Kanika Mathur
github.com/KanikaGenesis

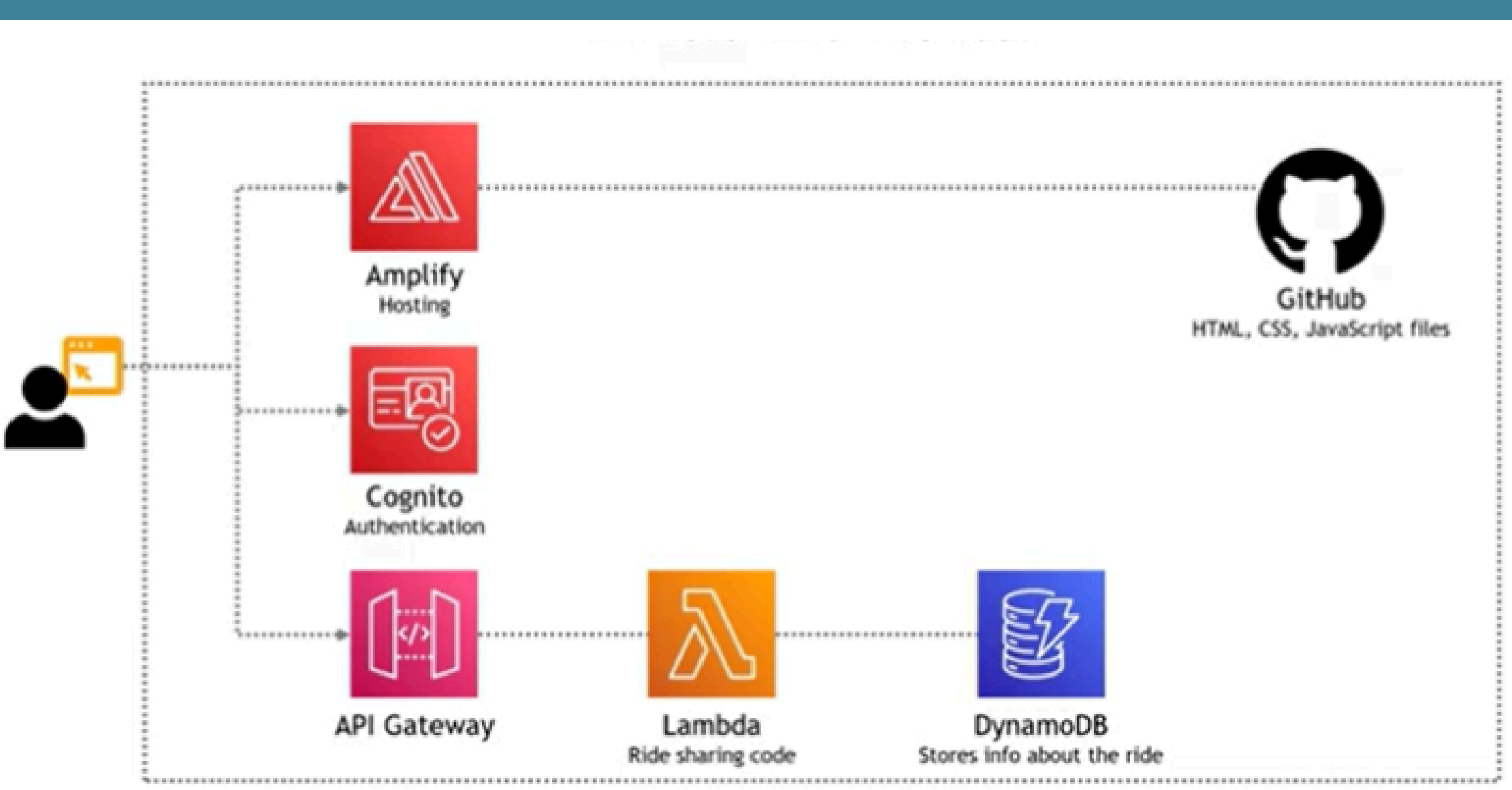
Wild Rydes Serverless Application

Kanika Mathur
github.com/KanikaGenesis

Project Overview

In this project, I created a fun and fictional web application for a unicorn ride-sharing service called Wild Rydes. The app uses 6 AWS services - IAM, Amplify, Cognito, Lambda, API Gateway and DynamoDB, with code stored in GitHub and incorporated into a CI/CD pipeline with Amplify.

