Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cloud Architecture Design

Project Report

VAAHAN

AWS-Based Vehicle Rental Platform

Submitted By: Sushen Grover
Reg No: 23BCE1728
Course Code: BCSE355L
Slot: F2 + TF2
Faculty: Dr. Priyaadharshini M

1. Project Objective

The primary objective of the Vaahan Marketplace project is to develop a secure, scalable, and user-friendly platform enabling individuals to efficiently list, browse, and book vehicles. By using a comprehensive suite of AWS serverless services for robust backend operations and a modern React frontend, Vaahan aims to deliver a seamless and engaging experience for both vehicle owners and prospective renters.

2. List of Modules

- User Authentication & Profiles
- Vehicle Listing & Management
- Booking System
- Secure & Scalable Backend API
- AI Chatbot Assistant

3. Detailed Description of Modules

3.1. User Authentication & Profiles

This module is the foundation of the application's security. It uses **AWS Cognito** to manage all user-facing authentication. This includes handling secure sign-up, sign-in, and session control. It also provides the JSON Web Tokens (JWTs) used to authorize API requests, ensuring that only logged-in users can access or modify data. The frontend profile page queries Cognito to display user-specific details like email and verification status.

3.2. Vehicle Listing & Management

This is the core marketplace module. It provides the functionality for authenticated users to add their own vehicles to the platform via a simple form. This data is securely sent to the backend, assigned a unique ID, and stored in the **Amazon DynamoDB** table. The main dashboard queries this module's API to display a comprehensive, real-time marketplace of all active vehicles available for booking. Users also have the ability to remove their own listings.

3.3. Booking System

This module handles the complete transaction logic. When a user clicks "Book Now" on a vehicle, the frontend sends a request to the /api/book endpoint. The **AWS Lambda** function then performs a conditional update in **DynamoDB** to change the vehicle's status to "Booked" and records the booking user's ID. This ensures that two users cannot book the same vehicle. Upon a successful booking, an **Amazon SNS** notification is published.

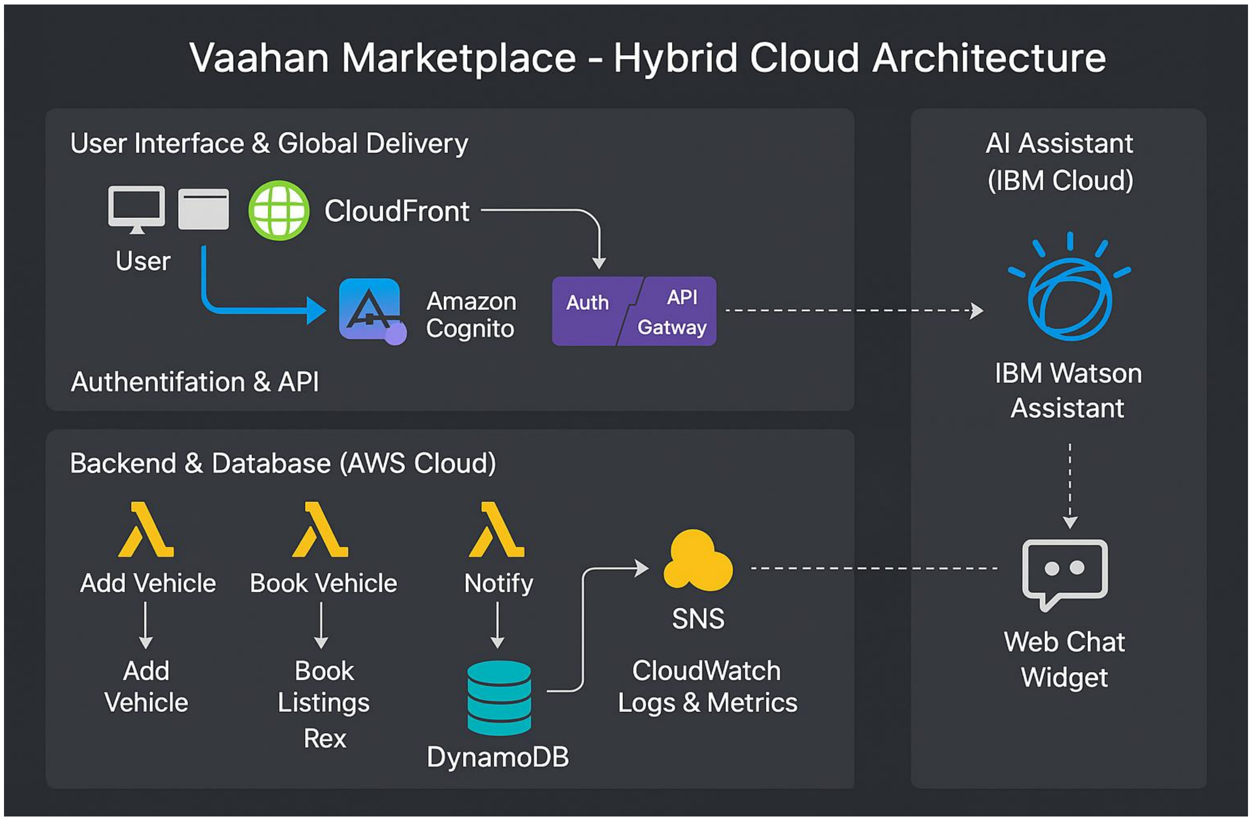
3.4. Secure & Scalable Backend API

The entire backend is powered by a serverless API. **AWS API Gateway** provides the secure, public endpoints (e.g., /api/vehicles, /api/book) that the React frontend calls. These endpoints are configured to trigger specific **AWS Lambda** functions. The API Gateway also enforces CORS (Cross-Origin Resource Sharing) policies, locking down the API so it only accepts requests from the deployed CloudFront frontend URL.

3.5. AI Chatbot Assistant

To enhance the user experience, this module integrates an external **IBM Watson Assistant**. A chat widget is embedded in the frontend application. This AI-powered chatbot provides real-time user support, helping users navigate the application or get instant answers to common questions about booking or listing vehicles, based on the intents configured in the IBM Cloud console.

4. Architecture Diagram



5. Individual Contribution

This was a solo project. As such, all aspects of the project were handled by Sushen Grover (23BCE1728), including:

- **Cloud Architecture Design:** Planning the interaction between all AWS services and the hybrid-cloud model with IBM Cloud.
- **Backend Development:** Writing the Python-based AWS Lambda functions using Flask,

setting up API Gateway endpoints, and designing the DynamoDB data schema.

- **Frontend Development:** Building the complete user interface and application logic using React.js, including components, state management, and API calls.
- **DevOps & Deployment:** Configuring AWS S3 and CloudFront for frontend hosting, setting up all IAM permissions for security, and using Amazon CloudWatch for extensive debugging.
- **Third-Party Integration:** Configuring and integrating the IBM Watson Assistant chatbot into the React frontend application.

6. Tools & Software Requirements

- **Programming Languages:**
  - **Python:** Used for the robust backend logic and AWS Lambda functions.
  - **JavaScript (React):** Used for building the interactive and responsive frontend user interface.
- **Database:**
  - **AWS DynamoDB:** Primary NoSQL database for flexible and scalable data storage.
- **IDEs & Design:**
  - **VS Code:** Used for all code development and debugging.
  - **Draw.io / Lucidchart:** Used for designing the cloud architecture.
- **Cloud Interaction:**
  - **AWS Console:** Used for direct management and configuration of all AWS services.
  - **IBM Cloud Console:** Used for configuring the Watson Assistant chatbot.
- **Frontend Libraries:**
  - **Tailwind CSS:** For modern, utility-first styling.
  - **AWS Amplify (JS Library):** For simplifying frontend authentication with Cognito.

7. AWS Services Used

This project successfully integrated 10 core cloud services to create the full application.

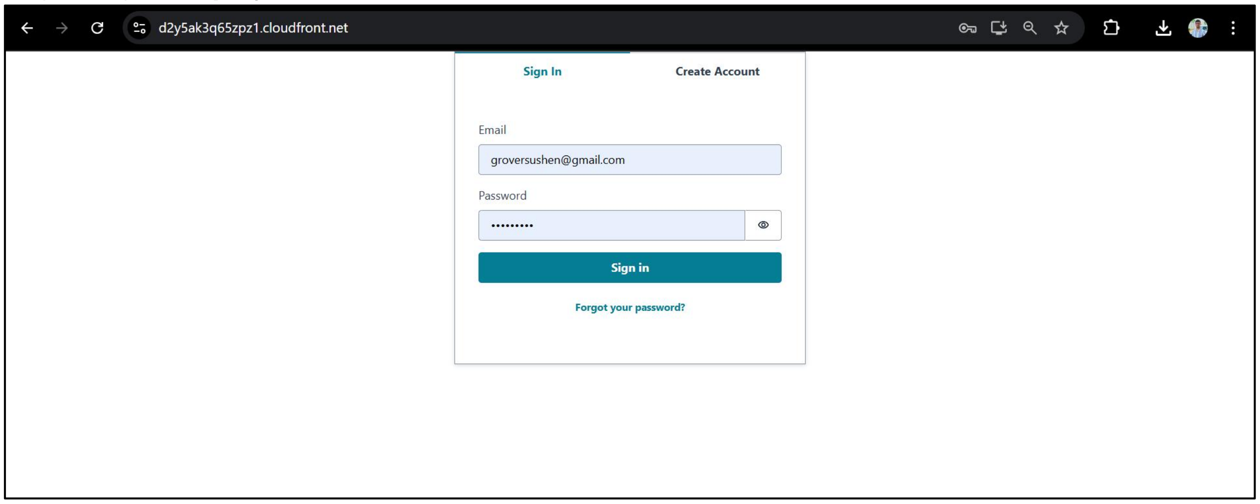
Service Used	Service Category	Role in 'Vaahan' Project
AWS Lambda + AWS API Gateway	Serverless Compute & Endpoints	Powers the backend API for all vehicle and booking logic.
AWS S3 + AWS CloudFront	Object Storage & CDN	Hosts and delivers the static React frontend app globally.
AWS DynamoDB	NoSQL Database	Stores all vehicle listings,

		user data, and bookings.
AWS Cognito	User Authentication & Identity	Manages all user sign-up, sign-in, and API security.
AWS SNS	Notifications Service	Sends automated notifications when a vehicle is booked.
AWS CloudWatch	Monitoring & Logging	Used for debugging Lambda errors and monitoring performance.
AWS IAM	Security & Permissions	Manages secure permissions between all AWS services.
IBM Watson Assistant	AI Chatbot Service (Hybrid)	Provides the intelligent, real-time chatbot for user support.

