

Travel Go: A Cloud-Based Instant Travel Reservation Platform

Project Overview:

Travel Go aims to streamline the travel planning process by offering a centralized platform to book buses, trains, flights, and hotels. Catering to the growing need for realtime and convenient travel solutions, it utilizes cloud infrastructure. The application employs Flask for backend services, AWS EC2 for scalable hosting, DynamoDB for fast data handling, and AWS SNS for immediate notifications. Users can sign up, log in, search, and book transport and accommodations. Additionally, they can review booking history and pick seats interactively, ensuring a user-friendly and responsive system.

Use Case Scenarios:

Scenario 1: Instant Travel Booking Experience

After logging into Travel Go, the user picks a travel type (bus/train/flight/hotel) and fills in preferences. Flask manages backend operations by retrieving matching options from DynamoDB. Once a booking is confirmed, AWS EC2 ensures swift performance even under high usage. This real-time setup allows users to secure bookings efficiently, even during rush periods.

Scenario 2: Instant Email Alerts

Upon booking confirmation, Travel Go uses AWS SNS to instantly send emails with booking info. For example, when a flight is booked, Flask handles the transaction and SNS notifies the user via email. The booking details are saved securely in DynamoDB. This smooth integration boosts reliability and enhances user confidence.

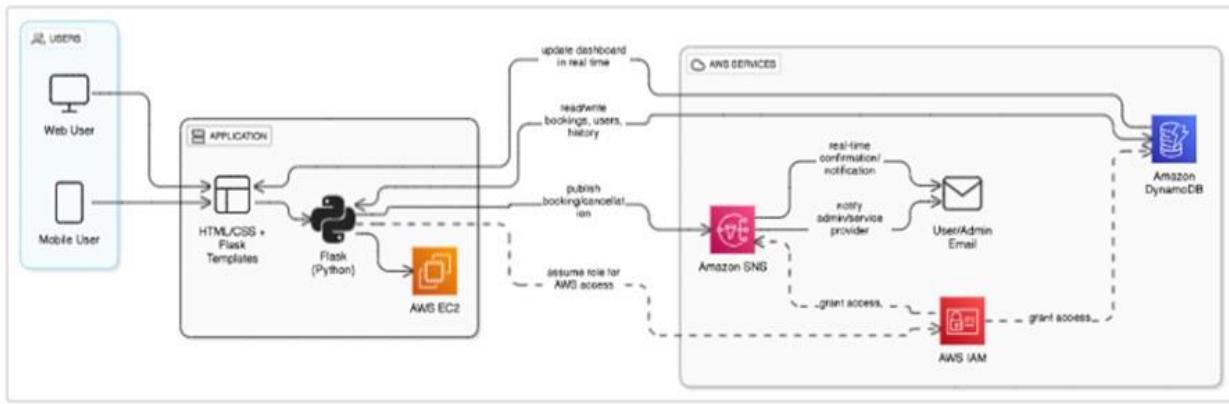
Scenario 3: Easy Booking Access and Control

Users can return anytime to manage previous bookings. For instance, a user checks hotel reservations made last week. Flask quickly fetches this from DynamoDB. Thanks to its cloud-first approach, Travel Go provides uninterrupted access, letting users easily cancel or make new plans. EC2 hosting guarantees consistent performance across multiple users.

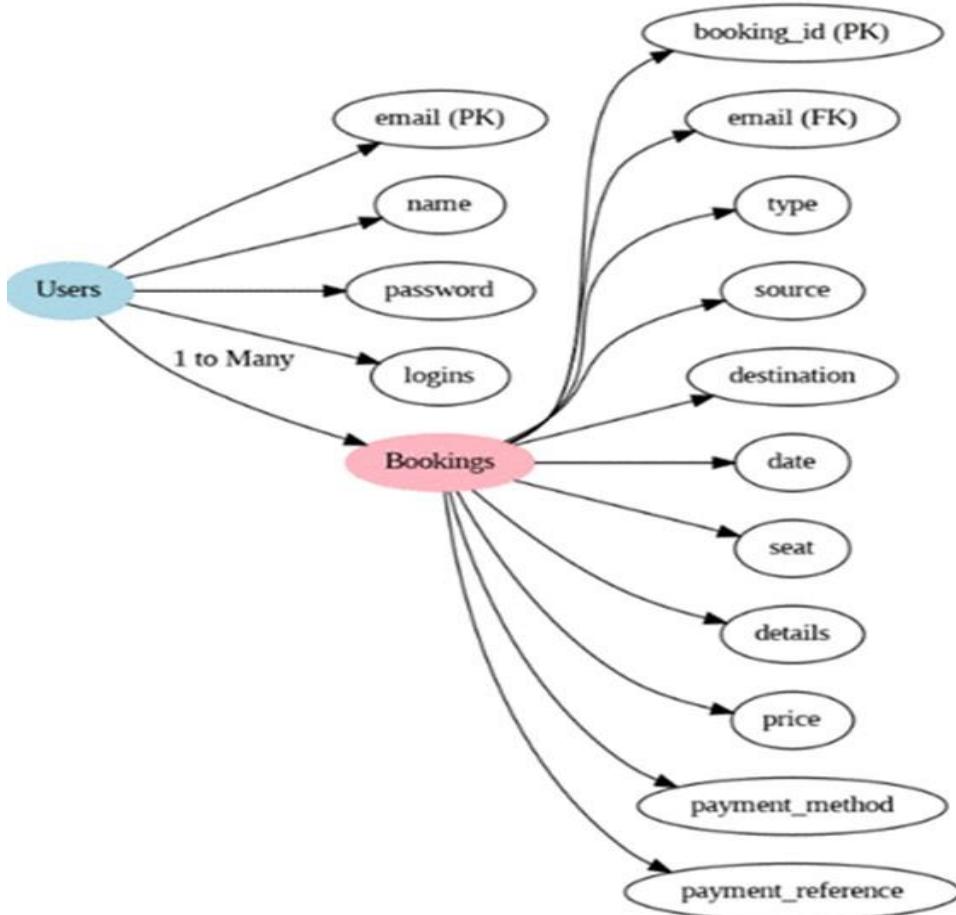
AWS Architecture Diagram

To build a scalable and responsive travel booking platform, a well-structured cloud architecture is essential. Travel Go leverages core AWS services such as EC2, DynamoDB, IAM, and SNS to ensure high availability and fault tolerance. The architecture is designed to support real-time booking, seamless data flow, and secure operations. Below is the visual representation of the AWS setup used in the project.

AWS Architecture



Entity Relationship Diagram



Requirements:

1. AWS Account Configuration
2. IAM Basics Overview
3. EC2 Introduction
4. DynamoDB Fundamentals
5. SNS Concepts
6. Git Version Control

Development Steps:

1. AWS Account Setup and Login

- Activity 1.1: Register for an AWS account if you haven't already.
- Activity 1.2: Access the AWS Management Console using your credentials.

2. DynamoDB Database Initialization

- Activity 2.1: Create a DynamoDB table.
- Activity 2.2: Define attributes for users and travel bookings.

3. SNS Notification Configuration

- Activity 3.1: Create SNS topics to notify users upon booking.
- Activity 3.2: Add user and provider emails as subscribers for updates.

4. Backend Development using Flask

- Activity 4.1: Build backend logic with Flask.
- Activity 4.2: Connect AWS services using the boto3 SDK.

5. IAM Role Configuration

- Activity 5.1: Set up IAM roles with specific permissions.
- Activity 5.2: Assign relevant policies to each role.

