

CAPSTONE PROJECT

CLOUD-POWERED TRAVEL DIARIES:

A FULL-STACK AMPLIFY REACT APPLICATION

- SUBMITTED BY: AYUSHI NAGPURE, DHANASHREE GIRIYA, NUPUR KIRWAI
- GUIDED BY: DR. PRADNYA BORKAR



CONTENT

1. Introduction
2. Objectives
3. System Design & Architecture
4. Technical Stack
5. Implementation
6. Results & Analysis
7. Conclusion & Future Work



INTRODUCTION

PROBLEM STATEMENT:

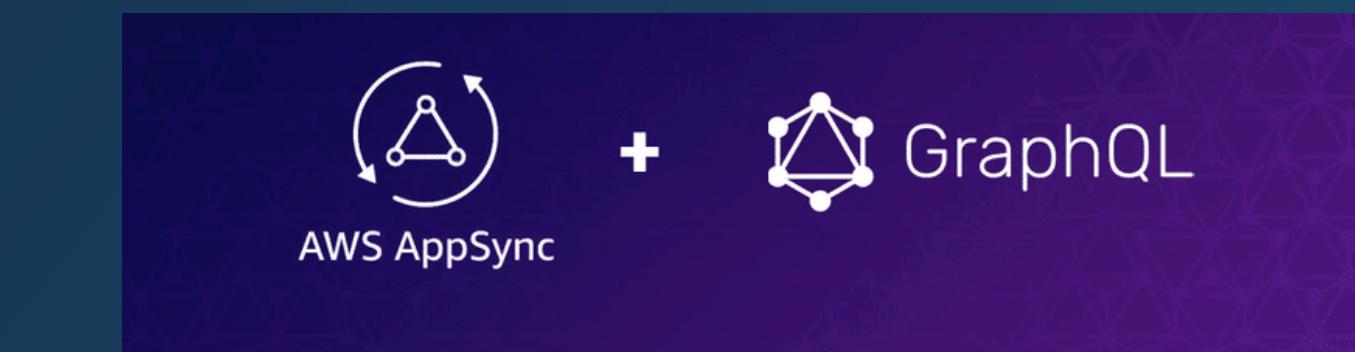
- Traditional web apps require heavy backend setup, leading to high costs and complexity.

SOLUTION:

- Use of AWS Amplify for a serverless, cloud-based travel diary app.

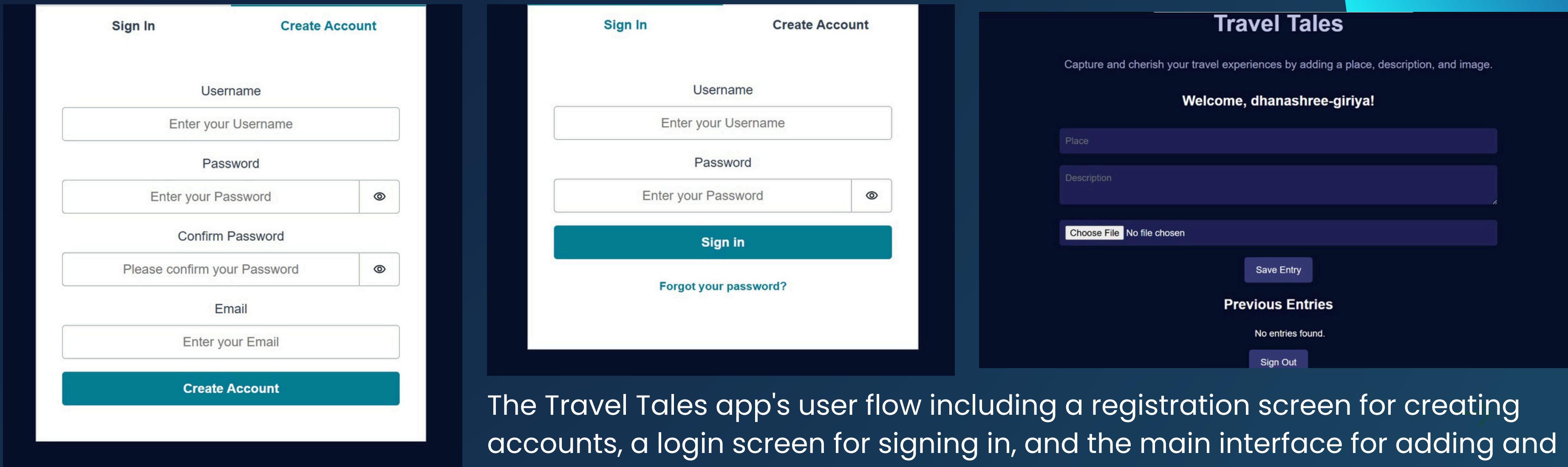
KEY FEATURES:

- Authentication: Amazon Cognito
- Database: DynamoDB
- Storage: Amazon S3
- API: GraphQL with AWS AppSync



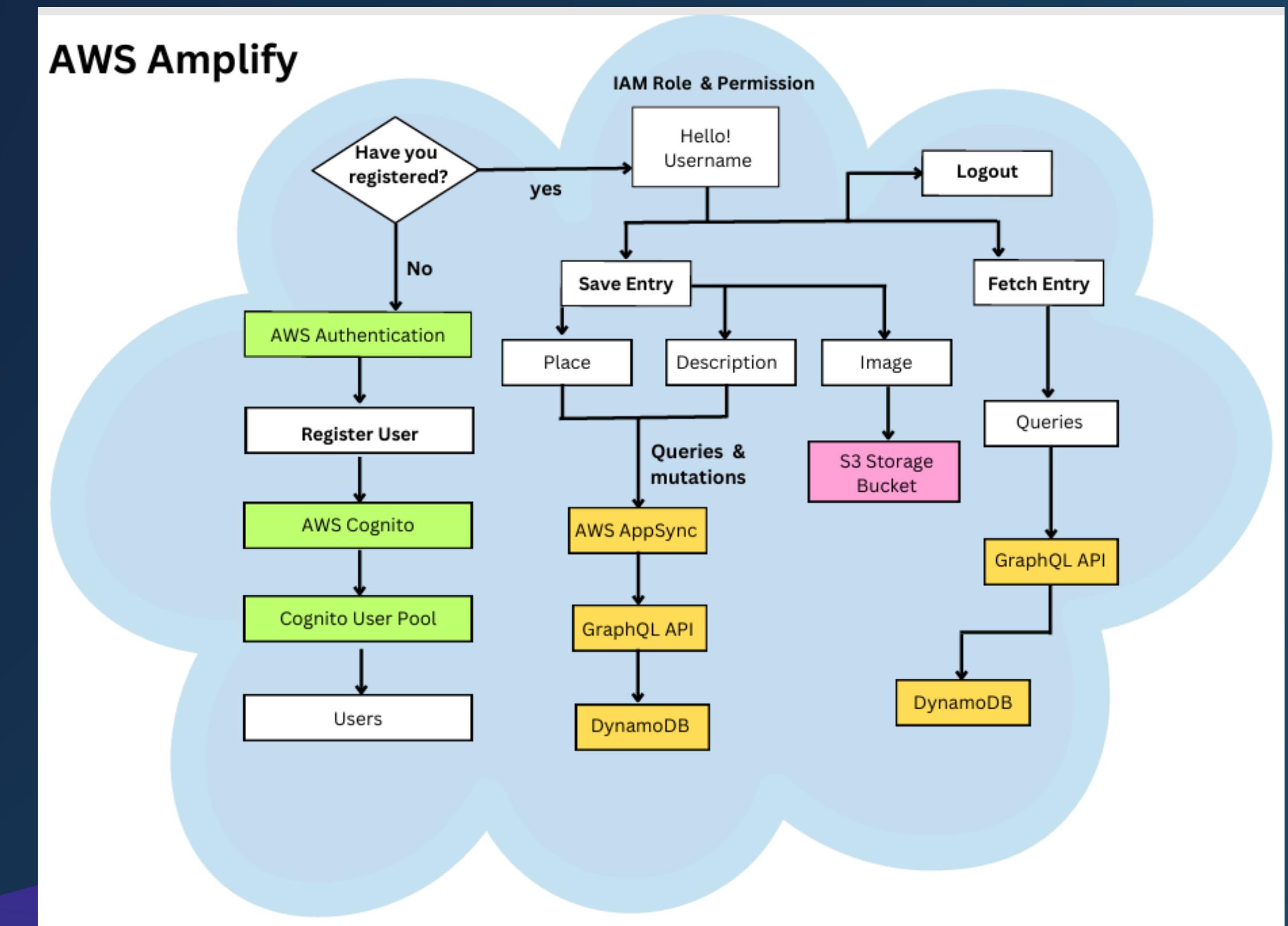
OBJECTIVES

- To build a full-stack travel journal using AWS Amplify.
- To leverage cloud services for scalability, security, and cost-efficiency.
- To simplify backend management with pre-configured services.
- To provide a user-friendly React frontend with real-time updates.



SYSTEM DESIGN & ARCHITECTURE

- Architecture: Serverless, cloud-native
- Components:
 - Frontend: React.js
 - Backend: AWS Amplify
 - Database: DynamoDB
 - Storage: Amazon S3