8. Screen Shots

8.1. Project Demo Screenshots

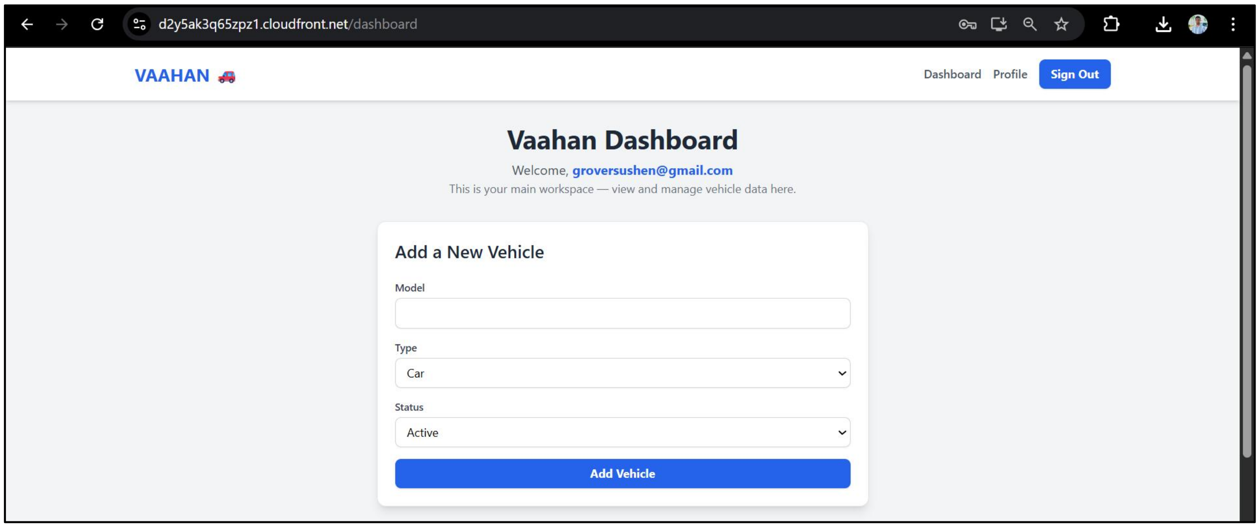
Login Page (Amplify UI)


d2y5ak3q65zpz1.cloudfront.net

Email: groversushen@gmail.com

Password: ••••••••••

Add Vehicle Form



d2y5ak3q65zpz1.cloudfront.net/dashboard

VAAHAN 🚙

Dashboard Profile Sign Out

Vaahan Dashboard

Welcome, **groversushen@gmail.com**

This is your main workspace — view and manage vehicle data here.

Add a New Vehicle

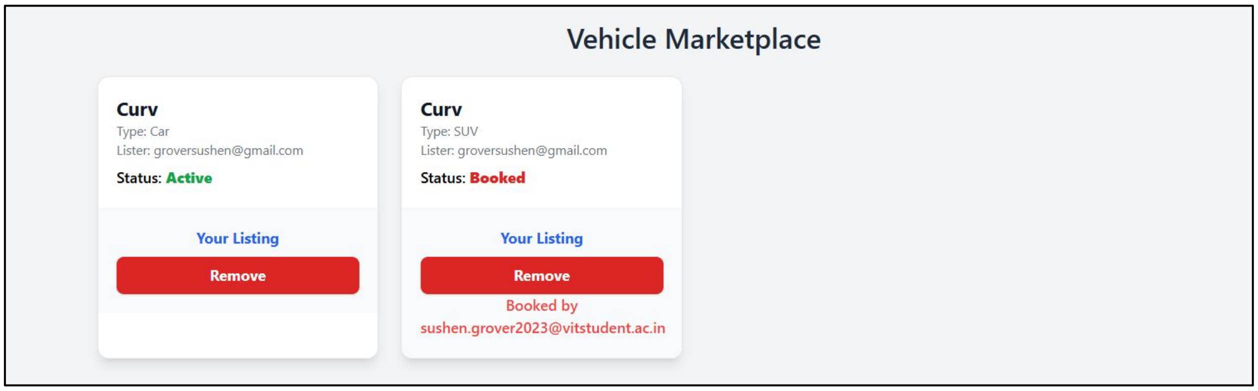
Model:

Type: Car

Status: Active

Add Vehicle

Dashboard

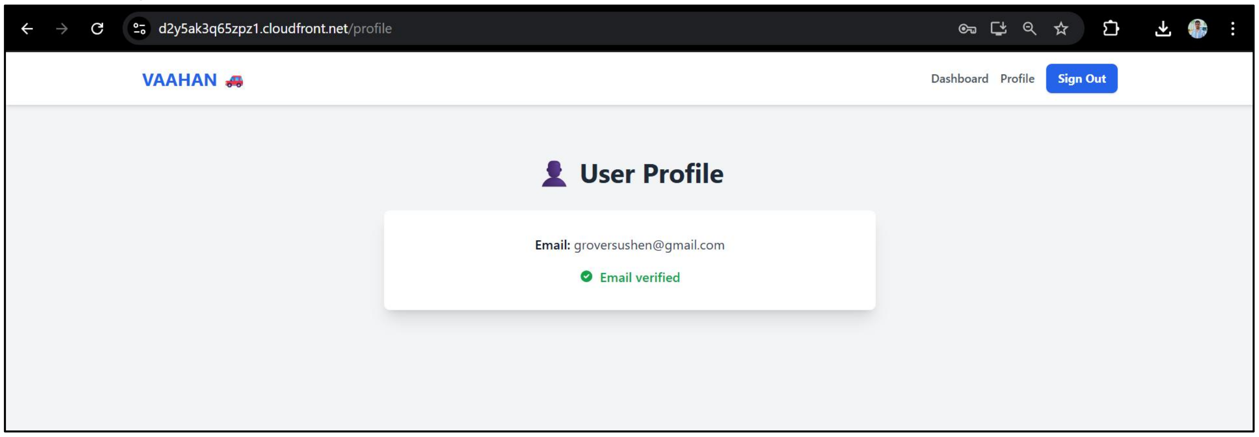


Vehicle Marketplace

Curv
Type: Car
Lister: groversushen@gmail.com
Status: **Active**
Your Listing
Remove

Curv
Type: SUV
Lister: groversushen@gmail.com
Status: **Booked**
Your Listing
Remove
Booked by sushen.grover2023@vitstudent.ac.in

Profile Page



d2y5ak3q65zpz1.cloudfront.net/profile

VAAHAN 🚙

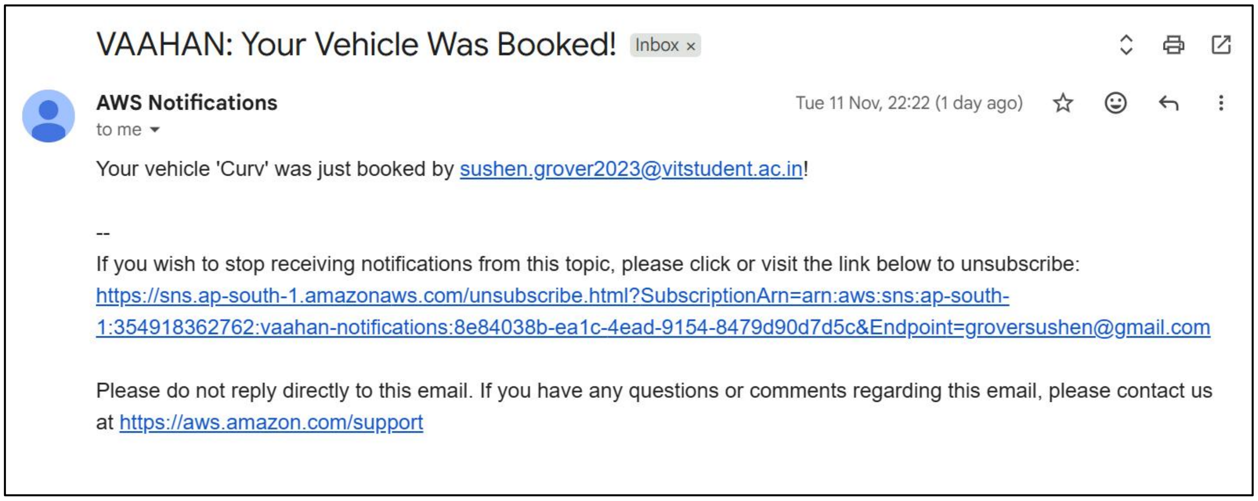
Dashboard Profile Sign Out

User Profile

Email: groversushen@gmail.com

✓ Email verified

SNS Email to owner


VAAHAN: Your Vehicle Was Booked! Inbox ×

AWS Notifications
to me

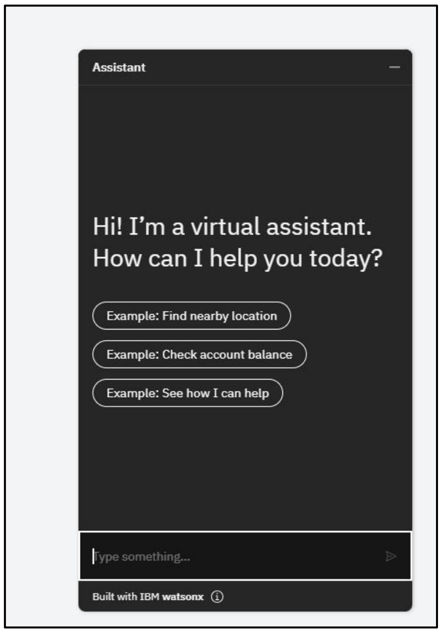
Tue 11 Nov, 22:22 (1 day ago)

Your vehicle 'Curv' was just booked by sushen.grover2023@vitstudent.ac.in!

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.ap-south-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:ap-south-1:354918362762:vaahan-notifications:8e84038b-ea1c-4ead-9154-8479d90d7d5c&Endpoint=groversushen@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

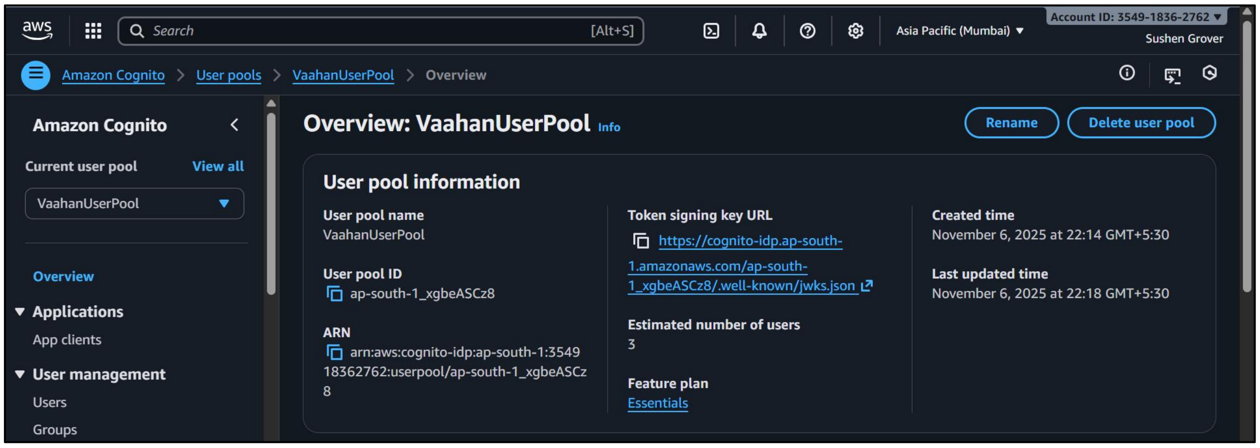
Chatbot


Hi! I'm a virtual assistant. How can I help you today?