The app will let you create an account and log in, then request a ride by clicking on a map.



Kanika Mathur
github.com/KanikaGenesis

Deploying the Application with GitHub and AWS Amplify

- Step 1: I set up a **GitHub** repository to manage version control and collaboration. The repository contains the application code, ensuring streamlined updates.
- Step 2: I configured **AWS Amplify** to deploy the Wild Rydes app directly from GitHub. With automated deployments for each code push, Amplify provided a robust and efficient hosting solution.

Kanika Mathur
github.com/KanikaGenesis

AWS Amplify All apps / Create new app

1. Choose source code provider
2. Add repository and branch
3. App settings
4. Review

Start building with Amplify

Amplify provides a fully-managed web hosting experience and a backend building service to build fullstack apps. If you need a starter project, please visit the docs.

Deploy your app

To deploy an app from a Git provider, select one of the options below:

GitHub | BitBucket | CodeCommit | GitLab

Amplify requires read-only access to your repository.

To manually deploy an app from Amazon S3 or a Zip file, select "Deploy without Git"

Deploy without Git

Start with a template

Cancel | Previous | Next

Looking to build an app with our Gen 1 tools (Amplify Studio/Amplify CLI)? Create an app with Gen 1

[Alt+S] United States (N. Virginia) Kanika-IAM-User @ 7172-7973-3653

erview Support Docs

wildrydes-site

Manage sandboxes | Visit deployed URL

App ID: d8cjydf59wmga

Branches 1 | Search... | Add branch

main

Deployed

★ Production branch

Domain	Last deployment	Last commit
https://main.d8cjydf59wmga.amplifyapp.com	1 minute ago	Update index.html / wildrydes-site:main

Kanika Mathur
github.com/KanikaGenesis

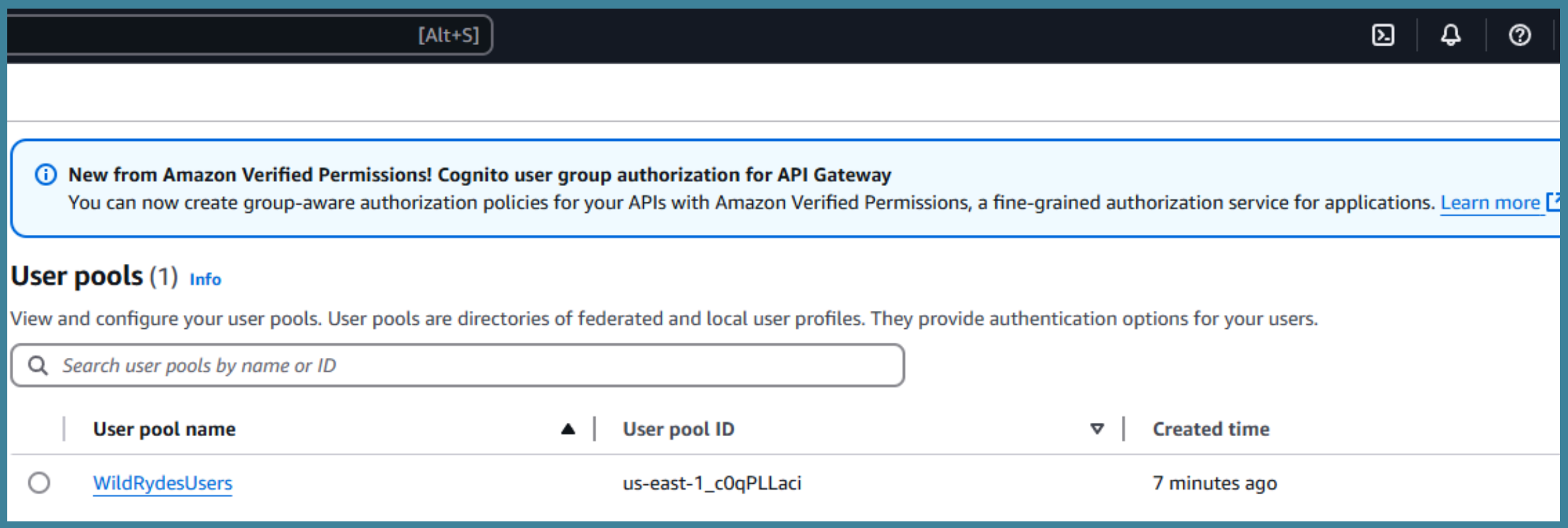
Implementing User Authentication with Amazon Cognito

