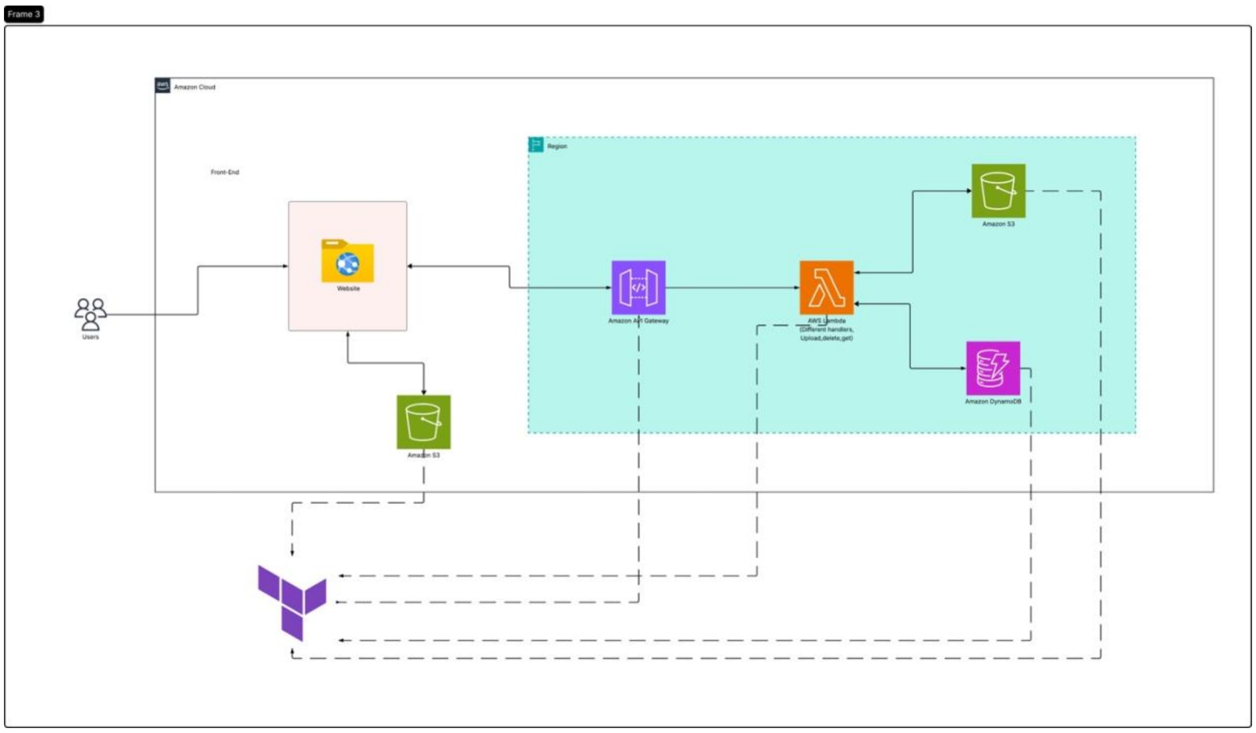
Building My Full AWS Serverless Application

A fully serverless, end-to-end application designed to manage, display, and store car inventory data with separate Admin & Customer interfaces.

Architecture Diagram



Situation

I wanted to build a real-world full-stack cloud project showcasing my AWS skills.
The goal: create a working Car Inventory System where admins manage inventory and customers browse cars online.

Task

Build both the backend and frontend using:

- AWS serverless services
- Terraform for infrastructure
- A public customer site
- An admin management panel
- Functional image uploads (presigned URLs)

- Database integration (DynamoDB)

Action

- Designed the system architecture
- Created Terraform modules for S3, Lambda, API Gateway, IAM, and DynamoDB
- Built & deployed all Lambda functions in Python
- Implemented presigned S3 upload URLs
- Built a functional admin dashboard
- Built a clean customer website
- Solved multiple AWS issues:
  - CORS errors
  - S3 Access Denied (403)
  - Block Public Access conflicts
  - Incorrect Lambda event parsing
  - API integration errors
- Validated and tested all routes end-to-end
- Finalized deployment for both websites

Result

- Fully working, professional AWS serverless app
- Admin site can add, edit, upload images, delete, and manage inventory
- Customer site displays the real cars from DynamoDB with real S3 images
- Image uploads and retrieval are working end-to-end
- Infrastructure is fully automated with Terraform
- Gained real industry-level AWS engineering experience

Technologies Used

- AWS Lambda
- AWS API Gateway (HTTP API)
- AWS DynamoDB
- AWS S3 (two buckets: admin + customer)
- IAM
- Terraform
- HTML, CSS, JavaScript

