

# Travel Go: A Cloud-Powered Real-Time Travel Booking Platform

## Project Description:

**TravelGo simplifies travel planning by providing a single platform to book buses, trains, flights, and hotels.**

As demand grows for real-time and hassle-free travel services, TravelGo meets this need using cloud tools.

It uses Flask for backend growth, AWS EC2 for hosting, DynamoDB for data storage, and AWS SNS to send instant Alerts.

Users can easily register, log in, search, and book transport or stay.

The system also includes features like booking history and dynamic seat selection.

These tools ensure a smooth, fast, and ready for every traveler.

## Case 1: Illustrates Real-Time Booking for Users.

Using TravelGo, users log in, choose travel type –bus, train, flight, or hotel—and enter their booking choice.

The Flask backend processes these inputs and retrieves matching options from DynamoDB.

AWS EC2 handles traffic and delivers fast response times once the user confirms a booking. Thanks to its real-time design, the platform allows users to book travel without delays or long page loads—even during peak seasons.

## Case 2: shows how TravelGo sends email alerts after a

## **booking.**

Once the user confirms a booking, TravelGo uses AWS SNS to send the booking details by email.

For example, when a user books a flight, Flask handles the transaction while AWS SNS sends an email to the user and notifies the service provider if needed.

DynamoDB securely stores all booking data.

This smooth and fast communication helps build user trust and enhances the overall experience.

## **Case 3: explains how users book easily and access their records without delays.**

Users can log in at any time to view or manage their previous bookings.

For example, a user may open the dashboard to check a hotel booking made a week earlier.

Flask retrieves this data from DynamoDB, ensuring instant access.

TravelGo's cloud-based design provides more handiness, while its simple interface makes it easy to cancel bookings or plan new trips.

With AWS EC2 handling the hosting, the platform runs smoothly—even when many users are online at once.

### [\*\*AWS setup\*\*](#)

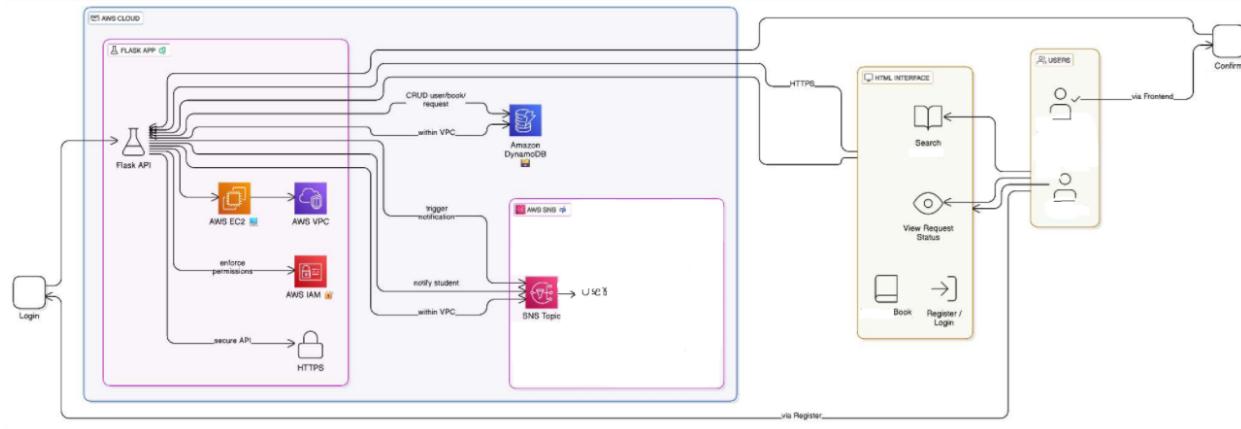
TravelGo uses Amazon Web Services (AWS) to build a fast and real-time travel booking platform..

- **AWS EC2** (Elastic Compute Cloud) hosts the Flask-based backend coating. It ensures high handiness and fast response times, even under heavy user load.
- **Amazon DynamoDB** serves as the primary database for storing user

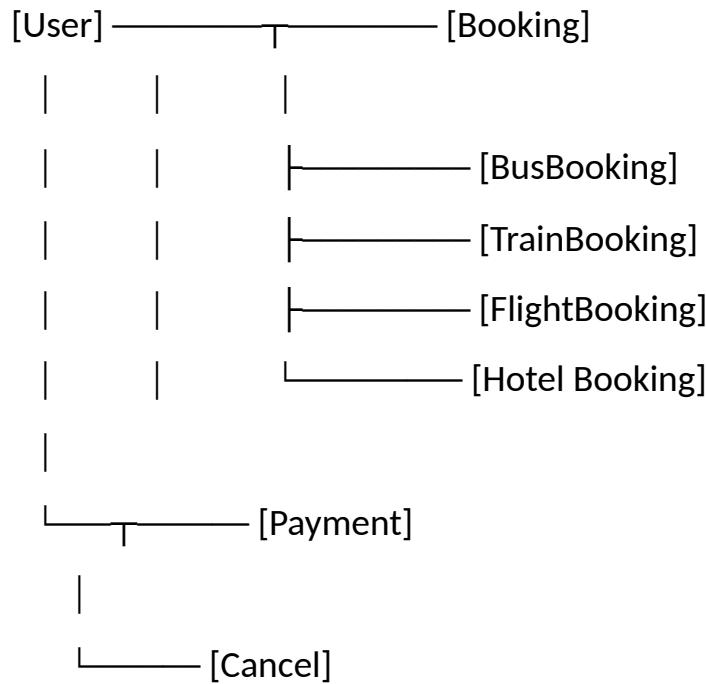
data, booking history, and travel records. Its low-latency performance supports instant data retrieval.

- **AWS SNS** enables real-time messages. *The system sends instant email alerts to users after they confirm a booking and can also notify service providers*

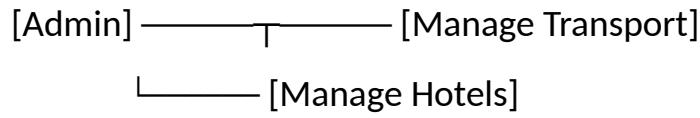
, IAM roles securely connect EC2 instances to other AWS services without hard coding credentials.