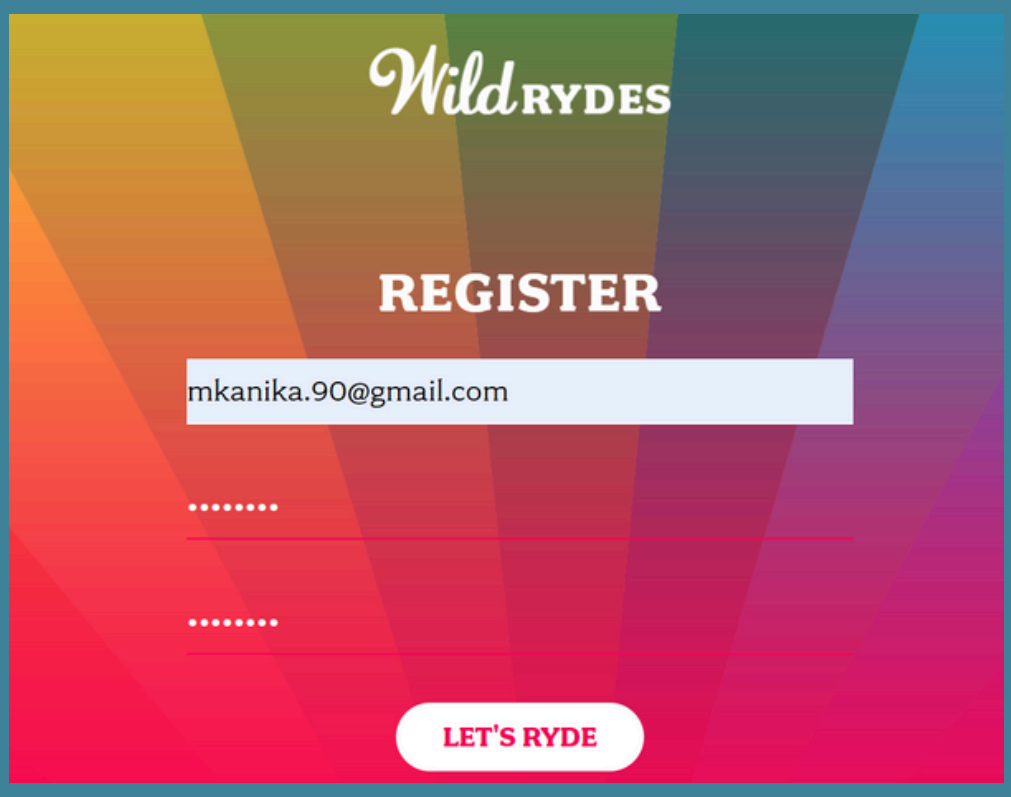
- Step 1: I created a **Cognito** User Pool named **WildRydesUsers**, along with an App Client for the web application. Username-based login and email verification were configured.

