OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Introduction to Cloud Computing – Mr. Schön – SoSe23

Assignment for

Introduction to Cloud Computing

Weather Report implemented on AWS

Advisor: Maximilian Schön
Students: Nguyễn Quốc Minh Thư - 3397231.

REGENSBURG, April 2023

Contents

1 Requirement Elicitation

1.1 Project Identify

Requirement: Weather Report implemented on AWS

Name	**Weather Report**
Description	• Goal: create a simple Web frontend to enable users to retrieve and display weather data from locations • Use a free-to-use weather services available on the Internet, e.g. https://openweathermap.org/ • Create a simple Web Site in the Cloud where a user can enter a location and the corresponding weather current information and weather forecast is displayed • Implement a connection in the Web Site to the weather service • Create a Cloud database and store all retrieved weather information in a cloud database • Your web page should display the current and also previous weather information stored in the database • **optional:** With a service like "Alexa" or comparable the web page is able to handle spoken user input
Implementation hints	Possible Cloud Services to be used (examples) • Virtual machine • Object Storage • API Gateway • Serverless Functions • PaaS Database • **optional:** Alexa Skills Kit

♣ IDENTIFY THE CONTEXT OF THIS PROJECT:

The project involves creating a simple web frontend that allows users to retrieve and display weather data from locations using a free-to-use weather service available on the Internet, such as OpenWeatherMap. The web frontend will be hosted on a cloud-based website and will have a connection to the weather service to retrieve and store weather information in a cloud database. The web page will also have the capability to display both current and previous weather information stored in the database. Optionally, the web page may also be integrated with a voice recognition service like "Alexa"or similar for spoken user input.

♣ WHO ARE RELEVANT SKATEHOLDERS ?

- Users: The primary stakeholders of this app would be the users who are interested in checking weather information for different locations.
- Weather Service Provider (e.g., OpenWeatherMap): The weather service provider would be a relevant stakeholder as the app would be using their API to retrieve weather data.
- AWS Cloud Database Provider: The cloud database provider would be a relevant stakeholder as the app would be storing weather information in a cloud database.

♣ WHAT ARE THEIR CURRENT NEEDS ?

User Stories:

- **Users:** Users need a simple and user-friendly web frontend that allows them to easily enter a location and retrieve current and forecasted weather information. They may also need access to previous weather information for reference.
- **Weather Service Provider:** The weather service provider may need to ensure that their API is properly integrated into the web frontend and that the data retrieved from their service is accurate and up-to-date.
- **Cloud Database Provider:** The cloud database provider may need to ensure that the database is properly set up and configured to securely store and retrieve weather information.

♣ IN YOUR OPINION, WHAT BENEFITS OF UWC 2.0 WILL BE FOR EACH STAKEHOLDER?

- **For User:**
  1. Users will benefit from a simple and convenient way to retrieve and display weather information for different locations.
  2. Get the update status faster including access to previous weather data for reference.
- **For Weather Service Provider:**
  1. The weather service provider may benefit from increased usage of their API through integration with the web frontend.
  2. Potentially leading to increased brand exposure and revenue.
- **For Cloud Database Provider:**
  1. The cloud database provider may benefit from increased usage of their database service as weather information is stored and retrieved from the cloud database.
  2. Gain more subcription in the future.

1.2 Describe System

Requirement: Describe all functional and non-functional requirements that can be inferred from the project description. Draw a use-case diagram for the whole system.

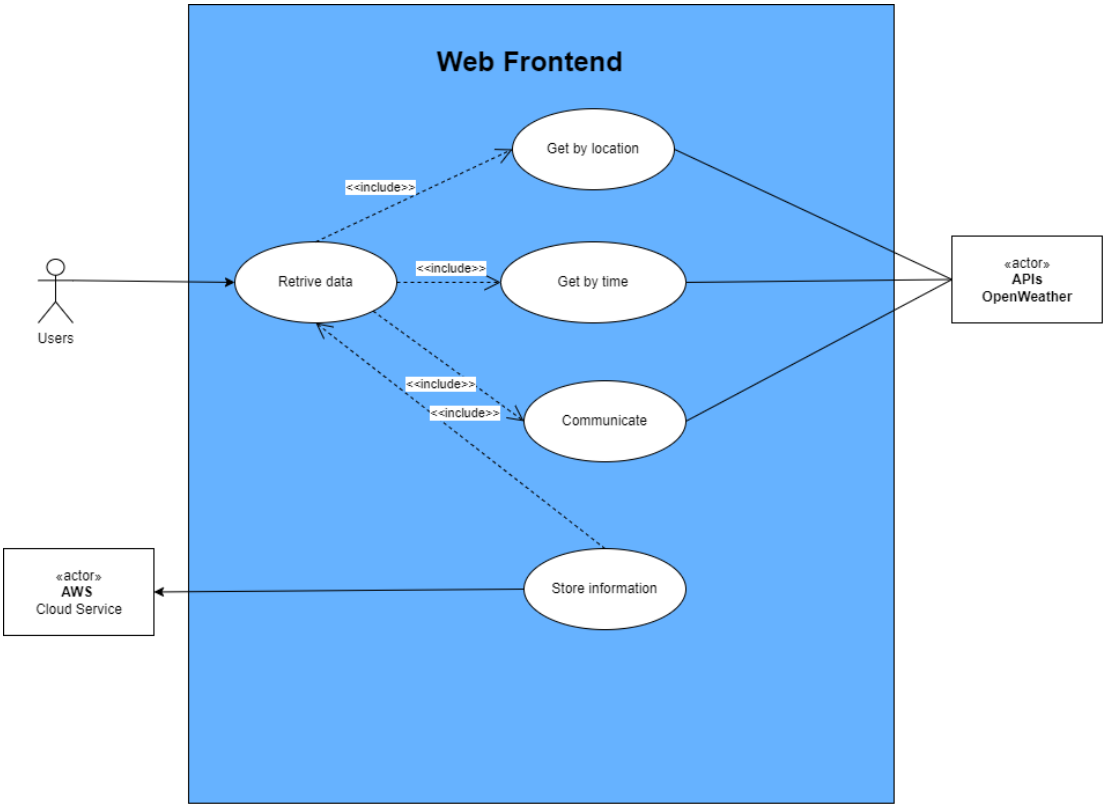
♣ FUNCTIONAL REQUIREMENTS:

1. **User Input:** The web frontend should allow the user to input a location, either manually or through voice recognition (if voice input is implemented), to retrieve weather information.
2. **Weather Data Retrieval:** The web frontend should connect to the weather service provider's API and retrieve weather data for the entered location, including current weather information and weather forecast.
3. **Weather Data Storage:** The web frontend should store the retrieved weather data in a cloud database for future reference, including current and previous weather information.
4. **Weather Data Display:** The web frontend should display the retrieved weather information to the user in a user-friendly format, including current weather conditions, temperature, humidity, wind speed, and weather forecast.
5. **Previous Weather Information:** The web frontend should allow the user to view previously stored weather information from the cloud database, including historical weather data for a specific location and date.

♣ NON-FUNCTIONAL REQUIREMENTS:

- **Performance requirements:** The web frontend should have fast response times and be able to handle a large number of concurrent user requests, ensuring that the weather information is retrieved and displayed quickly and efficiently.
- **Reliability requirements:** The web frontend should be reliable and available for users to access weather information at any time without frequent downtime or system failures.
- **Security requirements** The web frontend should implement appropriate security measures to protect user data, including encryption of sensitive information such as location data and secure storage of retrieved weather information in the cloud database.
- **Scalability requirements** The web frontend should be designed to handle potential future expansions or updates, such as adding new weather service providers or integrating with additional voice recognition services.

♣ USE-CASE DIAGRAM FOR THE WHOLE SYSTEM:



Retrieve Weather Information

Use Case Name	Retrieve Weather Information
Description	The user interacts with the web frontend to enter a location and retrieve weather information.
Actor(s)	User
Trigger	Users type on the "Get by place"box.
Main flow	1. User enters a location on the web frontend. 2. Web frontend sends a request to the weather service provider's API to retrieve weather data for the entered location. 3. Weather service provider's API responds with the requested weather data. 4. Web frontend displays the retrieved weather information to the user.

Display Previous Weather Information

Use Case Name	**Display Previous Weather Information**
Description	The user interacts with the web frontend to view previously stored weather information from the cloud database.
Actor(s)	User
Trigger	Users type on the "Get by time"box.
Main flow	1. User requests to view previous weather information on the web frontend.. 2.Web frontend retrieves previous weather data from the cloud database. 3. Web frontend displays the retrieved previous weather information to the user.

Store Weather Information

Use Case Name	**Store Weather Information**
Description	The web frontend stores the retrieved weather information in a cloud database for future reference.
Actor(s)	Web frontend, Cloud Database
Trigger	Users clicks on the "Assign to janitors"button.
Main flow	1. Web frontend receives weather data from the weather service provider's API. 2.Web frontend stores the weather data in the cloud database. 3. Cloud database confirms successful storage of the weather data.

Handle Spoken User Input (Optional)

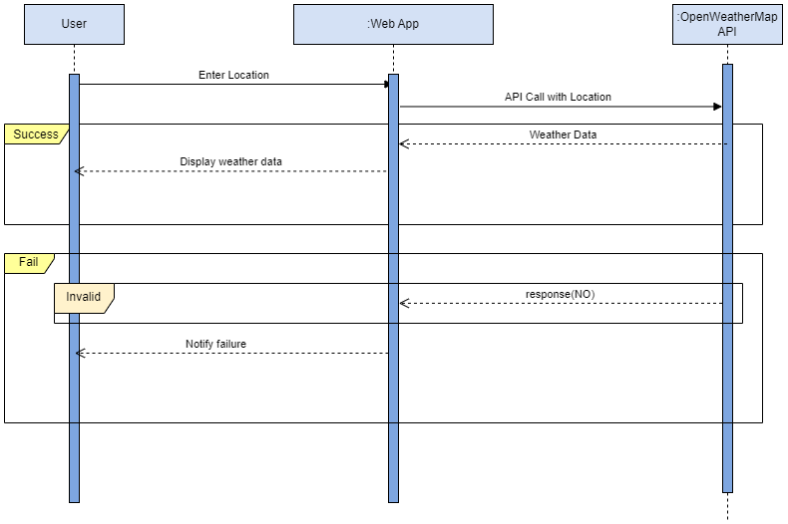
Use Case Name	**Retrieve Weather Information**
Description	The web frontend integrates with a voice recognition service to handle spoken user input for location entry.
Actor(s)	User, Voice Recognition Service (e.g., "Alexa")
Trigger	Users use the voice searching function.
Main flow	1. User provides spoken input for location on the web frontend through the voice recognition service. 2.Voice recognition service converts spoken input into text. 3.Web frontend processes the converted text input as if it was entered manually. 4.Web frontend retrieves and displays weather information based on the processed text input.