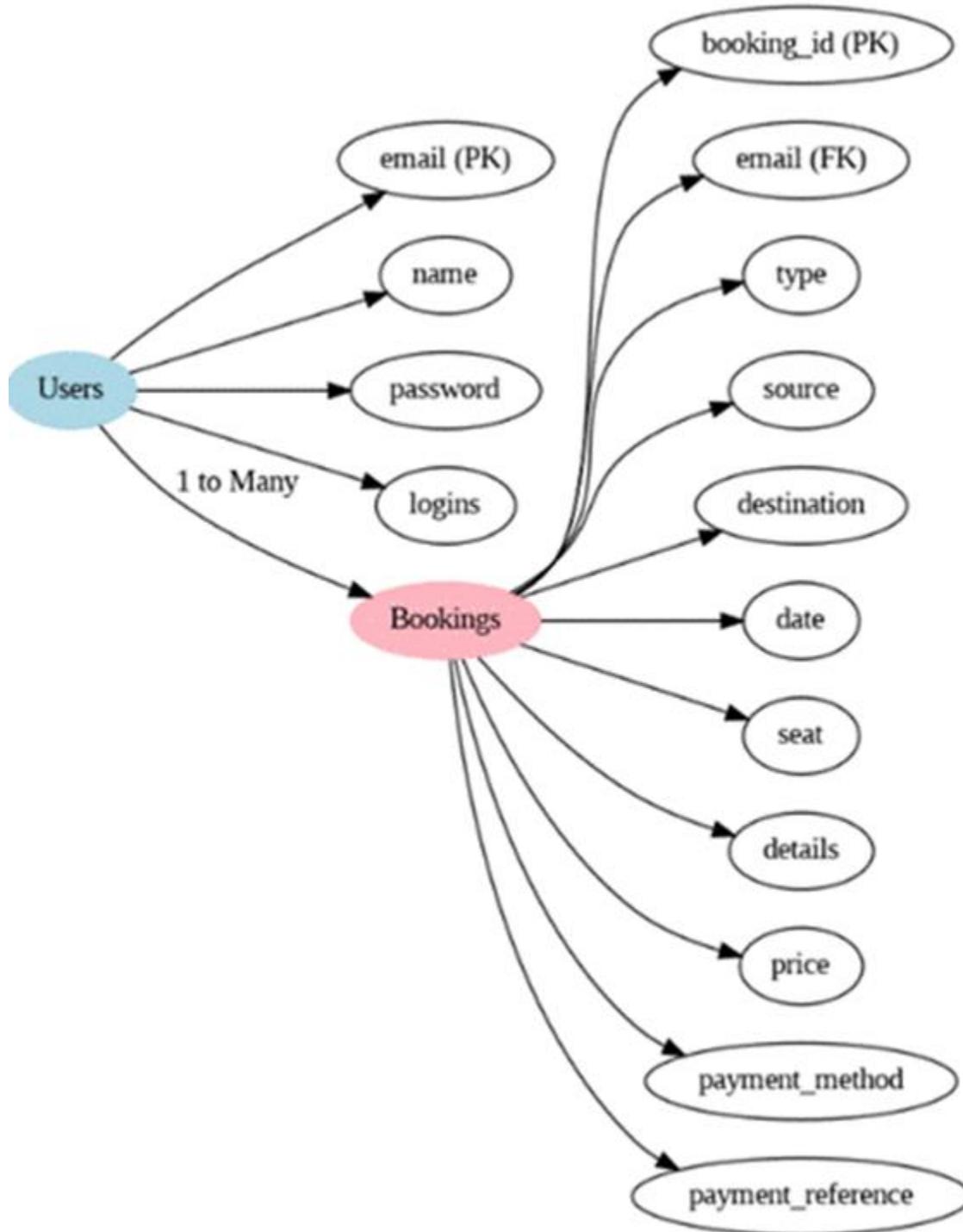


(ER)Diagram



# TRAVELGO



**Pre-requisites:**

1. Set up an AWS Account Setup: [AWS Account Setup](#)

2. see IAM: [IAM Overview](#)
3. Set up an Amazon EC2 Basics: [EC2](#)
4. Get DynamoDB Basics: [DynamoDB](#)
5. Set up SNS: [SNS support](#)
6. Use Git: [Git Document](#)

## The Work Flow Of TravelGo Project

### 1. Start With AWS Account Setup and Login

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

**action 1.2:** Log in to the AWS Console

### 2. DynamoDB Database Creation and Setup

**action 2.1:** Create a DynamoDB Table.

**action 2.2:** Configure Attributes for User Data and Book Requests.

### 3. SNS Setup

**action 3.1:** Create SNS topics for book request notice.

**action 3.2:** Subscribe users and library staff to SNS email.

### 4. Setup Backend and App

**action 4.1:** Develop the Backend Using Flask.

**action 4.2:** Integrate AWS Services Using boto3.

### 5. Setup IAM Role

**action 5.1:** Create IAM Role

**action 5.2:** Attach Policies

### 6. Setup EC2 Instance

**action 6.1:** Launch an EC2 instance to host the Flask.

**action 6.2:** Configure protection groups for HTTP, and SSH access.

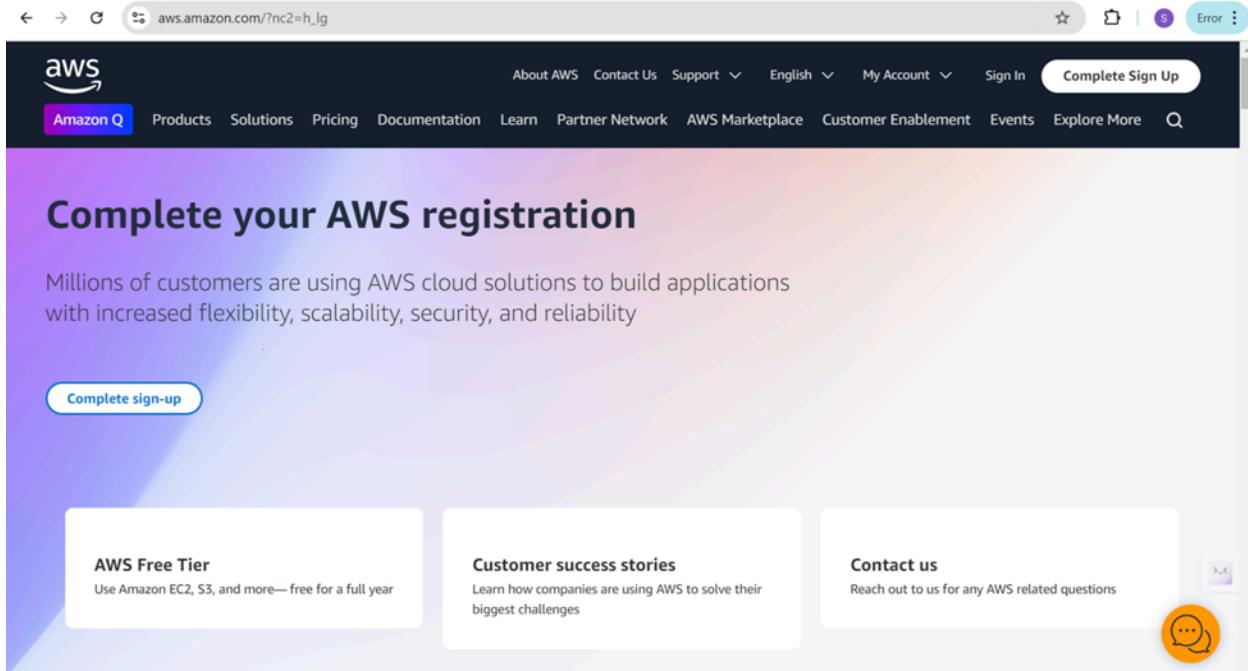
### 7. Now Deployment on EC2

**action 7.1:** Upload Flask Files

**action 7.2:** Run the Flask App

### 8. Let us Testing and Deploy

#### 1: Setup AWS Account and Login



## 2: Next Log in to the AWS Console

The left side of the image shows the AWS sign-in interface. It features the AWS logo and a 'Sign in' button. Two radio button options are shown: 'Root user' (selected) and 'IAM user'. Below these are fields for 'Root user email address' (containing 'username@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.'

The right side of the image shows a dark purple sidebar titled 'AI Use Case Explorer'. The title is in large white font. Below it, text reads: 'Discover AI use cases, customer success stories, and expert-curated implementation plans'. At the bottom of the sidebar is a 'Explore now >' button.

## 3. Create IAM Roles

# TRAVELGO

The screenshot shows the AWS IAM Dashboard. On the left, a sidebar lists 'Identity and Access Management (IAM)' and various management options like User groups, Policies, and Account settings. The main area displays the 'IAM Dashboard' with a message about new access analyzers. It highlights two 'Access denied' errors:

- User:** arnawssts:713881794827:assumed-role/rsoaccount-new/6801da4369  
**Action:** iam:GetAccountSummary  
**Context:** no identity-based policy allows the action
- User:** arnawssts:713881794827:assumed-role/rsoaccount-new/6801da4369  
**Action:** iam:ListAccountAliases  
**Context:** no identity-based policy allows the action

Below these, sections for 'What's new' and 'Tools' are visible. The bottom of the screen shows the Windows taskbar with various pinned icons.

The screenshot shows the 'Create role' wizard, Step 1: Select trusted entity. The sidebar shows steps 2 and 3. The main area is titled 'Select trusted entity' and includes a 'Trusted entity type' section with five options:

- AWS service** (selected): Allows AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account**: Allows 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**: Allows 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.

Below this is a 'Use case' section where 'EC2' is selected as the service or use case. The bottom of the screen shows the Windows taskbar.

**Permissions policies (3/1060) Info**

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

Filter by Type: All types | 33 matches

| Policy name                            | Type        | Description  |
|--|-------------|--|
| AmazonEC2ContainerRegistryFullAccess   | AWS managed | Provides administrative access to Amazon EC2 Container Registry                |
| AmazonEC2ContainerRegistryPowerUser    | AWS managed | Provides full access to Amazon EC2 Container Registry                          |
| AmazonEC2ContainerRegistryPullOnly     | AWS managed | Provides access to pull images from Amazon EC2 Container Registry              |
| AmazonEC2ContainerRegistryReadOnly     | AWS managed | Provides read-only access to Amazon EC2 Container Registry                     |
| 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 for Amazon ECS                          |
| AmazonEC2ContainerServiceRole          | AWS managed | Default policy for Amazon ECS service role                                     |
| <b>AmazonEC2FullAccess</b>             | AWS managed | Provides full access to Amazon EC2 via the AWS SDKs and command-line interface |
| AmazonEC2ReadOnlyAccess                | AWS managed | Provides read only access to Amazon EC2  |
| AmazonEC2RoleforAWSCodeDeploy          | AWS managed | Provides EC2 access to S3 bucket to do code deployment                         |

**Role details**

**Role name**  
Enter a meaningful name to identify this role.  
**studentuser**  
Maximum 64 characters. Use alphanumeric and "+\_-.\_" characters.