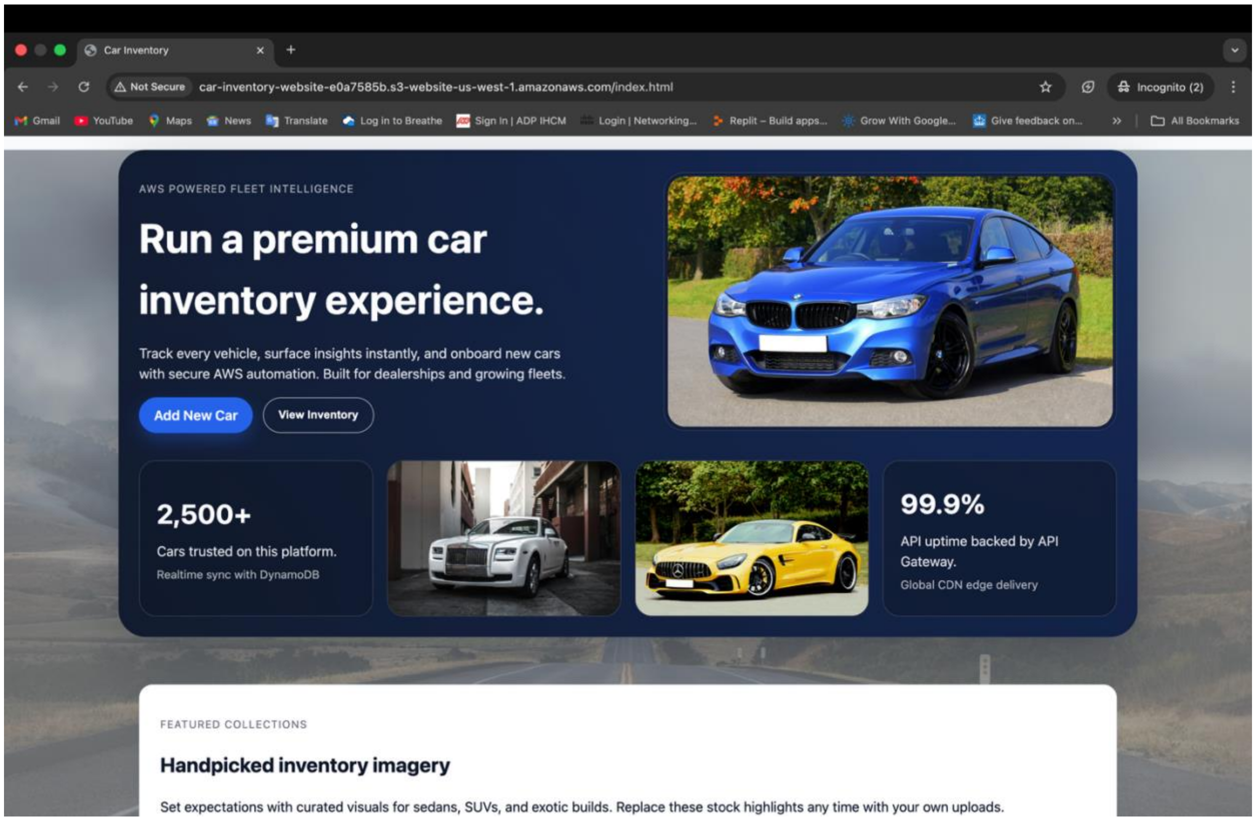
Project Overview

This system is split into **two applications**:

Admin Dashboard

Used to:

- Add new cars
- Edit car details
- Upload images
- Delete cars
- Manage inventory



AWS POWERED FLEET INTELLIGENCE

Run a premium car inventory experience.

Track every vehicle, surface insights instantly, and onboard new cars with secure AWS automation. Built for dealerships and growing fleets.