6. EC2 Instance Launch

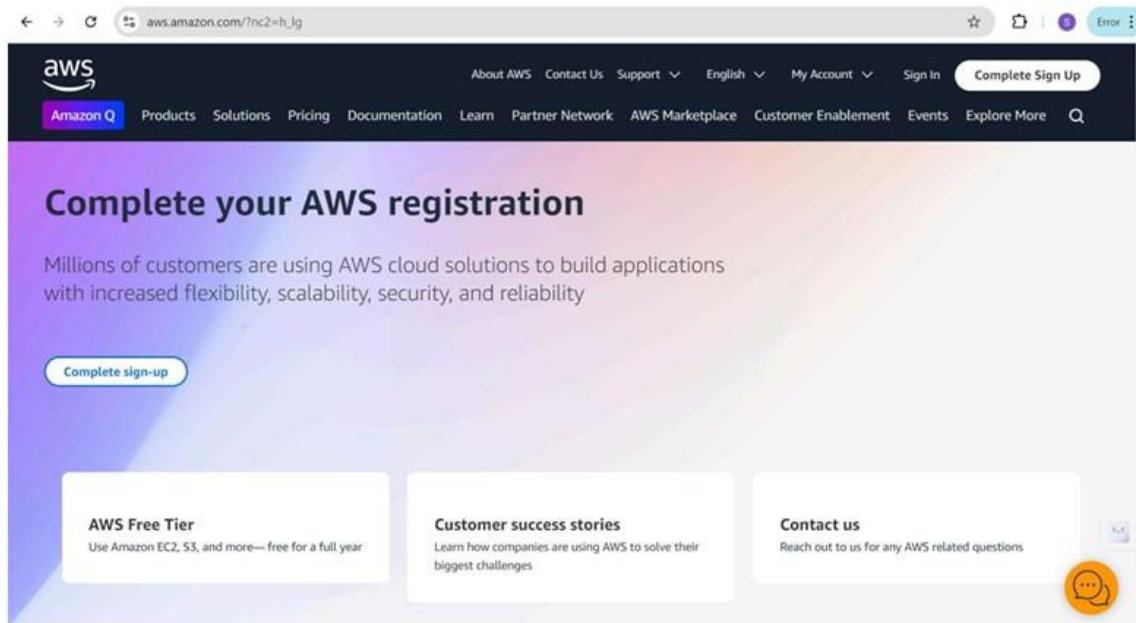
- Activity 6.1: Start an EC2 instance to host the backend.
- Activity 6.2: Set up security rules to allow HTTP and SSH traffic.

7. Flask Application Deployment

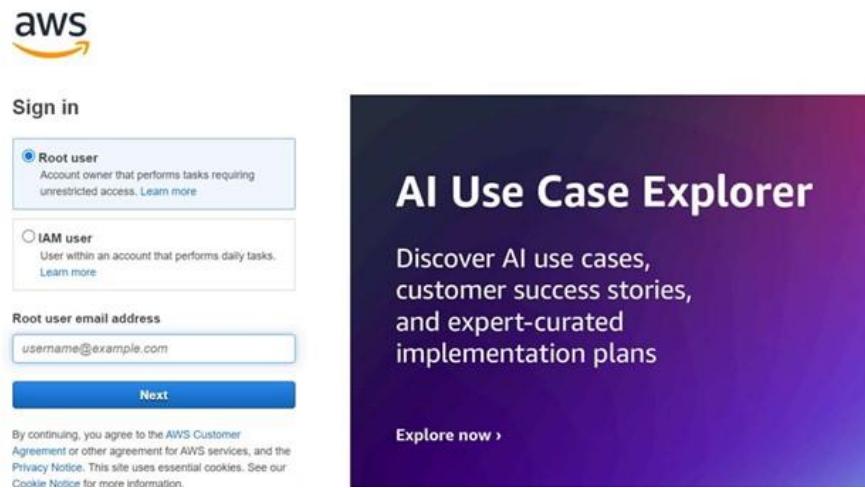
- Activity 7.1: Upload your Flask files to the EC2 instance.
- Activity 7.2: Launch your Flask server from the instance.

8. Testing and Launch

1. AWS Account Setup and Login



2. Log in to the AWS Management Console



After logging into the AWS Console:

Once you're logged in, you can access a wide range of AWS services from the dashboard. Use the search bar at the top to quickly navigate to services like EC2, DynamoDB, SNS, or IAM as needed for your project setup.

3. IAM Roles Creation

The screenshot shows two browser windows side-by-side. The top window is the 'IAM Dashboard' under the 'Identity and Access Management (IAM)' section. It displays a 'New access analyzers available' notification and several error messages related to 'Access denied'. One message states: 'You don't have permission to iam:GetAccountSummary. To request access, copy the following text and send it to your AWS administrator.' Below this is a snippet of AWS Lambda code:

```
User: arn:aws:sts::490004646397:assumed-role/rsouser-new/6801da4369d20120be221457
Action: iam:GetAccountSummary
Context: no identity-based policy allows the action
```

A 'Diagnose with Amazon Q' button is present. The bottom window shows the 'Create role | IAM | Global' wizard, specifically the 'Use case' step. It asks to allow an AWS service like EC2, Lambda, or others to perform actions in the account. A dropdown menu is set to 'EC2'. Under 'Use case', the 'EC2' option is selected, with a description: 'Allows EC2 instances to call AWS services on your behalf.' Other options listed include 'EC2 Role for AWS Systems Manager', 'EC2 Spot Fleet Role', 'EC2 - Spot Fleet Auto Scaling', 'EC2 - Spot Fleet Tagging', 'EC2 - Spot Instances', 'EC2 - Spot Fleet', and 'EC2 - Scheduled Instances'. At the bottom right of the wizard are 'Cancel' and 'Next' buttons.

Before assigning permissions, ensure that the IAM role is created and ready to be attached with appropriate AWS-managed policies for each required service.

The screenshot shows the 'Add permissions' step of the IAM role creation wizard. The user has searched for 'ec2' and found 33 matches. The 'AmazonEC2FullAccess' policy is selected.

Policy name	Type	Description
AmazonEC2ContainerRegistryFullAccess	AWS managed	Provides administrative access to Amazon ECR
AmazonEC2ContainerRegistryPowerUser	AWS managed	Provides full access to Amazon EC2 Container Registry
AmazonEC2ContainerRegistryPullOnly	AWS managed	Provides access to pull images from Amazon ECR
AmazonEC2ContainerRegistryReadOnly	AWS managed	Provides read-only access to Amazon ECR
AmazonEC2ContainerServiceAutoscaleRole	AWS managed	Policy to enable Task AutoScaling for Amazon ECS
AmazonEC2ContainerServiceEventsRole	AWS managed	Policy to enable CloudWatch Events for Amazon ECS
AmazonEC2ContainerServiceforEC2Role	AWS managed	Default policy for the Amazon EC2 Role
AmazonEC2ContainerServiceRole	AWS managed	Default policy for Amazon ECS service
AmazonEC2FullAccess	AWS managed	Provides full access to Amazon EC2 via...
AmazonEC2ReadOnlyAccess	AWS managed	Provides read only access to Amazon E...

The screenshot shows the 'Name, review, and create' step of the IAM role creation wizard. The 'Role details' section includes the role name 'Studentuser' and a description 'Allows EC2 instances to call AWS services on your behalf.'

Step 1: Select trusted entities

Trust policy

```

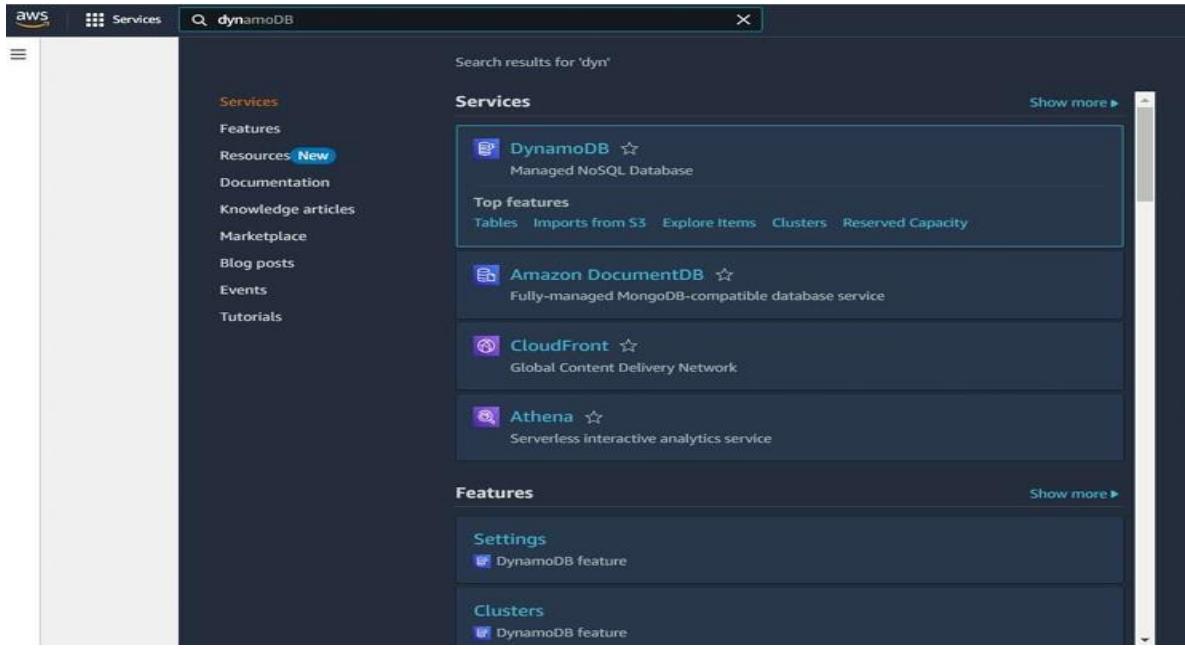
1 < {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "sts:AssumeRole"
8       ]
9     }
10  ]
11 }
  
```

4. DynamoDB Database Creation and Setup

Familiarize yourself with the DynamoDB service interface. DynamoDB offers a fast and flexible NoSQL database ideal for serverless applications like Travel Go. You will now proceed to create tables to store user and booking data. In this setup, you'll define partition keys to uniquely identify records—such as using “Email” for users or “Train Number” for trains. Properly configuring your attributes is crucial to ensure efficient querying and data retrieval. DynamoDB’s schema-less

design allows flexibility while maintaining performance, making it well-suited for handling diverse booking data types in Travel Go.

