



Kanika Mathur  
[github.com/KanikaGenesis](https://github.com/KanikaGenesis)

# Wild Rydes Serverless Application



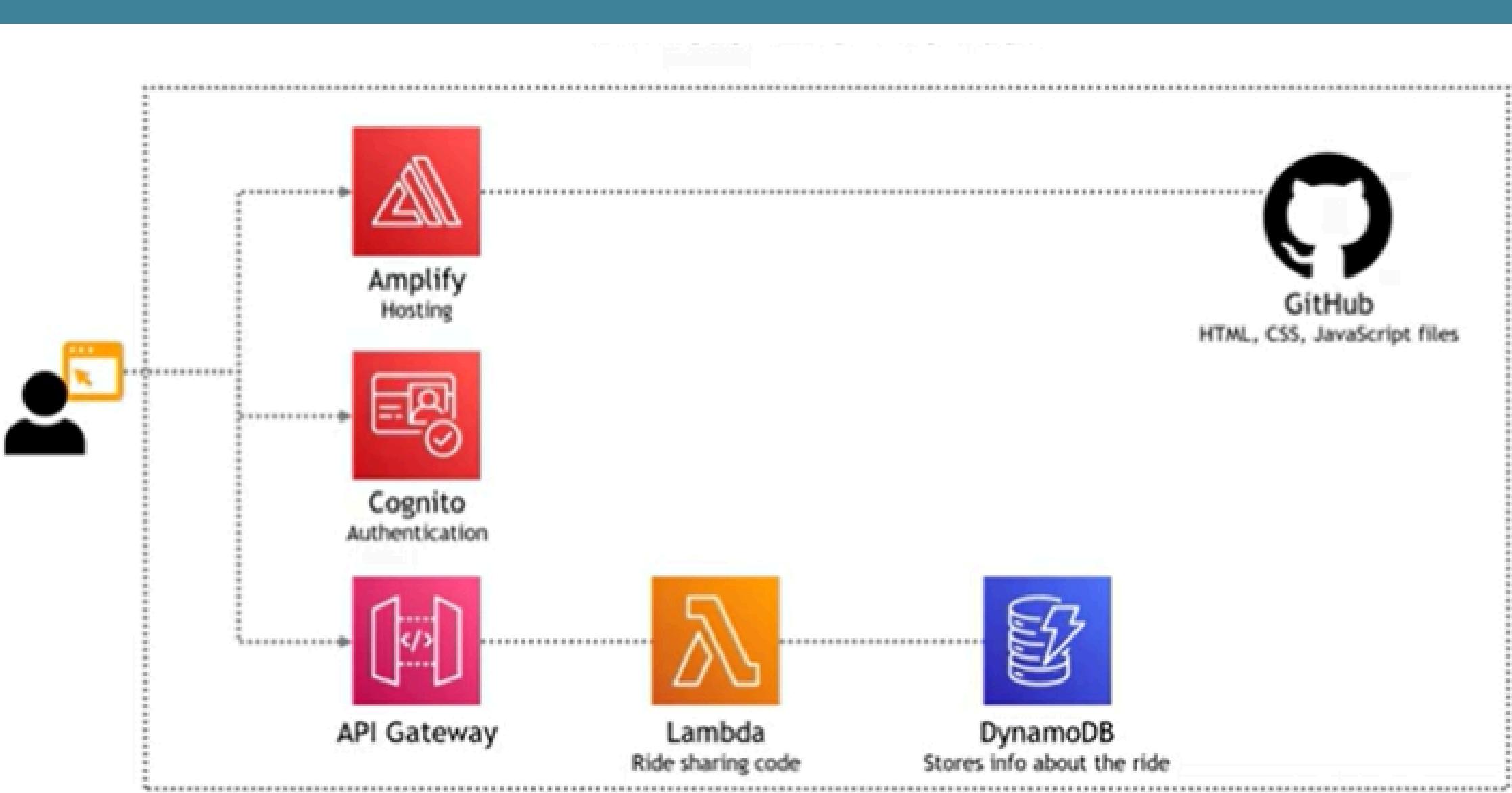


Kanika Mathur  
[github.com/KanikaGenesis](https://github.com/KanikaGenesis)

# 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.

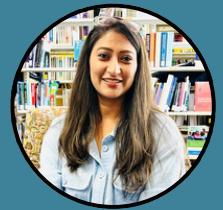




Kanika Mathur  
[github.com/KanikaGenesis](https://github.com/KanikaGenesis)