2,500+
Cars trusted on this platform.
Realtime sync with DynamoDB

99.9%
API uptime backed by API Gateway.
Global CDN edge delivery

FEATURED COLLECTIONS

Handpicked inventory imagery

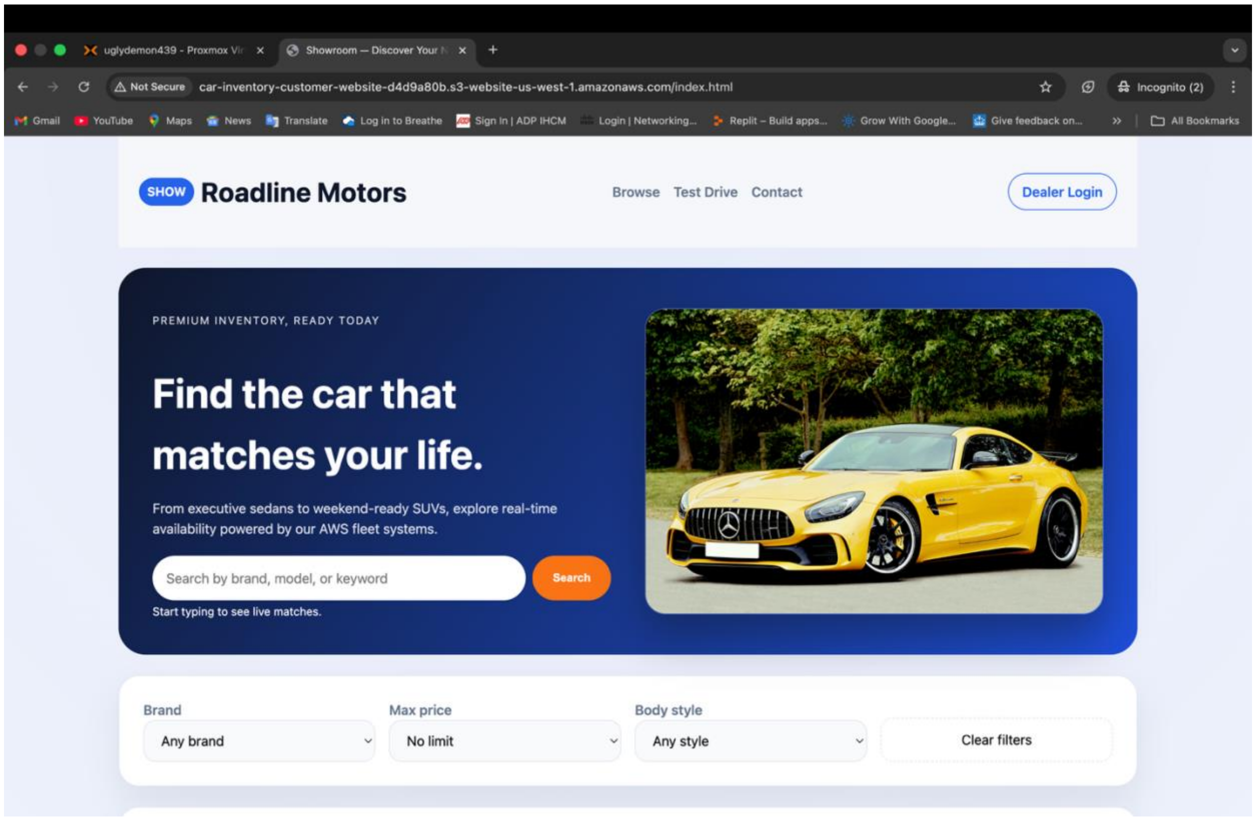
Set expectations with curated visuals for sedans, SUVs, and exotic builds. Replace these stock highlights any time with your own uploads.

Customer Website

Used for:

- Viewing all cars
- Seeing details and images
- Browsing the available inventory

Each site is hosted in its own S3 bucket for isolation and professionalism.



car-inventory-customer-website-d4d9a80b.s3-website-us-west-1.amazonaws.com/index.html

