

**RainHound**

Weather Tracker and Alerts Generator



Stancu Gabriel – Iulian

https://github.com/GabrielStancu/RainHound

**Table of contents**

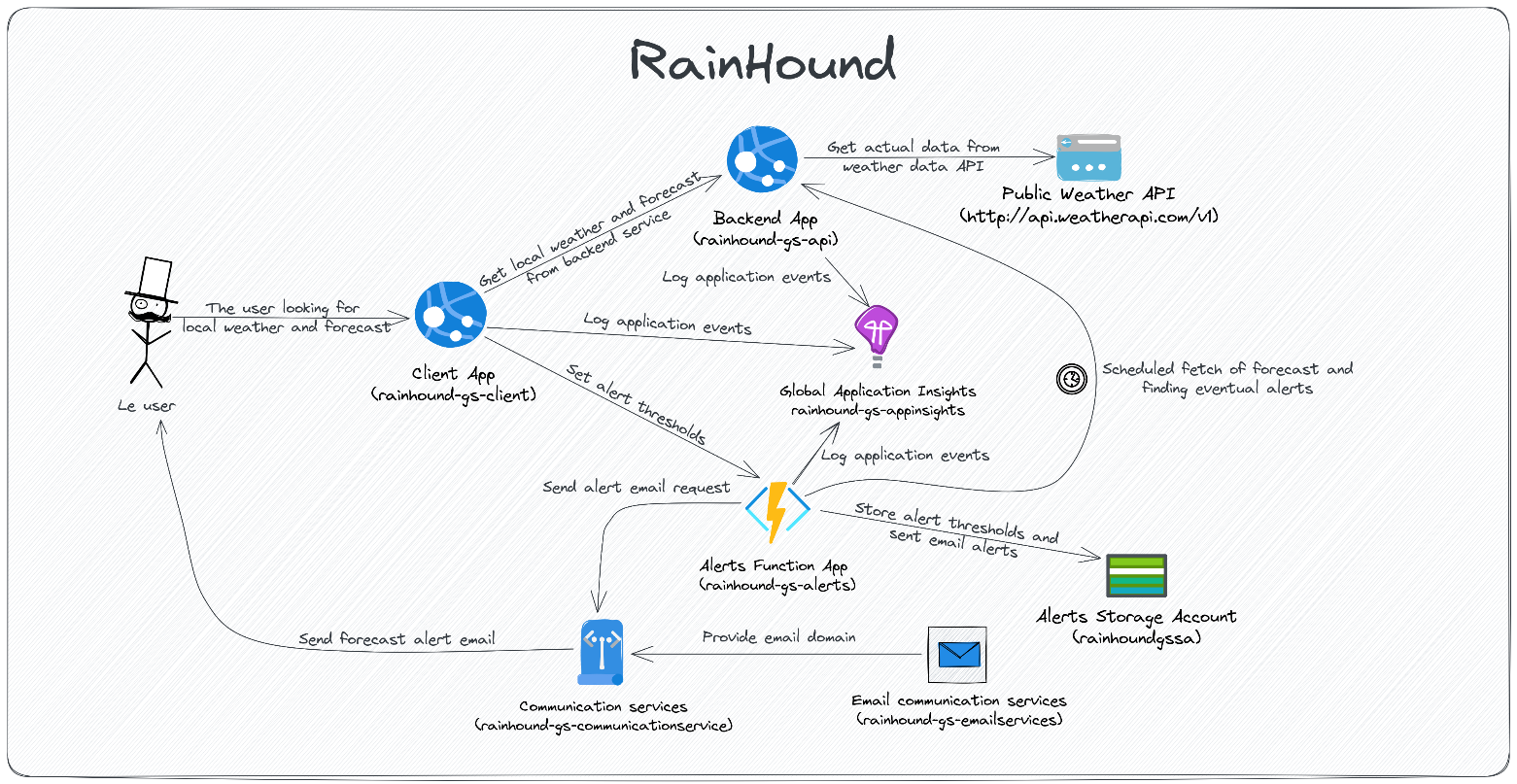
|  |  |
| --- | --- |
| 1. [Introduction](#Introduction)…………………………………………………………………………………... | 2 |
| 1. [Architecture](#Architecture) …………………………………………………………………………………. | 2 |
| 1. [User manual](#User_manual) …………………………………………………………………………………. | 3 |
| 1. [Deployment](#Deployment) ………………………………………………………………………………….. | 5 |
| 1. [Important notions](#Notions) ……………………………………………………………………………. | 8 |
| 1. [Future improvements](#Imporvements) ………………………………………………………………………... | 10 |

