templates, etc.), and Images. The main content area features a large banner for 'Amazon Elastic Compute Cloud (EC2)' with the tagline 'Create, manage, and monitor virtual servers in the cloud.' Below the banner, a paragraph describes the service, mentioning over 600 instance types and various purchase models. To the right of the banner is a callout box titled 'Launch a virtual server' containing two buttons: 'Launch instance' (orange) and 'View dashboard' (blue). At the bottom of the main content area, there's a summary of the current launch configuration.

Launch an instance

Number of instances: 1

Software Image (AMI): Amazon Linux 2023 AMI 2023.7.2... [read more](#)

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Summary

Launch instance

Preview code

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

The screenshot shows the AWS EC2 'Launch an instance' wizard. In the 'Key pair (login)' section, a dropdown menu is open for selecting a key pair, with 'Select' highlighted. A 'Create new key pair' button is visible. In the 'Network settings' section, 'Network' is set to 'vpc-063cbc01c43c74b9a' and 'Subnet' is set to 'No preference (Default subnet in any availability zone)'. The 'Auto-assign public IP' option is enabled. On the right side, the 'Summary' section shows 'Number of instances' as 1, 'Software Image (AMI)' as 'Amazon Linux 2023 AMI 2023.7.2...', 'Virtual server type (instance type)' as 't2.micro', and a 'New security group' is selected. Buttons for 'Cancel', 'Launch instance', and 'Preview code' are at the bottom.

Create and download the key pair for Server access.

The screenshot shows the 'Create key pair' dialog box. The 'Key pair name' field contains 'travelgo (1)'. The 'Key pair type' section has 'RSA' selected, with 'ED25519' also listed as an option. Under 'Private key file format', the '.pem' option is selected. At the bottom, there are 'Cancel' and 'Create key pair' buttons.



InstantLibrary.pem

The screenshot shows the AWS EC2 Instances Launch Wizard. The top section is titled "Launch an instance" and includes a "Firewall (security groups)" step where a new security group named "launch-wizard-1" is being created. It allows SSH and HTTPS traffic from anywhere. The "Configure storage" step shows a 1x 8 GiB gp3 volume. The "Summary" step shows one instance launching with the Amazon Linux 2023.7.2 AMI. The bottom section shows a success message: "Successfully initiated launch of instance i-0ca4bb2320edd50cb". Below this are "Next Steps" including "Create billing and free tier usage alerts", "Connect to your instance", "Connect an RDS database", and "Create EBS snapshot policy".

Configure security groups for HTTP, and SSH access:

The screenshot shows the AWS EC2 Instances page. A green success message at the top states: "Successfully attached studentuser to instance i-0ca4bb2320edd50cb". The main table lists one instance: "TravelGoproject" (Instance ID: i-0ca4bb2320edd50cb, State: Running, Type: t2.micro). The "Actions" dropdown menu for this instance includes options like "Connect", "View alarms", and "Launch instances".

AWS CloudWatch Metrics

Search [Alt+S] United States (N. Virginia) rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew71 ▾

EC2 > Security Groups > sg-001cb936ef308ba42 - launch-wizard-1 > Edit inbound rules

Inbound rules

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0485a2bbdb90566bf	SSH	TCP	22	Cus... ▾	<input type="text"/> 0.0.0.0/0 X
sgr-0a3b97d577de88311	HTTPS	TCP	443	Cus... ▾	<input type="text"/> 0.0.0.0/0 X
-	Custom TCP	TCP	5000	An... ▾	<input type="text"/> 0.0.0.0/0 X

Add rule

⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel Preview changes Save rules

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Student - Skill W trovenin/student trovenin/student Instances chargpt - Search Confirm Bus Book TravelG0/app.py app.py for final d...

https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:

aws Search [Alt+S] United States (N. Virginia) rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew83 ▾

EC2 > Instances

EC2 Successfully attached studentuser to instance i-0ca4bb2320edd50cb

Instances (1/1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
TravelGoproject	i-0ca4bb2320edd50cb	Running	t2.micro	Initializing	View alarms +	us-east-1d	ec2-44-

i-0ca4bb2320edd50cb (TravelGoproject)

Details Status and alarms Monitoring Security Networking Storage Tags

Instance summary

Instance ID i-0ca4bb2320edd50cb	Public IPv4 address 44.212.43.168 [open address]	Private IPv4 addresses 172.31.88.224
IPv6 address -	Instance state Running	Public DNS ec2-44-212-43-168.compute-1.amazonaws.com [open address]
Hostname type IP name in 172.31.88.224.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-88-224.ec2.internal	

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Modify IAM role Info

Attach an IAM role to your instance.