Find the car that matches your life.

From executive sedans to weekend-ready SUVs, explore real-time availability powered by our AWS fleet systems.

Start typing to see live matches.

Backend Architecture

The backend is built entirely on AWS serverless services:

API Workflow

1. Frontend calls API Gateway
2. API Gateway routes request to Lambda
3. Lambda reads/writes to DynamoDB
4. For image uploads: Lambda generates presigned S3 PUT URLs
5. Images get stored in S3 and displayed publicly

Infrastructure (Terraform)

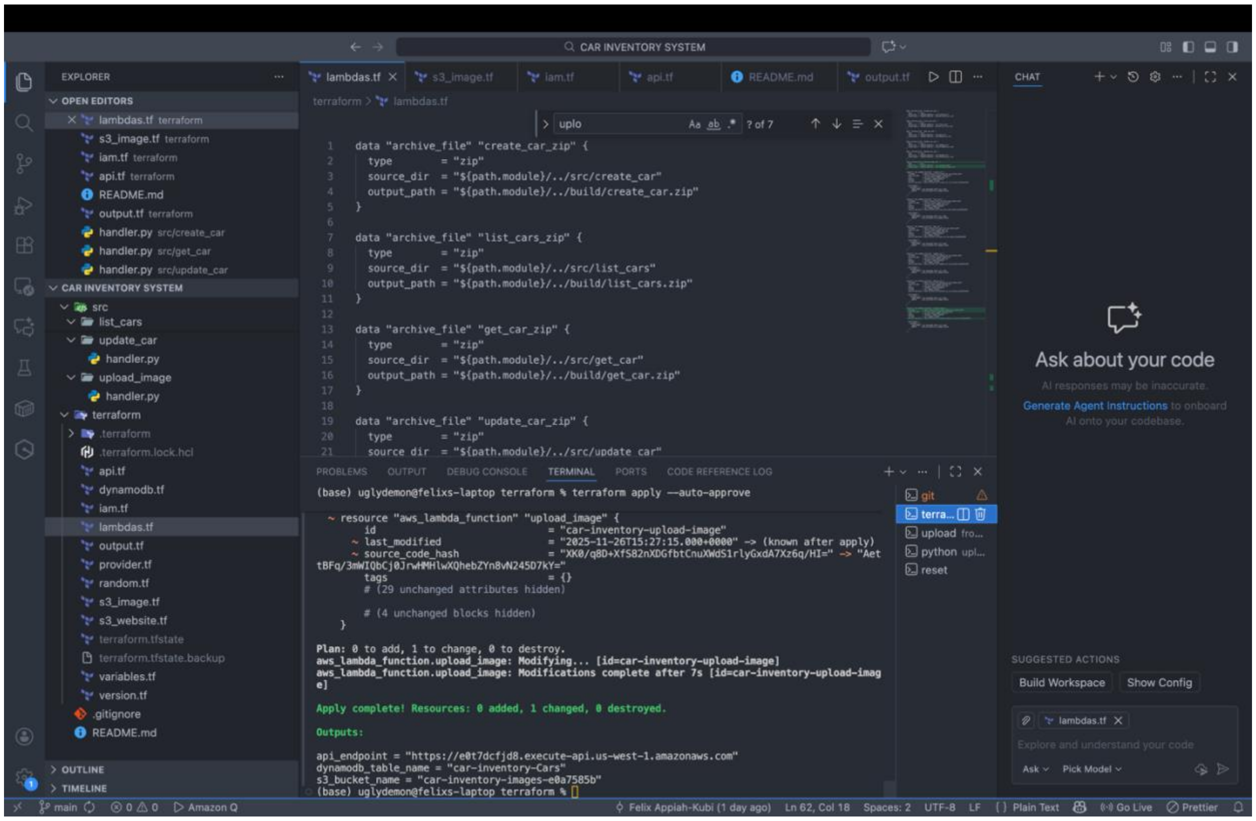
All AWS resources are provisioned through Terraform:

- S3 buckets (admin + customer + image storage)
- Bucket policies
- IAM roles & policies
- Lambda functions & environment variables
- DynamoDB table
- API Gateway routes & integrations

This ensures repeatable, consistent deployments.

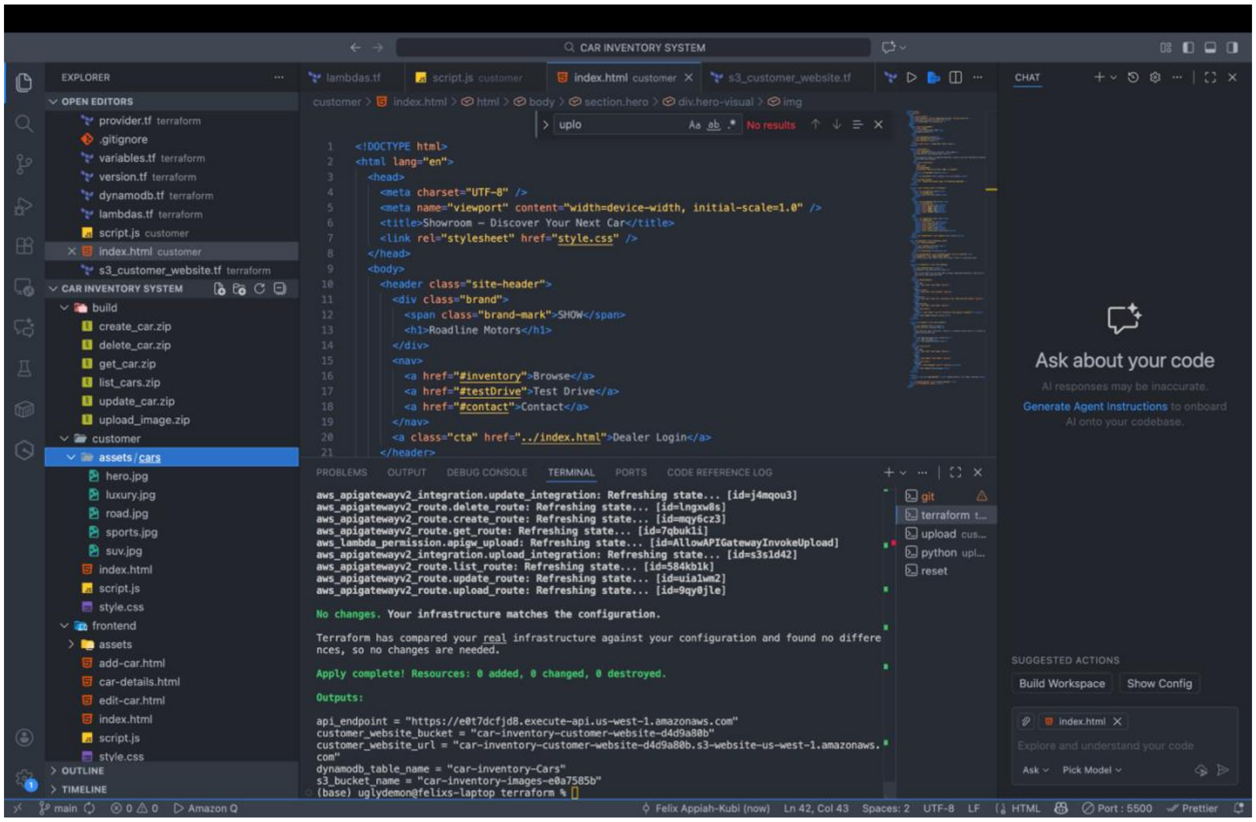
What I built:

- A full serverless backend using AWS Lambda, API Gateway, S3, and DynamoDB
- An Admin Dashboard to upload cars, edit details, upload images, and manage inventory
- A Customer Website hosted separately for public access
- Presigned URL image uploads
- Full AWS Terraform deployment



```
data "archive_file" "create_car_zip" {
  type        = "zip"
  source_dir  = "${path.module}/../src/create_car"
  output_path = "${path.module}/../build/create_car.zip"
}

data "archive_file" "list_cars_zip" {
  type        = "zip"
  source_dir  = "${path.module}/../src/list_cars"
  output_path = "${path.module}/../build/list_cars.zip"
}

data "archive_file" "get_car_zip" {
  type        = "zip"
  source_dir  = "${path.module}/../src/get_car"
  output_path = "${path.module}/../build/get_car.zip"
}

data "archive_file" "update_car_zip" {
  type        = "zip"
  source_dir  = "${path.module}/../src/update_car"
```

```
(base) uglydemon@felixs-laptop terraform % terraform apply —auto-approve

  ~ resource "aws_lambda_function" "upload_image" {
        id                             = "car-inventory-upload-image"
      ~ last_modified                  = "2025-11-26T15:27:15.000+0000" -> (known after apply)
      ~ source_code_hash               = "XK0/q8D+XfS82nXDGfbtCnuXWdS1rlyGxdA7Xz6q/HI=" -> "Aet
tBFq/3mWIQbCj0JrwHMHlwXQhebZYn8vN245D7kY="
        tags                           = {}
        # (29 unchanged attributes hidden)

        # (4 unchanged blocks hidden)
    }

Plan: 0 to add, 1 to change, 0 to destroy.
aws_lambda_function.upload_image: Modifying... [id=car-inventory-upload-image]
aws_lambda_function.upload_image: Modifications complete after 7s [id=car-inventory-upload-imag
e]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.

Outputs:

api_endpoint = "https://e0t7dcfjd8.execute-api.us-west-1.amazonaws.com"
dynamodb_table_name = "car-inventory-Cars"
s3_bucket_name = "car-inventory-images-e0a7585b"
(base) uglydemon@felixs-laptop terraform %
```



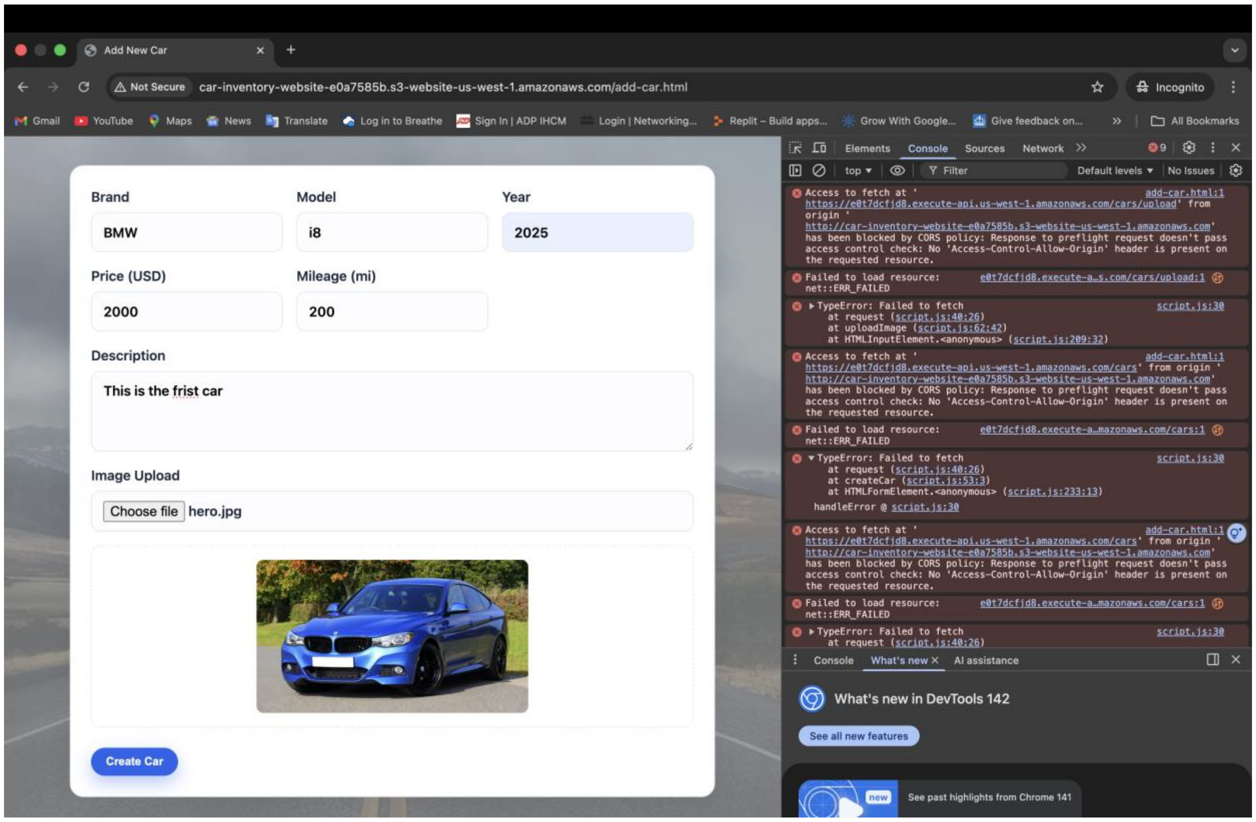
```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Showroom — Discover Your Next Car</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <header class="site-header">
      <div class="brand">
        <span class="brand-mark">SHOW</span>
        <h1>Roadline Motors</h1>
      </div>
      <nav>
        <a href="#inventory">Browse</a>
        <a href="#testDrive">Test Drive</a>
        <a href="#contact">Contact</a>
      </nav>
      <a class="cta" href="../index.html">Dealer Login</a>
    </header>
```

```
aws_apigatewayv2_integration.update_integration: Refreshing state... [id=j4mqou3]
aws_apigatewayv2_route.delete_route: Refreshing state... [id=lngxw8s]
aws_apigatewayv2_route.create_route: Refreshing state... [id=mqy6cz3]
aws_apigatewayv2_route.get_route: Refreshing state... [id=7qbukli]
aws_lambda_permission.apigw_upload: Refreshing state... [id=AllowAPIGatewayInvokeUpload]
aws_apigatewayv2_integration.upload_integration: Refreshing state... [id=s3s1d42]
aws_apigatewayv2_route.list_route: Refreshing state... [id=584kb1k]
aws_apigatewayv2_route.update_route: Refreshing state... [id=uia1wm2]
aws_apigatewayv2_route.upload_route: Refreshing state... [id=9qy0jle]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differe
nces, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

api_endpoint = "https://e0t7dcfjd8.execute-api.us-west-1.amazonaws.com"
customer_website_bucket = "car-inventory-customer-website-d4d9a80b"
customer_website_url = "car-inventory-customer-website-d4d9a80b.s3-website-us-west-1.amazonaws.
com"
dynamodb_table_name = "car-inventory-Cars"
s3_bucket_name = "car-inventory-images-e0a7585b"
(base) uglydemon@felixs-laptop terraform %
```