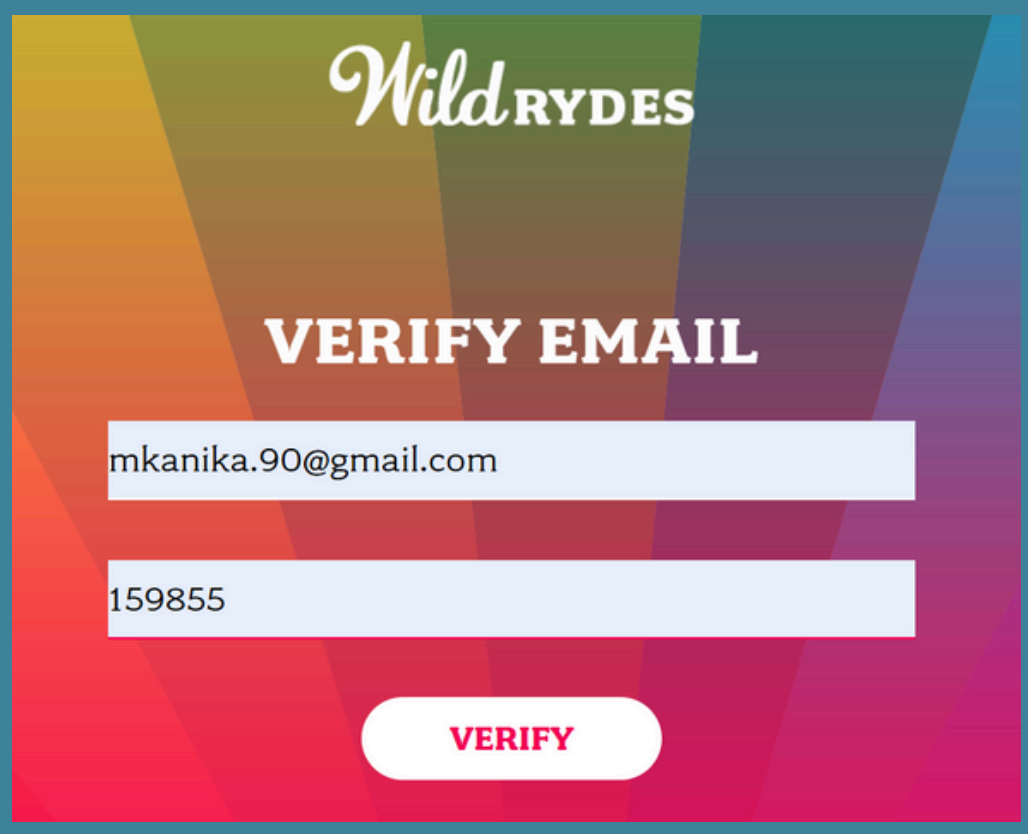
- Step 2: I recorded the User Pool ID and Client ID for integration purposes.

- Step 3: I updated the config.js file with these IDs to enable authentication. This allowed users to register, verify their accounts, and log in to request rides on the “**Giddy Up**” page.