 - Authentication: Amazon Cognito



TECHNICAL STACK

Frontend :

- [React.js](#):
A JavaScript library for building dynamic and responsive user interfaces.

Backend :

- [AWS Amplify](#):
Provides pre-configured backend services like authentication, database, and storage.

Database :

- [Amazon DynamoDB](#):
A NoSQL database for storing travel entries.

Storage :

- [Amazon S3](#):
Scalable cloud storage for user-uploaded images.

Authentication :

- [Amazon Cognito](#):
Manages user authentication (sign-up, login, session management).

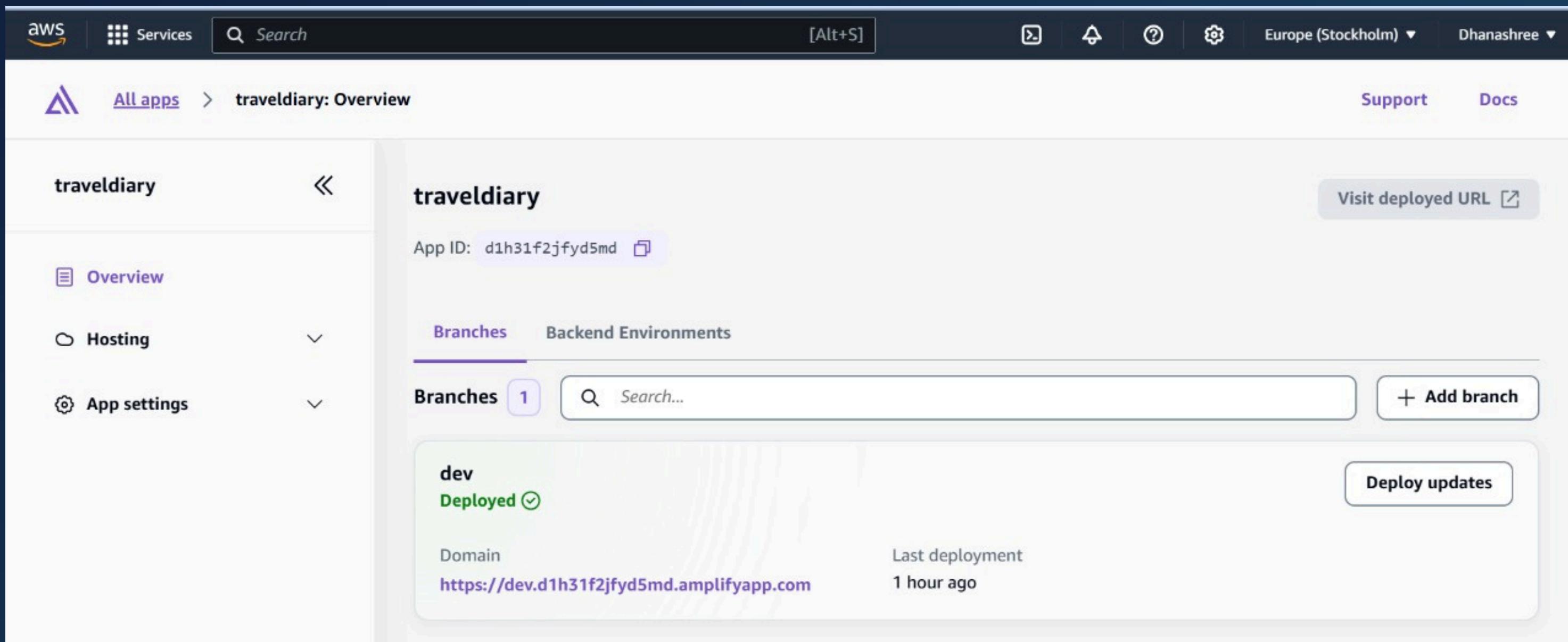
API :