What I learned:

This project taught me far more than I expected:

- Debugging real AWS issues (CORS, 403 Forbidden, presigned URL mismatch, permissions…)
- Fixing S3 Block Public Access settings properly
- Understanding how API Gateway integrates with Lambda
- Handling image uploads and retrieval with S3
- Structuring a real production-like serverless backend



Brand	Model	Year
BMW	i8	2025

Price (USD)	Mileage (mi)
2000	200

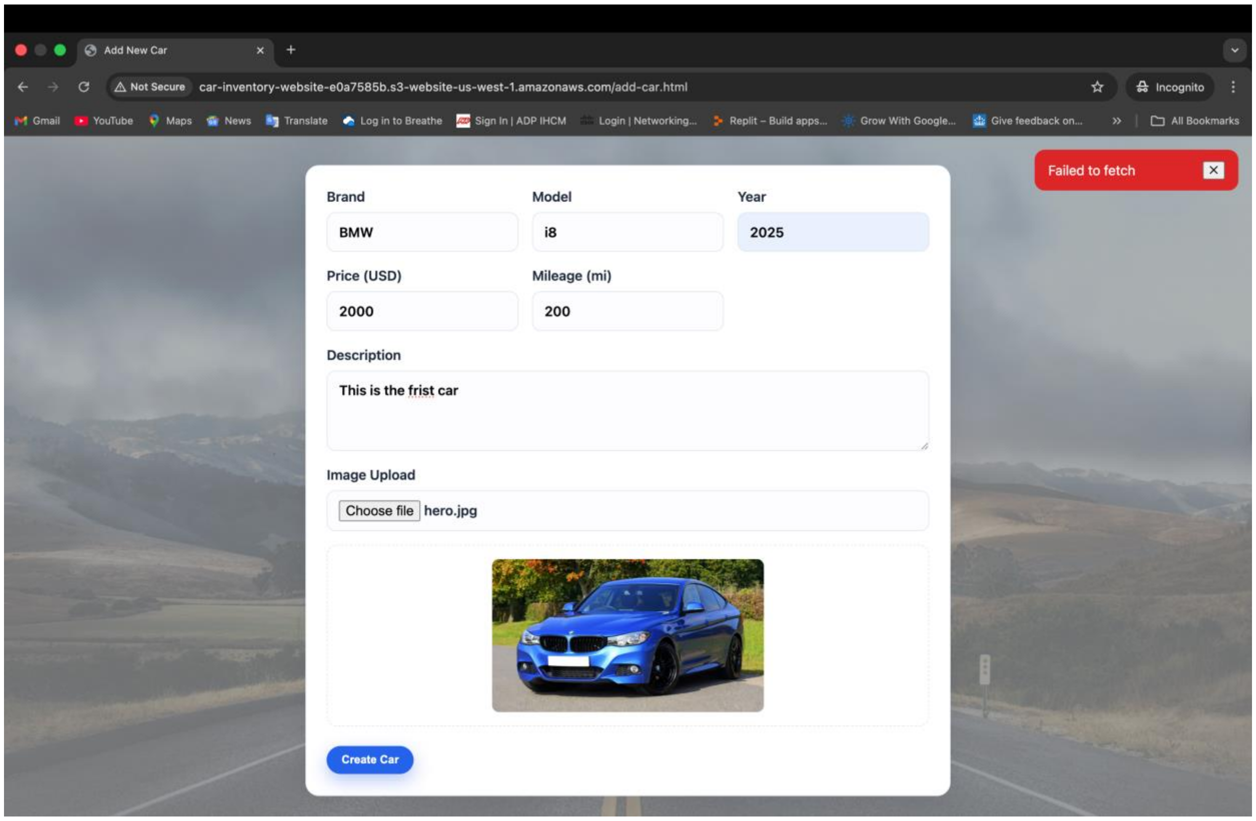
Description

This is the frist car

Image Upload

Choose file hero.jpg

```
Access to fetch at '                                                 add-car.html:1
https://e0t7dcfjd8.execute-api.us-west-1.amazonaws.com/cars/upload' from
origin '
http://car-inventory-website-e0a7585b.s3-website-us-west-1.amazonaws.com'
has been blocked by CORS policy: Response to preflight request doesn't pass
access control check: No 'Access-Control-Allow-Origin' header is present on
the requested resource.
Failed to load resource:        e0t7dcfjd8.execute-a…s.com/cars/upload:1
net::ERR_FAILED
▶TypeError: Failed to fetch                                       script.js:30
    at request (script.js:40:26)
    at uploadImage (script.js:62:42)
    at HTMLInputElement.<anonymous> (script.js:209:32)
Access to fetch at '                                                 add-car.html:1
https://e0t7dcfjd8.execute-api.us-west-1.amazonaws.com/cars' from origin '
http://car-inventory-website-e0a7585b.s3-website-us-west-1.amazonaws.com'
has been blocked by CORS policy: Response to preflight request doesn't pass
access control check: No 'Access-Control-Allow-Origin' header is present on
the requested resource.
Failed to load resource:        e0t7dcfjd8.execute-a…mazonaws.com/cars:1
net::ERR_FAILED
▼TypeError: Failed to fetch                                       script.js:30
    at request (script.js:40:26)
    at createCar (script.js:53:3)
    at HTMLFormElement.<anonymous> (script.js:233:13)
  handleError @ script.js:30
Access to fetch at '                                                 add-car.html:1
https://e0t7dcfjd8.execute-api.us-west-1.amazonaws.com/cars' from origin '
http://car-inventory-website-e0a7585b.s3-website-us-west-1.amazonaws.com'
has been blocked by CORS policy: Response to preflight request doesn't pass
access control check: No 'Access-Control-Allow-Origin' header is present on
the requested resource.
Failed to load resource:        e0t7dcfjd8.execute-a…mazonaws.com/cars:1
net::ERR_FAILED
▶TypeError: Failed to fetch                                       script.js:30
    at request (script.js:40:26)
```