Kanika Mathur
github.com/KanikaGenesis



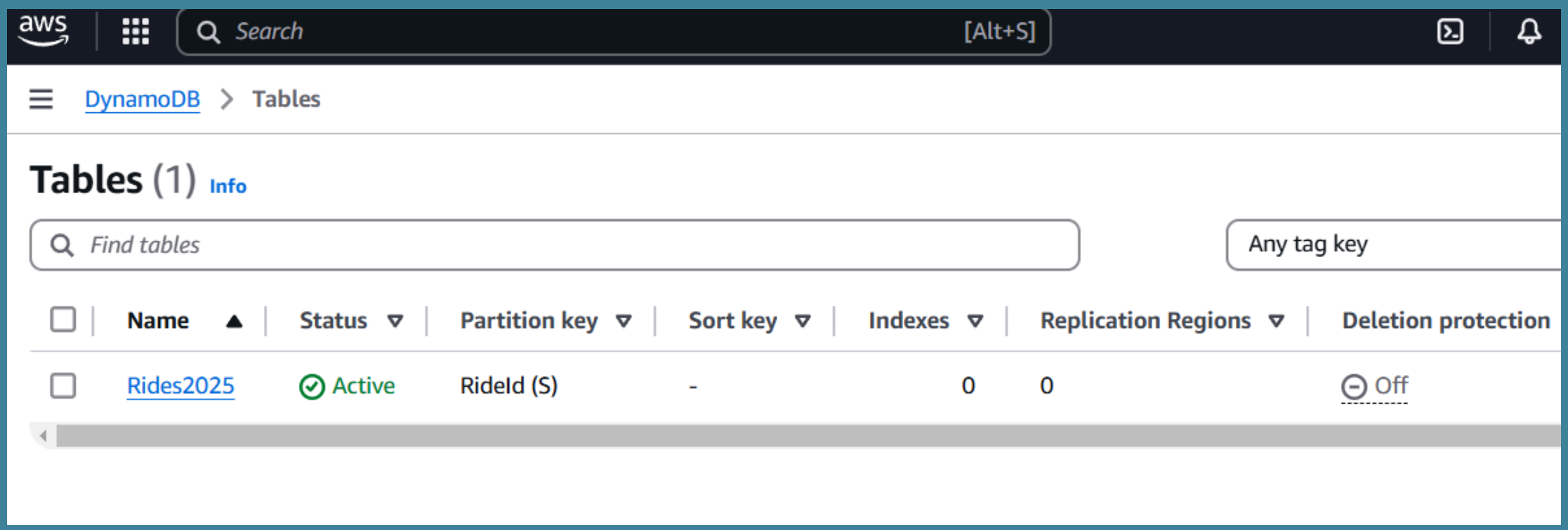




Kanika Mathur
github.com/KanikaGenesis

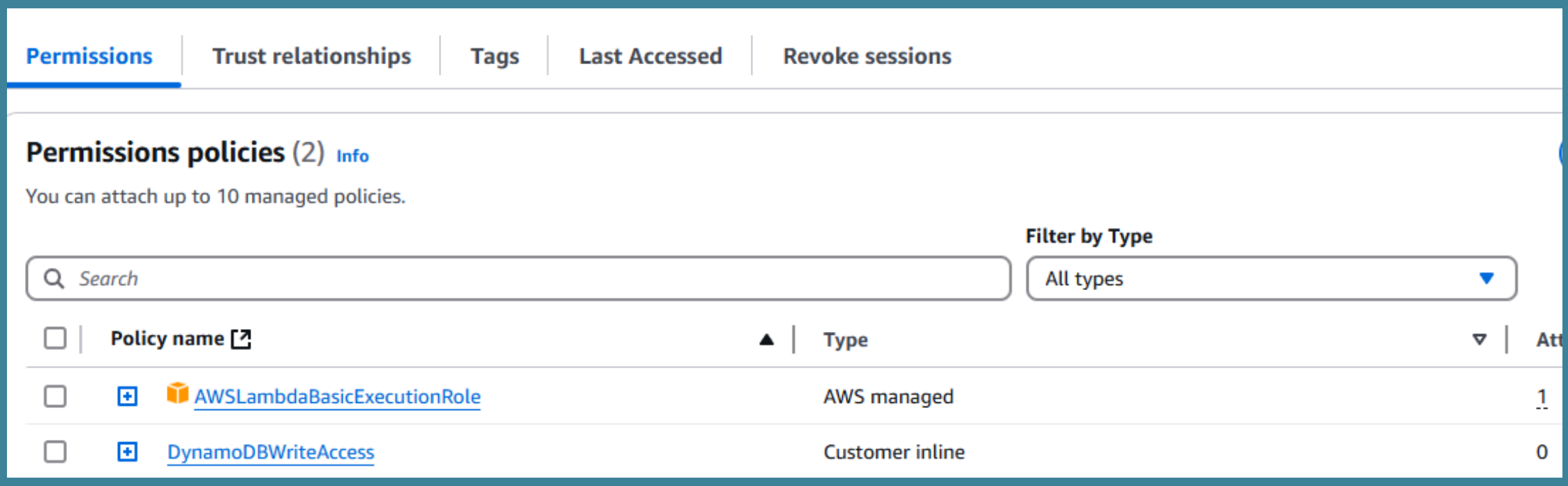
Developing the Backend with DynamoDB and Lambda

- Step 1: I created a **DynamoDB** table named **Rides2025** to store ride request data. I saved the ARN for the table, which was required for permissions configuration.