2 System modelling

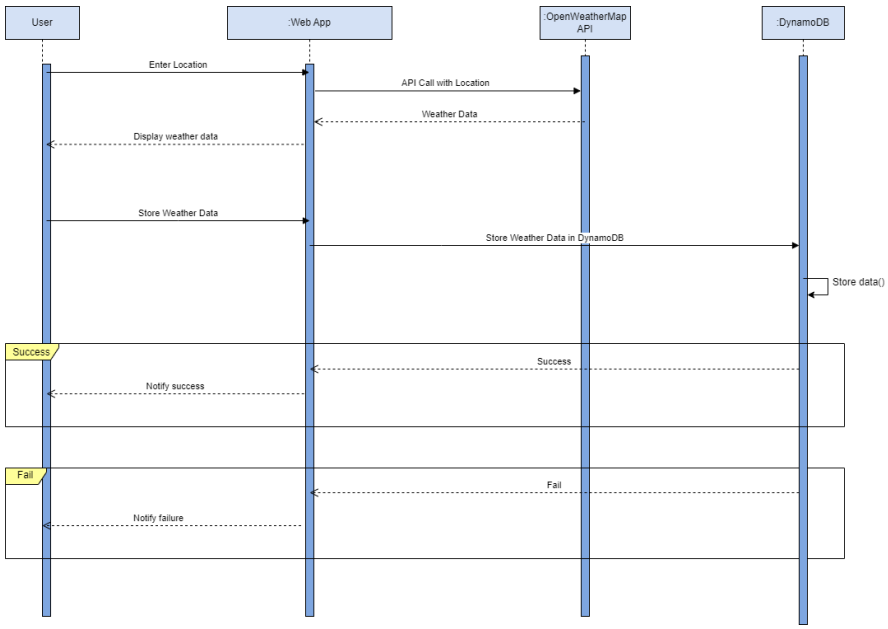
2.1 Sequence Diagram

In Brief: Show the sequence of some basic functionality of the weather web app.

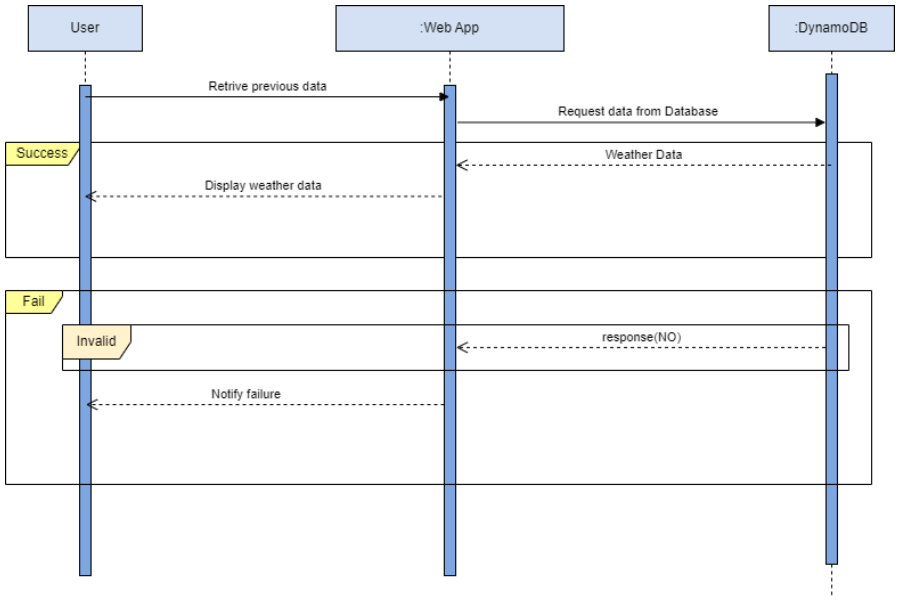
2.1.1 Searching for weather data:



2.1.2 Store weather data to database:



2.1.3 Retrieve previous stored weather data from database:



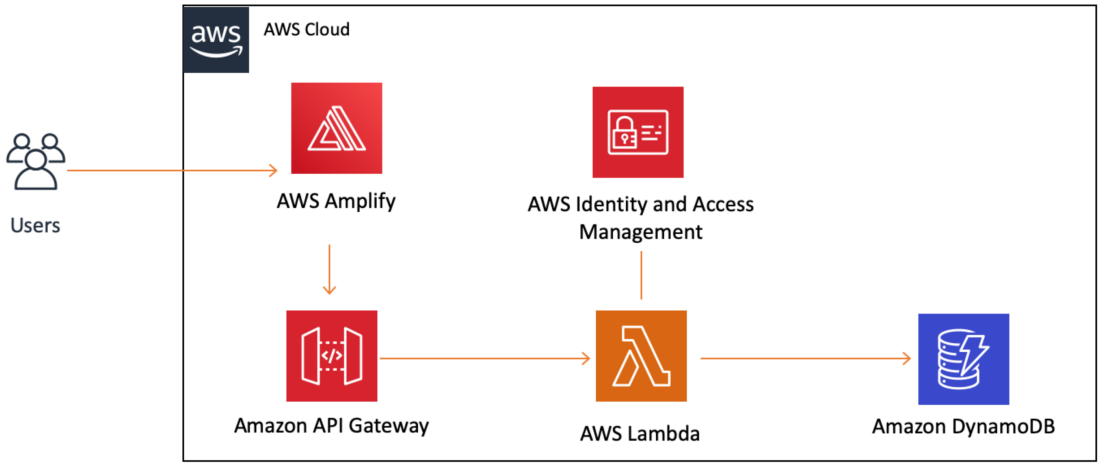
2.2 Detailed description :