**Description**  
Add a short explanation for this role.  
Allows EC2 instances to call AWS services on your behalf.  
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: \_+-.@~![]#\$%^&`~-`

**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-       "Principal": [
10-         "service::ec2.amazonaws.com"
11-       ]
12-     }
13-   ]
}

```

## 4. DynamoDB Database Creation and Setup

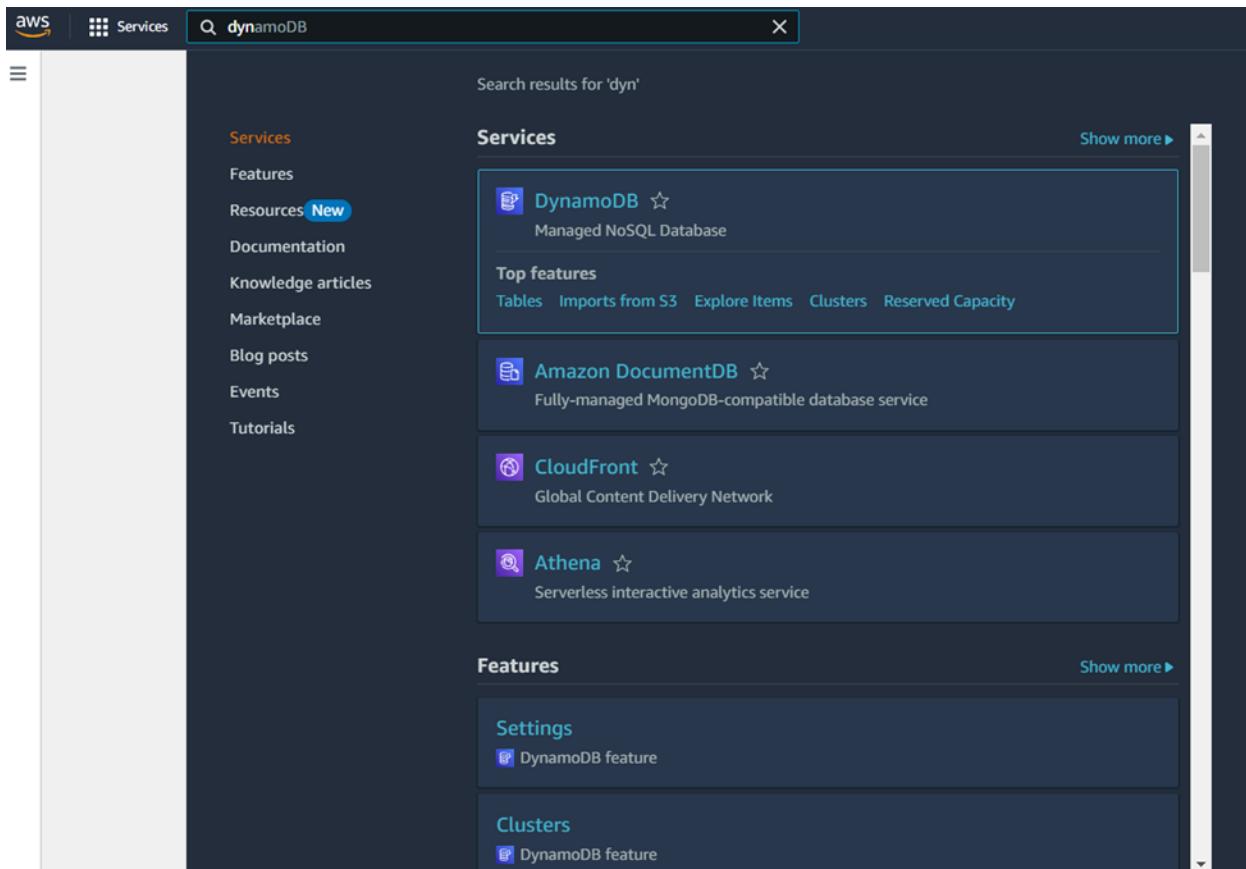
- In the AWS Console, navigate to DynamoDB and click on create tables

To manage user data and bookings, TravelGo uses **Amazon DynamoDB**, a

fast and scalable NoSQL database service offered by AWS.

To begin, you open the AWS Console and navigate to the DynamoDB section. From there, you can create a new table by giving it a name—such as "Users" or "Bookings"—and defining a primary key. The primary key could be something like a user ID or booking ID, which uniquely places each record. You can also add a sort key if you want to organize the data further, like by date.

Next, you choose the content mode.



The screenshot shows two tabs open in a browser window:

- DynamoDB > Dashboard**: This tab displays the "Dashboard" section. It includes sections for "Alarms (0)" and "DAX clusters (0)". Both sections show a search bar, a table header (e.g., "Cluster name" for DAX clusters), and a "Create" button ("Create alarm" for alarms, "Create cluster" for DAX clusters). A sidebar on the left lists various management options like "Tables", "Explore items", and "Integrations".
- DynamoDB > Tables**: This tab displays the "Tables" section. It shows a table header with columns like "Name", "Status", "Partition key", etc. A message at the top states "You have no tables in this account in this AWS Region." A prominent orange "Create table" button is located at the bottom.

## Create a DynamoDB table for storing details and book

### Requests

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

## TRAVELGO

The screenshot shows the 'Create table' wizard in the Amazon DynamoDB console. The table name is set to 'travelgo\_users'. The partition key is 'email' of type String. There is no sort key defined. Under 'Table settings', the 'Default settings' radio button is selected. The status bar at the bottom indicates it's 29°C and mostly cloudy.

Follow the same steps to create a train table with train number as the partition key.

Create a Bookings table with the partition key set as the user's email and the sort key as the booking date.

The screenshot shows the 'Create table' wizard in the Amazon DynamoDB console. The table name is set to 'trains'. The partition key is 'train\_number' of type String. There is no sort key defined. Under 'Table settings', the 'Default settings' radio button is selected. The status bar at the bottom indicates it's 29°C and mostly cloudy.

# TRAVELGO

The screenshot shows the 'Create table' wizard in the Amazon DynamoDB console. The 'Table details' section is active, displaying fields for 'Table name' (set to 'booking') and 'Partition key' (set to 'user\_email'). The 'Sort key - optional' section is also visible. Below this, the 'Table settings' section offers 'Default settings' (selected) or 'Customize settings'. The browser's address bar shows the URL for creating a table in the us-east-1 region.

**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.

**Table settings**

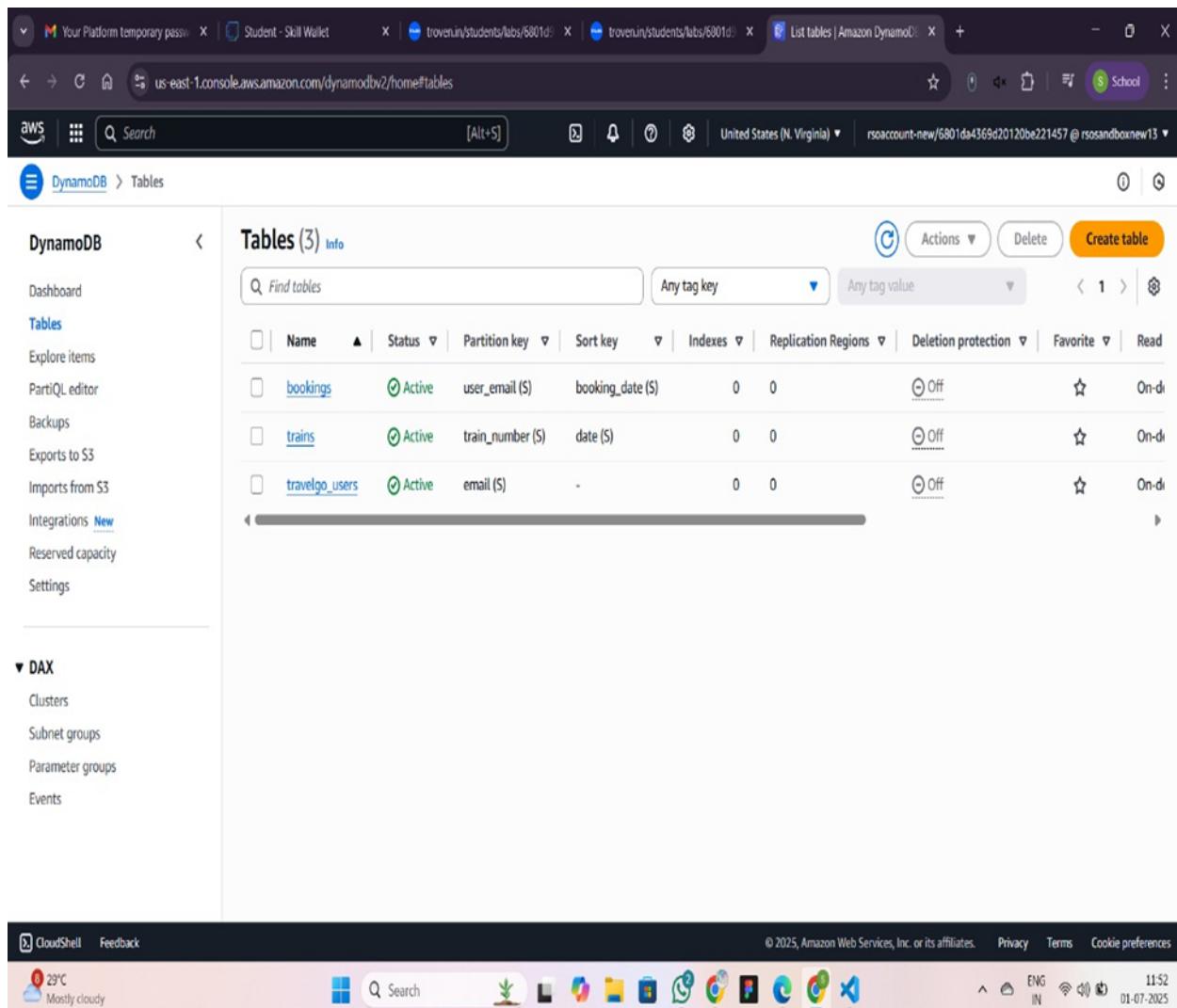
**Default settings**  
The fastest way to create your table. You can modify most of these settings after your table has been created. To learn more about table settings, see Table Settings.

**Customize settings**  
Use these advanced features to make DynamoDB work better for your needs.