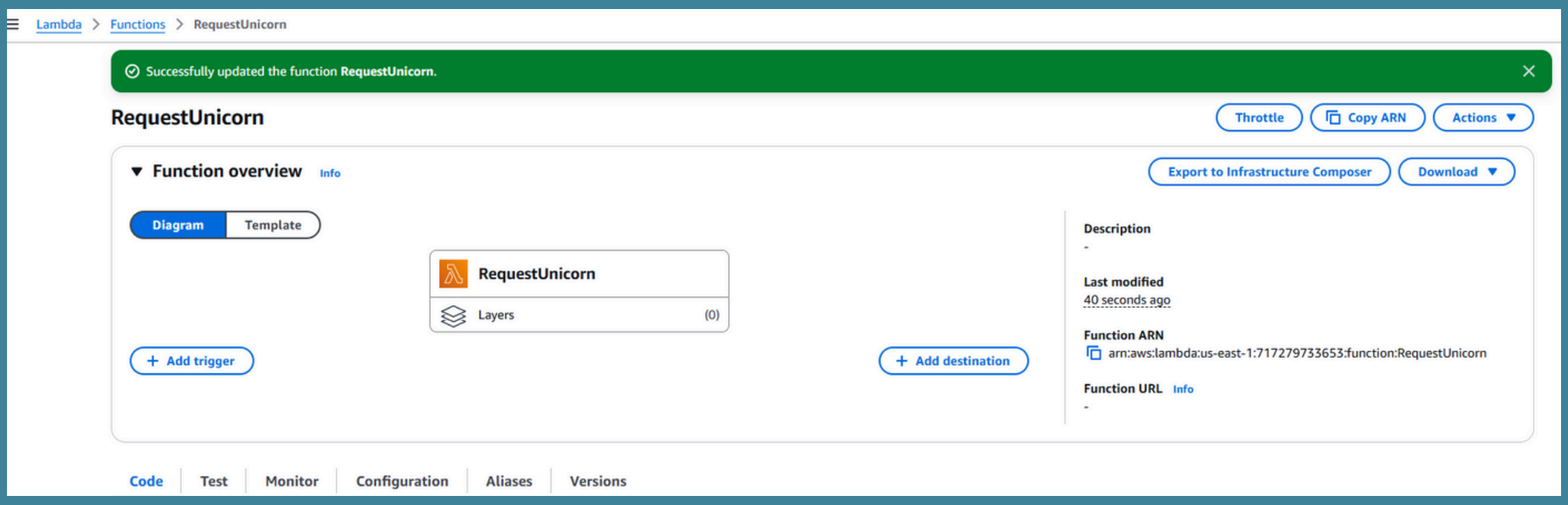


Kanika Mathur
github.com/KanikaGenesis

- Step 2: I set up an **IAM** role for Lambda with policies granting permissions to write to the DynamoDB table.



- Step 3: I created a **Lambda** function named **RequestUnicorn**, which handled all backend logic for processing ride requests. This included selecting a unicorn from a predefined list, capturing the user's request details (location, time, etc.), and writing this data to the DynamoDB table.



Kanika Mathur
github.com/KanikaGenesis

Configuring APIs with API Gateway

- Step 1: I set up a **REST API** in **API Gateway** to manage incoming requests, bridging the frontend and backend.

- Step 2: I implemented a resource path /ride under the API to serve as the endpoint for ride requests.

- Step 3: I configured a **Cognito-based Authorizer** to restrict API access to authenticated users, ensuring secure interaction with the application.

- Step 4: I created **POST** method for /ride resource responsible for invoking the RequestUnicorn Lambda function to pass data between the API Gateway and Lambda seamlessly.

- Step 5: I deployed the API to the dev stage, generating a public invoke URL. Added it to the config.js file in the application code.

Kanika Mathur
github.com/KanikaGenesis

APIs (1/1)

Find APIs

Name	Description	ID	Protocol	API endpoint type
WildRydes		k17kh1sg77	REST	Regional

Authorizers (1) Info

WildRydes

Authorizer ID
449fx0

Cognito pool
WildRydesUsers - c0qPLLaci (us-east-1)

Token source
Authorization

Token validation - optional
none

Resources

API actions | Deploy API

Create resource

- /
  - /ride
    - OPTIONS
    - POST

Resource details

Delete | Update documentation | Enable CORS

Path
/ride

Resource ID
ciolkf

Methods (2)