1. **User:** The user interacts with the web app by entering a location (e.g., city name or zip code).
2. **Web App:** The web app receives the user's input and initiates the weather retrieval process.
3. **OpenWeatherMap API:** The web app makes an API call to the OpenWeatherMap API with the entered location to retrieve current weather information.
4. **DynamoDB:** The web app stores the retrieved weather data in a DynamoDB NoSQL database on AWS, and can also retrieve previous weather data from DynamoDB.
5. **Display Weather Data:** The web app displays the retrieved weather data on the web page for the user to see, including the current temperature, weather description, and humidity.
6. **Store Weather Data:** The web app stores the retrieved weather data in DynamoDB, including the location, temperature, weather description, and humidity as attributes.
7. **Display Previous Weather Data:** The web app can also retrieve and display previous weather data from DynamoDB, allowing the user to view historical weather search for a location.

3 Architecture design

3.1 AWS Cloud Services Architectural Approach

In Brief: Describe an architectural approach of all the AWS cloud services that I will use to implement the desired system.



Hình 1: Application architecture

♣ **Static Web Hosting:** **AWS Amplify** hosts static web resources including HTML, CSS, JavaScript, and image files which are loaded in the user's browser.

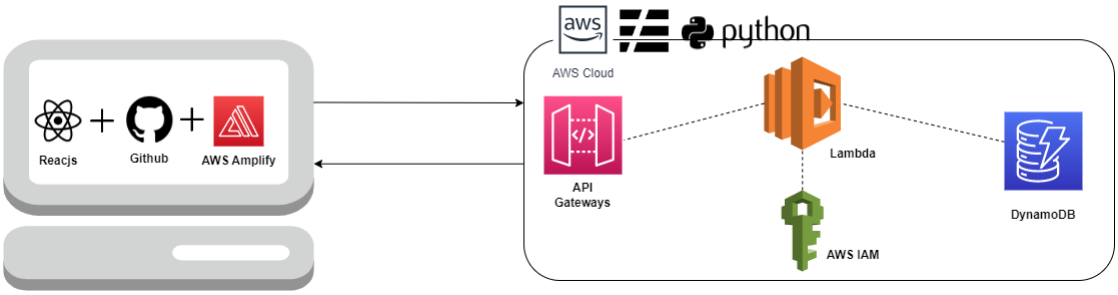
♣ **Securely manage identities and resources:** **AWS Identity and Access Management (IAM)** specify who or what can access services and resources in AWS, centrally manage fine-grained permissions, and analyze access to refine permissions across AWS.

♣ **Serverless Backend:** **Amazon DynamoDB** provides a persistence layer where data can be stored by the API's Lambda function.

♣ **RESTful API:** JavaScript executed in the browser sends and receives data from a public backend API built using **Lambda and API Gateway.**

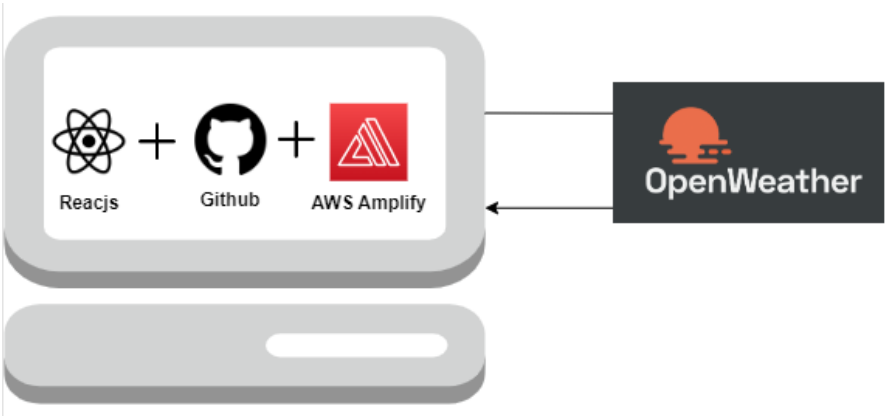
3.2 Full-stack Architecture Approach

In Brief: Describe an architectural approach that I will use to implement the desired system.