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

CloudShell Feedback 29°C Mostly cloudy Search

## TRAVELGO



The screenshot shows the AWS DynamoDB console with the following details:

- Left Sidebar:** Shows the "DynamoDB" navigation bar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings.
- Tables Overview:** A table titled "Tables (3) Info" lists three tables:
  - bookings:** Active, Partition key: user\_email (\$), Sort key: booking\_date (\$), Indexes: 0, Replication Regions: 0, Deletion protection: Off, Favorite: On-disk.
  - trains:** Active, Partition key: train\_number (\$), Sort key: date (\$), Indexes: 0, Replication Regions: 0, Deletion protection: Off, Favorite: On-disk.
  - travelgo\_users:** Active, Partition key: email (\$), Sort key: -, Indexes: 0, Replication Regions: 0, Deletion protection: Off, Favorite: On-disk.
- Bottom Navigation:** Includes CloudShell, Feedback, a weather icon (29°C, Mostly cloudy), a search bar, and various AWS service icons (Lambda, CloudWatch Metrics, CloudWatch Logs, CloudWatch Metrics Insights, CloudWatch Metrics Dashboards, CloudWatch Metrics Insights Insights, CloudWatch Metrics Insights Insights Insights, CloudWatch Metrics Insights Insights Insights Insights).

### SNS Notified Setup

1. Create SNS topics for sending email to users.
2. In the AWS Console, search for “SNS” and go to the SNS dashboard.

X

Search results for 'sns'

**Services** Show more ▶

- 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

**Features** Show more ▶

- Events**  
ElastiCache feature
- SMS**  
AWS End User Messaging feature
- Hosted zones**  
Route 53 features

**Amazon SNS** X

**New Feature** Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Application Integration

## Amazon Simple Notification Service

Pub/sub messaging for microservices and serverless applications.

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.

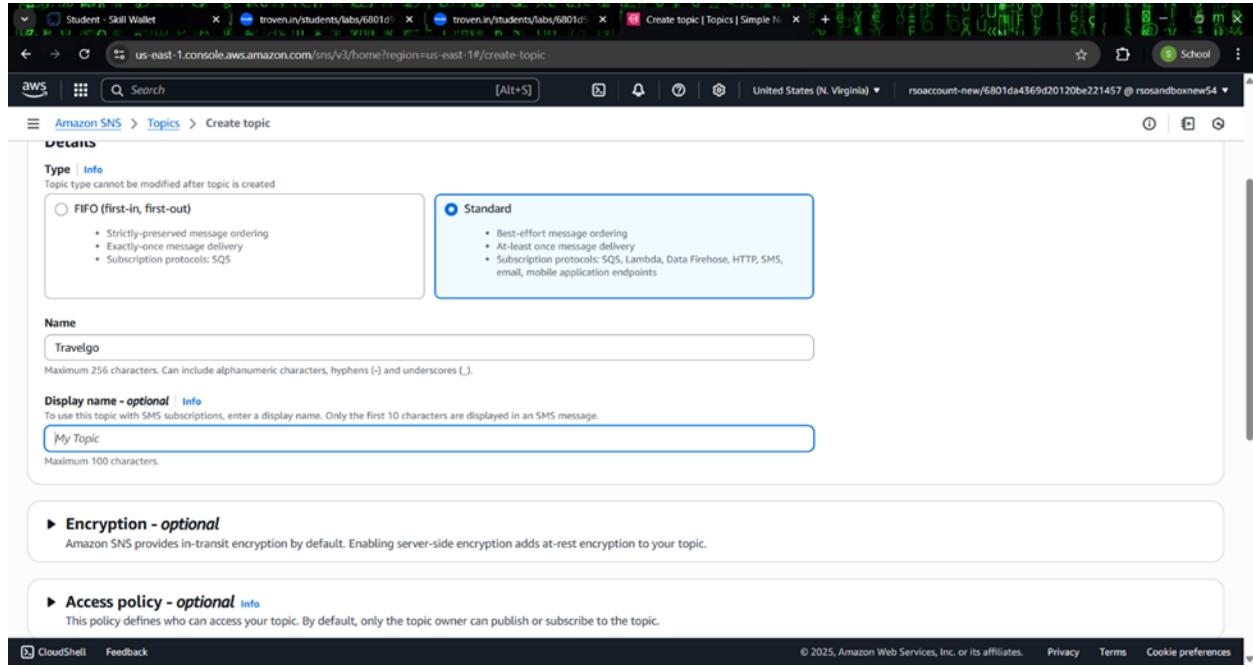
**Create topic**

Topic name  
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.

**Pricing**

# TRAVELGO

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



2. Click on **create topic**

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](#)

3. Configure the SNS topic and note down the **Topic ARN**.

4. Click on **create subscription**

## TRAVELGO

### ► Subscription filter policy - optional Info

This policy filters the messages that a subscriber receives.

### ► Redrive policy (dead-letter queue) - optional Info

Send undeliverable messages to a dead-letter queue.

Cancel

Create subscription

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

The screenshot shows an email titled "AWS Notification - Subscription Confirmation" from "Zuveriya <no-reply@sns.amazonaws.com>" to the recipient. The email was sent on "Mon 30 Jun, 16:44 (2 days ago)". It contains a message about being identified as spam and a "Report as not spam" button. Below this, it states the topic subscription chosen: "arn:aws:sns:us-east-1:194722438347:TravelGoBooking". It provides a link to "Confirm subscription" and a note about not replying directly and opting out via "sns-opt-out".

AWS Notification - Subscription Confirmation External Spam x

Zuveriya <no-reply@sns.amazonaws.com>  
to me ▾ Mon 30 Jun, 16:44 (2 days ago) ☆ ↵ :

Why is this message in spam? This message is similar to messages that were identified as spam in the past.

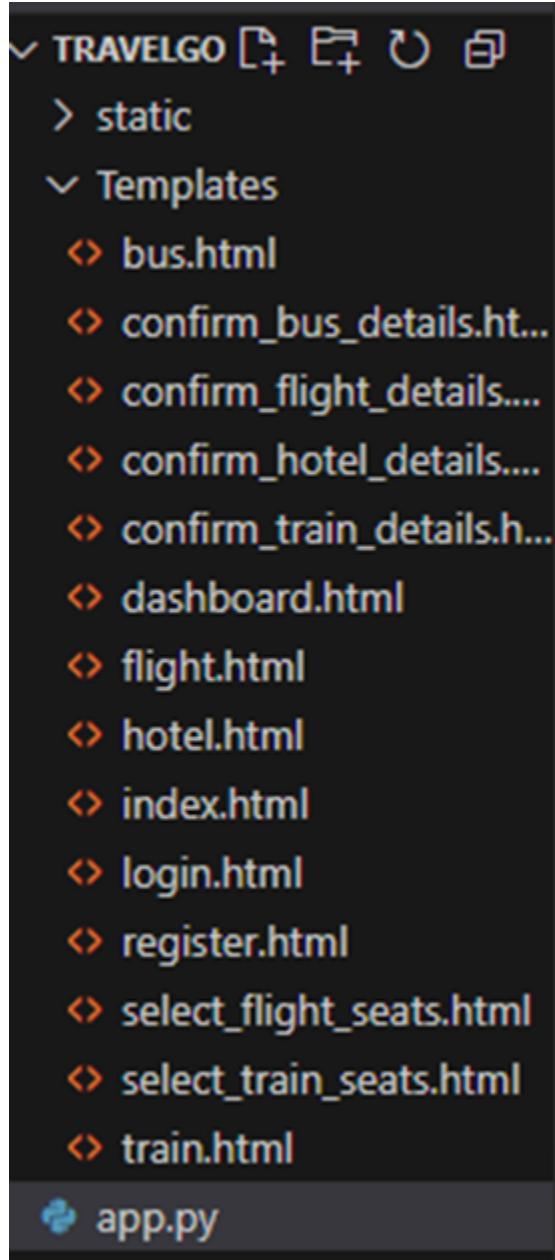
Report as not spam ⓘ

You have chosen to subscribe to the topic:  
arn:aws:sns:us-east-1:194722438347:TravelGoBooking

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](#)

Setup Backend Develop and App



### Code Description:

```
from flask import Flask, render_template, request, redirect, url_for, session, jsonify
from pymongo import MongoClient
from werkzeug.security import generate_password_hash, check_password_hash
import json
from datetime import datetime
from bson.objectid import ObjectId # Still needed for other potential ObjectIds from DB
from bson.errors import InvalidId # Import this for specific error handling
```

② Flask: format the web app.

- ❑ render template: Renders HTML templates using Jinja2.
- ❑ request: Handles incoming data from forms or API calls.
- ❑ redirect and url\_for: Used for redirect between pages (e.g., login → dashboard).
- ❑ session: Stores user-specific data across routes (e.g., user login session).
- ❑ jsonify: Converts Python data into JSON format (commonly used in API responses).

```
app = Flask(__name__)
```

format the Flask app instance using Flask(\_\_name\_\_) to start building the web app.

```
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')
```

SNS connection:

```
# Function to send SNS notifications
# This function is duplicated in the original code, removing the duplicate.
def send_sns_notification(subject, message):
    try:
        sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Subject=subject,
            Message=message
        )
    except Exception as e:
        print(f"SNS Error: Could not send notification - {e}")
        # Optionally, flash an error message to the user or log it more robustly.
```

### Routes:

- The app.py file in the TravelGo project manages all the backend routes for user interactions and booking functionality. It starts with **authentication routes** such as /signup, /login, and /logout to handle user registration, login, and logout functionalities. Once authenticated, users are redirected to the **dashboard** via the /dashboard route, which acts as a central hub for navigating through different booking options. The root route / usually leads to the landing or home page of the platform. The /bus, /train, /flight, and /hotel routes render respective booking pages where users can input their travel details.
- Each booking type has a check route like /confirm\_bus\_details, /confirm\_train\_details, etc., to validate and preview booking data.
- After check, final routes such as /final\_confirm\_bus\_booking or /final\_confirm\_flight\_booking handle storing the booking details in the database.

```
@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')

@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'))
```

## TRAVELGO

```
@app.route('/dashboard')
def dashboard():
    if 'email' not in session:
        return redirect(url_for('login'))
    user_email = session['email']

    # Query bookings for the logged-in user using the primary key 'user_email'
    # No GSI is needed here as 'user_email' is likely the partition key for the bookings_table.
    response = bookings_table.query(
        KeyConditionExpression=Key('user_email').eq(user_email),
        ScanIndexForward=False # Get most recent bookings first
    )
    bookings = response.get('Items', [])

    # Convert Decimal types from DynamoDB to float for display if necessary
    for booking in bookings:
        if 'total_price' in booking:
            try:
                booking['total_price'] = float(booking['total_price'])
            except (TypeError, ValueError):
                booking['total_price'] = 0.0 # Default value if conversion fails
    return render_template('dashboard.html', username=user_email, bookings=bookings)

@app.route('/train')
def train():
    if 'email' not in session:
        return redirect(url_for('login'))
    return render_template('train.html')

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

    booking_details = {
        'name': request.args.get('name'),
        'train_number': request.args.get('trainNumber'),
        'source': request.args.get('source'),
        'destination': request.args.get('destination'),
        'departure_time': request.args.get('departureTime'),
        'arrival_time': request.args.get('arrivalTime'),
        'price_per_person': Decimal(request.args.get('price')),
        'travel_date': request.args.get('date'),
        'num_persons': int(request.args.get('persons')),
        'item_id': request.args.get('trainId'), # This is the train ID
        'booking_type': 'train',
        'user_email': session['email'],
        'total_price': Decimal(request.args.get('price')) * int(request.args.get('persons'))
    }
```

```

@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'))

```

Deployment Code:

```

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

```

start the Flask server to listen on all network port (0.0.0.0) at port 5000 with debug mode enabled for testing

Setup EC2 Instance:

|   |                        |
|---|------------------------|
|  static    | Initial commit         |
|  templates | Update statistics.html |
|  app.py    | Update app.py          |

Launch an EC2 instance to host the Flask app.

## 1. Click on launch Instance:

# TRAVELGO

## Modify IAM Roles"

The screenshot shows the 'Modify IAM role' page in the AWS IAM console. The instance ID is 'i-0a069314eedd37bcc (TravelGoproject)'. In the 'IAM role' section, a dropdown menu is open, showing 'Studentuser' selected. Other options include 'No IAM Role' (with a note to choose this to detach an IAM role) and another 'Studentuser' entry. A 'Create new IAM role' button is also visible. At the bottom right are 'Cancel' and 'Update IAM role' buttons.

The screenshot shows the 'Launch an instance' page in the AWS EC2 console. A green success message at the top states 'Successfully initiated launch of instance (i-0a069314eedd37bcc)'. Below it, a 'Launch log' link is available. The 'Next Steps' section contains several cards: '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', and 'Manage CloudWatch alarms'. Each card has a corresponding 'Create' or 'Learn more' button.

Create Key Pair:

# TRAVELGO

## ▼ Instance type [Info](#) | [Get advice](#)

### Instance type

t2.micro

Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true  
On-Demand Linux base pricing: 0.0124 USD per Hour  
On-Demand Windows base pricing: 0.017 USD per Hour  
On-Demand RHEL base pricing: 0.0268 USD per Hour  
On-Demand SUSE base pricing: 0.0124 USD per Hour

All generations

[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

## ▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

Select

[Create new key pair](#)

### Key pair type

RSA

RSA encrypted private and public key pair

ED25519

ED25519 encrypted private and public key pair

### Private key file format

.pem

For use with OpenSSH

.ppk

For use with PuTTY

 When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#) 

[Cancel](#)

[Create key pair](#)



Travelgo.pem  
Type: PEM File

## TRAVELGO

The screenshot shows the AWS CloudWatch Metrics Insights interface. A search bar at the top contains the query: "AWS CloudWatch Metrics Insights". Below the search bar, there are two tabs: "Metrics" and "Logs". Under the "Metrics" tab, there is a table with three columns: "Metric Name", "Dimensions", and "Time Range". The first row shows "AWS CloudWatch Metrics Insights" with dimensions "Region:us-east-1" and a time range from "1 hour ago" to "Now". The second row shows "AWS CloudWatch Metrics Insights" with dimensions "Region:us-east-1" and a time range from "1 hour ago" to "Now". At the bottom of the interface, there is a "Run" button.

Start by attaching an IAM role to your EC2 instance to connect using EC2 Instance Connect. You can do this by selecting your instance, clicking on **Actions**, then navigate to **surety** and selecting **Modify IAM Role** to attach the suitable role.c. Select the **EC2 instance** you wish to connect to. In the top of the **EC2 Dashboard**, click the **Connect** button. From the menu, choose **EC2 Instance Connect**. Finally, click **Connect** again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser

## TRAVELGO

The screenshot shows the AWS Management Console with the URL <https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#SecurityGroup:group-id=sg-072f73d3e236a3496>. The page displays the details of the security group **sg-072f73d3e236a3496 - launch-wizard-1**. The **Inbound rules** section lists four entries:

| Name | Security group rule ID | IP version | Type  | Protocol | Port range |
|------|------------------------|------------|-------|----------|------------|
| -    | sgr-0950ba6cd12516aa6  | IPv4       | HTTPS | TCP      | 443        |
| -    | sgr-01fd87def5fa08a2b  | IPv4       | SSH   | TCP      | 22         |
| -    | sgr-0f8323e1c3b1f02c3  | IPv4       | HTTP  | TCP      | 80         |

Below the table, there is a note: "Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>".