Instance ID
 i-0ca4bb2320edd50cb (TravelGoproject)

IAM role
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

Studentuser

Connect Info

Connect to an instance using the browser-based client.

EC2 Instance Connect

⚠️ Unable to verify public subnet
You are not authorized to perform this operation. User: arn:aws:sts::863518417312:assumed-role/rsoaccount-new/i-001da4369d20120be221457 is not authorized to perform: ec2:DescribeRouteTables because no identity-based policy allows the ec2:DescribeRouteTables action
Unable to verify if associated subnet [subnet-0d87074c98246f372](#) is a public subnet.
To use EC2 Instance Connect, your instance must be in a public subnet. [To make the subnet a public subnet, add a route in the subnet route table to an internet gateway.](#)

Instance ID
 i-0ca4bb2320edd50cb (TravelGoproject)

Connect using a Public IP
Connect using a public IPv4 or IPv6 address

Connect using a Private IP
Connect using a private IP address and a VPC endpoint

Public IPv4 address

© 2025, Amazon Web Services, Inc. or its affiliates.

Connect Info

Connect to an instance using the browser-based client.

EC2 Instance Connect | **Session Manager** | **SSH client** | **EC2 serial console**

Instance ID

i-0ca4bb2320edd50cb (TravelGoProject)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is travel-go-apl1.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
chmod 400 "travelgo (1).pem"
4. Connect to your instance using its Public DNS:
ec2-44-212-43-168.compute-1.amazonaws.com

Example:

```
ssh -i "travel-go-apl1.pem" ec2-user@ec2-50-19-159-121.compute-1.amazonaws.com
```

Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

- Now connect the EC2 with the files

```
PS C:\Users\gudap> ssh -i "C:\Users\gudap\Downloads\travelgo (1).pem" ec2-user@ec2-44-212-43-168.compute-1.amazonaws.com
The authenticity of host 'ec2-44-212-43-168.compute-1.amazonaws.com (64:ff9b:2cd4:2ba8)' can't be established.
ED25519 key fingerprint is SHA256:8Bt+LQC2P00x4D05LBgEPXESBKn0Z+weuj/vViGqnNk.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-44-212-43-168.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

          _ _ _ _ _ 
         / \_ _ _ \_ 
        / \# \# \# \_ 
       / \# / \# \# \# \_ 
      / \# / \# \# \# \# \_ 
     / \# / \# \# \# \# \# \_ 
    / \# / \# \# \# \# \# \# \_ 
   / \# / \# \# \# \# \# \# \# \_ 
  / \# / \# \# \# \# \# \# \# \# \_ 
 [ec2-user@ip-172-31-88-224 ~]$ sudo yum install git -y
Amazon Linux 2023 Kernel Livepatch repository                               165 kB/s | 17 kB     00:00
Dependencies resolved.
=====
 Package           Architecture      Version       Repository      Size
=====
 Installing:
 git              x86_64          2.47.1-1.amzn2023.0.3      amazonlinux      52 k
 Installing dependencies:
 git-core          x86_64          2.47.1-1.amzn2023.0.3      amazonlinux      4.5 M
 git-core-doc      noarch          2.47.1-1.amzn2023.0.3      amazonlinux      2.8 M
 perl-Error        noarch          1:0.17029-5.amzn2023.0.2      amazonlinux      41 k
 perl-File-Find    noarch          1.37-477.amzn2023.0.7      amazonlinux      25 k
 perl-Git          noarch          2.47.1-1.amzn2023.0.3      amazonlinux      40 k
 perl-TermReadKey x86_64          2.38-9.amzn2023.0.2       amazonlinux      36 k
 perl-lib          x86_64          0.65-477.amzn2023.0.7      amazonlinux      15 k

 Transaction Summary
=====
 Install 8 Packages

 Total download size: 7.5 M
 Installed size: 37 M
```