Assistant
Example: Find nearby location
Example: Check account balance
Example: See how I can help
Type something...
Built with IBM watsonx

8.2. AWS Service Configuration Screenshots

Amazon Cognito User Pool


Amazon Cognito > User pools > VaahanUserPool > Overview

Account ID: 3549-1836-2762 · Asia Pacific (Mumbai) · Sushen Grover

Current user pool: VaahanUserPool

Overview: VaahanUserPool

User pool information

User pool name
VaahanUserPool

User pool ID
ap-south-1_xgbeASCz8

ARN
arn:aws:cognito-idp:ap-south-1:354918362762:userpool/ap-south-1_xgbeASCz8

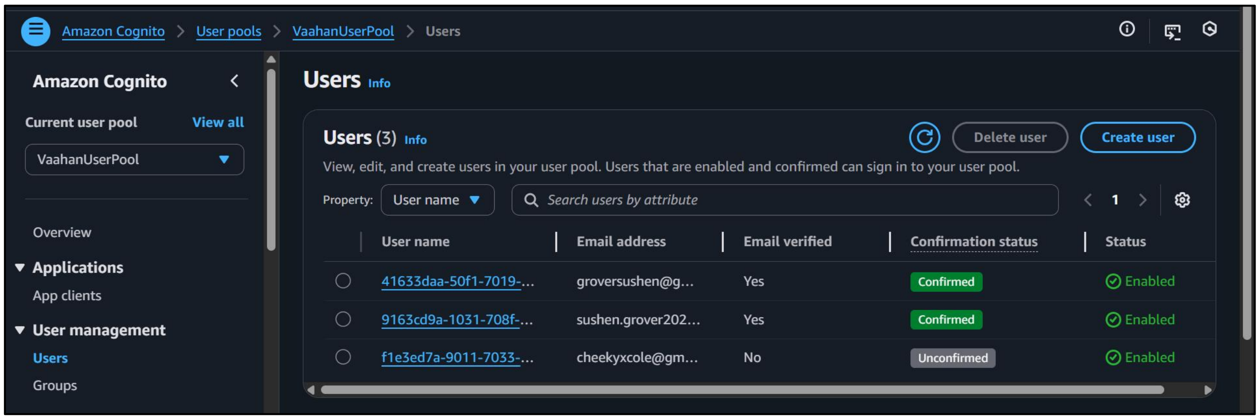
Token signing key URL
https://cognito-idp.ap-south-1.amazonaws.com/ap-south-1_xgbeASCz8/.well-known/jwks.json

Estimated number of users
3

Feature plan
Essentials

Created time
November 6, 2025 at 22:14 GMT+5:30

Last updated time
November 6, 2025 at 22:18 GMT+5:30


Amazon Cognito > User pools > VaahanUserPool > Users

Current user pool: VaahanUserPool

Users

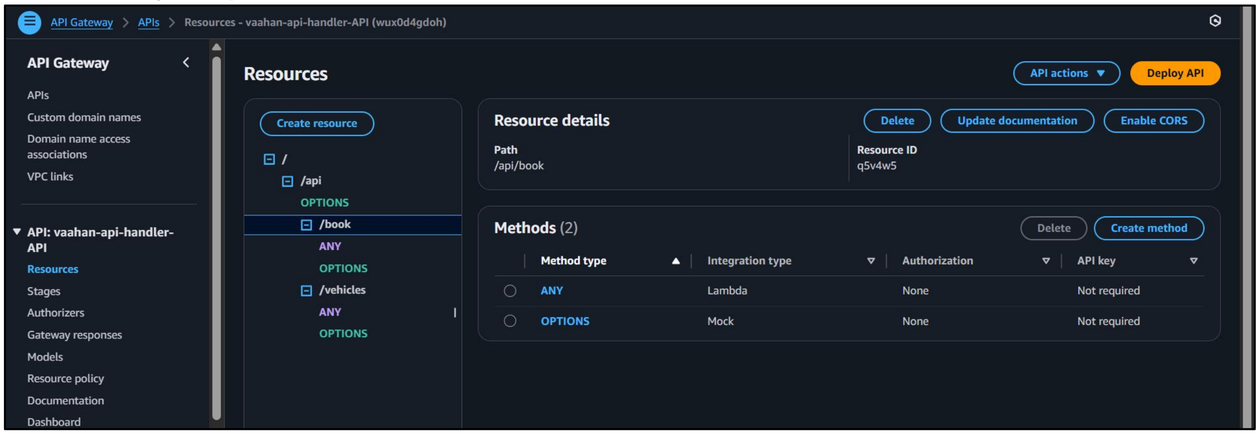
Users (3)

View, edit, and create users in your user pool. Users that are enabled and confirmed can sign in to your user pool.

Property: User name · Search users by attribute

User name	Email address	Email verified	Confirmation status	Status
41633daa-50f1-7019-...	groversushen@g...	Yes	Confirmed	Enabled
9163cd9a-1031-708f-...	sushen.grover202...	Yes	Confirmed	Enabled
f1e3ed7a-9011-7033-...	cheekyxcole@gm...	No	Unconfirmed	Enabled

API Gateway Endpoints (Resource Tree)


API Gateway > APIs > Resources - vaahan-api-handler-API (wux0d4gdoh)

Resources

```
/
  /api
    OPTIONS
    /book
      ANY
      OPTIONS
    /vehicles
      ANY
      OPTIONS
```

Resource details

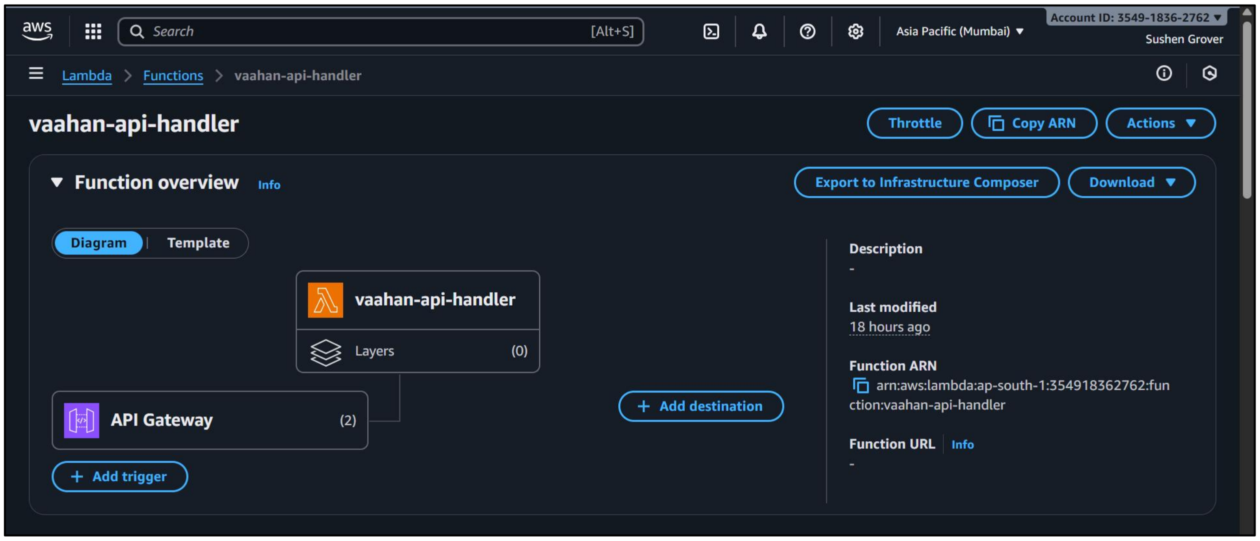
Path
/api/book

Resource ID
q5v4w5

Methods (2)

Method type	Integration type	Authorization	API key
ANY	Lambda	None	Not required
OPTIONS	Mock	None	Not required

Lambda Function Configuration

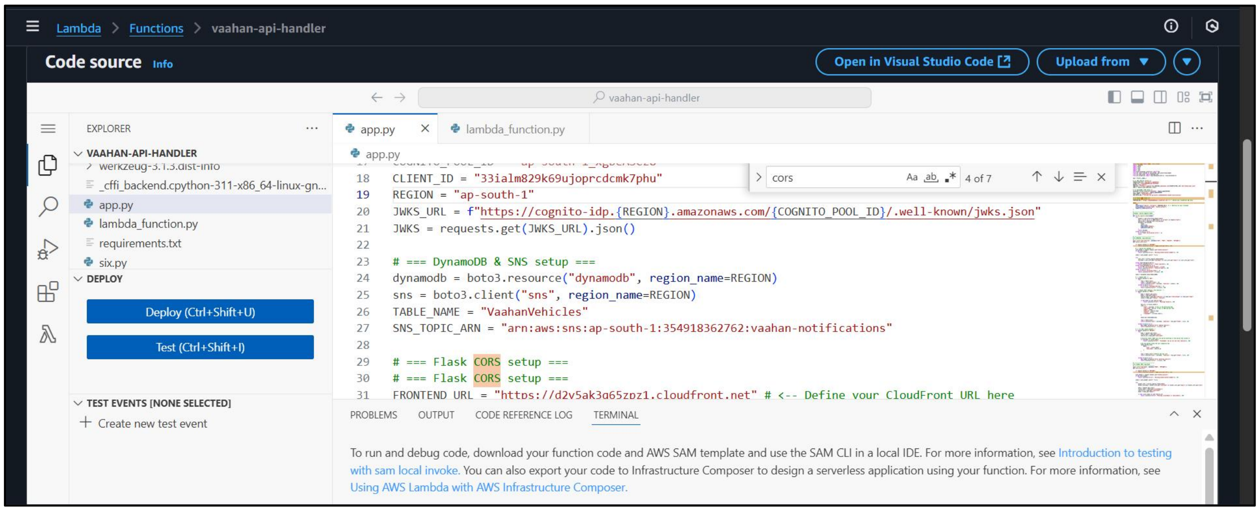


Description
-

Last modified
18 hours ago

Function ARN
arn:aws:lambda:ap-south-1:354918362762:function:vaahan-api-handler

Function URL Info
-

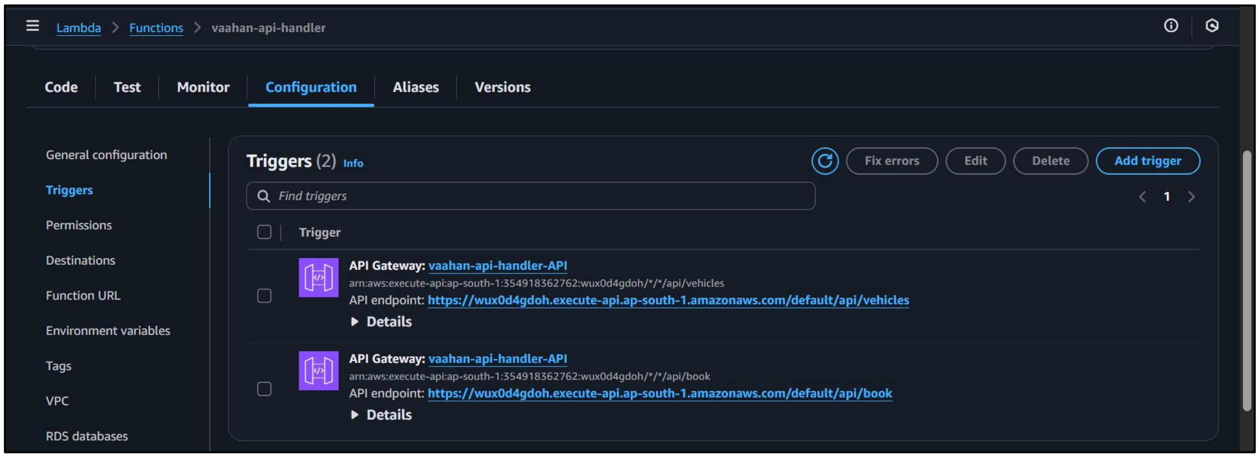


```python
CLIENT_ID = "33ialm829k69ujoprcdcmk7phu"
REGION = "ap-south-1"
JWKS_URL = f"https://cognito-idp.{REGION}.amazonaws.com/{COGNITO_POOL_ID}/.well-known/jwks.json"
JWKS = requests.get(JWKS_URL).json()

# === DynamoDB & SNS setup ===
dynamodb = boto3.resource("dynamodb", region_name=REGION)
sns = boto3.client("sns", region_name=REGION)
TABLE_NAME = "VaahanVehicles"
SNS_TOPIC_ARN = "arn:aws:sns:ap-south-1:354918362762:vaahan-notifications"

# === Flask CORS setup ===
# === Flask CORS setup ===
FRONTEND_URL = "https://d2v5ak3q65zpz1.cloudfront.net" # <-- Define your CloudFront URL here
```