```
PS C:\Users\HP> ssh -i "C:\Users\HP\Downloads\Travelgo.pem" ec2-user@ec2-35-172-135-176.compute-1.amazonaws.com
The authenticity of host 'ec2-35-172-135-176.compute-1.amazonaws.com (35.172.135.176)' can't be established.
ED25519 key fingerprint is SHA256:ydiIf1dUE8jHE2hu/EvOQX0cmX4WJwNaOTB7EedoNU8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-35-172-135-176.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

  _#
 /_#####
 \#####
  \##|
   \|/ --- https://aws.amazon.com/linux/amazon-linux-2023
    V~' '-->
     ~~-.-
      / \
     _/_/
```

First, install and set up the AWS CLI with your credentials to connect to an AWS EC2 instance using PowerShell. The EC2 instance must have a public IP address, port 22 open in its protection group (for Linux), and the correct IAM role attached if you're using EC2 Instance Connect.

Next, open PowerShell and use the SSH command with your .pem key file

to connect. Before that, ensure the `.pem` file has the correct permissions to avoid access errors.

If you are using EC2 Instance Connect, you can connect directly through the AWS Console or by sending your SSH public key using a CLI command.

This process allows you to securely access your EC2 instance from your local machine using PowerShell.

```
~/m/
[ec2-user@ip-172-31-90-127 ~]$ sudo yum install git -y
Amazon Linux 2023 Kernel Livepatch repository
7 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-URI          noarch      1.37-477.amzn2023.0.7   amazonlinux
36 k
=====
9.65 MB/s | 1
=====
1/1 [done]
```

Once you've connected to your EC2 instance using PowerShell, the next step is to install the required Python libraries. These include Flask for building the web app and PyMongo for connecting to MongoDB. You can install them by running the suitable `pip` commands.

If your project includes a `requirements.txt` file, it's recommended to use that file to install all essential libraries in one go. This helps ensure that the EC2 surroundings have everything needed to run your app smoothly.

## TRAVELGO

```
Complete!
[ec2-user@ip-172-31-90-127 ~]$ git clone https://github.com/Zuveriya2527/TravelGo.git
Cloning into 'TravelGo'...
remote: Enumerating objects: 41, done.
remote: Counting objects: 100% (41/41), done.
remote: Compressing objects: 100% (33/33), done.
remote: Total 41 (delta 6), reused 34 (delta 3), pack-reused 0 (from 0)
Receiving objects: 100% (41/41), 2.46 MiB | 33.14 MiB/s, done.
Resolving deltas: 100% (6/6), done.
[ec2-user@ip-172-31-90-127 ~]$ cd TravelGo
[ec2-user@ip-172-31-90-127 TravelGo]$ sudo yum install python3 -y

sudo yum install python3-pip -y

pip install flask

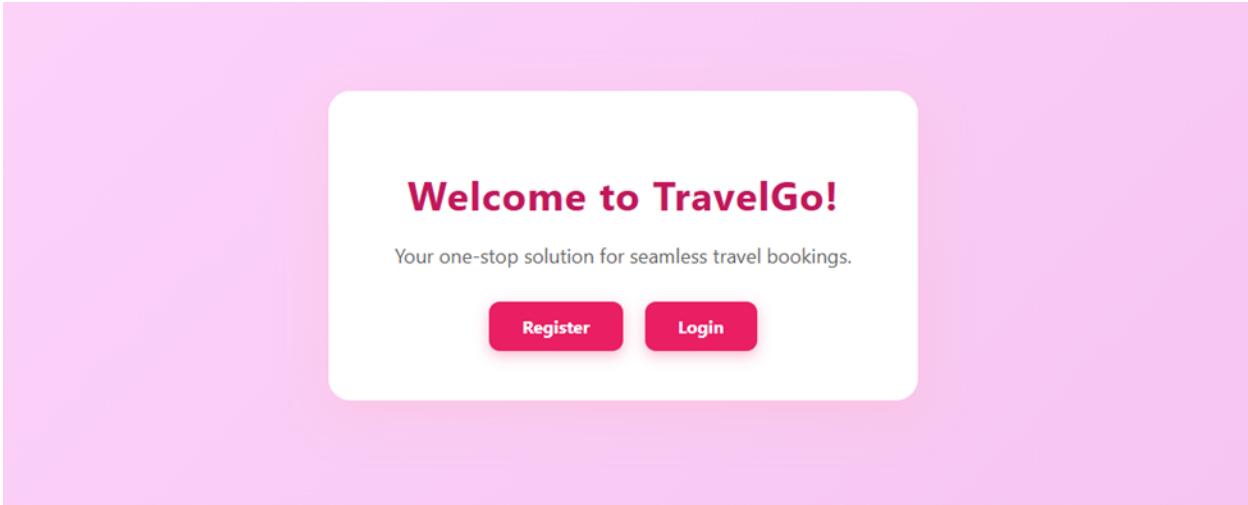
pip install boto3
Last metadata expiration check: 0:03:23 ago on Mon Jun 30 11:29:21 2025.
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:03:23 ago on Mon Jun 30 11:29:21 2025.
Dependencies resolved.
=====
Package           Architecture   Version      Repository      Size
=====
Installing:
python3          x86_64        3.9.23-1    amzn2023.0.1   1.9 M
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.39.0,>=1.38.46->boto3) (2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.39.0,>=1.38.46->boto3) (1.25.10)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.39.0,>=1.38.46->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.38.46 botocore-1.38.46 s3transfer-0.13.0
[ec2-user@ip-172-31-90-127 TravelGo]$ python3 app.py
python3: can't open file '/home/ec2-user/TravelGo/app.py': [Errno 2] No such file or directory
[ec2-user@ip-172-31-90-127 TravelGo]$ ls -la
total 4
drwxr-xr-x. 4 ec2-user ec2-user 49 Jun 30 11:31 .
drwxr-----. 6 ec2-user ec2-user 118 Jun 30 11:32 ..
drwxr-xr-x. 8 ec2-user ec2-user 163 Jun 30 11:31 .git
-rw-r--r--. 1 ec2-user ec2-user 26 Jun 30 11:31 README.md
drwxr-xr-x. 4 ec2-user ec2-user 86 Jun 30 11:31 Travel
[ec2-user@ip-172-31-90-127 TravelGo]$ cd Travel
[ec2-user@ip-172-31-90-127 Travel]$ ls -la
total 48
drwxr-xr-x. 4 ec2-user ec2-user 86 Jun 30 11:31 .
drwxr-xr-x. 4 ec2-user ec2-user 49 Jun 30 11:31 ..
-rw-r--r--. 1 ec2-user ec2-user 46 Jun 30 11:31 .gitignore
-rw-r--r--. 1 ec2-user ec2-user 158 Jun 30 11:31 README.md
-rw-r--r--. 1 ec2-user ec2-user 22920 Jun 30 11:31 app.py
drwxr-xr-x. 4 ec2-user ec2-user 47 Jun 30 11:31 static
drwxr-xr-x. 2 ec2-user ec2-user 16384 Jun 30 11:31 templates
```