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



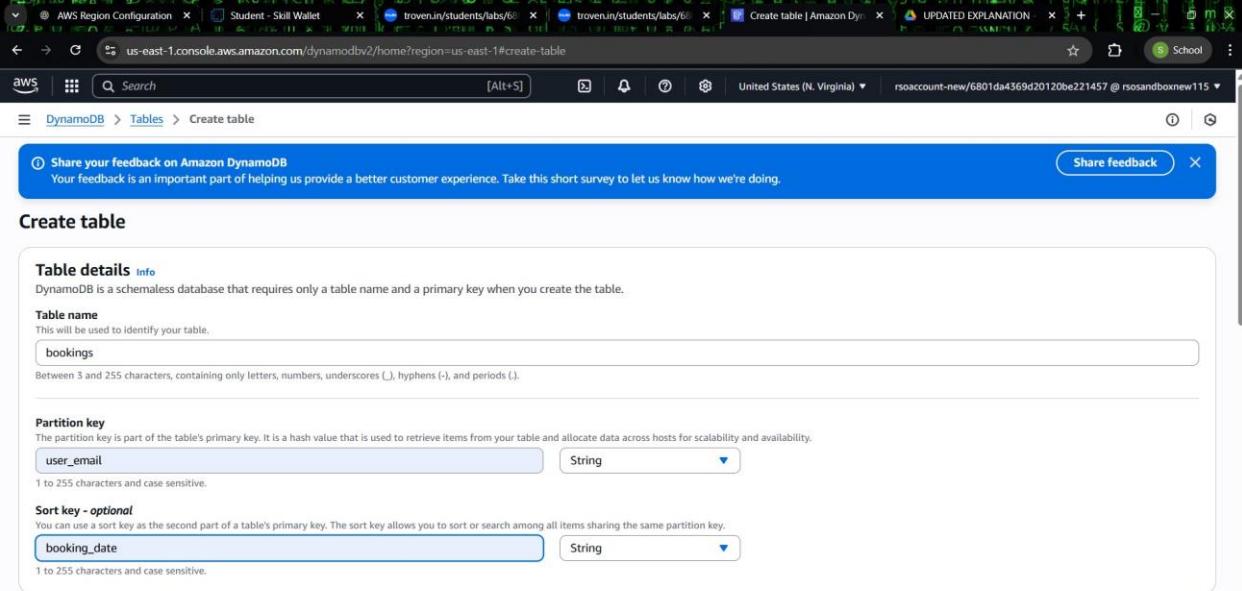
Create a DynamoDB table for storing registration details and book Requests

1. Create Users table with partition key “Email” with type String and click on create tables.

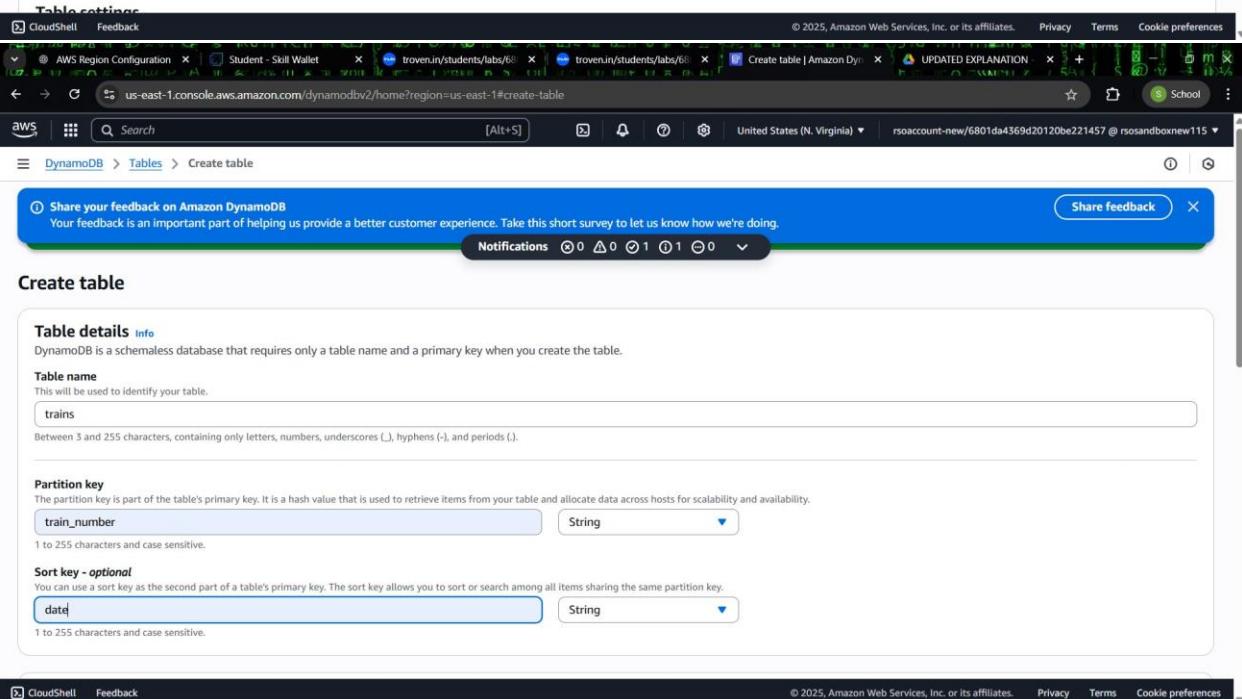
A screenshot of the "Create table" wizard in the AWS DynamoDB console. The table name is "travelgo_users". The partition key is "email" of type String. The sort key is optional and currently empty. The table settings are not yet configured. The status bar at the bottom shows "CloudShell" and "Feedback".

Repeat the same procedure to create additional tables:

- **Train Table:** Use “Train Number” as the partition key to uniquely identify each train.
- **Bookings Table:** Set “User Email” as the partition key and “Booking Date” as the sort key to efficiently organize and retrieve individual user booking records based on date.



The screenshot shows the 'Create table' wizard for the 'bookings' table. It includes sections for 'Table details' (info), 'Table name' (set to 'bookings'), 'Partition key' (set to 'user_email' of type String), and 'Sort key - optional' (set to 'booking_date' of type String). A feedback survey banner is at the top.



The screenshot shows the 'Create table' wizard for the 'trains' table. It includes sections for 'Table details' (info), 'Table name' (set to 'trains'), 'Partition key' (set to 'train_number' of type String), and 'Sort key - optional' (set to 'date' of type String). A feedback survey banner is at the top. The bottom of the screen shows the AWS navigation bar.

The screenshot shows the AWS DynamoDB console. On the left, there's a sidebar with 'DynamoDB' selected. Under 'Tables', it lists 'bookings', 'trains', and 'travelpo_users'. Below that is a section for 'DAX' with options like Clusters, Subnet groups, Parameter groups, and Events. At the top right, there's a search bar, a 'Create table' button, and other navigation controls. The main area displays a table titled 'Tables (3) Info' with columns for Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read. The three tables listed are:

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read
bookings	Active	user_email (\$)	booking_date (\$)	0	0	Off	☆	On-di
trains	Active	train_number (\$)	date (\$)	0	0	Off	☆	On-di
travelpo_users	Active	email (\$)	-	0	0	Off	☆	On-di

5. SNS Notification Setup

1. Create SNS topics for sending email notifications to users.

The screenshot shows the AWS search results for 'sns'. On the left, there's a sidebar with 'Services', 'Features', 'Resources New', 'Documentation', 'Knowledge articles', 'Marketplace', 'Blog posts', 'Events', and 'Tutorials'. The search bar at the top has 'sns' entered. The results show:

- Simple Notification Service ☆**
SNS managed message topics for Pub/Sub
- Route 53 Resolver**
Resolve DNS queries in your Amazon VPC and on-premises network.
- Route 53 ☆**
Scalable DNS and Domain Name Registration
- AWS End User Messaging ☆**
Engage your customers across multiple communication channels

The screenshot shows the Amazon Simple Notification Service (SNS) homepage. At the top, there is a blue banner with the text "New Feature" and "Amazon SNS now supports High Throughput FIFO topics. [Learn more](#)". Below the banner, the page title is "Amazon Simple Notification Service" with the subtitle "Pub/sub messaging for microservices and serverless applications". A brief description follows: "Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging." To the right, there is a "Create topic" form with a "Topic name" field containing "MyTopic" and a "Next step" button. Below the form is a link "Start with an overview". On the left, there is a "Benefits and features" section. On the right, there is a "Pricing" section stating "Amazon SNS has no upfront costs. You pay based on the number of messages you publish, the number of messages you deliver, and any additional API calls for managing topics and". The bottom of the page includes standard browser navigation and status bars.