# 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 ▾

Support Docs

wildrydes-site

App ID: d8cjydf59wmga [Copy](#)

Branches 1  + Add branch

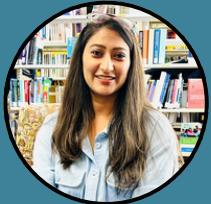
main ★ Production branch

Deployed [\(1\)](#)

Domain <https://main.d8cjydf59wmga.amplifyapp.com>

Last deployment 1 minute ago

Last commit Update index.html / wildrydes-site:main



Kanika Mathur  
[github.com/KanikaGenesis](https://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

[Alt+S] | [ ] | [ ] | [ ]

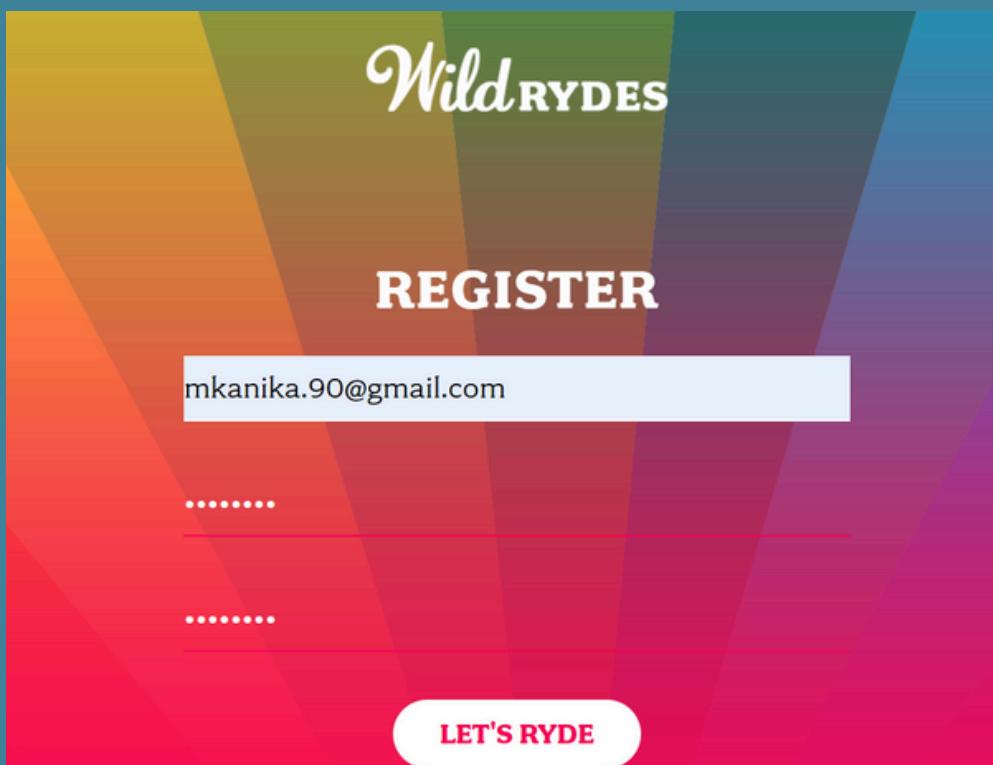
**i** New from Amazon Verified Permissions! Cognito user group authorization for API Gateway  
You can now create group-aware authorization policies for your APIs with Amazon Verified Permissions, a fine-grained authorization service for applications. [Learn more](#) [?]

### User pools (1) [Info](#)

View and configure your user pools. User pools are directories of federated and local user profiles. They provide authentication options for your users.

Search user pools by name or ID

User pool name	User pool ID	Created time
WildRydesUsers	us-east-1_c0qPLLaci	7 minutes ago