Failed to fetch

Brand	Model	Year
BMW	i8	2025

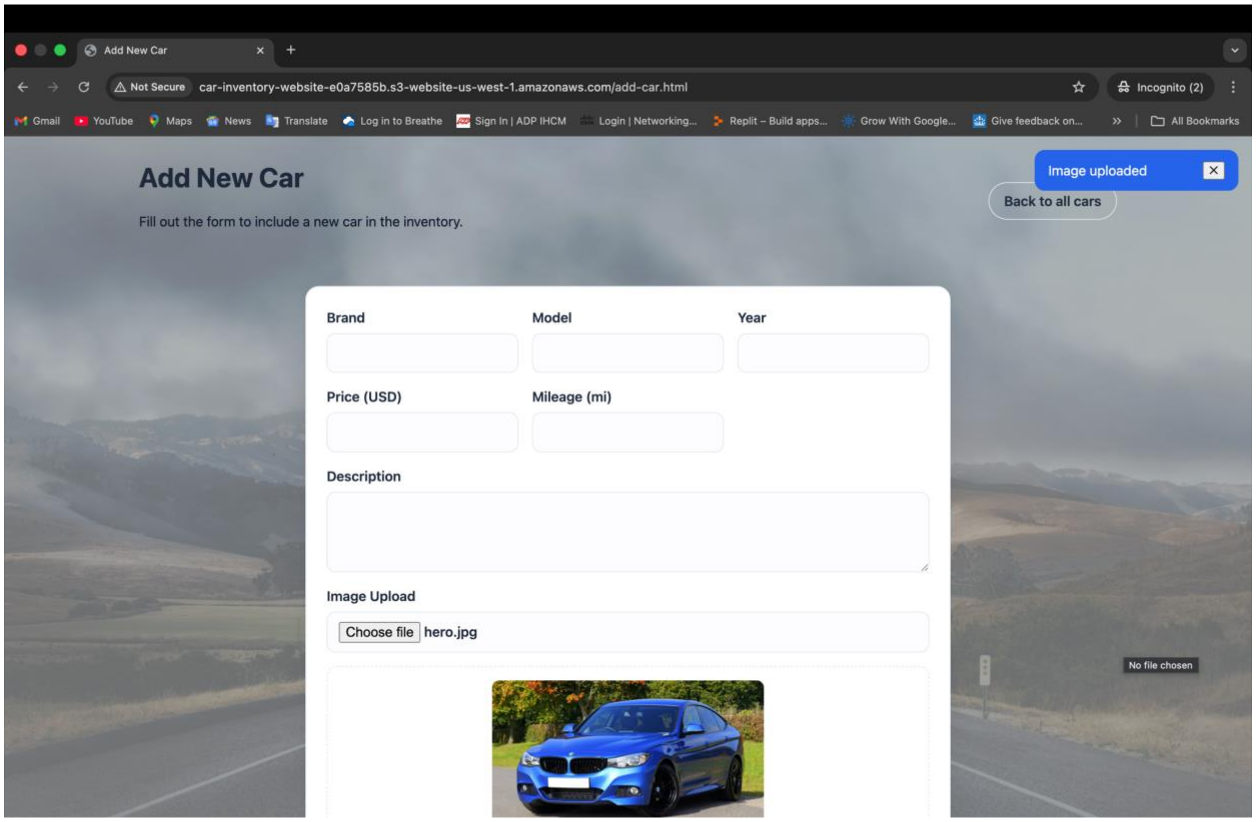
Price (USD)	Mileage (mi)
2000	200

Description

This is the frist car

Image Upload

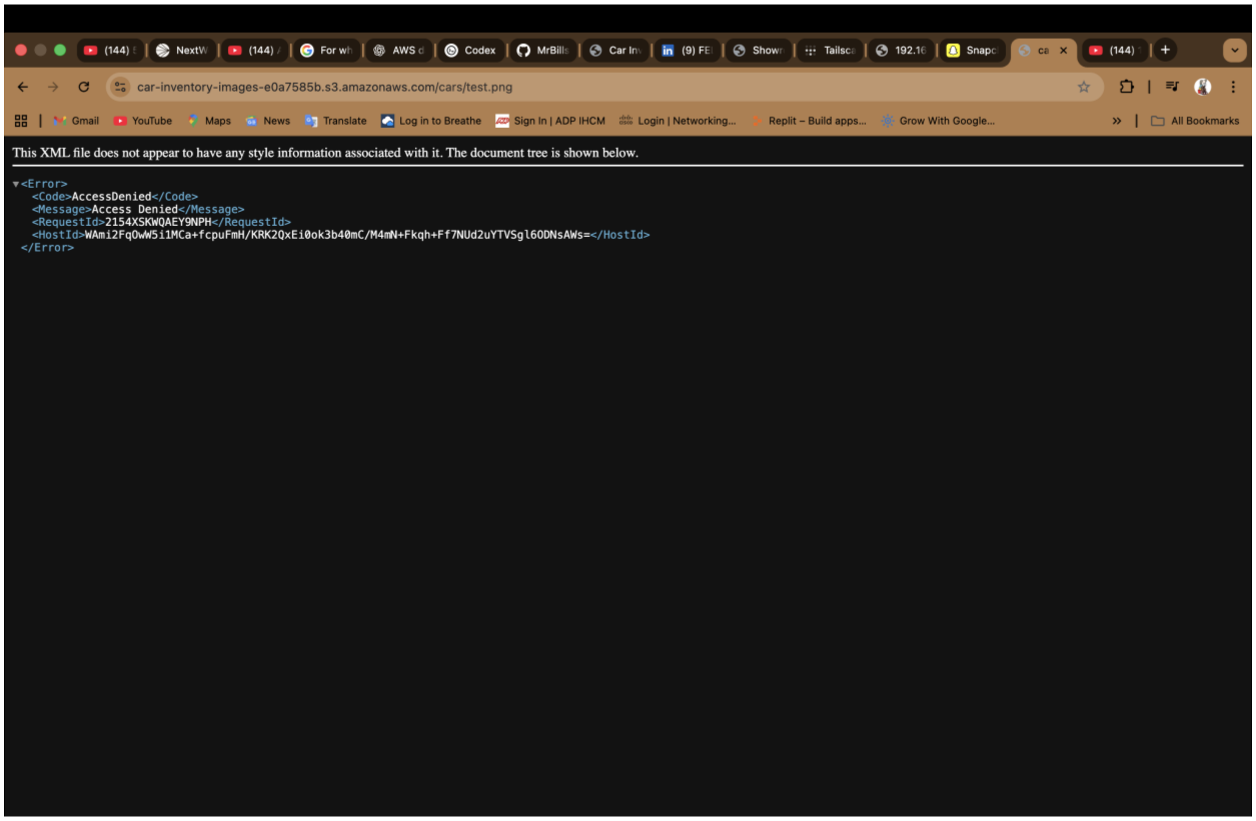
Choose file hero.jpg



Add New Car

Fill out the form to include a new car in the inventory.

Brand:
Model:
Year:
Price (USD):
Mileage (mi):
Description:
Image Upload: hero.jpg


car-inventory-images-e0a7585b.s3.amazonaws.com/cars/test.png

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Error>
   <Code>AccessDenied</Code>
   <Message>Access Denied</Message>
   <RequestId>2154XSKWQAEY9NPH</RequestId>
   <HostId>WAmi2FqOwW5i1MCa+fcpuFmH/KRK2QxEi0ok3b40mC/M4mN+Fkqh+Ff7NUd2uYTVSgl6ODNsAWs=</HostId>
 </Error>
```