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

The screenshot shows the "Create topic" wizard on the "Details" step. The "Type" section is selected, showing two options: "FIFO (first-in, first-out)" and "Standard". "Standard" is selected, with a list of protocols: "Best-effort message ordering", "At-least once message delivery", and "Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints". The "Name" field contains "TravelLog". The "Display name - optional" field contains "My Topic". The "Encryption - optional" section notes that Amazon SNS provides in-transit encryption by default. The bottom of the page includes standard browser navigation and status bars.

2. Click on create topic

To begin configuring notifications, navigate to the SNS dashboard and click on “Create Topic.” Choose the topic type (Standard or FIFO) based on your requirements. Provide a meaningful name

for the topic that reflects its purpose (e.g., booking-alerts). This topic will serve as the communication channel for sending notifications to subscribed users.

The screenshot shows the 'Create topic' configuration page for an AWS SNS topic. It includes sections for optional settings:

- Access policy - optional**: Defines who can access the topic.
- Data protection policy - optional**: Defines which sensitive data to monitor and prevent from being exchanged.
- Delivery policy (HTTP/S) - optional**: Defines how Amazon SNS retries failed deliveries to HTTP/S endpoints.
- Delivery status logging - optional**: Configures message delivery status to CloudWatch Logs.
- Tags - optional**: Metadata labels for tracking costs.
- Active tracing - optional**: AWS X-Ray active tracing for traces and service map.

At the bottom right are 'Cancel' and 'Create topic' buttons.

3. Configure the SNS topic and note down the **Topic ARN**.
4. Click on create subscription.

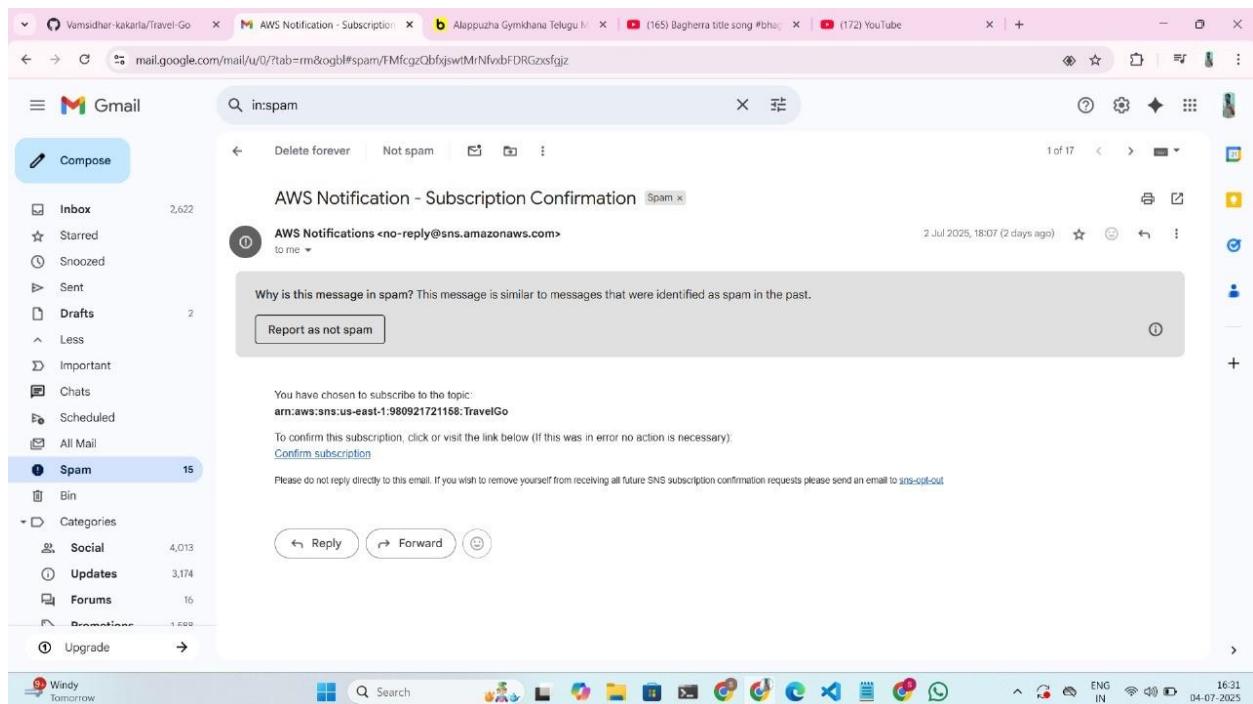
The screenshot shows the 'Create subscription' configuration page for an AWS SNS topic. It includes sections for optional settings:

- Subscription filter policy - optional**: Filters messages for subscribers.
- Redrive policy (dead-letter queue) - optional**: Sends undeliverable messages to a dead-letter queue.

At the bottom right are 'Cancel' and 'Create subscription' buttons.

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

Once the email subscription is confirmed, the endpoint becomes active for receiving notifications. This ensures that users will instantly get booking confirmations or alerts. You can manage or remove subscriptions anytime via the SNS dashboard. It's recommended to test the setup by publishing a sample message to verify successful delivery.



Backend Configuration and Coding

```

1  from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
2  from pymongo import MongoClient
3  from werkzeug.security import generate_password_hash, check_password_hash
4  from datetime import datetime
5  from bson.objectid import ObjectId
6  from bson.errors import InvalidId

```

- Flask: Initiates web application
- render_template: Loads Jinja2 HTML templates
- request: Collects input from forms/API
- redirect/url_for: Page redirection logic
- session: Keeps user-specific info
- jsonify: Returns data in JSON structure

```

8  app = Flask(__name__)

```

Begin building the web application by initializing the Flask app instance using `Flask(__name__)`, which sets up the core of your backend framework.

```

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

```

SNS connection:

This function sends alerts using AWS SNS by publishing a subject and message to a predefined topic. It uses `sns_client.publish()` with error handling to catch failures and print debug info. This ensures users receive real-time booking notifications.

```

22 SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:490004646397:Travelgo:372db6a7-7131-4571-8397-28b62c9e08a8'
23
24 # Function to send SNS notifications
25 # This function is duplicated in the original code, removing the duplicate.
26 def send_sns_notification(subject, message):
27     try:
28         sns_client.publish(
29             TopicArn=SNS_TOPIC_ARN,
30             Subject=subject,
31             Message=message
32         )
33     except Exception as e:
34         print(f"SNS Error: Could not send notification - {e}")
35         # Optionally, flash an error message to the user or log it more robustly.

```

Routes:

Route Overview of TravelGo:

The application begins with user onboarding through the Register and Login routes, securing user access. Once authenticated, users are redirected to the Dashboard, which serves as the central hub. From there, they can explore and book through Bus, Train, Flight, and Hotel modules. Each booking passes through a Confirmation step where details are reviewed before final submission. The Final Confirmation route stores the booking and triggers email alerts. Users also have the option to manage or cancel their reservations via the Cancel Booking route, ensuring full control and flexibility.