Milestone 7: Deployment using EC2

Install Software on the EC2 Instance

Install Python3, Flask, and Git:

- On Amazon Linux 2:

```
sudo yum update -y  
sudo yum install python3 git  
sudo pip3 install flask boto3
```

- Verify Installations:

```
flask --version  
git --version
```

Clone Your Flask Project from GitHub

Clone your project repository from GitHub into the EC2 instance using Git.

- Run: '[git clone https://github.com/your-github-username/your-repository-name.git](https://github.com/your-github-username/your-repository-name.git)'

Note: change your-github-username and your-repository-name with your credentials

- Here : <https://github.com/2004rakesh/TravelGo-A-Travel-booking-platfrom>

This will download your project to the EC2 instance.

- To navigate to the project directory, run the following command: cd Travelgo1
- Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:
- **Run the Flask Application**
- `export SNS_TOPIC_ARN=ARN:arn:aws:sns:us-east-1:047719615036:TravelGo1:2888f19c-04ee-4400-96cc-ec1576d79fb4`
- `git pull origin main` (Only if you made changes in git and want to reflect them in EC2 terminal.)
- # Install requirements

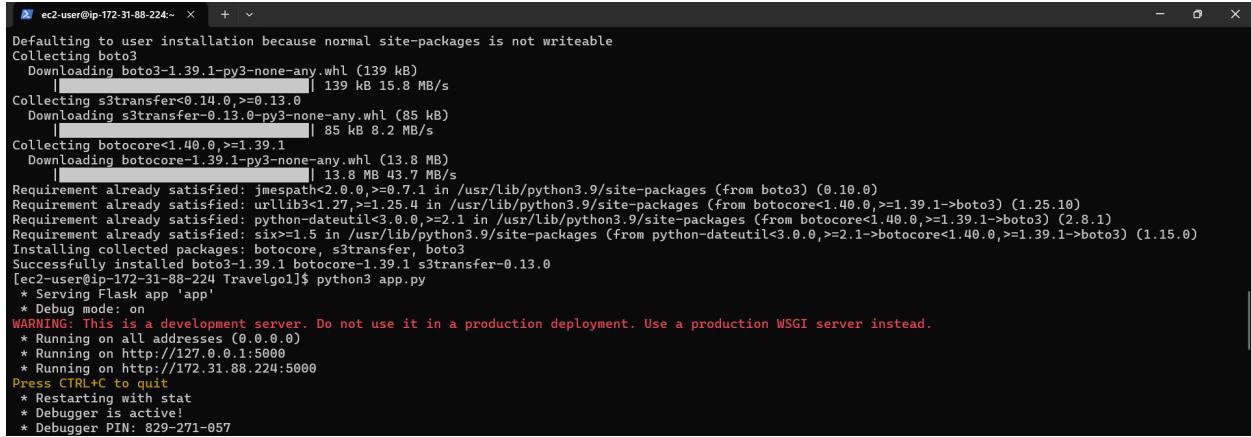
```
pip install flask boto3
```

- Launch the Flask app:
`sudo -E venv/bin/python3 app.py`

Verify the Flask app is running:

`http://your-ec2-public-ip`

- Run the Flask app on the EC2 instance



```
ec2-user@ip-172-31-88-224:~ + ~
Collecting boto3
  Downloading boto3-1.39.1-py3-none-any.whl (139 kB)
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
Collecting botocore<1.40.0,>=1.39.1
  Downloading botocore-1.39.1-py3-none-any.whl (13.8 MB)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.1->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.1->boto3) (2.8.1)
Requirement already satisfied: six>1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.1->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.1 botocore-1.39.1 s3transfer-0.13.0
[ec2-user@ip-172-31-88-224 Travelgo1]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.88.224:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 829-271-057
```

- Access the website through:

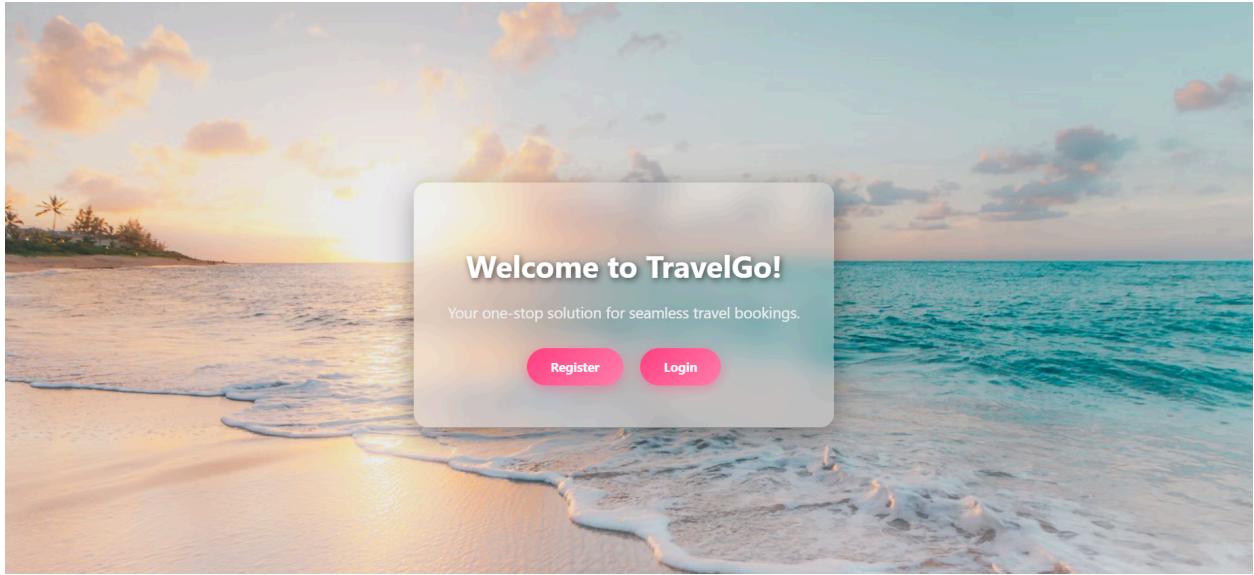
Public IPs: `http://44.212.43.168:5000`

Milestone 8 : Testing and Deployment

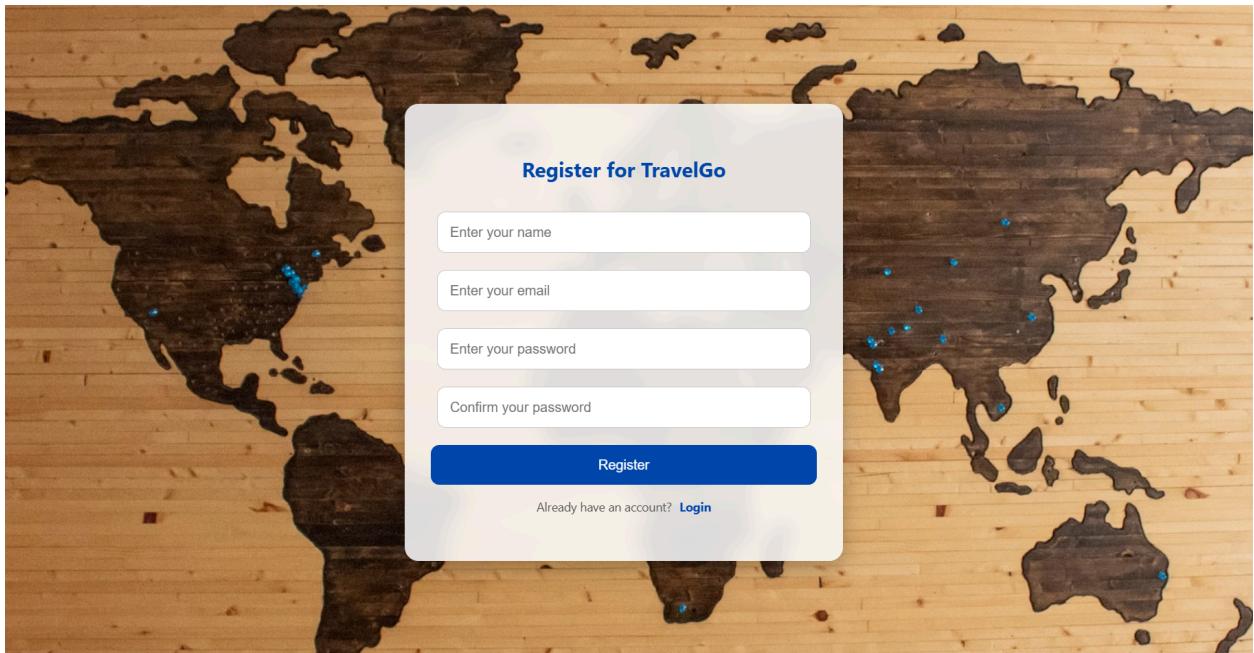
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment.