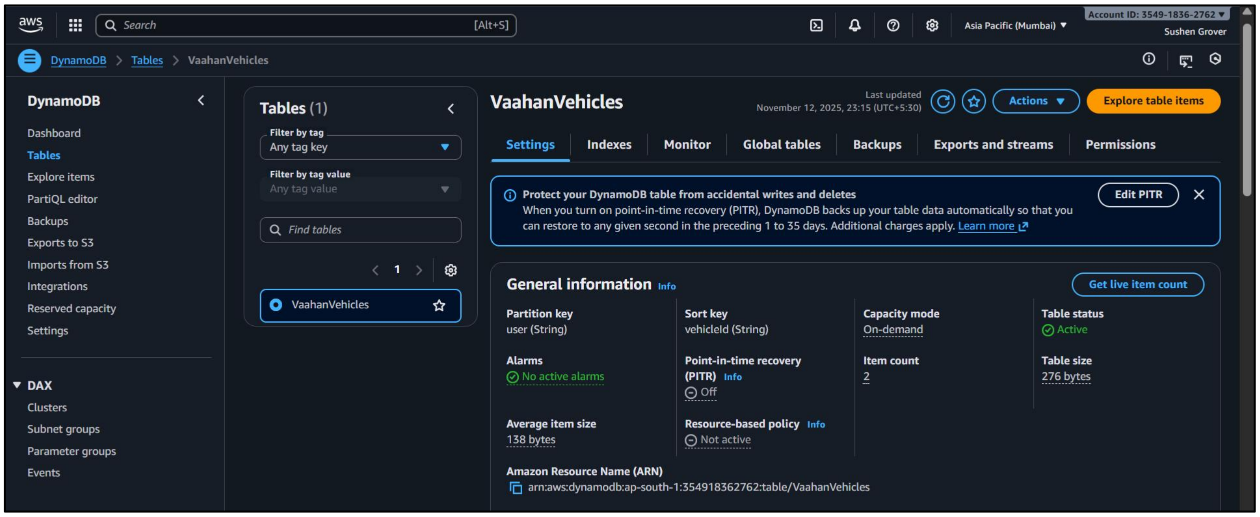
To run and debug code, download your function code and AWS SAM template and use the SAM CLI in a local IDE. For more information, see Introduction to testing with sam local invoke. You can also export your code to Infrastructure Composer to design a serverless application using your function. For more information, see Using AWS Lambda with AWS Infrastructure Composer.



API Gateway: vaahan-api-handler-API
arn:aws:execute-api:ap-south-1:354918362762:wux0d4gdoh/*/*/api/vehicles
API endpoint: https://wux0d4gdoh.execute-api.ap-south-1.amazonaws.com/default/api/vehicles

API Gateway: vaahan-api-handler-API
arn:aws:execute-api:ap-south-1:354918362762:wux0d4gdoh/*/*/api/book
API endpoint: https://wux0d4gdoh.execute-api.ap-south-1.amazonaws.com/default/api/book

DynamoDB Table



DynamoDB > Tables > VaahanVehicles

VaahanVehicles

Last updated November 12, 2025, 23:15 (UTC+5:30)

Settings | Indexes | Monitor | Global tables | Backups | Exports and streams | Permissions

Protect your DynamoDB table from accidental writes and deletes
When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 1 to 35 days. Additional charges apply. Learn more

General information

Partition key user (String)	**Sort key** vehicleId (String)	**Capacity mode** On-demand	**Table status** Active
Alarms No active alarms	**Point-in-time recovery (PITR)** Off	**Item count** 2	**Table size** 276 bytes
Average item size 138 bytes	**Resource-based policy** Not active		

Amazon Resource Name (ARN)
arn:aws:dynamodb:ap-south-1:354918362762:table/VaahanVehicles

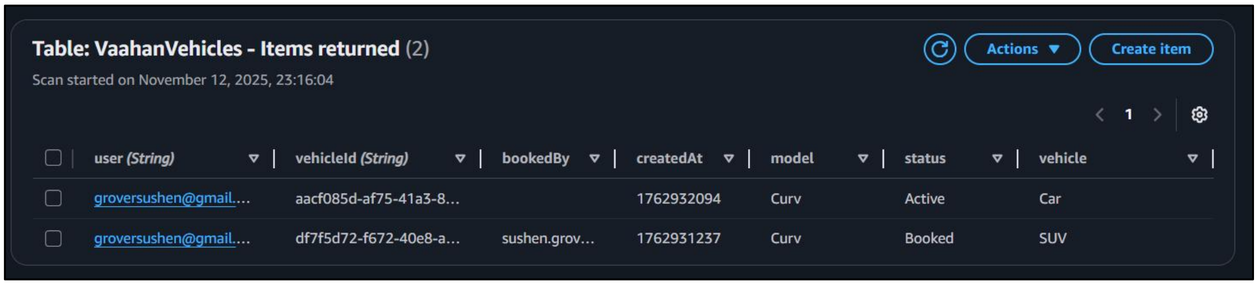
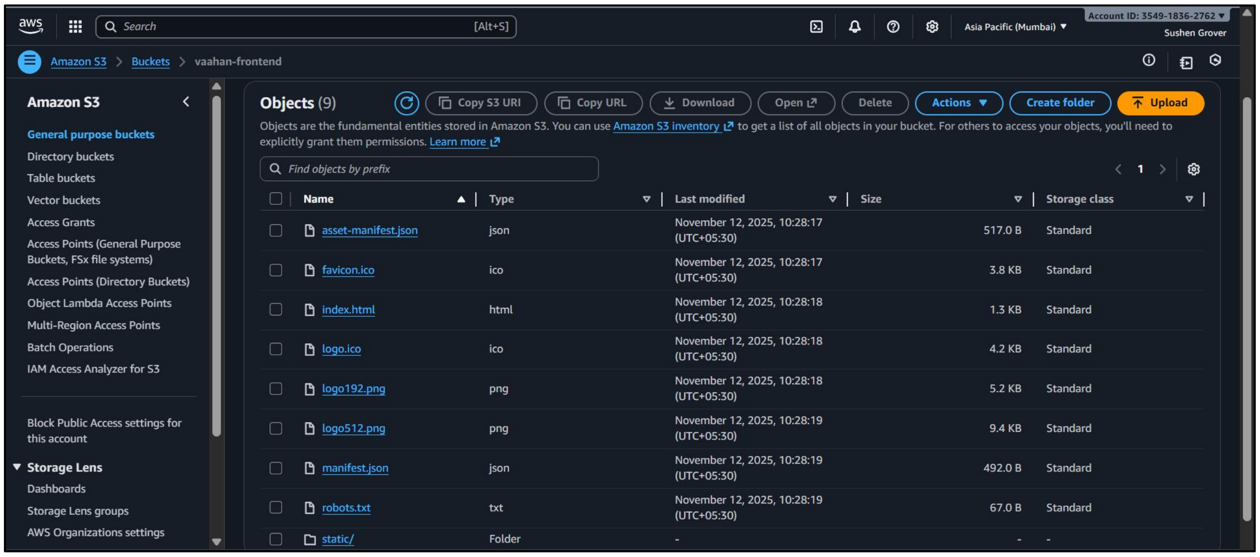


Table: VaahanVehicles - Items returned (2)

Scan started on November 12, 2025, 23:16:04

user (String)	vehicleId (String)	bookedBy	createdAt	model	status	vehicle
groversushen@gmail....	aacf085d-af75-41a3-8...		1762932094	Curv	Active	Car
groversushen@gmail....	df7f5d72-f672-40e8-a...	sushen.grov...	1762931237	Curv	Booked	SUV

S3 Bucket Configuration



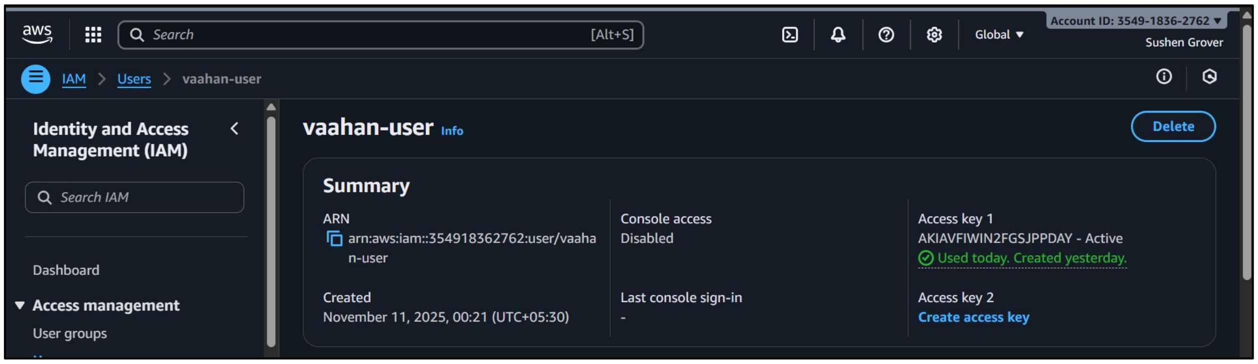
Amazon S3 > Buckets > vaahan-frontend

Objects (9)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
asset-manifest.json	json	November 12, 2025, 10:28:17 (UTC+05:30)	517.0 B	Standard
favicon.ico	ico	November 12, 2025, 10:28:17 (UTC+05:30)	3.8 KB	Standard
index.html	html	November 12, 2025, 10:28:18 (UTC+05:30)	1.3 KB	Standard
logo.ico	ico	November 12, 2025, 10:28:18 (UTC+05:30)	4.2 KB	Standard
logo192.png	png	November 12, 2025, 10:28:18 (UTC+05:30)	5.2 KB	Standard
logo512.png	png	November 12, 2025, 10:28:19 (UTC+05:30)	9.4 KB	Standard
manifest.json	json	November 12, 2025, 10:28:19 (UTC+05:30)	492.0 B	Standard
robots.txt	txt	November 12, 2025, 10:28:19 (UTC+05:30)	67.0 B	Standard
static/	Folder	-	-	-