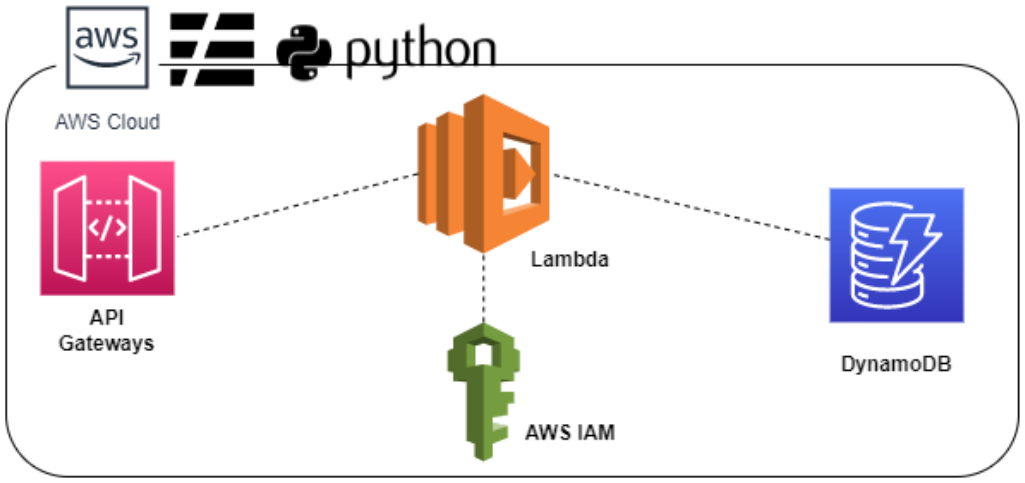
Hình 2: Application architecture

1. **Technologies Used:** The technologies used in the project, including:
   - Front-end: ReactJS with Bulma Library, GitHub, AWS Amplify, OpenWeatherAPI
   - Back-end: Python functions in AWS Lambda, API Gateways, DynamoDB, IAM
2. **Front-end Architecture:** Include an architecture diagram of the front-end system, showing how ReactJS with Bulma Library and AWS Amplify are used.And get the weather Data from OpenWeatherMap API.



Hình 3: Application architecture

3. **Back-end Architecture:** Include an architecture diagram of the back-end system, showing how Python functions in AWS Lambda, API Gateways, DynamoDB, and IAM are used.



Hình 4: Application architecture

In this chapter, provide a detailed description of the implementation of the system.

4. **Front-end Implementation:** In this section, the implementation of the front-end system, including:
   - ReactJS with Bulma Library was used to create the front-end
   - AWS Amplify and Github was used to deploy and manage the front-end

5. **Back-end Implementation:** In this section, the implementation of the back-end system, including:
   - Python functions were used to implement the back-end
   - API Gateways were used to manage API requests
   - DynamoDB was used to store weather data
   - IAM was used to manage access to the system

4 Implementation

4.1 Setting up.

Requirement: Creating an online repository (github, bitbucket, etc) for version control.

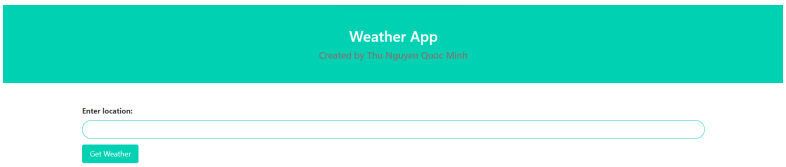
Link Github: WEATHER-APP-AWS-OTH-S23.

4.2 Frontend

Requirement: Implement– design an interface of either a Desktop-view central dashboard .

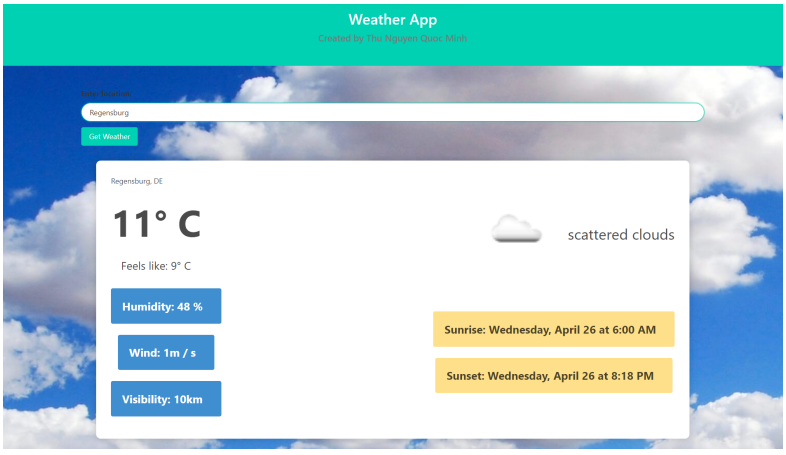
Link Amplify deploy app: WEATHER APP REACT AWS S23 OTH.