Let's take a closer look at the backend code that powers these application routes.

```

38 @app.route('/')
39 def index():
40     return render_template('index.html')
41
42 @app.route('/register', methods=['GET', 'POST'])
43 def register():
44     if request.method == 'POST':
45         email = request.form['email']
46         password = request.form['password']
47
48         # Check if user already exists
49         # This uses get_item on the primary key 'email', so no GSI needed.
50         existing = users_table.get_item(Key={'email': email})
51         if 'Item' in existing:
52             flash('Email already exists!', 'error')
53             return render_template('register.html')
54
55         # Hash password and store user
56         hashed_password = generate_password_hash(password)
57         users_table.put_item(Item={'email': email, 'password': hashed_password})
58         flash('Registration successful! Please log in.', 'success')
59         return redirect(url_for('login'))
60     return render_template('register.html')

```

```

62     @app.route('/login', methods=['GET', 'POST'])
63     def login():
64         if request.method == 'POST':
65             email = request.form['email']
66             password = request.form['password']
67
68             # Retrieve user by email (primary key)
69             user = users_table.get_item(Key={'email': email})
70
71             # Authenticate user
72             if 'Item' in user and check_password_hash(user['Item']['password'], password):
73                 session['email'] = email
74                 flash('Logged in successfully!', 'success')
75                 return redirect(url_for('dashboard'))
76             else:
77                 flash('Invalid email or password!', 'error')
78                 return render_template('login.html')
79
80     return render_template('login.html')

```

```

81     @app.route('/logout')
82     def logout():
83         session.pop('email', None)
84         flash('You have been logged out.', 'info')
85         return redirect(url_for('index'))
86
87     @app.route('/dashboard')
88     def dashboard():
89         if 'email' not in session:
90             return redirect(url_for('login'))
91         user_email = session['email']
92
93         # Query bookings for the logged-in user using the primary key 'user_email'
94         # No GSI is needed here as 'user_email' is likely the partition key for the bookings_table.
95         response = bookings_table.query(
96             KeyConditionExpression=Key('user_email').eq(user_email),
97             ScanIndexForward=False # Get most recent bookings first
98         )
99         bookings = response.get('Items', [])
100
101         # Convert Decimal types from DynamoDB to float for display if necessary
102         for booking in bookings:
103             if 'total_price' in booking:
104                 try:
105                     booking['total_price'] = float(booking['total_price'])
106                 except (TypeError, ValueError):
107                     booking['total_price'] = 0.0 # Default value if conversion fails
108
109         return render_template('dashboard.html', username=user_email, bookings=bookings)
110
111     @app.route('/train')
112     def train():
113         if 'email' not in session:
114             return redirect(url_for('login'))
115         return render_template('train.html')

```

```

116 @app.route('/confirm_train_details')
117 def confirm_train_details():
118     if 'email' not in session:
119         return redirect(url_for('login'))
120
121     booking_details = {
122         'name': request.args.get('name'),
123         'train_number': request.args.get('trainNumber'),
124         'source': request.args.get('source'),
125         'destination': request.args.get('destination'),
126         'departure_time': request.args.get('departureTime'),
127         'arrival_time': request.args.get('arrivalTime'),
128         'price_per_person': Decimal(request.args.get('price')),
129         'travel_date': request.args.get('date'),
130         'num_persons': int(request.args.get('persons')),
131         'item_id': request.args.get('trainId'), # This is the train ID
132         'booking_type': 'train',
133         'user_email': session['email'],
134         'total_price': Decimal(request.args.get('price')) * int(request.args.get('persons'))
135     }

```

```

492 @app.route('/cancel_booking', methods=['POST'])
493 def cancel_booking():
494     if 'email' not in session:
495         return redirect(url_for('login'))
496
497     booking_id = request.form.get('booking_id')
498     user_email = session['email']
499     booking_date = request.form.get('booking_date') # This is crucial as it's the sort key
500
501     if not booking_id or not booking_date:
502         flash("Error: Booking ID or Booking Date is missing for cancellation.", 'error')
503         return redirect(url_for('dashboard'))
504
505     try:
506         # Delete item using the primary key (user_email and booking_date)
507         # This does not use GSI, so it remains unchanged.
508         bookings_table.delete_item(
509             Key={'user_email': user_email, 'booking_date': booking_date}
510         )
511         flash(f"Booking {booking_id} cancelled successfully!", 'success')
512     except Exception as e:
513         flash(f"Failed to cancel booking {booking_id}: {str(e)}", 'error')
514
515     return redirect(url_for('dashboard'))

```

Note: The implementation logic for train booking, train details, and cancellation is consistent across other modules such as bus, flight, and hotel. Each follows a similar structure for handling user input, storing booking data, and managing cancellations using the same backend principles. This modular approach promotes code reusability and simplifies maintenance across the application. Minor adjustments are made in each module to accommodate specific fields like transport type or room preferences. Overall, the core flow—search, select, book, and cancel—remains consistent for all services.

Deployment code:

```
518     if __name__ == '__main__':
519         # IMPORTANT: In a production environment, disable debug mode and specify a production-ready host.
520         app.run(debug=True, host='0.0.0.0')
```

Start the Flask server by configuring it to run on all network interfaces (0.0.0.0) at port 5000, with debug mode enabled to support development and live testing.

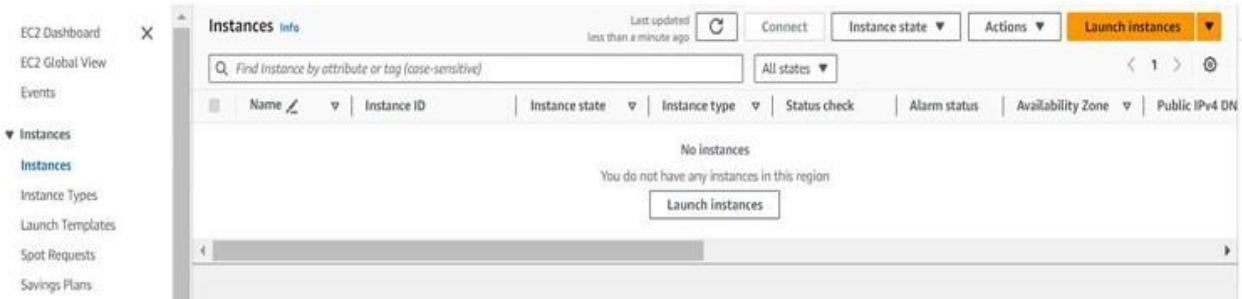
EC2 Instance Setup:

The screenshot shows a GitHub repository named 'TravelGo'. The repository is public and has 1 branch and 0 tags. The commit history shows a single commit from user 'HEMALATHA9542' titled 'First commit' made 2 weeks ago. The commit details show files like static, templates, venv, venv_new, README.md, and app.py. The README file is visible. On the right side, there is an 'About' section with no description, website, or topics provided. It also lists Readme, Activity, 0 stars, 0 watching, and 0 forks. The 'Releases' and 'Packages' sections are empty.

Launch an EC2 instance to host the Flask application.

The screenshot shows the AWS Services search results for 'EC2'. The search bar at the top has 'EC2' typed into it. The results are categorized under 'Services' and 'Features'. Under 'Services', the EC2 service is listed as 'Virtual Servers in the Cloud'. Other services listed include EC2 Image Builder, AWS Compute Optimizer, and AWS Firewall Manager. Under 'Features', the Export snapshots to EC2 feature is listed. The sidebar on the left shows other service categories like Features, Blogs, Documentation, Knowledge Articles, Tutorials, Events, and Marketplace.

1. Click on launch Instance:



Once the EC2 instance is launched, it acts as the server backbone for your application. Naming the instance as Travelgoproject helps in easy identification during future scaling or monitoring. You can manage performance, logs, and connectivity from the EC2 dashboard. This instance will host and run your Flask backend reliably.

The screenshot shows the 'Launch an instance' wizard on the AWS EC2 console. The top navigation bar shows multiple tabs open, including 'AWS Region Configuration', 'Student - Skill Wallet', 'troven.in/students/labs/68...', 'troven.in/students/labs/68...', 'Launch an instance | EC2', and 'UPDATED EXPLANATION...'. The main page title is 'Launch an instance'. A blue banner at the top says 'It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices' with a 'Take a walkthrough' button. Below this, the 'Launch an instance' section starts with 'Name and tags' where 'Travelgoproject' is entered. The 'Application and OS Images (Amazon Machine Image)' section lists 'Amazon Linux 2023 AMI 2023.7.2...read more' (ami-05ffe3c48a9991133). The 'Summary' section shows 'Number of instances: 1'. Other settings include 'Virtual server type (instance type): t2.micro', 'Firewall (security group): New security group', and 'Storage (volumes): 1 volume(s) - 8 GiB'. A note about the 'Free tier' is visible on the right. At the bottom, there are buttons for 'CloudShell', 'Feedback', and links to '© 2025, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

Create a key pair named Travelgo to securely connect to your EC2 instance via SSH. Download and store the .pem file safely, as it will be required for future logins. While setting up the firewall, configure the security group to allow inbound traffic on ports 22 (SSH) and 5000 (Flask). This ensures both secure access and proper functioning of the web application. It's recommended to restrict SSH access to your specific IP range for added security. The Flask port (5000) should be open to all only during development—limit access in production. Properly configured security groups help prevent unauthorized access while keeping your app responsive and accessible.

The screenshot shows the AWS CloudShell interface with multiple tabs open. The main tab displays the process of launching an EC2 instance. In the 'Instance type' section, a 't2.micro' instance is selected. Below it, the 'Key pair (login)' section is expanded, showing a 'Create key pair' modal. The modal asks for a 'Key pair name' (Travelgo) and 'Key pair type' (RSA). It also provides instructions for storing the private key securely. The 'Configure storage' section shows a 1x 8 GiB gp3 volume. The 'Firewall (security group)' section shows a new security group named 'launch-wizard-1' with rules for SSH, HTTPS, and HTTP traffic from anywhere. The 'Summary' section indicates 1 instance will be launched. The 'Software Image (AMI)' is set to Amazon Linux 2023.7.2. A 'Free tier' message is visible. The bottom right of the main window has a 'Launch instance' button.