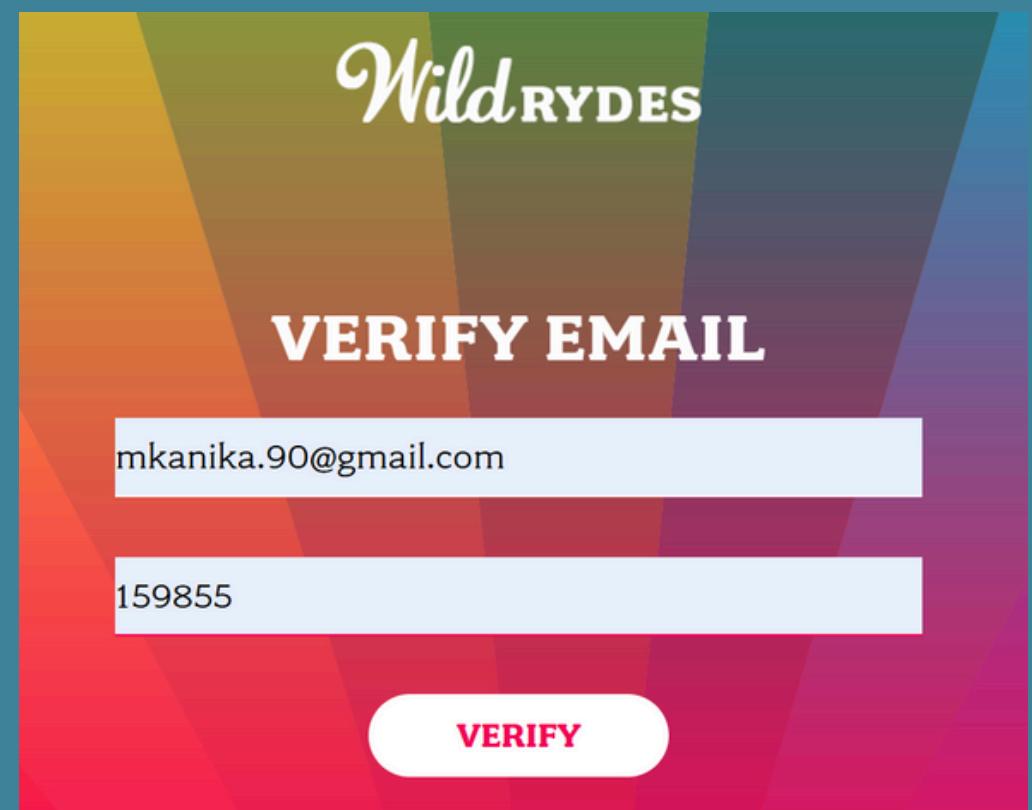
Wild RYDES

REGISTER

mkanika.90@gmail.com

.....  
.....

LET'S RYDE



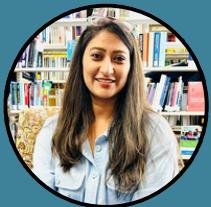
Wild RYDES

VERIFY EMAIL

mkanika.90@gmail.com

159855

VERIFY



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.

The screenshot shows the AWS DynamoDB console. The top navigation bar includes the AWS logo, a search bar, and a [Alt+S] keyboard shortcut. Below the navigation, the breadcrumb trail shows 'DynamoDB > Tables'. The main area displays a table titled 'Tables (1)'. The table has a header row with columns: Name, Status, Partition key, Sort key, Indexes, Replication Regions, and Deletion protection. A search bar labeled 'Find tables' and a filter labeled 'Any tag key' are also present. The table body contains one row for the 'Rides2025' table, which is listed as Active with a partition key of 'RidId (S)' and no sort key, indexes, or replication regions. Deletion protection is set to 'Off'.

	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection
<input type="checkbox"/>	<a href="#">Rides2025</a>	<span>Active</span>	RidId (S)	-	0	0	<span>Off</span>



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.

The screenshot shows the 'Permissions' tab selected in the AWS IAM console. Under 'Permissions policies (2)', it lists two policies: 'AWSLambdaBasicExecutionRole' (AWS managed) and 'DynamoDBWriteAccess' (Customer inline). A search bar and a filter dropdown are also visible.

Policy name	Type	Attributes
<a href="#">AWSLambdaBasicExecutionRole</a>	AWS managed	1
<a href="#">DynamoDBWriteAccess</a>	Customer inline	0

- 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.

The screenshot shows the 'RequestUnicorn' Lambda function details. It includes sections for 'Function overview', 'Layers' (empty), 'Triggers' (+ Add trigger), 'Destinations' (+ Add destination), 'Throttle', 'Copy ARN', 'Actions', 'Export to Infrastructure Composer', 'Download', 'Description', 'Last modified' (40 seconds ago), 'Function ARN' (arn:aws:lambda:us-east-1:717279733653:function:RequestUnicorn), and 'Function URL'. Navigation tabs at the bottom include Code, Test, Monitor, Configuration, Aliases, and Versions.