- [AWS AppSync \(GraphQL\)](#):
Enables real-time data synchronization between frontend and backend.

IMPLEMENTATION

- AWS Amplify Setup:

AWS Amplify Hosting : Automated deployment and hosting of the React app.



- Amazon S3 for Image Storage :

Scalable storage for user-uploaded images.

The screenshot shows the AWS S3 console interface. The left sidebar is titled "Amazon S3" and includes sections for "General purpose buckets", "Multi-Region Access Points", "Batch Operations", "IAM Access Analyzer for S3", "Block Public Access settings for this account", "Storage Lens", "Dashboards", "Storage Lens groups", and "AWS Organizations settings". The main content area is titled "Objects (6)" and displays a list of six PNG files. The table columns are "Name", "Type", "Last modified", "Size", and "Storage class". The objects listed are:

Name	Type	Last modified	Size	Storage class
14420.png	png	UU:U1:16 (UTC+05:30)		
1742324173597-14420.png	png	March 19, 2025, 00:26:16 (UTC+05:30)	115.3 KB	Standard
1742367013118-14420.png	png	March 19, 2025, 12:20:17 (UTC+05:30)	115.3 KB	Standard
1742449209103-14420.png	png	March 20, 2025, 11:10:13 (UTC+05:30)	115.3 KB	Standard
1742449211666-14420.png	png	March 20, 2025, 11:10:13 (UTC+05:30)	115.3 KB	Standard

- DynamoDB for Database Management:

NoSQL database for storing travel entries.

The screenshot shows the AWS DynamoDB console interface. The left sidebar has a navigation menu with options like Dashboard, Tables, Explore items (which is selected), PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations (New), Reserved capacity, and Settings. Below this is a DAX section with Clusters, Subnet groups, Parameter groups, and Events. The main content area shows the 'Tables (1)' section with a search bar for tag keys and values, and a 'Find tables' input field. A table named 'TravelEntry-ubcdqxuejjcl3pvyehgcak2qua-dev' is listed. The main panel displays the table details for 'TravelEntry-ubcdqxuejjcl3pvyehgcak2qua-dev'. It includes an 'Autopreview' toggle (which is on), a 'View table details' button (which is highlighted in blue), and a 'Scan or query items' section with a note about consumed read capacity units. Below this is a green notification box stating 'Completed. Read capacity units consumed: 2'. The 'Items returned (2)' section shows a table with two items. The columns are labeled 'id (String)', 'description', 'image', and 'name'. Item 1 has a value of 2 for 'id', 'malls' for 'description', '14420.png' for 'image', and 'pune' for 'name'. Item 2 has a value of 1 for 'id', 'orange city' for 'description', '14420.png' for 'image', and 'ngp' for 'name'. There are 'Actions' and 'Create item' buttons at the top of this table view. The bottom of the page includes standard AWS footer links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences, along with a copyright notice for 2025.