Wait for the confirmation message like “Successfully initiated launch of instance i0e7f9bfc28481d812,” indicating the instance is being provisioned. During this process, create a key pair named Travelgo (RSA type) with .pem file format. Save this private key securely, as it will be needed for future SSH access.

The screenshot shows the AWS Lambda console with a success message: "Successfully initiated launch of instance (i-0e7f9bfc28481d812)". Below this, there's a "Launch log" section and a "Next Steps" summary.

Next Steps:

- Create billing and free tier usage alerts
- Connect to your instance
- Connect an RDS database
- Create EBS snapshot policy
- Manage detailed monitoring
- Create Load Balancer
- Create AWS budget
- Manage CloudWatch alarms

At the bottom, there's a navigation bar with links for CloudShell, Feedback, and various AWS services like S3, Lambda, and CloudWatch. The date and time shown are 02-07-2025 at 13:51.

Edit Inbound Rules:

Select the EC2 instance you just launched and ensure it's in the “running” state. Navigate to the “Security” tab, then click “Edit inbound rules.” Add a new rule with the following settings: Type – Custom TCP, Protocol – TCP, Port Range – 5000, Source – Anywhere (IPv4) 0.0.0.0/0. This allows external access to your Flask application running on port 5000.

The screenshot shows the AWS Security Groups console in the "Edit inbound rules" wizard. It lists existing rules and allows adding a new one.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-01084cb5c8716b9df	SSH	TCP	22	Custom	0.0.0.0/0
sgr-0ce5f84230320a5a2	HTTP	TCP	80	Custom	0.0.0.0/0
sgr-0bfd52d31ccca377e	HTTPS	TCP	443	Custom	0.0.0.0/0
-	Custom TCP	TCP	5000	Anywh...	0.0.0.0/0

Add rule

A warning message at the bottom states: "⚠️ 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."

Buttons at the bottom right include: Cancel, Preview changes, Save rules, and a large orange "Save rules" button.

The screenshot shows the AWS EC2 Security Groups console. A success message at the top indicates that inbound security group rules were successfully modified on security group `sg-069e3954cb2b6f875`. The main page displays the details for the security group `sg-069e3954cb2b6f875 - launch-wizard-1`. The **Details** section shows the security group name (`launch-wizard-1`), security group ID (`sg-069e3954cb2b6f875`), owner (`084828560977`), description (`launched-wizard-1 created 2025-07-02T08:18:46.54Z`), and VPC ID (`vpc-0e150e14f6bb76cf`). Below this, the **Inbound rules** tab is selected, showing four entries:

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-01084cb5c8716b9df	IPv4	SSH	TCP	22
-	sgr-0aff0e76ef524cd65	IPv4	Custom TCP	TCP	5000
-	sgr-0ce5f84230520a5a2	IPv4	HTTP	TCP	80

Modify IAM ROLE

Attach the IAM role `Studentuser` to your EC2 instance to grant secure access to AWS services like DynamoDB and SNS. This avoids hardcoding credentials and ensures your app functions with the required permissions. Make sure the role includes all necessary policies for smooth integration.

The screenshot shows the AWS EC2 Instances console. It lists one instance, `i-0e7f9bfc28481d812`, which is running and of type `t2.micro`. The instance is associated with the security group `sg-067a674658fb162b (launch-wizard-1)`. In the Actions dropdown menu, the **Modify IAM role** option is highlighted. The bottom of the screen shows the AWS taskbar with various open tabs and system status indicators.

Modify IAM role Info

Attach an IAM role to your instance.

Instance ID
i-0426bada174a7ef93 (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 ▼ Create new IAM role

Cancel Update IAM role

CloudShell Feedback

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

Verify it's you

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

EC2 Instances

Instances (1/1) Info

Success Successfully attached Studentuser to instance i-0e7f9bfc28481d812

Connect Instance state Actions Launch instances

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
TravelGoproject	i-0e7f9bfc28481d812	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b	ec2-18-205-

i-0e7f9bfc28481d812 (TravelGoproject)

Details Status and alarms Monitoring Security Networking Storage Tags

Instance summary Info

Instance ID i-0e7f9bfc28481d812	Public IPv4 address 18.205.24.220 [open address]	Private IPv4 addresses 172.31.82.161
IPv6 address	Instance state Running	Public DNS ec2-18-205-24-220.compute-1.amazonaws.com [open address]

Wait until the confirmation message appears indicating that the Studentuser role has been successfully attached to the instance. This confirms that all required permissions are now active. Once attached, proceed to connect to your EC2 instance and run your GitHub-hosted Flask application code.

The screenshot shows the AWS EC2 Connect interface. At the top, there are several browser tabs open, including 'Student - Skill Wallet', 'troven.in/students/labs/68', 'troven.in/students/labs/69', 'Connect to instance | EC2', 'Instances | EC2 | us-east-1', and 'Online Python Compiler'. The main navigation bar shows 'EC2 > Instances > i-0e7f9bfc28481d812 > Connect to instance'. Below this, a 'Connect' section is visible with an 'Info' link. A note says 'Connect to an instance using the browser-based client.' There are three tabs: 'EC2 Instance Connect', 'Session Manager', and 'SSH client' (which is selected). An 'EC2 serial console' tab is also present. Under 'Instance ID', it shows 'i-0e7f9bfc28481d812 (TravelGoproject)'. Step-by-step instructions are provided: 1. Open an SSH client. 2. Locate your private key file. The key used to launch this instance is 'Travelgo.pem'. 3. Run this command, if necessary, to ensure your key is not publicly viewable. (chmod 400 "Travelgo.pem") 4. Connect to your instance using its Public DNS: 'ec2-18-205-24-220.compute-1.amazonaws.com'. An example command is shown: 'ssh -i "Travelgo.pem" ec2-user@ec2-18-205-24-220.compute-1.amazonaws.com'. A note at the bottom states: '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.'



 The following are the terminal outputs after executing the commands shown below the image. These outputs indicate that the setup steps have been successfully carried out. The commands summary will be provided at last.

- sudo yum install git----- 24
 - git clone your_repository_url • cd your_project_directory • sudo yum install python3----- 24
 - sudo yum install python3-pip----- 24

```

Windows PowerShell
Total download size: 1.9 M
Installed size: 11 M
Downloading Packages:
(1/2): libcrypt-compat-4.4.33-7.amzn2023.x 2.5 MB/s | 92 kB    00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.11.n 31 MB/s | 1.8 MB    00:00
Total          20 MB/s | 1.9 MB  00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing      : libcrypt-compat-4.4.33-7.amzn2023.x86_64           1/1
Installing   : libcrypt-compat-4.4.33-7.amzn2023.x86_64           1/2
Installing   : python3-pip-21.3.1-2.amzn2023.0.11.noarch        2/2
Running scriptlet: python3-pip-21.3.1-2.amzn2023.0.11.noarch        2/2
Verifying     : libcrypt-compat-4.4.33-7.amzn2023.x86_64           1/2
Verifying     : python3-pip-21.3.1-2.amzn2023.0.11.noarch        2/2
Installed:
libcrypt-compat-4.4.33-7.amzn2023.x86_64
python3-pip-21.3.1-2.amzn2023.0.11.noarch

Complete!
[ec2-user@ip-172-31-94-133 Travel-Go]$ pip install flask
Defaulting to user installation because normal site-packages is not writeable
Collecting flask
  Downloading Flask-3.1.1-py3-none-any.whl (103 kB)
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting markupsafe>=2.1.1
  Downloading MarkupSafe-3.0.2-cp39-cp39-manylinux2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting click>=8.1.3
  Downloading click-8.1.8-py3-none-any.whl (98 kB)
Collecting blinker>=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting importlib-metadata>=3.6.0
  Downloading importlib_metadata-8.7.0-py3-none-any.whl (27 kB)
Collecting jinja2>=3.1.2