Functional Testing to verify the Project

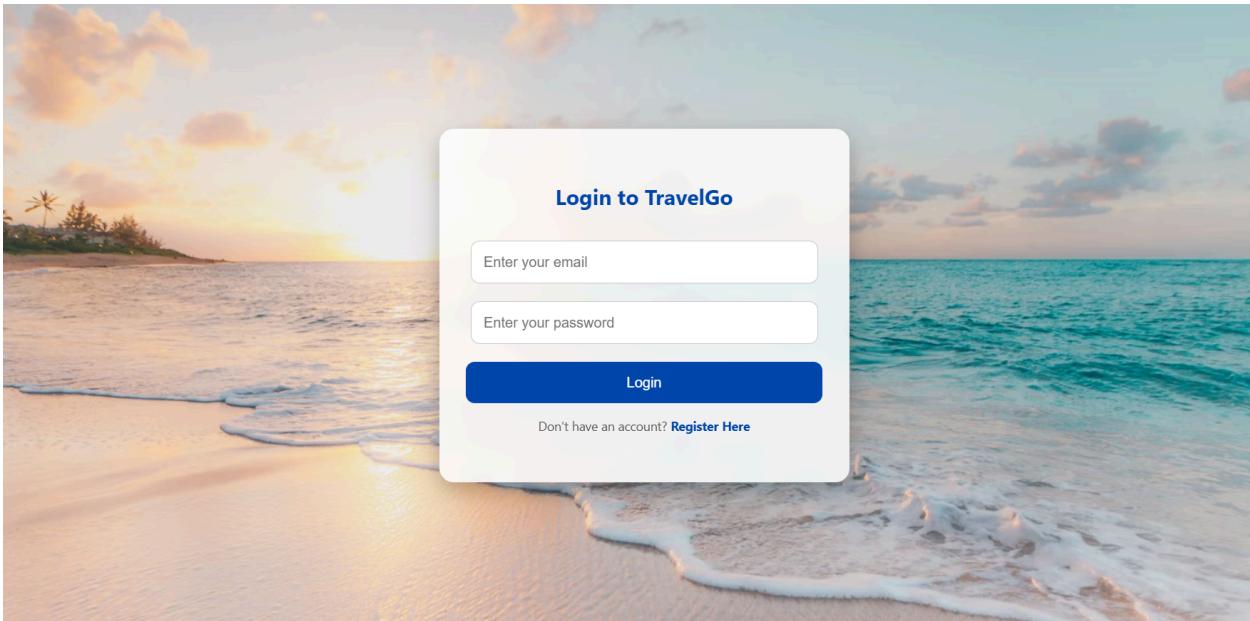
Home Page:



Sign In Page:



Sign Up Page:



Dashboard Page:

Welcome, gudapatiudayarakesh@gmail.com!

User gudapatiudayarakesh@gmail.com

Total Bookings	Upcoming	Completed	Cancelled
0	0	0	0

Bus Train Flight Hotel

All Bus Train Flight Hotel dd-mm-yyyy

Your Bookings

You haven't made any bookings yet. Start your travel planning now!