IAM User Configuration



vaahan-user Info

Summary

ARN
arn:aws:iam::354918362762:user/vaahan-user

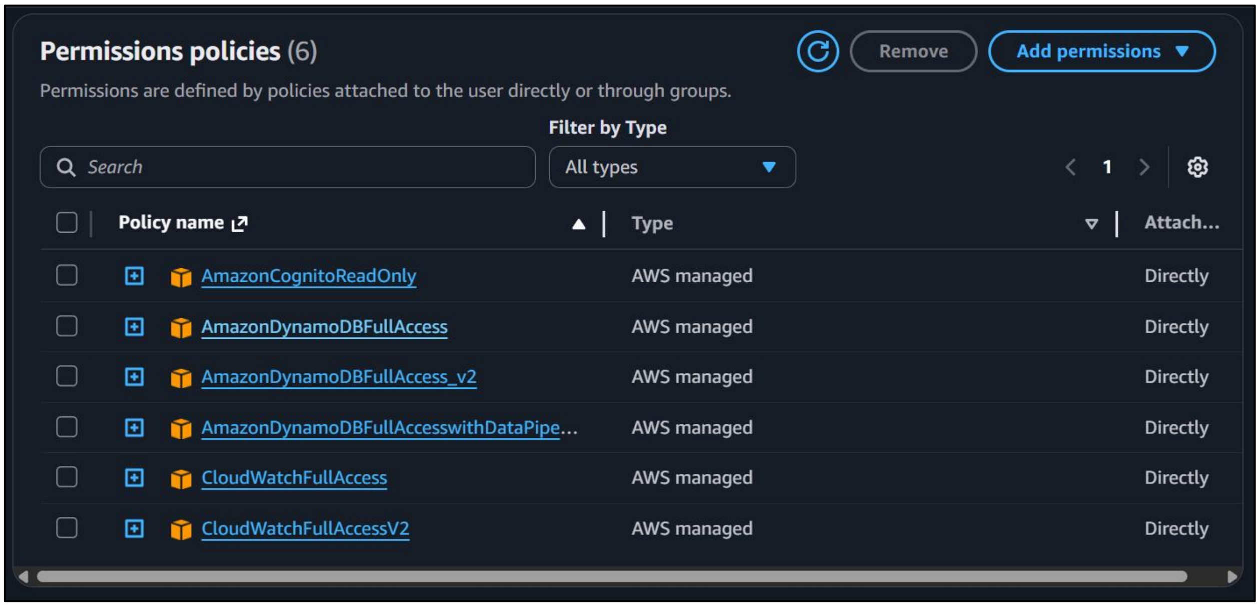
Created
November 11, 2025, 00:21 (UTC+05:30)

Console access
Disabled

Last console sign-in
-

Access key 1
AKIAVFIWIN2FGSJPPDAY - Active
Used today. Created yesterday.

Access key 2
Create access key



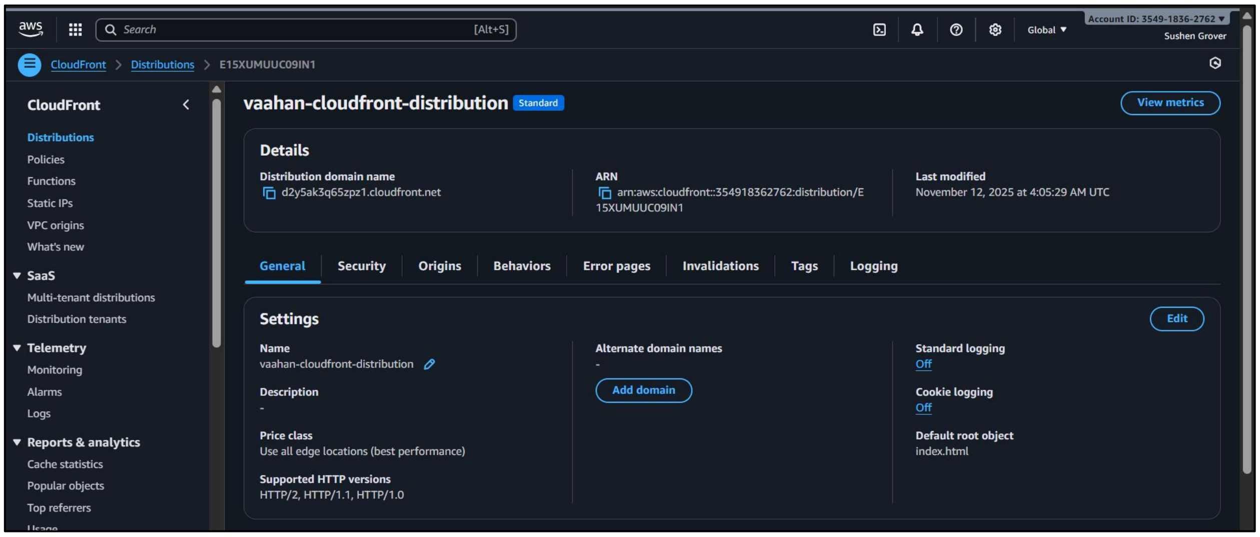
Permissions policies (6)

Permissions are defined by policies attached to the user directly or through groups.

Filter by Type: All types

Policy name	Type	Attach...
AmazonCognitoReadOnly	AWS managed	Directly
AmazonDynamoDBFullAccess	AWS managed	Directly
AmazonDynamoDBFullAccess_v2	AWS managed	Directly
AmazonDynamoDBFullAccesswithDataPipe...	AWS managed	Directly
CloudWatchFullAccess	AWS managed	Directly
CloudWatchFullAccessV2	AWS managed	Directly

CloudFront Distribution



vaahan-cloudfront-distribution Standard

Details

Distribution domain name
d2y5ak3q65zpz1.cloudfront.net

ARN
arn:aws:cloudfront::354918362762:distribution/E15XUMUUC09IN1

Last modified
November 12, 2025 at 4:05:29 AM UTC

Settings

Name
vaahan-cloudfront-distribution

Description
-

Price class
Use all edge locations (best performance)

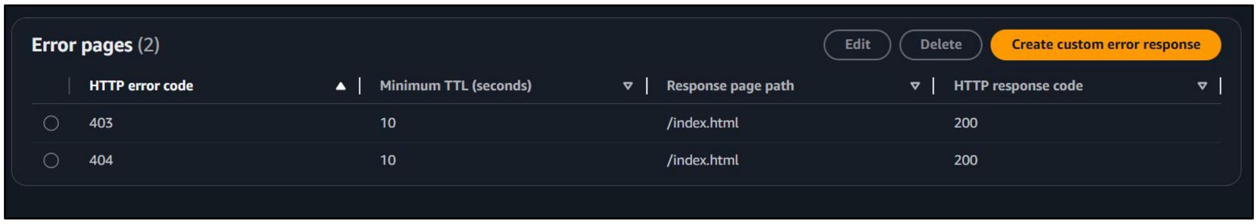
Supported HTTP versions
HTTP/2, HTTP/1.1, HTTP/1.0

Alternate domain names
-

Standard logging
Off

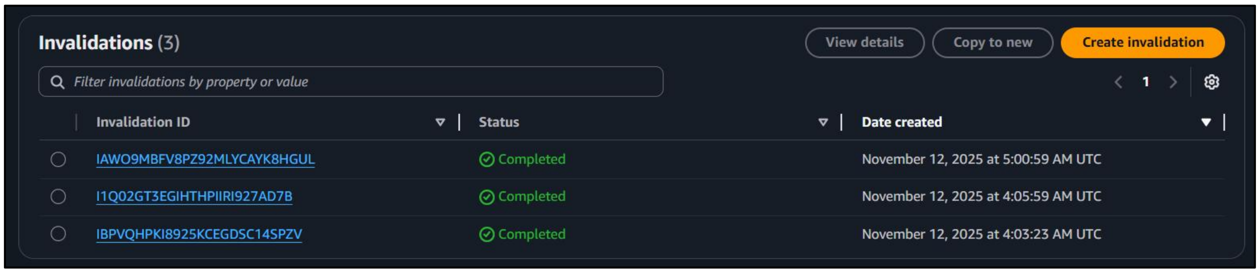
Cookie logging
Off

Default root object
index.html



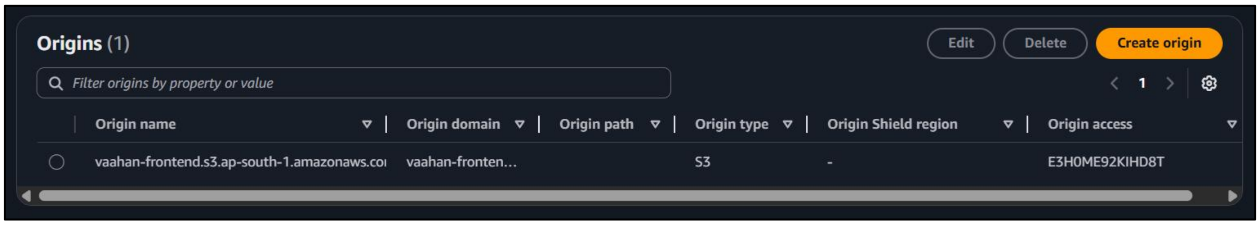
Error pages (2)

HTTP error code	Minimum TTL (seconds)	Response page path	HTTP response code
403	10	/index.html	200
404	10	/index.html	200



Invalidations (3)

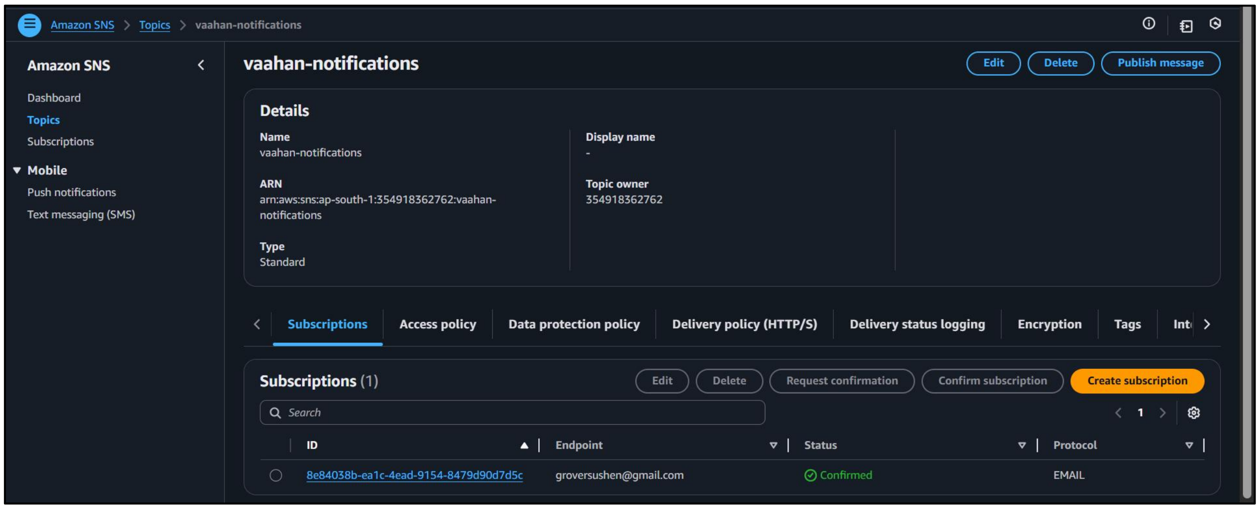
Invalidation ID	Status	Date created
IAWO9MBFV8PZ92MLYCAYK8HGUL	Completed	November 12, 2025 at 5:00:59 AM UTC
I1Q02GT3EGIHTHPIIRI927AD7B	Completed	November 12, 2025 at 4:05:59 AM UTC
IBPVQHPKI8925KCEGDSC14SPZV	Completed	November 12, 2025 at 4:03:23 AM UTC