```

```

Windows PowerShell
Python 3.9.23
[ec2-user@ip-172-31-88-102 Travel-Go]$ python app.p
-bash: python: command not found
[ec2-user@ip-172-31-88-102 Travel-Go]$ python app.py
-bash: python: command not found
[ec2-user@ip-172-31-88-102 Travel-Go]$ Python app.py
-bash: Python: command not found
[ec2-user@ip-172-31-88-102 Travel-Go]$ python app.py
-bash: python: command not found
[ec2-user@ip-172-31-88-102 Travel-Go]$ 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://172.31.88.102:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 526-078-484
103.42.250.170 - - [01/Jul/2025 08:30:33] "GET / HTTP/1.1" 200 -
103.42.250.170 - - [01/Jul/2025 08:30:34] "GET /static/images/travel-bg.jpg HTTP/1.1" 404 -
103.42.250.170 - - [01/Jul/2025 08:30:34] "GET /favicon.ico HTTP/1.1" 404 -
103.42.250.170 - - [01/Jul/2025 08:30:39] "GET /Login HTTP/1.1" 200 -
103.42.250.170 - - [01/Jul/2025 08:30:43] "GET /register HTTP/1.1" 200 -
103.42.250.170 - - [01/Jul/2025 08:30:58] "POST /register HTTP/1.1" 500 -
Traceback (most recent call last):
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 1536, in __call__
    return self.wsgi_app(environ, start_response)
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 1514, in wsgi_app
    response = self.handle_exception(e)
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 1511, in wsgi_app
    response = self.full_dispatch_request()
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 919, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 917, in full_dispatch_request
    rv = self.dispatch_request()
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 902, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args) # type: ignore[no-any-return]
  File "/home/ec2-user/Travel-Go/app.py", line 50, in register
    existing = users_table.get_item(Key={'email': email})

```

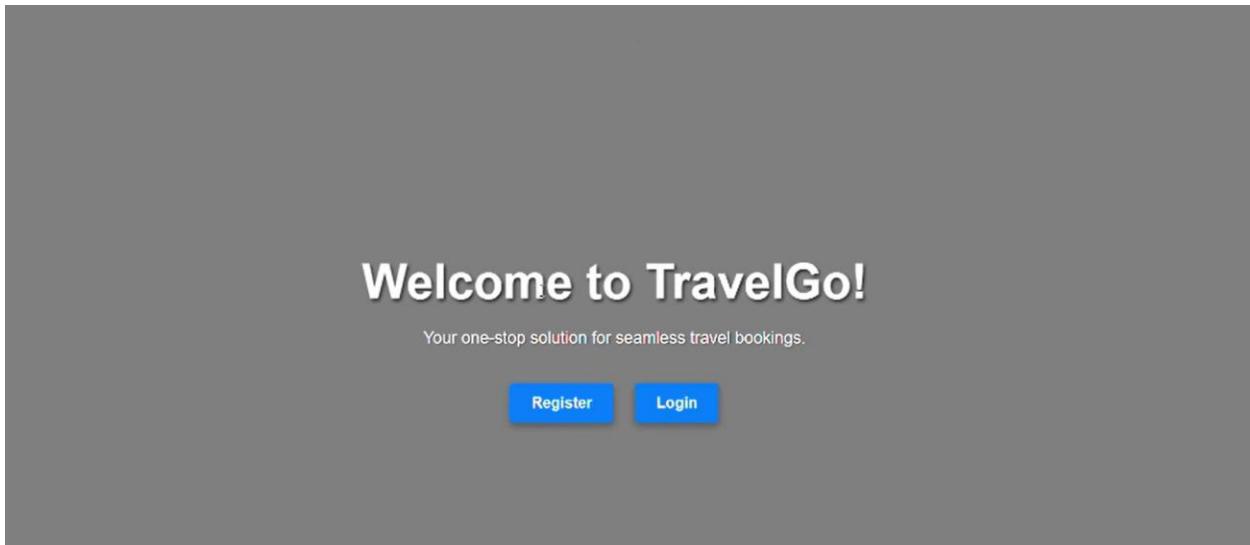
- pip install flask
- pip install boto3
- python3 app.py

Deployment Steps Summary:

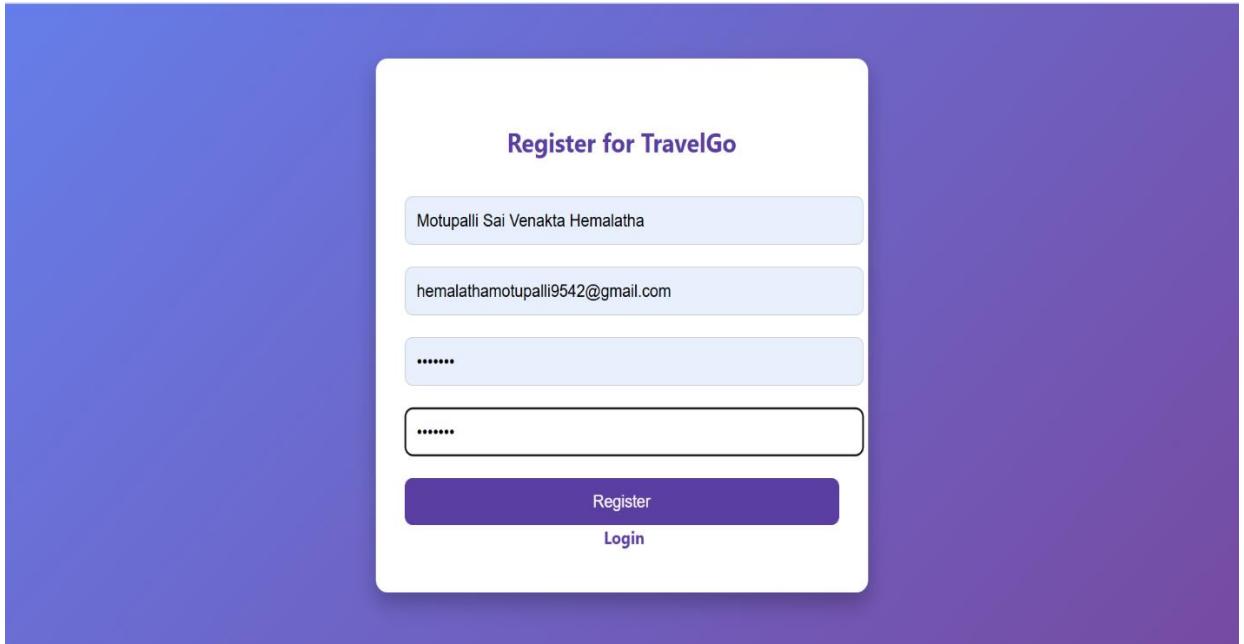
- sudo yum install git -y
- git clone your_repository_url
- cd your_project_directory
- sudo yum install python3 -y
- sudo yum install python3-pip -y
- pip install flask
- pip install boto3 • python3 app.py

User Interface Screens:

- Homepage:



- Register and Login Portals:



- **Dashboard View:**

The dashboard serves as the central hub for users to access all travel services in one place. It displays a personalized greeting along with quick navigation cards for Bus, Train, Flight, and Hotel bookings. A success alert confirms the user's login, enhancing the user experience. Below, recent and upcoming bookings are listed with full travel details and cancellation options. This intuitive layout ensures users can manage their journeys effortlessly and efficiently.

Lets have a look at my dashboard page!!!

The screenshot shows the TravelGo dashboard. At the top, there is a dark blue header bar with the "TravelGo" logo on the left and "Home" (in white), "Dashboard" (in white), and "Logout" (in white) on the right. Below the header, a large white box contains a "Welcome" message: "Welcome, hemalathamotupalli9542@gmail.com!" followed by the text "Manage all your upcoming and completed bookings here." Below this, there are four rounded rectangular buttons with icons and text: "Bus" (with a bus icon), "Train" (with a train icon), "Flight" (with a flight icon), and "Hotel" (with a hotel icon). Further down, a section titled "Your Bookings" displays a "Hotel Booking" for "Taj Falaknuma Palace" located in "Hyderabad" from "2025-07-16" to "2025-07-18" for "Suite" room type, "1" room, and "1" guest.

- **Booking Tabs: Bus, Train, Flight, Hotel**

For demonstration purposes, a seat selection section has also been included in the bus booking module. This allows users to choose their preferred seats during the booking process, enhancing the overall user experience.

This interactive feature simulates real-world booking platforms and showcases the system's dynamic capabilities. It also helps users visualize the layout before confirming their reservation.

- **Bus booking:**

The screenshot displays the TravelGo application interface. At the top, there is a search bar with fields for 'From' (Hyderabad), 'To' (Vijayawada), 'Date' (02-07-2025), and 'Passenger Count' (1). Below the search bar are filters for bus types: AC, Non-AC, Sleeper, Semi-Sleeper, and Seater. A dropdown menu for 'Sort by Price' is set to 'None'. The search results show three bus options:

- Orange Travels**: AC Sleeper at 08:00 AM for ₹800/person. A green 'Book' button is available.
- Garuda Express**: Non-AC Seater at 11:00 AM for ₹400/person. A green 'Book' button is available.
- Greenline Travels**: AC Sleeper at 09:00 PM for ₹850/person. A green 'Book' button is available.