```
Last login: Mon Jun 30 11:28:37 2025 from 103.174.110.191
[ec2-user@ip-172-31-90-127 ~]$ cd TravelGo/Travel
[ec2-user@ip-172-31-90-127 Travel]$ source venv/bin/activate
-bash: venv/bin/activate: No such file or directory
[ec2-user@ip-172-31-90-127 Travel]$ sudo yum update -y
sudo yum install python3 -y
Last metadata expiration check: 0:21:27 ago on Mon Jun 30 11:29:21 2025.
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:21:28 ago on Mon Jun 30 11:29:21 2025.
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-90-127 Travel]$ python3 -m venv venv
[ec2-user@ip-172-31-90-127 Travel]$ source venv/bin/activate
(venv) [ec2-user@ip-172-31-90-127 Travel]$ pip install flask boto3 pymongo
Collecting flask
  Using cached flask-3.1.1-py3-none-any.whl (103 kB)
Collecting boto3
  Using cached boto3-1.38.46-py3-none-any.whl (139 kB)
Collecting pymongo
  Downloading pymongo-4.13.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (938 kB)
    |████████████████████████████████| 938 kB 15.1 MB/s
Collecting markupsafe>=2.1.1
  Using cached MarkupSafe-3.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting jinja2>=3.1.2
(venv) [ec2-user@ip-172-31-90-127 Travel]$ python app.py
* Serving Flask app 'app'
* Debug mode: on
Address already in use
Port 5000 is in use by another program. Either identify and stop that program, or start the server with a different port.
(venv) [ec2-user@ip-172-31-90-127 Travel]$ sudo lsof -i :5000
COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
python3  27007 ec2-user    6u  IPv4  44850      0t0  TCP *:commplex-main (LISTEN)
python3  27008 ec2-user    6u  IPv4  44850      0t0  TCP *:commplex-main (LISTEN)
python3  27008 ec2-user    7u  IPv4  44850      0t0  TCP *:commplex-main (LISTEN)
(venv) [ec2-user@ip-172-31-90-127 Travel]$ sudo kill -9 27007 27008
(venv) [ec2-user@ip-172-31-90-127 Travel]$ python 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://172.31.90.127:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 844-840-745
client_loop: send disconnect: Connection reset
```

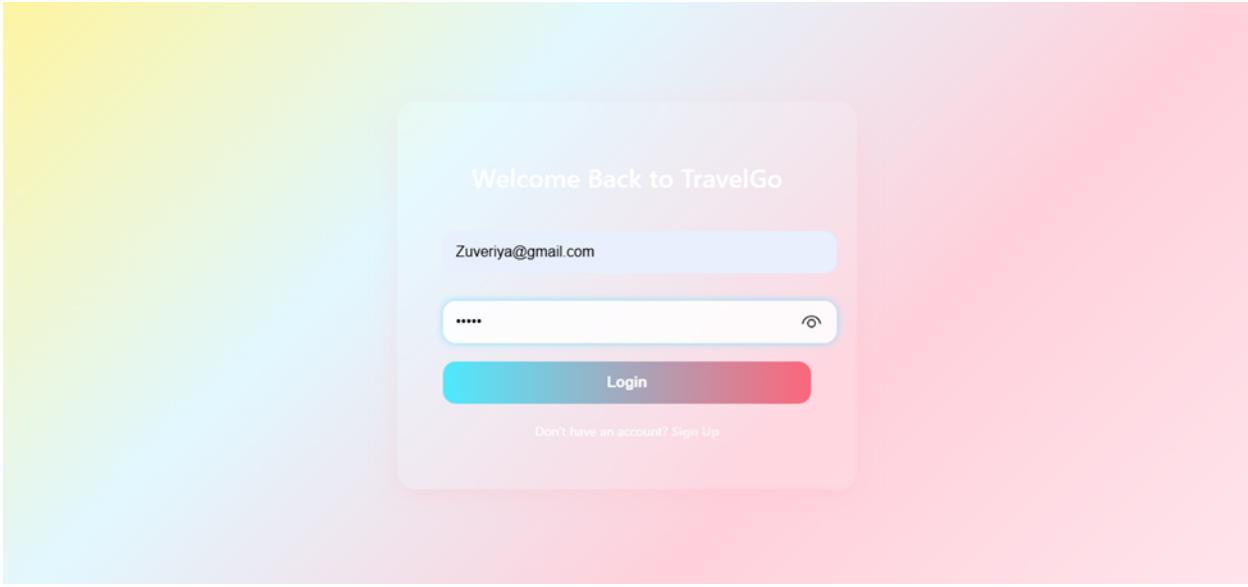
## Checking Testing and Deployment Home Page:

The **Home Page** of TravelGo serves as the entry point for users, providing a simple and intuitive interface to explore travel booking.

options. It encloses the platform's core features—allowing users to book buses, trains, flights, and hotels—all from one place. The page includes piloting to log in or sign up, along with quick links to useable booking services. Designed for ease of use, the home page highlights TravelGo's goal of making travel planning fast.



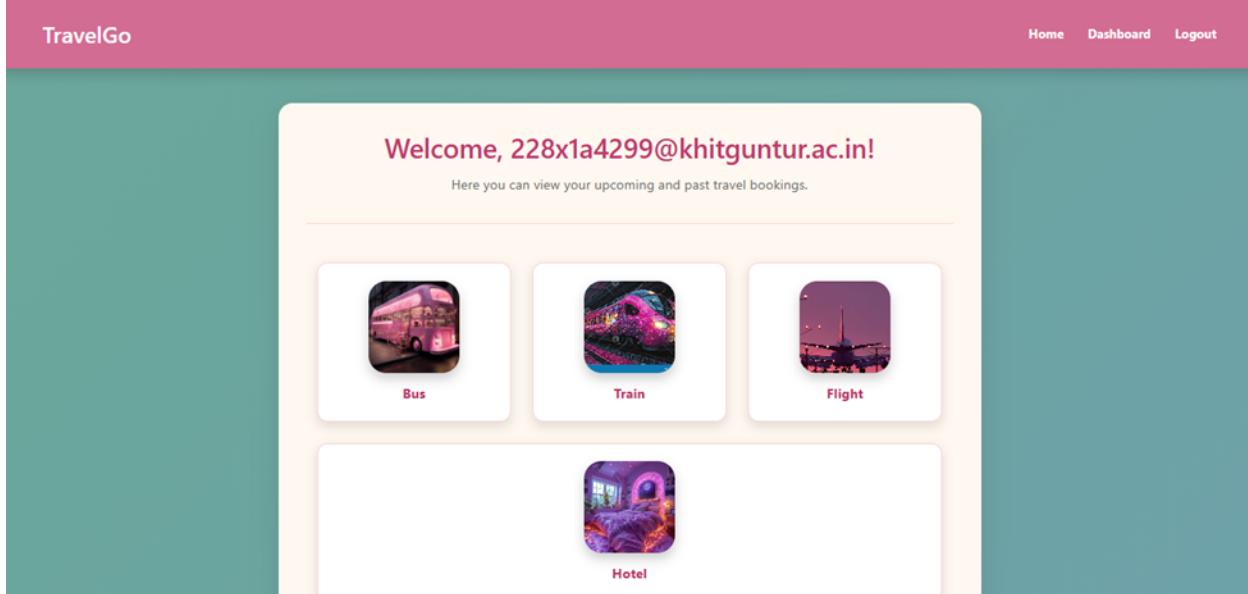
Login Page:



Dashboard:

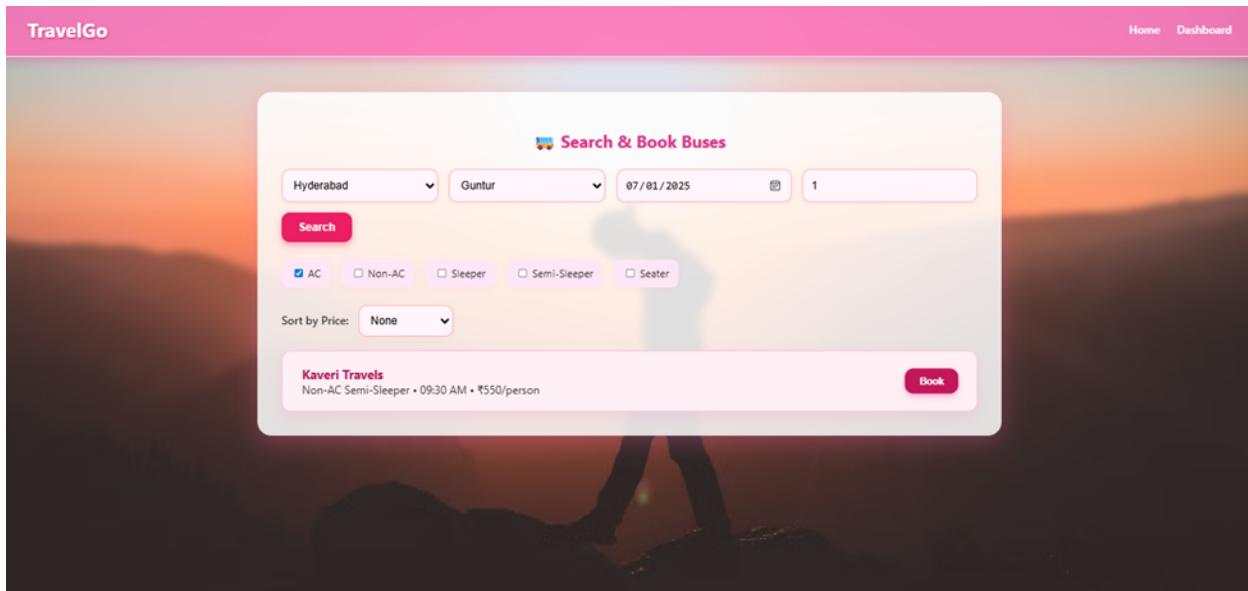
The **Dashboard Page** provides users with a personal interface to manage their travel action. After login in, users can choose to book buses, trains, flights, or hotels from the dashboard. It displays recent bookings, upcoming

trips, and account details in a clean, organized layout. The dashboard serves as a central hub, allowing users to track bookings, novice cancel, and receive updates—all in one convenient location.



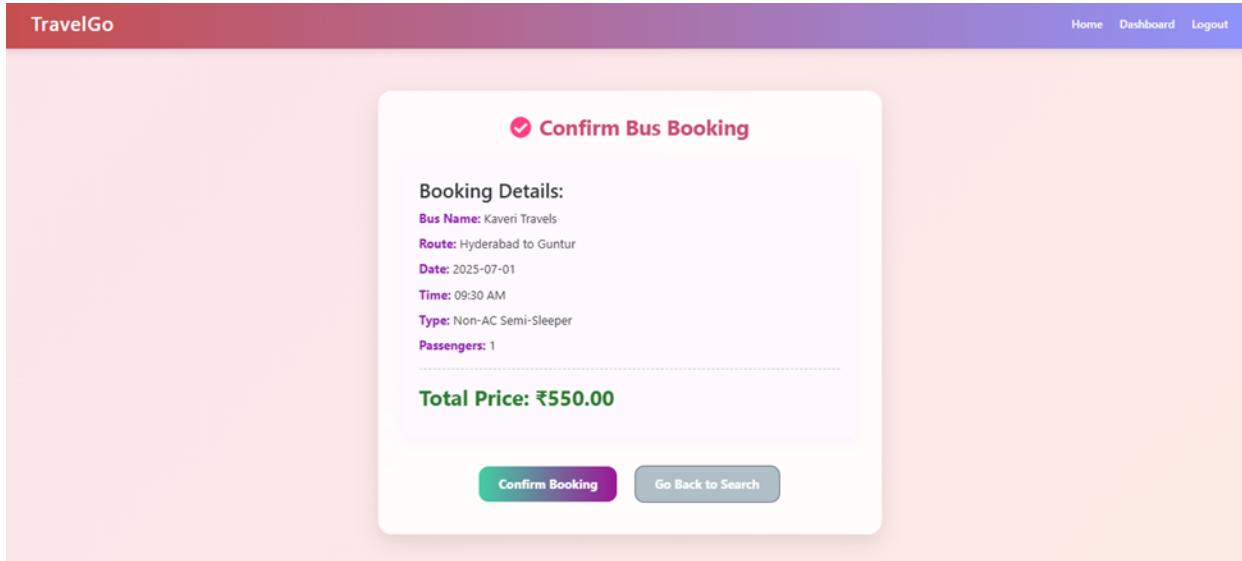
## Bookings:

- Bus



The **Bus Booking Page** allows users to search and book bus tickets by entering details such as the source, goal, journey date, and number of passengers. After users submit the form, the system displays available bus options, and users can select their preferred

service. After selection, users proceed to confirm seat numbers and fare details. The page ensures a smooth and intuitive booking flow, helping users reserve bus tickets and efficient within the TravelGo platform.



## YOUR BOOKINGS

### Non-AC Semi-Sleeper Booking: Kaveri Travels

Route: Hyderabad to Guntur

₹550.00

Travel Date: 2025-07-01

Time: 09:30 AM

Passengers: 1