Origins (1)

Origin name	Origin domain	Origin path	Origin type	Origin Shield region	Origin access
vaahan-frontend.s3.ap-south-1.amazonaws.coı	vaahan-fronten...		S3	-	E3HOME92KIHD8T

SNS Configuration



vaahan-notifications

Details

Name: vaahan-notifications

Display name: -

ARN: arn:aws:sns:ap-south-1:354918362762:vaahan-notifications

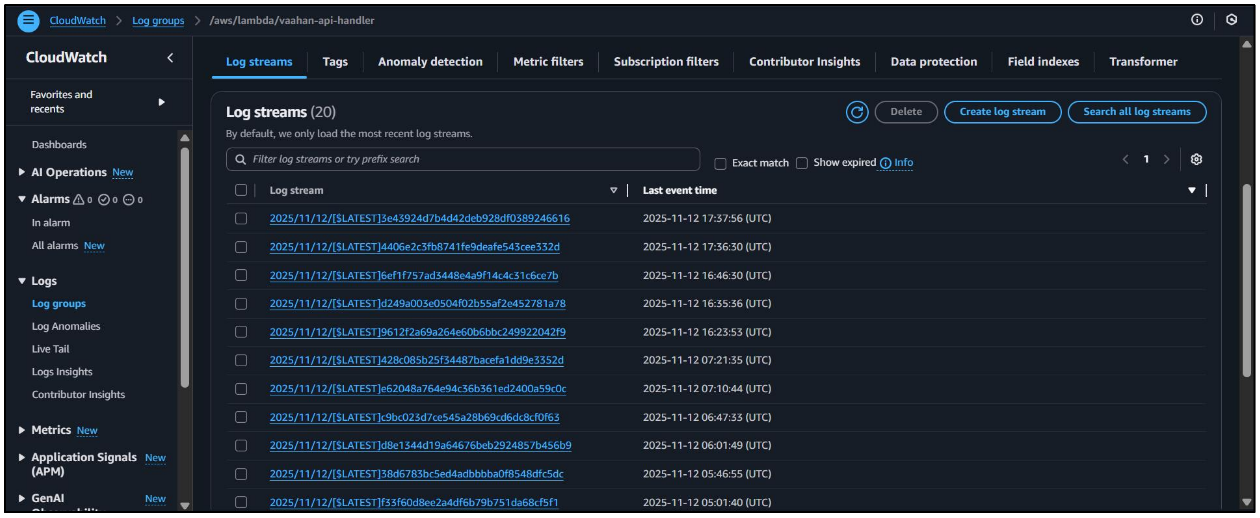
Topic owner: 354918362762

Type: Standard

Subscriptions (1)

ID	Endpoint	Status	Protocol
8e84038b-ea1c-4ead-9154-8479d90d7d5c	groversushen@gmail.com	Confirmed	EMAIL

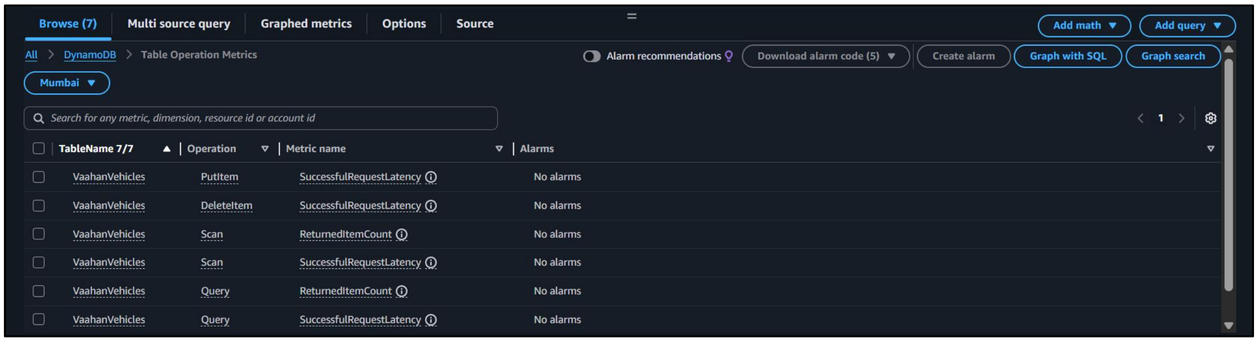
CloudWatch Logs (Lambda + DynamoDB)



Log streams (20)

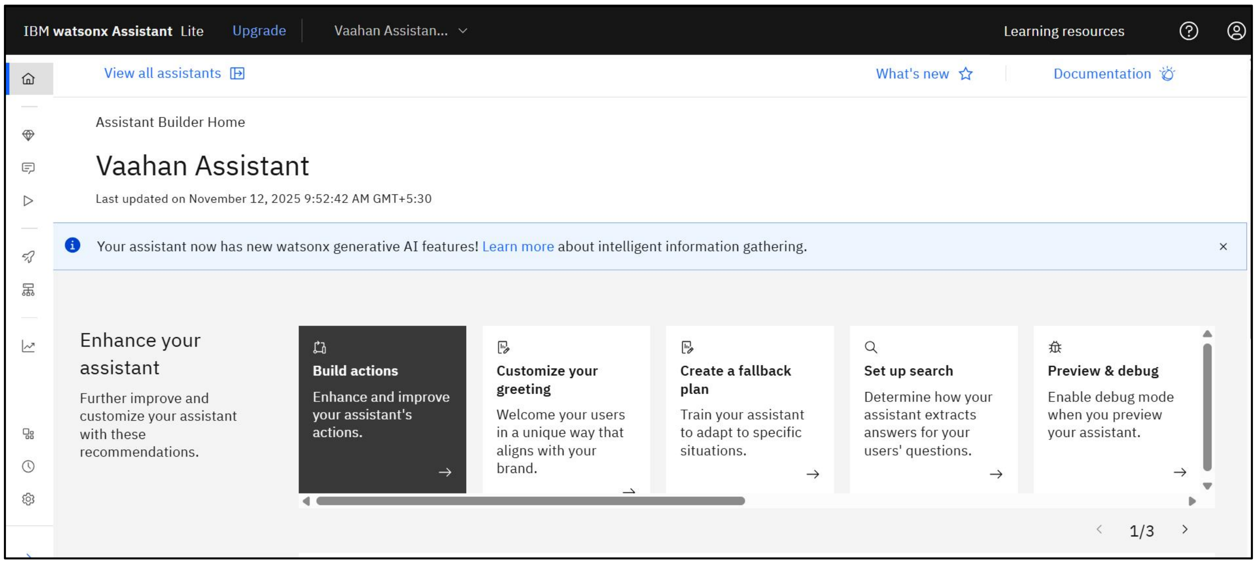
By default, we only load the most recent log streams.

Log stream	Last event time
2025/11/12/[$LATEST]3e43924d7b4d42deb928df0389246616	2025-11-12 17:37:56 (UTC)
2025/11/12/[$LATEST]4406e2c3fb8741fe9deafe543cee332d	2025-11-12 17:36:30 (UTC)
2025/11/12/[$LATEST]6ef1f757ad3448e4a9f14c4c31c6ce7b	2025-11-12 16:46:30 (UTC)
2025/11/12/[$LATEST]d249a003e0504f02b55af2e452781a78	2025-11-12 16:35:36 (UTC)
2025/11/12/[$LATEST]9612f2a69a264e60b6bbc249922042f9	2025-11-12 16:23:53 (UTC)
2025/11/12/[$LATEST]428c085b25f34487bacefa1dd9e3352d	2025-11-12 07:21:35 (UTC)
2025/11/12/[$LATEST]e62048a764e94c36b361ed2400a59c0c	2025-11-12 07:10:44 (UTC)
2025/11/12/[$LATEST]c9bc023d7ce545a28b69cd6dc8cf0f63	2025-11-12 06:47:33 (UTC)
2025/11/12/[$LATEST]d8e1344d19a64676beb2924857b456b9	2025-11-12 06:01:49 (UTC)
2025/11/12/[$LATEST]38d6783bc5ed4adbbbba0f8548dfc5dc	2025-11-12 05:46:55 (UTC)
2025/11/12/[$LATEST]f33f60d8ee2a4df6b79b751da68cf5f1	2025-11-12 05:01:40 (UTC)



TableName 7/7	Operation	Metric name	Alarms
VaahanVehicles	PutItem	SuccessfulRequestLatency	No alarms
VaahanVehicles	DeleteItem	SuccessfulRequestLatency	No alarms
VaahanVehicles	Scan	ReturnedItemCount	No alarms
VaahanVehicles	Scan	SuccessfulRequestLatency	No alarms
VaahanVehicles	Query	ReturnedItemCount	No alarms
VaahanVehicles	Query	SuccessfulRequestLatency	No alarms

IBM Watson Assistant



Assistant Builder Home

Vaahan Assistant

Last updated on November 12, 2025 9:52:42 AM GMT+5:30

Your assistant now has new watsonx generative AI features! Learn more about intelligent information gathering.

Enhance your assistant

Further improve and customize your assistant with these recommendations.

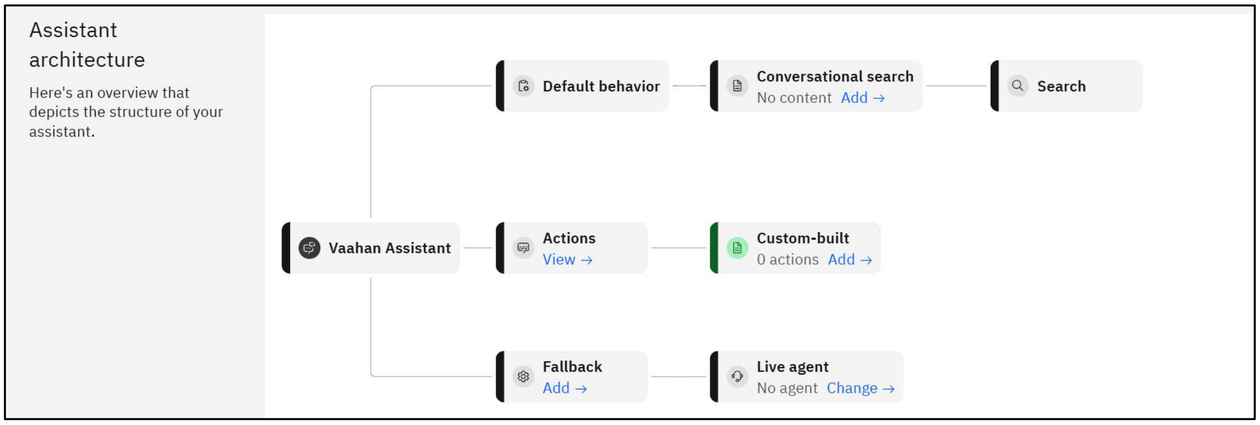
Build actions Enhance and improve your assistant's actions.