	id (String)	description	image	name
<input type="checkbox"/>	2	malls	14420.png	pune
<input type="checkbox"/>	1	orange city	14420.png	ngp

- AWS AppSync for GraphQL API : GraphQL API for real-time data synchronization.

The screenshot shows the AWS AppSync Data sources page. The left sidebar lists options like APIs, Schema, Data sources, Functions, Queries, Caching, Settings, Monitoring, and Custom domain names. The 'Data sources' option is selected. The main area displays a table titled 'Data sources (2)' with columns for Name, Type, and Resource. The first entry is 'NONE_DS' (Type: NONE). The second entry is 'TravelEntryTable' (Type: AMAZON_DYNAMODB), which is associated with the resource 'TravelEntry-ubcdqxuejjcl3pvyehgcak2qua-dev'.

Name	Type	Resource
NONE_DS	NONE	
TravelEntryTable	AMAZON_DYNAMODB	TravelEntry-ubcdqxuejjcl3pvyehgcak2qua-dev

- Amazon Cognito for Authentication : Secure user authentication and session management.

The screenshot shows the Amazon Cognito User pools page. The left sidebar includes sections for Current user pool (traveldiary9b77b31d_us...), Applications (App clients, New), User management (Users, Groups), Authentication (Authentication methods), and Sign-in (New). The 'Users' section is selected. The main area displays a table titled 'Users (5)' with columns for User name, Email address, Email verified, Confirmation status, and Status. All users listed are confirmed and enabled.

User name	Email address	Email verified	Confirmation status	Status
aachal	dhanashreegiriya2...	Yes	Confirmed	Enabled
dhanashree	coordinator01080...	Yes	Confirmed	Enabled
dhanashree-giriya	dhanashreegiriya2...	Yes	Confirmed	Enabled
ayushi	ayushinagpure9@...	Yes	Confirmed	Enabled
pradnya	pradnya.borkar@s...	Yes	Confirmed	Enabled

ANALYSIS

- **Scalability** : Serverless architecture allows auto-scaling.
- **Security** : AWS Cognito and IAM policies ensure secure access.
- **Cost Efficiency** : Pay-as-you-go model reduces operational costs.
- **User Experience** : React-based UI provides a smooth experience.

COMPARISON WITH TRADITIONAL WEB APPS

Feature	Traditional Web App	AWS Amplify-Based App
Backend Setup	Manual	Pre-configured
Authentication	Custom-built	AWS Cognito
Database Management	Requires setup	DynamoDB (Auto-scaled)
Image Storage	External service needed	AWS S3 (Integrated)
Scalability	Limited	High (Serverless)
Deployment	Manual	Automated with AWS Hosting

CONCLUSION

Key Takeaways :

- Serverless architecture reduces infrastructure management.
- Cloud-native solutions enhance scalability, security, and cost-efficiency. Successfully built a full-stack travel diary application using AWS Amplify and React.js.
- Demonstrated the effectiveness of cloud-native, serverless architecture in modern web development.

Future Work :

- **Offline Functionality:**
Allowing users to add and store travel entries without an internet connection.
- **Location Tagging:**
Automatically tagging travel entries with location data for better trip reporting.
- **Social Media Integration:**
Enabling users to share their travel experiences directly from the app.
- **AI-Based Recommendations:**
Using AI to suggest travel destinations based on user preferences.
- **Mobile Compatibility:**
Developing a mobile version using React Native for broader accessibility.
- **Collaboration Features:**
Allowing multiple users to contribute to a shared travel diary.

THANK YOU