Below the search results, the TravelGo navigation bar includes links for Home, Dashboard, and Logout. The main content area shows a confirmation message: "✓ Confirm Bus Booking". The booking details are listed as follows:

Booking Details:

- Bus Name:** TSRTC Travels
- Route:** Hyderabad to Vijayawada
- Date:** 2025-07-22
- Time:** 16:54 PM
- Type:** Non-AC Seater
- Passengers:** 1
- Selected Seats:** S7
- Total Price:** ₹833.00

At the bottom of the confirmation page are two buttons: 'Confirm Booking' (blue) and 'Go Back to Search' (grey).

- **Train booking:**

The train booking module enables users to search and reserve train tickets by selecting routes, dates, and times. It fetches real-time data and displays available options tailored to user input.

Once a booking is made, the system confirms the reservation and stores the details in the database. Bookings are reflected instantly on the user dashboard for easy tracking. This module ensures a seamless and efficient booking experience for train travelers.

The screenshot shows a search results page for a train booking. At the top, there is a search bar with dropdowns for 'From' (Hyderabad), 'To' (Delhi), date (27-06-2025), passengers (1), and a 'Search' button. Below the search bar, the results for 'Duronto Express (12285)' are displayed. The results include the train name, departure and arrival times, date, price per person (₹1800), and total price for 1 person (₹1800). A green 'Book Now' button is visible at the bottom of the result card.

The screenshot shows a confirmation page for a train booking. At the top, it says 'TravelGo' and has links for 'Home', 'Dashboard', and 'Logout'. The main content area has a title '✓ Confirm Train Booking' and a green message box stating 'Train booking confirmed successfully!'. Below this, under 'Booking Details:', it lists the train name (Duronto Express (12285)), route (Hyderabad to Delhi), date (2025-06-27), departure and arrival times (07:00 AM and 05:00 AM respectively), type, and passengers (1). It also displays the total price (₹1,800.00). At the bottom, there are two buttons: 'Confirm Booking' (green) and 'Go Back to Search' (grey).

- **Flight booking:**

✈ The flight booking section enables users to search and reserve flights quickly based on their preferred source, destination, and date. Upon entering the details, the system fetches available flight options and displays them with timing, pricing, and route information. The interface is clean and responsive, allowing users to confirm bookings with just a few clicks. Once booked,

flight details are stored securely and reflected in the dashboard. This module ensures a smooth and efficient booking experience tailored for air travel.

The screenshot shows the TravelGo flight booking interface. At the top, there's a search bar with fields for departure (Hyderabad), destination (Mumbai), date (02.07.2025), passengers (1), and a search button. Below the search bar, a flight result is displayed for IndiGo 6E-234, departing from Hyderabad to Mumbai on 2025-07-02 at 08:00 and arriving at 09:30. The total price is ₹3500. A green "Book" button is visible. A modal window titled "Confirm Your Flight Booking" lists the flight details and passenger information, ending with a "Confirm Booking" button.

- **Hotel booking:**

The hotel booking section allows users to search and reserve accommodations based on their destination and travel dates. The interface displays available hotels with details such as location, price, and amenities. Users can easily compare options and proceed with booking in just a few

clicks. Once confirmed, the booking details appear on the dashboard for easy tracking. The process is designed to be seamless, intuitive, and efficient for all types of travelers.

The screenshot displays a travel booking interface. At the top, a search bar includes dropdowns for location ('Hyderabad'), check-in date ('02-07-2025'), check-out date ('03-07-2025'), room count ('1'), and guest count ('1'). Below the search bar are several filter buttons: '5-Star', '4-Star', '3-Star', 'WiFi', 'Pool', and 'Parking'. A 'Sort by' dropdown is set to 'None'. Two hotel results are shown: 'Taj Falaknuma Palace' (5-Star, ₹25000/night) and 'The Park' (3-Star, ₹5500/night). Each result includes amenities like WiFi, Pool, and Restaurant, and a green 'Book' button. Below the main search area is a 'Confirm Your Hotel Booking' dialog box. It lists the selected details: Hotel: Taj Falaknuma Palace, Location: Hyderabad, Check-in: 2025-07-02, Check-out: 2025-07-03, Rooms: 1, Guests: 1, Price/night: ₹25000, Total nights: 1, and Total Cost: ₹25000. A large green 'Confirm Booking' button is at the bottom.

At the bottom of the dashboard, users can view a summary of all their bookings, including travel details, dates, and status. A dedicated “Cancel” button is provided next to each entry, allowing users to easily cancel any upcoming reservations if needed. This section provides a centralized view for managing bookings efficiently. It ensures transparency by displaying every confirmed ticket in an organized format. The cancel feature updates the backend in real time, removing the booking from the list and database. This functionality enhances user control and flexibility while planning or modifying travel. It also improves overall convenience by reducing the need to revisit individual booking sections. [All Bookings:](#)

Flight Booking
Flight: IndiGo GE-234
Route: Hyderabad → Mumbai
Date: 2025-07-02
Departure: 06:00
Arrival: 09:30
Passengers: 1
Booked On: 2025-07-02

₹3,500.00
[Cancel Booking](#)

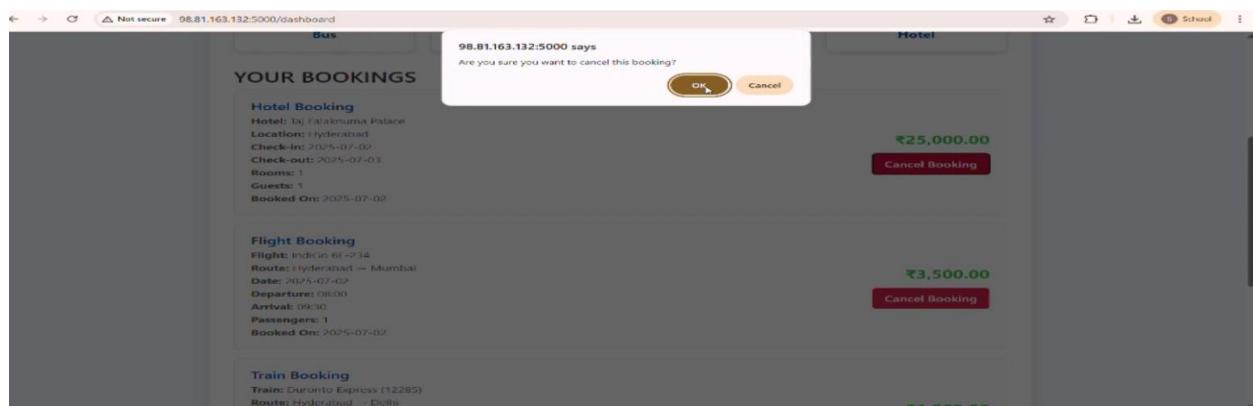
Train Booking
Train: Duronto Express (12285)
Route: Hyderabad → Delhi
Date: 2025-06-27
Time: 07:00 AM - 05:00 AM
Passengers: 1
Seats: S26
Booked On: 2025-07-02

₹1,800.00
[Cancel Booking](#)

Bus Booking
Bus: Orange Travels
Route: Hyderabad → Vijayawada
Date: 2025-07-02
Time: 08:00 AM
Seats: S14
Passengers: 1
Booked On: 2025-07-02

₹800.00
[Cancel Booking](#)

Cancel Bookings:



Once a booking is cancelled, a confirmation message appears indicating the cancellation was successful. This real-time feedback reassures the user that their action has been completed without issues. It also helps avoid confusion or repeated attempts. The booking entry is immediately removed from the dashboard view. This seamless flow enhances the overall usability and responsiveness of the application.

Let's have a look at the page indicating the cancellation was successful.

Welcome, hemalathamotupalli9542@gmail.com!

Manage all your upcoming and completed bookings here.

 Bus

 Train

 Flight

 Hotel

Your Bookings

Hotel Booking: Taj Falaknuma Palace

Location: Hyderabad

Check-in: 2025-07-16

Check-out: 2025-07-18

Room Type: Suite

Rooms: 1

Guests: 1

Final Conclusion – Elevating Travel with TravelGo:

TravelGo stands as a dynamic and cloud-powered solution that redefines how users plan and book their journeys. By seamlessly integrating Flask for backend logic, AWS EC2 for scalable deployment, DynamoDB for reliable data storage, and SNS for real-time notifications, the platform delivers a smooth, responsive, and user-centric experience.

From login to booking, and from instant alerts to effortless cancellations, every feature is built with efficiency and accessibility in mind. Whether it's buses, trains, flights, or hotels—TravelGo offers a unified platform that adapts to diverse travel needs. Its modern architecture ensures high performance, security, and scalability even under peak load conditions.  **With**

TravelGo, travel planning is no longer a hassle—it's an experience.

 Smart. Fast. Reliable. That's the future of travel.