Cancel Booking

Booked On: 2025-07-01

- Trains

## Search & Book Trains

Hyderabad ▾ Delhi ▾ 07/01/2025 ⏳ 5 ⏴ Search

**Search & Book Trains**

Hyderabad ▾ | Delhi ▾ | 07/01/2025 | 5 | Search

**Duronto Express (12285)** From: Hyderabad To: Delhi Departure: 07:00 AM Arrival: 05:00 AM Date of Travel: 2025-07-01 Price per person: ₹1800 Total for 5 person(s): ₹9000

[Book Now](#)

**Rajdhani Express (12433)** From: Hyderabad To: Delhi Departure: 03:00 PM Arrival: 09:00 AM (next day) Date of Travel: 2025-07-01 Price per person: ₹2200

Total for 5 person(s): ₹11000

[Book Now](#)

**TravelGo**

[Home](#) [Dashboard](#) [Logout](#)

**✓ Confirm Train Booking**

**Booking Details:**

Train Name: Duronto Express (12285)  
Route: Hyderabad to Delhi  
Date: 2025-07-01  
Departure: 07:00 AM Arrival: 05:00 AM  
Type:  
Passengers: 5  
Total Price: ₹9,000.00

[Confirm Booking](#) [Go Back to Search](#) [Select Seats](#)

The **Train Booking Page** enables users to book train tickets by entering essential travel details such as source, goal, date of journey, and number of passengers. Based on the input, the system displays useable trains with options to view timing, class, and fare. Users can select a preferred train, choose seats, and proceed to confirm the booking.

## YOUR BOOKINGS

### Train 12285 Booking: Duronto Express

Route: Hyderabad to Delhi

₹9,000.00

Travel Date: 2025-07-01

Time: 07:00 AM - 05:00 AM

[Cancel Booking](#)

Passengers: 5

Booked On: 2025-07-01

- Flight

## TravelGo

### Search & Book Flights

Hyderabad

Bengaluru

07/01/2025

3

[Search](#)

The **Flight Booking Page** enables users to search and book flights by entering the departure and arrival locations, travel date, and number of passengers. After users submit the search, the system shows a list of available flights with details like airline name, departure and arrival times,

duration, and fare. Users can select a flight, choose seats if there, and proceed to confirm the booking. This page ensures a smooth and efficient flight booking undergo within the TravelGo platform.

**Search & Book Flights**

Hyderabad ▾ Mumbai ▾ 07/01/2025 3 **Search**

**IndiGo SE-234**  
Hyderabad → Mumbai  
2025-07-01 | 08:00 - 09:30  
₹3500 x 3 = ₹10500

**Book**

- Hotel



**Hotel**

Booking details:

**hotel Booking: Oberoi**

Route: to **₹192,000.00**  
Travel Date:  
Passengers:  
Booked On: 2025-06-27

**Cancel Booking**

The **Hotel Booking Page** allows users to search for hotels by entering the location, check-in and check-out dates, number of guests, and room druthers. Based on the input, the platform displays a list of useable hotels

with details like room types, prices, and comforts. Users can select a hotel, view room options, and proceed with booking check. The page offers a seamless undergo for reserving accommodations, making it easy for users to plan and manage their stays through TravelGo.

## **□Conclusion of Travel Go Project**

The **Travel Go** project delivers a robust, cloud-powered travel booking platform that simplifies the process of booking **buses, trains, flights, and hotels** for users in real time. By using **Flask** for backend growth, **AWS EC2** for authentic deployment, **DynamoDB** for scalable data storage, and **AWS SNS** for instant notices, the system offers a seamless and responsive user undergo.

From user hallmark to booking manage and real-time alerts, Travel Go addresses the critical challenges of modern travel planning with and abstractness. The platform's ensures scalable, protection, and performance—making it a practical solution for travelers. This project demonstrates how cloud applied science can be in effect used to build smart, user-centric apps that operate smoothly under real-world conditions

[228x1a4299@khitguntur.ac.in](mailto:228x1a4299@khitguntur.ac.in)

7/2/2025