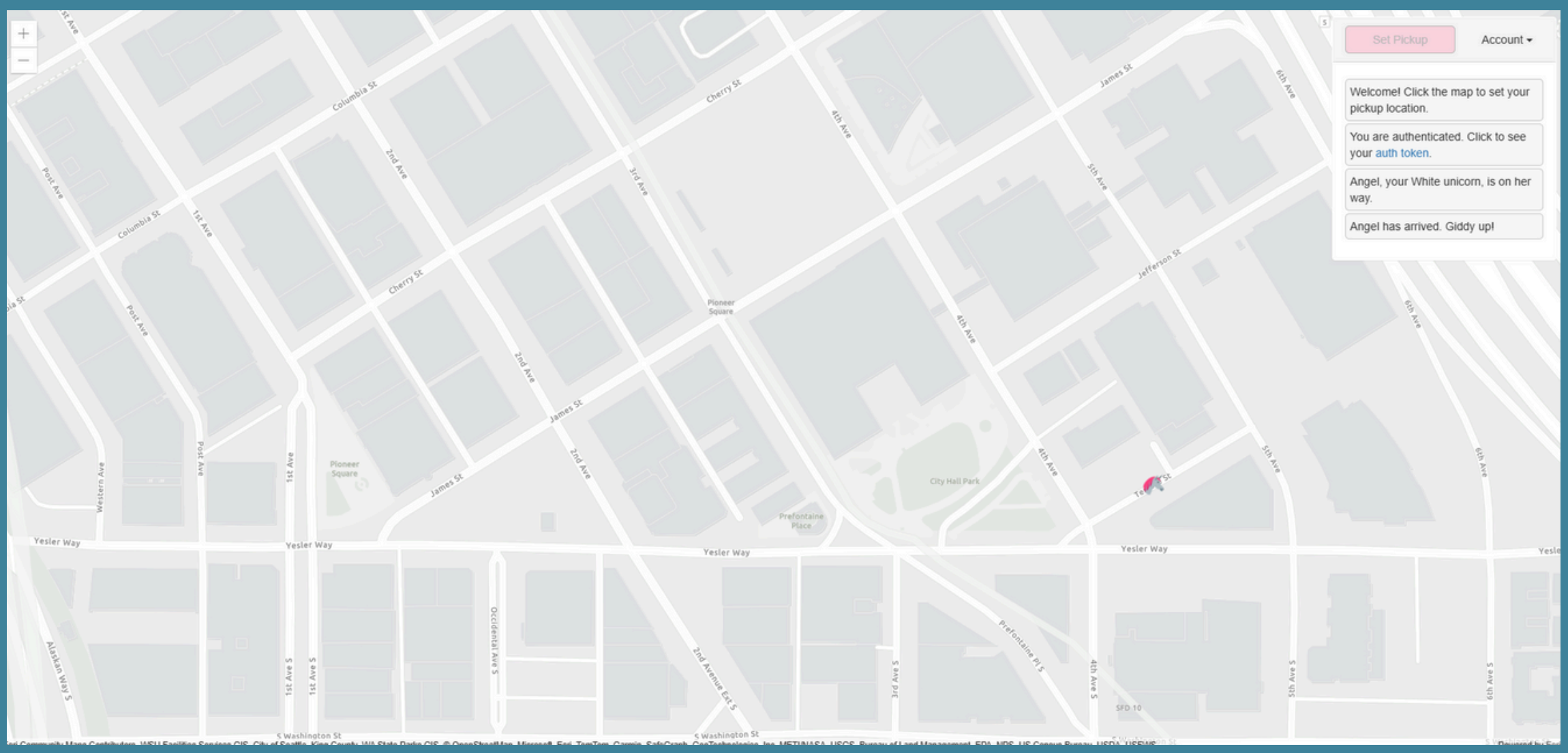
Delete | Create method

Method type	Integration type	Authorization	API key
OPTIONS	Mock	None	Not required
POST	Lambda	Cognito user pools	Not required

Kanika Mathur
github.com/KanikaGenesis

Testing Deployed App

Finally, I tested the workflow by logging in as a user, submitting a ride request from the frontend map, and verifying the expected response in the application. The API handled requests smoothly, triggering the Lambda function and updating the DynamoDB table with ride details.



Kanika Mathur
github.com/KanikaGenesis

Thank You