Kanika Mathur  
[github.com/KanikaGenesis](https://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
<a href="#">WildRydes</a>				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

**Path**  
/ride

[Delete](#)

[Update documentation](#)

[Enable CORS](#)

**Resource ID**  
ciolkf

### Methods (2)

[Delete](#)

[Create method](#)

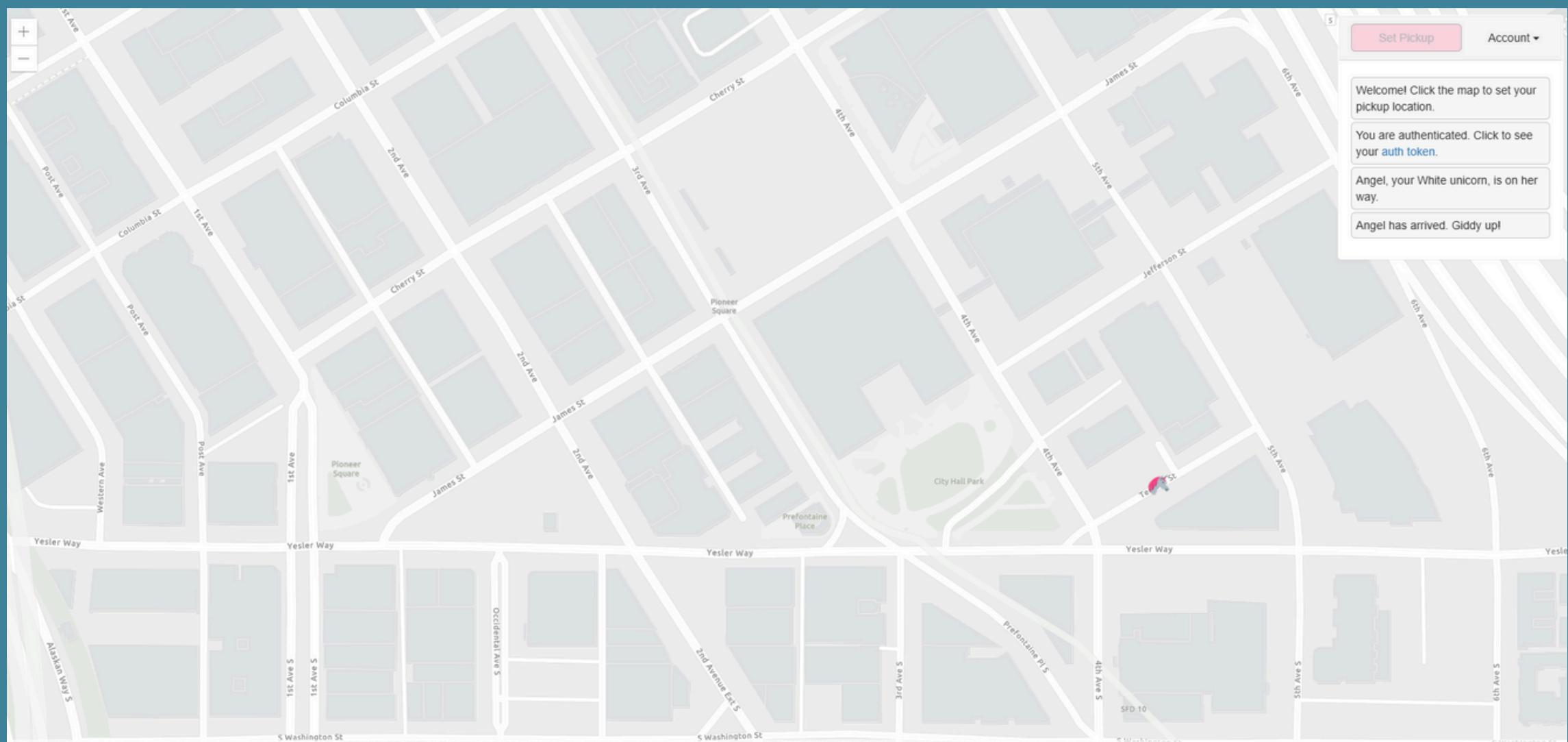
Method type	▲	Integration type	▼	Authorization	▼	API key	▼
<a href="#">OPTIONS</a>		Mock		None		Not required	
<a href="#">POST</a>		Lambda		Cognito user pools		Not required	



Kanika Mathur  
[github.com/KanikaGenesis](https://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](https://github.com/KanikaGenesis)

# Thank You