4.2.1 Welcome page



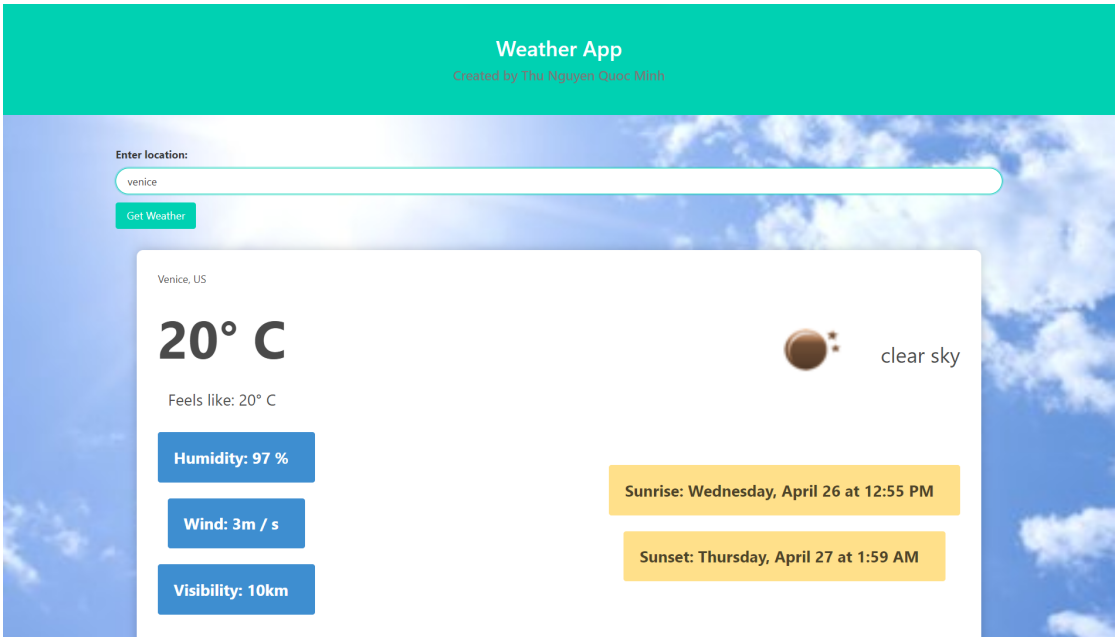
Hình 5: Started Page

4.2.2 Information page



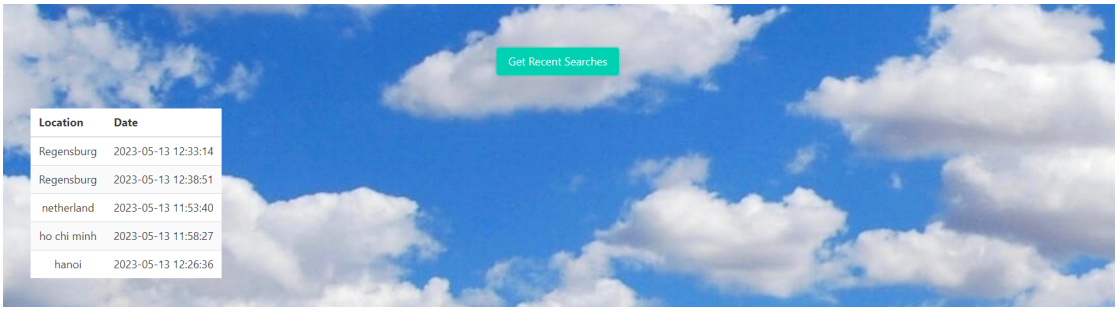
Hình 6: Cloudy weather information

4.2.3 Information page



Hình 7: Sunny weather Information

4.2.4 Get data from database

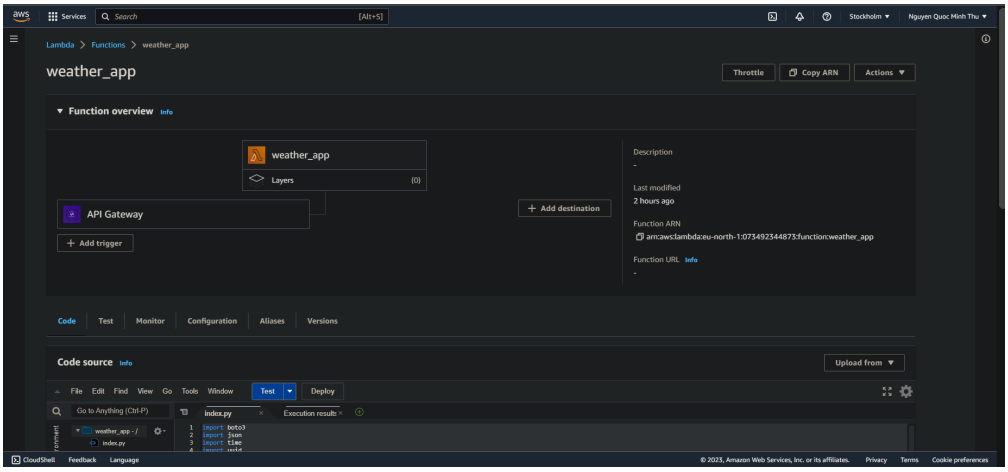


Hình 8: Get 5 recent searchs

4.3 Backend

4.3.1 AWS Lambda

♣ General Description:



Hình 9: Lambda Function

The API written in Python code that defines a lambda function to handle requests related to a weather app. The code includes two main operations:

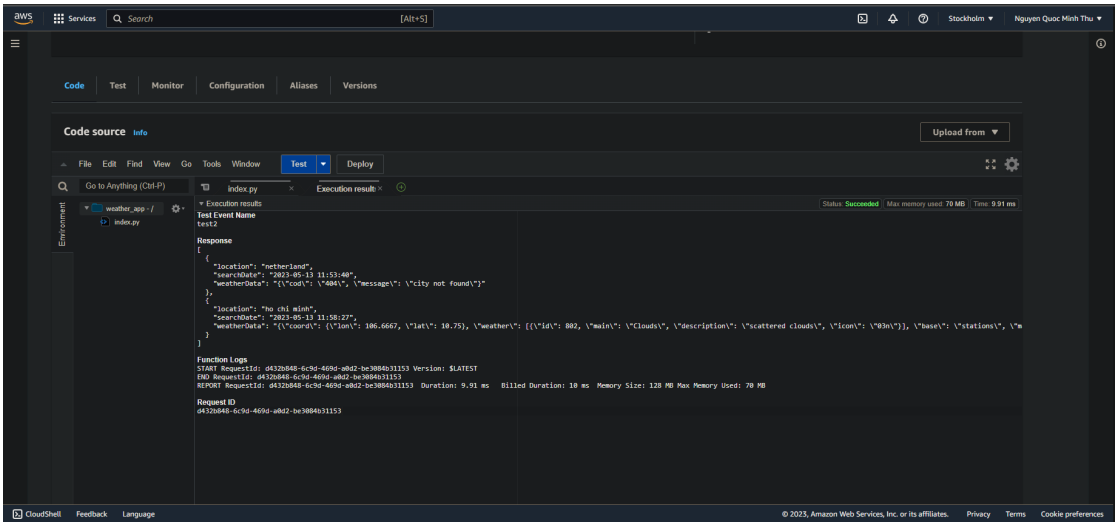
- **saveSearchData:** saves search data (location and weather data) in a DynamoDB table with a unique ID generated using UUID.
- **getRecentSearches:** retrieves the latest 5 searches made within the last 5 days from the DynamoDB table and returns them as a list of dictionaries, each containing the location, search date, and weather data. The code also includes error handling for unexpected exceptions that may occur during the execution of the lambda function. The code imports necessary libraries including boto3 for AWS SDK, json for JSON parsing, time for working with timestamps, and datetime for manipulating date and time values.

♣ API Python sample code:

```
Operation == 'saveSearchData':
location = event['location']
weatherData = event['weatherData']
new_id = str(uuid.uuid4()) # generate a new id using UUID
table.put_item(
Item={
'id': new_id,
'location': location,
'searchDate': datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S'),
'weatherData': json.dumps(weatherData)
}
)
return {'message': 'Search data saved successfully'}
```

```
Operation == 'getRecentSearches':
            start_date = int((datetime.now() - timedelta(days=5)).timestamp() *
    1000)
            projection_expression = "#loc, searchDate, weatherData"
            expression_attribute_names = {"#loc": "location"}
            response = table.scan(
                FilterExpression=Key('searchDate').gt(datetime.utcfromtimestamp(
    start_date/1000).strftime('%Y-%m-%d %H:%M:%S')),
                ProjectionExpression=projection_expression,
                ExpressionAttributeNames=expression_attribute_names,
                Limit=5
            )
            items = response['Items']
            recent_searches = []
            for item in items:
                search_date = datetime.fromisoformat(item['searchDate']).strftime(
    '%Y-%m-%d %H:%M:%S')
                recent_searches.append({'location': item['location'], 'searchDate'
    : search_date, 'weatherData': item['weatherData']})
            return recent_searches
        else:
            return {'message': 'Invalid operation'}
    except Exception as e:
        print(e)
        return {'message': 'Error processing request'}
```

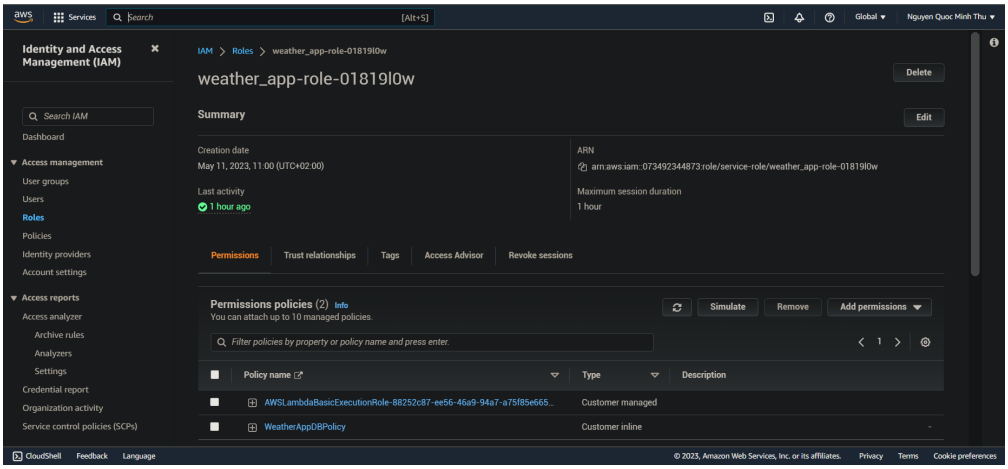
♣ API sample test:



Hình 10: Lambda Test

4.3.2 AWS IAM

♣ General Description:



Hình 11: AWS IAM

The IAM policy that grants permissions to perform certain actions on a specific DynamoDB table called "Weather_App_Database"in the eu-north-1 region.

The policy allows the following actions to be performed on the table: PutItem, DeleteItem, GetItem, Scan, Query, and UpdateItem. These actions correspond to the basic CRUD (create, read, update, and delete) operations on the table.