Customize your greeting Welcome your users in a unique way that aligns with your brand.

Create a fallback plan Train your assistant to adapt to specific situations.

Set up search Determine how your assistant extracts answers for your users' questions.

Preview & debug Enable debug mode when you preview your assistant.



9. Source Code

9.1. GitHub Repository Link

https://github.com/SushenGrover/VAAHAN-aws-based-vehicle-rental-platform

9.2. CloudFront Deployment Link

https://d2y5ak3q65zpz1.cloudfront.net/

9.3. Backend Code (Python / Lambda)

app.py (Main Flask Application for Lambda)

```
# app.py
from flask import Flask, request, jsonify
from flask_cors import CORS
import requests
import boto3
import uuid
import time
import json
from boto3.dynamodb.conditions import Key
from jwt import decode as jwt_decode, get_unverified_header
from jwt.algorithms import RSAAlgorithm
from jwt.exceptions import ExpiredSignatureError, InvalidTokenError

app = Flask(__name__)

# === Flask CORS setup ===
# Restrict access to only the deployed CloudFront URL
FRONTEND_URL = "[https://d2y5ak3q65zpc1.cloudfront.net](https://d2y5ak3q65zpc1.cloudfront.net)"
```

```
CORS(
    app,
    resources={r"/api/*": {"origins": FRONTEND_URL}},
    allow_headers=["Content-Type", "Authorization"],
    supports_credentials=True
)

# === AWS Cognito config ===
COGNITO_POOL_ID = "ap-south-1_xgbeASCz8"
CLIENT_ID = "33ialm829k69ujoprcdcmk7phu"
REGION = "ap-south-1"
JWKS_URL                                        =                                        f"https://cognito-
idp.{REGION}[.amazonaws.com/](https://.amazonaws.com/){COGNITO_POOL_ID}/.well-known/jwks.json"
JWKS = requests.get(JWKS_URL).json()

# === DynamoDB & SNS setup ===
dynamodb = boto3.resource("dynamodb", region_name=REGION)
sns = boto3.client("sns", region_name=REGION)
TABLE_NAME = "VaahanVehicles"
SNS_TOPIC_ARN = "arn:aws:sns:ap-south-1:354918362762:vaahan-notifications"


# -------------------------------
# Helper: Verify Cognito token
# -------------------------------
def verify_cognito_token(token):
    try:
        headers = get_unverified_header(token)
        # Find the key in the JWKS that matches the 'kid' from the token header
        key = next(k for k in JWKS["keys"] if k["kid"] == headers["kid"])
        # Construct the public key
        public_key = RSAAlgorithm.from_jwk(key)
        # Decode and verify the token
        decoded = jwt_decode(
            token,
            public_key,
            algorithms=["RS256"],
            audience=CLIENT_ID, # Check that the token's audience matches our client ID
        )
        return decoded
    except Exception as e:
        print("Token verification error:", e)
        raise

# -------------------------------
# GET, POST, DELETE /api/vehicles
# -------------------------------
```

```
@app.route("/api/vehicles", methods=["GET", "POST", "DELETE"])
def handle_vehicles():

    # --- Secure all methods ---
    auth_header = request.headers.get("Authorization")
    if not auth_header:
        return jsonify({"error": "Missing Authorization header"}), 401

    token = auth_header.split(" ")[-1]

    try:
        user_info = verify_cognito_token(token)
        # Get username from Cognito token (sub is a unique ID)
        username = user_info.get("username") or user_info.get("email") or user_info.get("sub")

    except ExpiredSignatureError:
        return jsonify({"error": "Token expired"}), 401
    except InvalidTokenError as e:
        print("JWT verification failed:", str(e))
        return jsonify({"error": "Invalid token"}), 401
    except Exception as e:
        print("Error verifying token:", e)
        return jsonify({"error": str(e)}), 500

    table = dynamodb.Table(TABLE_NAME)

    # --- Handle GET ---
    if request.method == "GET":
        try:
            # Scan the entire table to get all vehicles
            resp = table.scan()
            items = resp.get("Items", [])
            # Return all vehicles, plus the username of the person asking
            return jsonify({"user": username, "vehicles": items}), 200
        except Exception as e:
            print("Error fetching vehicles:", e)
            return jsonify({"error": str(e)}), 500

    # --- Handle POST (Adding a new vehicle) ---
    if request.method == "POST":
        try:
            body = request.get_json()
            model = body.get("model")
            vehicle_type = body.get("vehicle") or body.get("vehicleType") or body.get("type")
            status = body.get("status", "Active")

            if not model or not vehicle_type:
```

```
            return jsonify({"error": "Missing fields"}), 400

        new_id = str(uuid.uuid4())
        item = {
            "user": username, # This is the Partition Key
            "vehicleId": new_id, # This is the Sort Key
            "model": model,
            "vehicle": vehicle_type,
            "status": status,
            "createdAt": int(time.time()),
        }

        table.put_item(Item=item)

        # Return the full updated list of vehicles
        resp = table.scan()
        return jsonify({"user": username, "vehicles": resp.get("Items", [])}), 201

    except Exception as e:
        app.logger.exception("Error adding vehicle")
        return jsonify({"error": str(e)}), 500

# --- Handle DELETE ---
if request.method == "DELETE":
    try:
        body = request.get_json()
        lister_email = body.get("user")
        vehicle_id = body.get("vehicleId")

        # Security check: Make sure the person deleting is the person who listed it
        if username != lister_email:
            return jsonify({"error": "Forbidden: You do not own this vehicle"}), 403

        # Delete using the full composite key
        table.delete_item(
            Key={
                'user': lister_email,
                'vehicleId': vehicle_id
            }
        )

        resp = table.scan() # Return the new list
        return jsonify({"user": username, "vehicles": resp.get("Items", [])}), 200

    except Exception as e:
        app.logger.exception("Error deleting vehicle")
        return jsonify({"error": str(e)}), 500
```

```
# --------------------------------
# POST /api/book
# --------------------------------
@app.route("/api/book", methods=["POST"])
def book_vehicle():

    auth_header = request.headers.get("Authorization")
    if not auth_header:
        return jsonify({"error": "Missing Authorization header"}), 401

    token = auth_header.split(" ")[-1]

    try:
        booker_info = verify_cognito_token(token)
        booker_username = booker_info.get("username") or booker_info.get("email") or
booker_info.get("sub")

        body = request.get_json()
        lister_email = body.get("listerEmail")
        vehicle_id = body.get("vehicleId")
        model = body.get("model")

        if not lister_email or not vehicle_id:
            return jsonify({"error": "Missing listerEmail or vehicleId"}), 400

        table = dynamodb.Table(TABLE_NAME)

        try:
            # Atomic update: only change status if it is still "Active"
            table.update_item(
                Key={
                    'user': lister_email,
                    'vehicleId': vehicle_id
                },
                UpdateExpression="SET #st = :s, bookedBy = :b",
                ConditionExpression="#st = :av", # Make sure it's still 'Active'
                ExpressionAttributeNames={
                    '#st': 'status'
                },
                ExpressionAttributeValues={
                    ':s': 'Booked',
                    ':b': booker_username,
                    ':av': 'Active'
                }
            )
        except dynamodb.meta.client.exceptions.ConditionalCheckFailedException:
            app.logger.warn("Conditional check failed, vehicle already booked")
            return jsonify({"error": "Vehicle is no longer available"}), 409 # 409 Conflict

        # Send SNS notification to the lister
        try:
            sns_message = f"Your vehicle '{model}' (ID: {vehicle_id}) was just booked by {booker_username}!"
            sns.publish(
                TopicArn=SNS_TOPIC_ARN,
                Message=json.dumps({"default": sns_message}),
                MessageStructure="json",
                Subject="VAAHAN: Your Vehicle Was Booked!"
            )
        except Exception as sns_error:
            # Don't fail the whole request if SNS fails, just log it
            print(f"Failed to send SNS message: {sns_error}")

        # Return the new list of all vehicles
        resp = table.scan()
        items = resp.get("Items", [])
        return jsonify({"user": booker_username, "vehicles": items}), 200

    except Exception as e:
        app.logger.exception("Error booking vehicle")
        return jsonify({"error": str(e)}), 500
```

lambda_function.py (Lambda Handler)

```
# lambda_function.py
import aws_lambda_wsgi
from app import app # Imports the 'app' variable from your app.py

def handler(event, context):
    # Use .response() which is the correct function for aws_lambda_wsgi
    return aws_lambda_wsgi.response(app, event, context)
```