The dashboard features a top navigation bar with "TravelGo", "Dashboard", and "Logout" links. Below this is a welcome message and the user's email. A summary table shows zero bookings across four categories. Below the table are four buttons for Bus, Train, Flight, and Hotel. A filter section allows selecting "All" or specific transport modes. A date input field is also present. The main area is titled "Your Bookings" and displays a message indicating no bookings have been made.

Buses Page:

The screenshot shows the TravelGo - Bus Booking interface. At the top, there's a purple header bar with the 'TravelGo' logo on the left and 'Home' and 'Dashboard' links on the right. Below the header is a search form with dropdowns for 'From' (Hyderabad), 'To' (Vijayawada), 'Date' (10-07-2025), and 'Passenger Count' (1). A green 'Search' button is to the right of the dropdowns. The main content area displays a search result for 'TSRTC Travels' with the route 'Hyderabad → Vijayawada' and departure time '6:34 PM | Non-AC Sleeper'. To the right of the route information is a price of ₹742 and a blue 'Choose Seats' button. The background of the main content area is light gray.

Select Your Seats Page:

Select Your Seats

This screenshot shows a seat selection interface. At the top, there are two buttons: 'Lower Berth' and 'Upper Berth'. Below these buttons is a 6x2 grid of seat labels. The grid is organized into two columns: the first column contains seats L1A, L2A, L3A, L4A, L5A, and L6A; the second column contains seats L1B, L2B, L3B, L4B, L5B, and L6B. At the bottom right of the grid are two large buttons: a green 'Confirm' button and a gray 'Cancel' button.

Bus Booking Page:

Your Bookings

Bus - VRL Travels

Route: Hyderabad → Vijayawada

Date: 2025-07-10

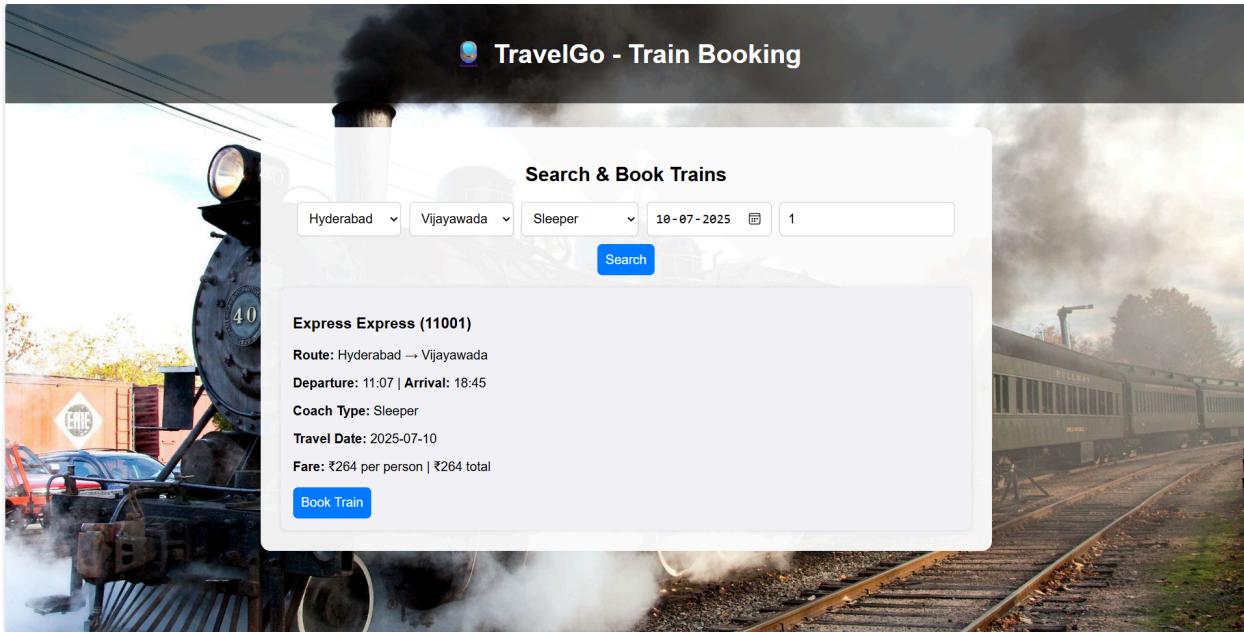
Seats: L3A

Passengers: 1

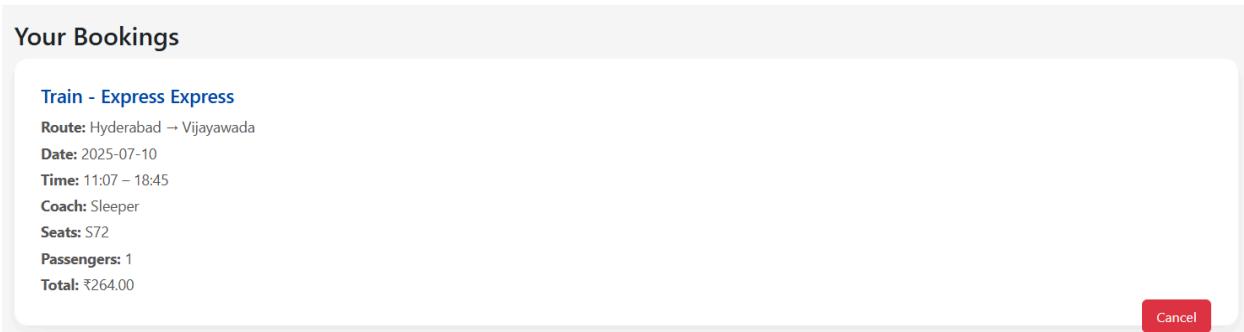
Total: ₹805.00

Cancel

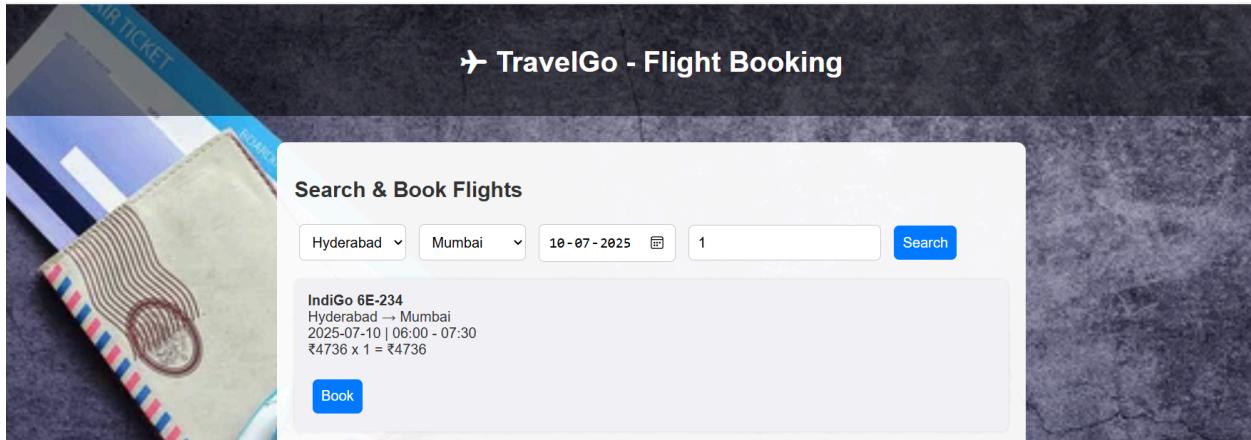
Trains Page:



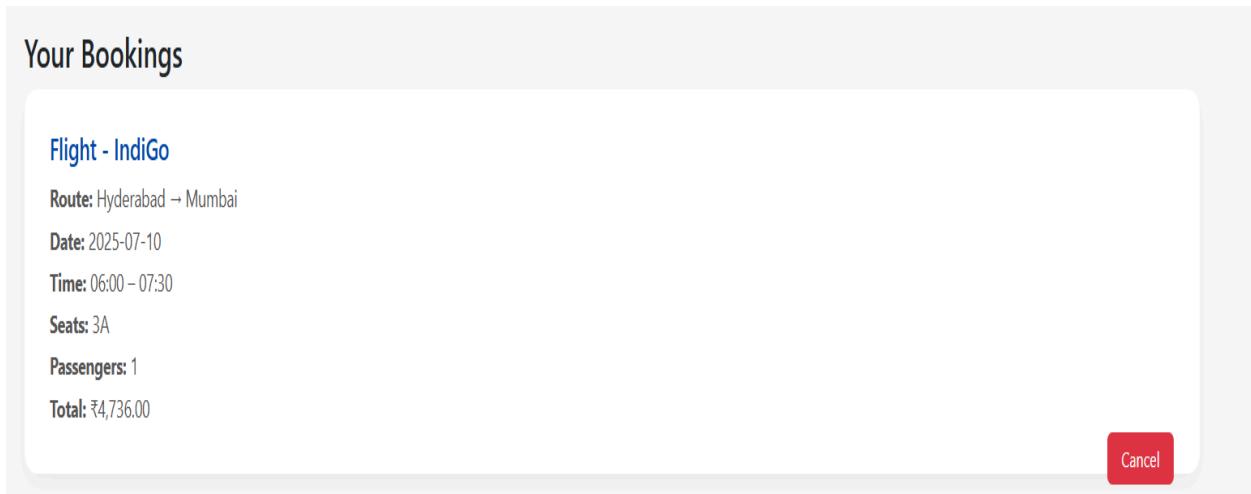
Train Booking Page:



Flights Page:



Flight Booking Page:



Hotels Page:

Travel Go - Hotel Booking

Hyderabad 10 - 07 - 2025 13 - 07 - 2025 1

5-Star 4-Star 3-Star WiFi Pool Parking

Sort by:

The Park
Hyderabad | 3-Star | ₹5500/night
Room Type: Deluxe Room
Amenities: WiFi

Taj Falaknuma Palace
Hyderabad | 5-Star | ₹25000/night
Room Type: Suite
Amenities: WiFi, Pool, Parking, Restaurant

Confirm Hotel Booking Page:

Your Bookings

Hotel - Taj Falaknuma Palace

Location: Hyderabad
Check-in: 2025-07-10
Check-out: 2025-07-13
Room Type:
Guests: 1
Total: ₹75,000.00

Bookings Page:

TravelGo

Welcome, gudapatiudayarakesh@gmail.com!

User gudapatiudayarakesh@gmail.com

Total Bookings 4 Upcoming 0 Completed 0 Cancelled 0

Bus Train Flight Hotel

All Bus Train Flight Hotel dd-mm-yyyy

Your Bookings

Hotel - Taj Falaknuma Palace

Location: Hyderabad
Check-in: 2025-07-10
Check-out: 2025-07-13
Room Type:
Guests: 1
Total: ₹75,000.00

Flight - IndiGo

Route: Hyderabad → Mumbai
Date: 2025-07-10
Time: 06:00 – 07:30
Seats: 3A
Passengers: 1
Total: ₹4,736.00

Train - Express Express

Route: Hyderabad → Vijayawada
Date: 2025-07-10
Time: 11:07 – 18:45
Coach: Sleeper
Seats: S72
Passengers: 1
Total: ₹264.00

Bus - VRL Travels

Route: Hyderabad → Vijayawada
Date: 2025-07-10
Seats: L3A
Passengers: 1
Total: ₹805.00

Conclusion

The TravelGo Website has been successfully developed and deployed using a scalable and cloud-native architecture. Leveraging AWS services such as EC2 for hosting, DynamoDB for real-time data management, and SNS for instant booking and cancellation notifications, the platform provides a seamless travel booking experience for users. TravelGo enables registered users to search and book buses, trains, flights, and hotels in a centralized, intuitive interface, eliminating the complexities of navigating multiple travel services.

The cloud infrastructure ensures high availability and smooth performance even during peak usage, while the Flask backend ensures efficient handling of user authentication, dynamic booking flows, and data transactions. Real-time notification integration via AWS SNS allows users to receive booking confirmations and cancellations immediately via email, improving communication and user engagement.

In summary, the TravelGo Website offers a modern, reliable, and user-friendly solution for managing travel and accommodation needs. It highlights the potential of cloud-based platforms in building unified travel systems, simplifying operations, and enhancing the overall user experience.