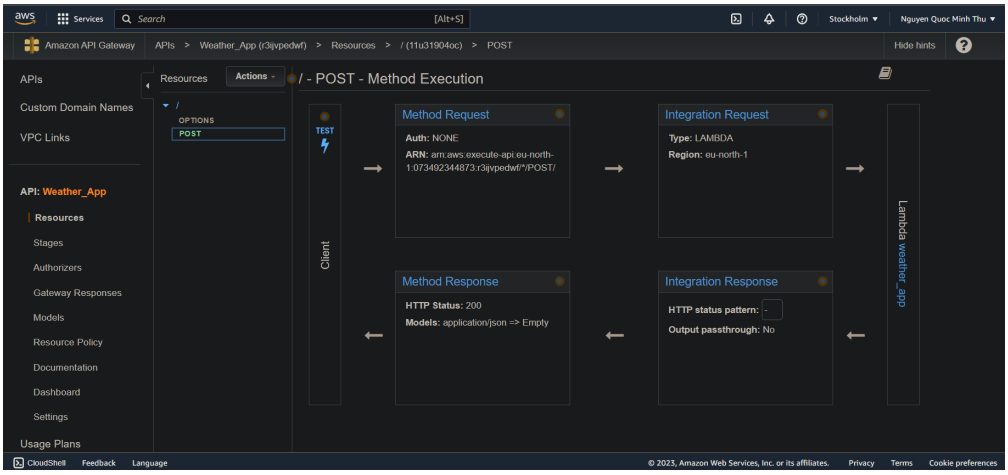
Overall, this IAM policy grants a user or role the necessary permissions to perform CRUD operations on the "Weather_App_Database"table in the eu-north-1 region.

♣ Sample JSON WeatherAppDBPolicy:

```
{
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": [
            "dynamodb:PutItem",
            "dynamodb:DeleteItem",
            "dynamodb:GetItem",
            "dynamodb:Scan",
            "dynamodb:Query",
            "dynamodb:UpdateItem"
        ],
        "Resource": "arn:aws:dynamodb:eu-north-1:073492344873:table/Weather_App_Database"
    }
]
}
```

4.3.3 AWS APIGateways

♣ General Description:



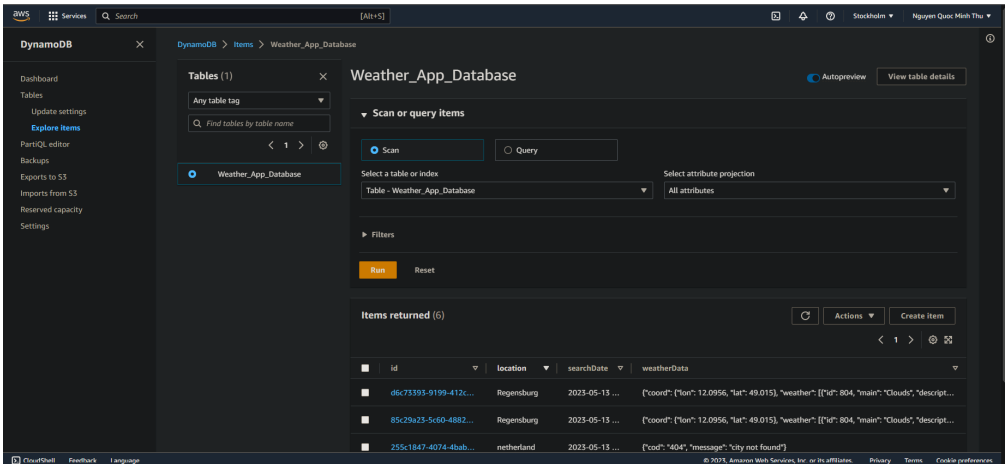
Hình 12: API Gateways

This configuration screen in the AWS console defines the settings for an API Gateway that is designed to handle a POST method. When the POST method is invoked, it triggers a Lambda function called "Lambda weather_app"to process the request. The API Gateway does not require authentication, meaning that any client can access it without providing credentials.

♣ Deploy API:

Invoke URL: https://r3ijvpedwf.execute-api.eu-north-1.amazonaws.com/dev.

4.3.4 AWS DynamoDB



Hình 13: DynamoDB

In brief: Description of the design a NoSQL database in DynamoDB to store weather information.

♣ Create a table in DynamoDB:

- **Table name:** WeatherData
- **Primary key:** Id (String) - representing the unique identifier for weather data.

♣ Define attributes (columns) for the WeatherData table:

- **Id (String):** The unique identifier for weather data (primary key). Using uuid format.
- **name (String):** The name of the location for the weather information.
- **country (String):** The country code of the location for the weather information.
- **date (String):** The date when the weather information was recorded.
- **temperature (Number):** The temperature (measurement unit can be chosen, e.g., Celsius).
- **feels_like (Number):** The feels like temperature (measurement unit can be chosen, e.g., Celsius).
- **humidity (Number):** The humidity (measurement unit can be chosen, e.g., percentage).
- **wind_speed (Number):** The wind speed (measurement unit can be chosen, e.g., km/h).
- **visibility (Number):** The visibility (measurement unit can be chosen, e.g., km).
- **sunrise (String):** The time of sunrise in ISO format.
- **sunset (String):** The time of sunset in ISO format.
- **description (String):** The description of the weather condition.

♣ Insert data into the WeatherData table:

```
{
"weatherId": "weather-001",
"name": "Ho Chi Minh City",
"country": "VN",
"date": "2023-05-13",
"temperature": 31.01,
"feels_like": 38.01,
"humidity": 79,
"wind_speed": 4.12,
"visibility": 10000,
"sunrise": "1683930710",
"sunset": "1683976059",
"description": "scattered clouds"
}
```

The actual data stored in the database is in JSON format and includes additional fields. However, we only store the necessary fields in our WeatherData table to reduce storage space and improve query performance.