9.4. Frontend Code (React.js)

src/pages/Dashboard.js

```
// src/pages/Dashboard.js
import React, { useEffect, useState } from "react";
import AddVehicle from "../components/AddVehicle";

// We use the globally available Amplify auth object from the script
const { fetchAuthSession } = window.aws_amplify.auth;
const                  API_BASE_URL                  =                  "[https://wxu0d4gdoh.execute-api-ap-south-
1.amazonaws.com/default](https://wxu0d4gdoh.execute-api-ap-south-1.amazonaws.com/default)";

function Dashboard({ user }) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // function to load all vehicles
  async function loadData() {
    setLoading(true);
    setError(null);
    try {
      const session = await fetchAuthSession();
      const idToken = session.tokens?.idToken?.toString();

      const res = await fetch(
        `${API_BASE_URL}/api/vehicles`,
        {
          method: "GET",
          headers: {
            Authorization: `Bearer ${idToken}`,
          },
        }
      );

      if (!res.ok) throw res;
      const resData = await res.json();
      setData(resData);
    } catch (err) {
      console.error("Error fetching vehicle data:", err);
      setError("Failed to load vehicle data");
    } finally {
      setLoading(false);
    }
  }

  // load once on mount
  useEffect(() => {
    loadData();
  }, []);

  // Handle Booking
  async function handleBook(vehicle) {
    if (
      !window.confirm(`Are you sure you want to book the ${vehicle.model}?`)
    ) {
      return;
    }
    setLoading(true);
    setError(null);
    try {
      const session = await fetchAuthSession();
      const idToken = session.tokens?.idToken?.toString();
      const payload = {
        listerEmail: vehicle.user,
        vehicleId: vehicle.vehicleId,
        model: vehicle.model,
      };
      const res = await fetch(
        `${API_BASE_URL}/api/book`,
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
            Authorization: `Bearer ${idToken}`,
          },
          body: JSON.stringify(payload),
        }
      );
      if (!res.ok) throw res;
      const newData = await res.json();
      setData(newData); // Refresh the list
    } catch (err) {
      console.error("Error booking vehicle:", err);
      try {
        const errorBody = await err.json();
        setError(errorBody.error || "Failed to book vehicle.");
      } catch (parseErr) {
        setError("Failed to book vehicle. Check console.");
      }
    } finally {
      setLoading(false);
    }
  }

  // Handle Remove
  async function handleRemove(vehicle) {
    if (
      !window.confirm(
        `Are you sure you want to REMOVE your listing for the ${vehicle.model}?`
      )
    ) {
      return;
    }
    setLoading(true);
    setError(null);
    try {
      const session = await fetchAuthSession();
      const idToken = session.tokens?.idToken?.toString();
      const payload = {
        user: vehicle.user,
        vehicleId: vehicle.vehicleId,
      };
      const res = await fetch(
        `${API_BASE_URL}/api/vehicles`,
        {
          method: "DELETE",
          headers: {
            "Content-Type": "application/json",
            Authorization: `Bearer ${idToken}`,
          },
          body: JSON.stringify(payload),
        }
      );
      if (!res.ok) throw res;
      const newData = await res.json();
      setData(newData); // Refresh the list
    } catch (err) {
      console.error("Error removing vehicle:", err);
      try {
        const errorBody = await err.json();
        setError(errorBody.error || "Failed to remove vehicle.");
      } catch (parseErr) {
        setError("Failed to remove vehicle. Check console.");
      }
    } finally {
      setLoading(false);
    }
  }

  return (
    <div className="space-y-8">
      {/* --- Welcome Header --- */}
      <div className="text-center">
        <h1 className="text-4xl font-bold text-gray-900">    Vaahan Dashboard</h1>
        <p className="text-lg text-gray-600 mt-2">
          Welcome back, <span className="font-semibold text-indigo-600">{user?.signInDetails?.loginId ||
"User"}</span>
        </p>
        <p className="text-gray-500">This is your main workspace — view and manage vehicle data
  here.</p>
      </div>

      {/* --- Add Vehicle Form --- */}
      <AddVehicle onAdded={setData} />

      {/* --- Vehicle Marketplace --- */}
      <div>
        <h3 className="text-2xl font-semibold text-gray-900 mb-5">    Vehicle Marketplace</h3>
        {loading && <p className="text-center text-gray-500">Loading vehicle data...</p>}
        {error && <p className="text-center text-red-500">{error}</p>}

        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
          {data && data.vehicles && data.vehicles.map((vehicle) => (
            <div
              key={vehicle.vehicleId}
              className="bg-white rounded-2xl shadow-lg border border-gray-100 overflow-hidden
  transition-all duration-300 hover:shadow-xl"
            >
              <div className="p-6">
                {/* --- Status Badge --- */}
                {vehicle.status === "Active" ? (
                  <span className="inline-block bg-green-100 text-green-800 text-xs font-semibold px-3 py-
  1 rounded-full uppercase tracking-wide">
                    {vehicle.status}
                  </span>
                ) : (
                  <span className="inline-block bg-yellow-100 text-yellow-800 text-xs font-semibold px-3
  py-1 rounded-full uppercase tracking-wide">
                    {vehicle.status}
                  </span>
                )}

                <h4 className="text-2xl font-bold text-gray-900 mt-3">{vehicle.model}</h4>

                <div className="mt-2 text-gray-500 space-y-1">
                  <p>Type: <span className="font-medium text-gray-700">{vehicle.vehicle}</span></p>
                  <p className="text-sm">Lister: <span className="font-medium text-gray-
  700">{vehicle.user}</span></p>
                </div>

                {vehicle.status === "Booked" && (
                  <p className="mt-3 text-sm font-semibold text-red-600">
                    Booked by {vehicle.bookedBy || "another user"}
                  </p>
                )}
              </div>

              {/* --- Card Footer with Buttons --- */}
              <div className="bg-gray-50 px-6 py-4">
                {/* 1. It's my listing */}
                {data.user === vehicle.user && (
                  <div className="text-center">
                    <p className="text-sm font-semibold text-indigo-600 mb-2">Your Listing</p>
                    <button
                      onClick={() => handleRemove(vehicle)}
                      className="w-full bg-red-500 text-white font-semibold py-2 px-4 rounded-lg shadow-
  md hover:bg-red-600 focus:outline-none focus:ring-2 focus:ring-red-500 focus:ring-offset-2 transition
  duration-150"
                    >
                      Remove
                    </button>
                  </div>
                )}

                {/* 2. It's available to book */}
                {data.user !== vehicle.user && vehicle.status === "Active" && (
                  <button
                    onClick={() => handleBook(vehicle)}
                    className="w-full bg-green-500 text-white font-semibold py-2 px-4 rounded-lg shadow-
  md hover:bg-green-600 focus:outline-none focus:ring-2 focus:ring-green-500 focus:ring-offset-2
  transition duration-150"
                  >
                    Book Now
                  </button>
                )}
              </div>
            </div>
          ))}
        </div>
      </div>
    </div>
    );
  }

  export default Dashboard;
```

src/components/AddVehicle.jsx

```
// src/components/AddVehicle.jsx
import React, { useState } from "react";
```

```
// We use the globally available Amplify auth object from the script
const { fetchAuthSession } = window.aws_amplify.auth;

const                API_BASE_URL                =                "[https://wxu0d4gdoh.execute-api-ap-south-
1.amazonaws.com/default](https://wxu0d4gdoh.execute-api-ap-south-1.amazonaws.com/default)";

export default function AddVehicle({ onAdded }) {
  const [model, setModel] = useState("");
  const [vehicleType, setVehicleType] = useState("Car");
  const [status, setStatus] = useState("Active");
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  async function handleSubmit(e) {
    e.preventDefault();
    setError(null);
    setLoading(true);
    try {
      const session = await fetchAuthSession();
      const idToken = session.tokens?.idToken?.toString();

      if (!idToken) {
        throw new Error("Failed to retrieve ID token from session");
      }

      const payload = { model, vehicle: vehicleType, status };

      const res = await fetch(
        `${API_BASE_URL}/api/vehicles`,
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
            Authorization: `Bearer ${idToken}`,
          },
          body: JSON.stringify(payload),
        }
      );

      if (!res.ok) {
        const text = await res.text();
        throw new Error(`${res.status} ${text}`);
      }

      const data = await res.json();
      setModel("");
      setVehicleType("Car");
      setStatus("Active");
      if (onAdded) onAdded(data);
    } catch (err) {
      console.error("Add vehicle error:", err);
      setError(err.message || "Failed to add vehicle");
    } finally {
      setLoading(false);
    }
  }

  return (
    <div className="bg-white p-6 md:p-8 rounded-2xl shadow-lg border border-gray-100 max-w-2xl
mx-auto">
      <h3 className="text-xl font-semibold text-gray-900 mb-5">Add a Vehicle to the Marketplace</h3>
      <form onSubmit={handleSubmit} className="space-y-4">
        <div>
          <label htmlFor="model" className="block text-sm font-medium text-gray-700 mb-1">
            Model
          </label>
          <input
            id="model"
            value={model}
            onChange={(e) => setModel(e.target.value)}
            required
            className="w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:ring-
indigo-500 focus:border-indigo-500"
            placeholder="e.g., Tesla Model 3"
          />
        </div>

        <div className="grid grid-cols-2 gap-4">
          <div>
            <label htmlFor="type" className="block text-sm font-medium text-gray-700 mb-1">
              Type
            </label>
            <select
              id="type"
              value={vehicleType}
              onChange={(e) => setVehicleType(e.target.value)}
              className="w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:ring-
indigo-500 focus:border-indigo-500"
            >
              <option>Car</option>
              <option>Bike</option>
              <option>SUV</option>
              <option>Van</option>
            </select>
          </div>
          <div>
            <label htmlFor="status" className="block text-sm font-medium text-gray-700 mb-1">
              Status
            </label>
            <select
              id="status"
              value={status}
              onChange={(e) => setStatus(e.target.value)}
              className="w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:ring-
indigo-500 focus:border-indigo-500"
            >
              <option>Active</option>
              <option>Inactive</option>
            </select>
          </div>
        </div>

        <button
          type="submit"
          disabled={loading}
          className="w-full py-3 px-4 bg-indigo-600 text-white font-semibold rounded-md shadow-md
hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:ring-offset-2 transition
duration-150 ease-in-out disabled:opacity-50"
        >
          {loading ? "Adding..." : "Add Vehicle"}
        </button>
        {error && <div className="text-red-600 text-sm text-center mt-2">{error}</div>}
      </form>
    </div>
  );
}
```

src/components/Navbar.jsx

```
import React from "react";

// The Navbar receives `onNavigate` from App.js to handle page changes.
const Navbar = ({ user, onLogout, onNavigate }) => {
  return (
    <nav className="bg-gradient-to-r from-gray-800 to-gray-900 text-white shadow-lg">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div className="flex items-center justify-between h-16">
          {/* --- Logo and Title --- */}
          <div className="flex-shrink-0">
            <h2 className="text-2xl font-bold">VAAHAN    </h2>
          </div>

          {/* --- Navigation Links --- */}
          <div className="hidden md:block">
            <div className="ml-10 flex items-baseline space-x-4">
              <button
                onClick={() => onNavigate('dashboard')}
                className="text-gray-300 hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-
sm font-medium transition"
              >
                Dashboard
              </button>
              <button
                onClick={() => onNavigate('profile')}
                className="text-gray-300 hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-
sm font-medium transition"
              >
                Profile
              </button>
            </div>
          </div>

          {/* --- Logout Button --- */}
          {user && (
            <button
              onClick={onLogout}
              className="ml-4 bg-indigo-500 text-white font-semibold px-4 py-2 rounded-lg shadow-md
hover:bg-indigo-600 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:ring-offset-2
focus:ring-offset-gray-800 transition duration-150"
            >
              Logout
            </button>
          )}
        </div>
      </div>
    </nav>
  );
};

export default Navbar;
```

src/pages/Profile.js

```
import React, { useEffect, useState } from "react";

// We use the globally available Amplify auth object from the script
```

```
const { fetchUserAttributes } = window.aws_amplify.auth;

function Profile() {
  const [attrs, setAttrs] = useState(null);

  useEffect(() => {
    async function load() {
      try {
        const data = await fetchUserAttributes();
        setAttrs(data);
      } catch (e) {
        console.error("Error fetching attributes", e);
      }
    }
    load();
  }, []);

  return (
    <div className="max-w-md mx-auto mt-10 bg-white p-8 rounded-2xl shadow-lg border border-
gray-100">
      <h1 className="text-3xl font-bold text-gray-900 text-center mb-6">    User Profile</h1>
      {attrs ? (
        <div className="space-y-4">
          <div className="text-lg">
            <span className="font-medium text-gray-500">Email:</span>
            <span className="ml-2 font-semibold text-gray-900">{attrs.email}</span>
          </div>

          {attrs.phone_number && (
            <div className="text-lg">
              <span className="font-medium text-gray-500">Phone:</span>
              <span className="ml-2 font-semibold text-gray-900">{attrs.phone_number}</span>
            </div>
          )}

          {attrs.email_verified && (
             <div className="flex items-center justify-center bg-green-100 text-green-700 p-3 rounded-
lg">
                <span className="font-semibold">✅ Email Verified</span>
             </div>
          )}
        </div>
      ) : (
        <p className="text-center text-gray-500">Loading your profile...</p>
      )}
    </div>
  );
}

export default Profile;
```

10. References

- Amazon API Gateway Developer Guide
- Amazon S3 Documentation
- Amazon CloudFront Developer Guide
- Amazon SNS Developer Guide
- Amazon CloudWatch User Guide
- IBM Watson Assistant Documentation
- How to Use AWS Cognito for User Authentication
- Amazon DynamoDB Documentation
- React Documentation
- AWS Lambda Documentation