1. **Introduction**

RainHound is a web application used for weather tracking in real time. It is capable of fetching weather forecast for the next 3 days, with the possibility of sending email alerts in case certain events happen:

* Temperature goes below a preset threshold
* Temperature goes above a preset threshold
* Rain chances going above a preset threshold

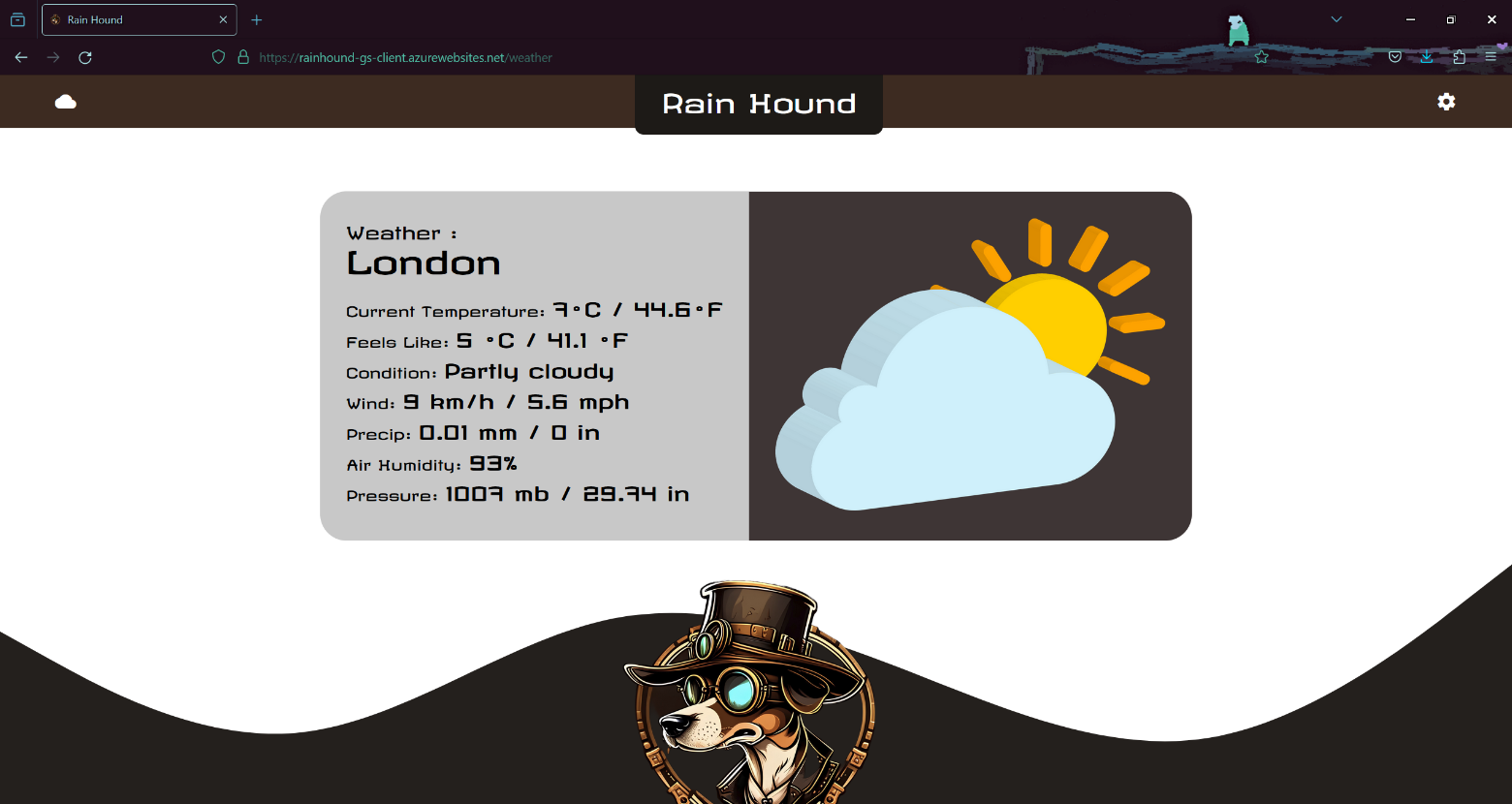
1. **Architecture**

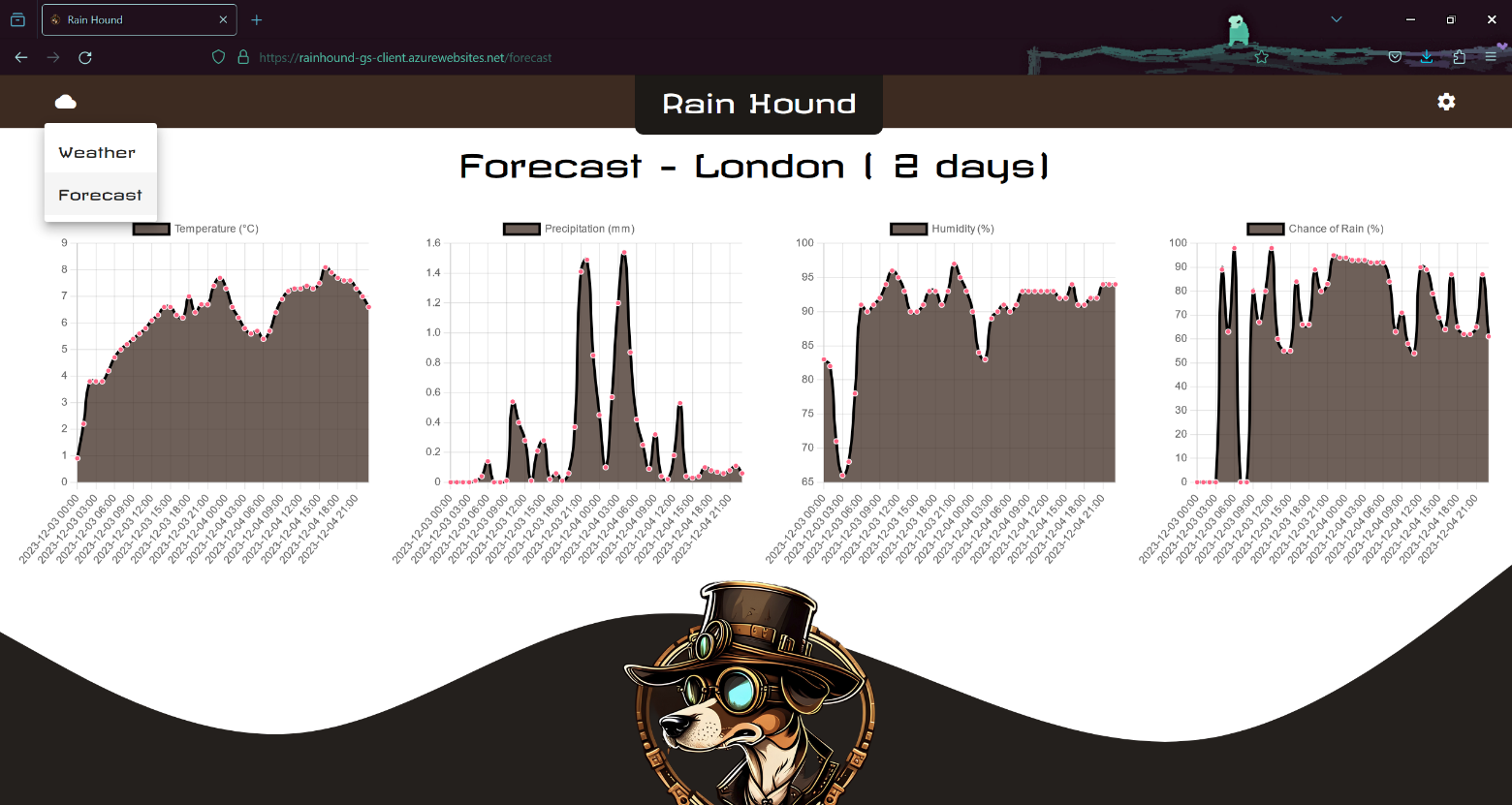
The application was built using the following architecture:

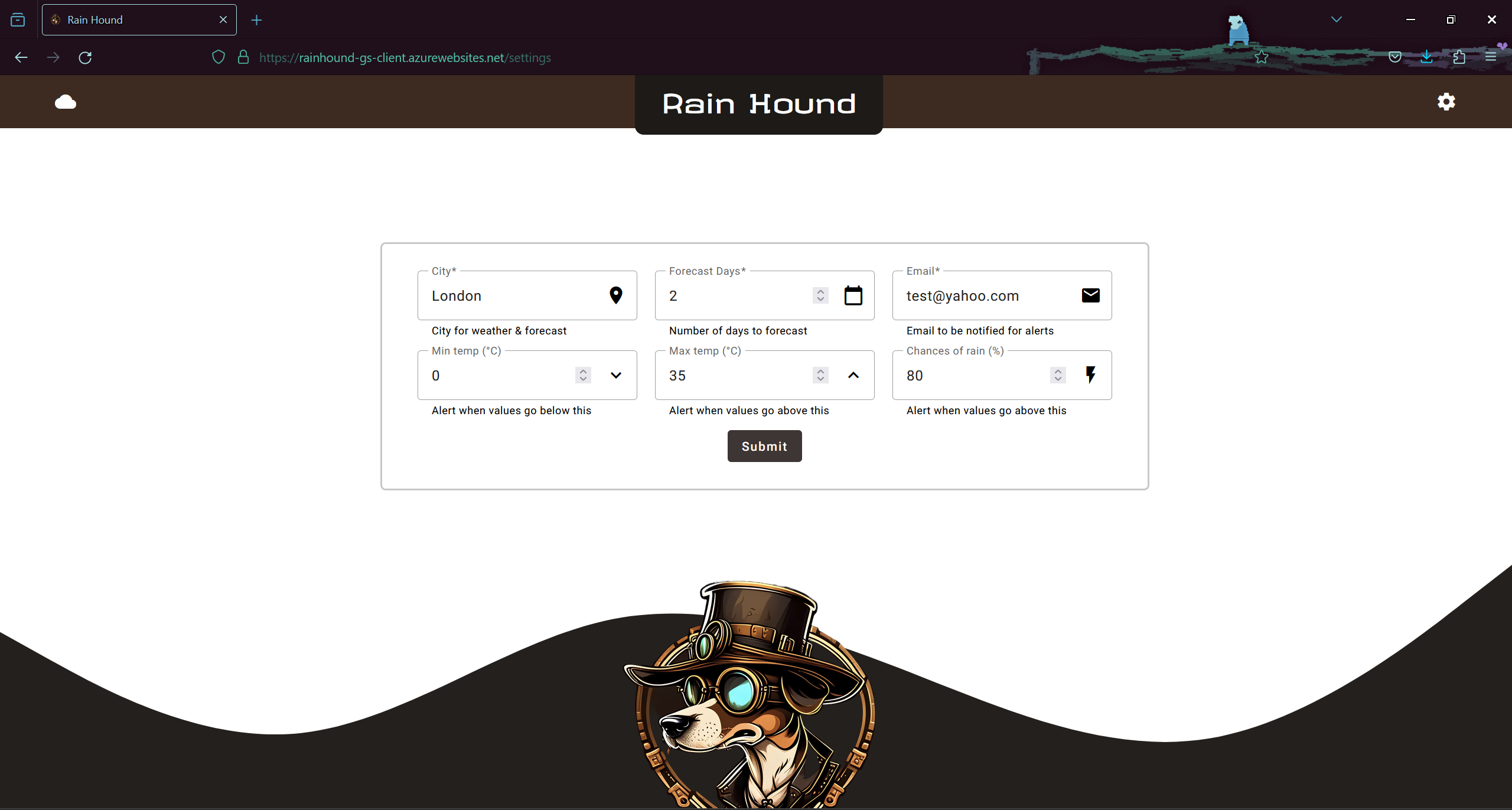
* **Client App (Web App):** Angular SPA application, allowing user to track weather in the desired location, obtaining forecast data, setting alert thresholds
* **Backend App (Web App)**: .NET 7 Web API, responsible for fetching data from the public weather API
* **Alerts Function App (Function App)**: .NET 7 Function App, responsible for storing the alerts set by the user and running periodically (every hour) a forecast check to search for new alerts
* **Alerts Storage Account**: Storage Account, the table storage is used for storing the alert thresholds and the send email alerts (to avoid duplicate emails every hour)
* **Global Application Insights**: Application Insights instance where all the logs from the first 3 components are sent
* **Email communication service**: provides an email domain used for sending the alerts to the users
* **Communication services**: uses the provided email domain to send the detected alerts to the users

1. **User manual**

The user goes to <https://rainhound-gs-client.azurewebsites.net> which will redirect the user to <https://rainhound-gs-client.azurewebsites.net/weather>. By default, the London weather is displayed. Displayed data consists of current temperature, how the temperature feels like, condition (assisted by a suggestive image), wind velocity, precipitations volume, air humidity and air pressure.

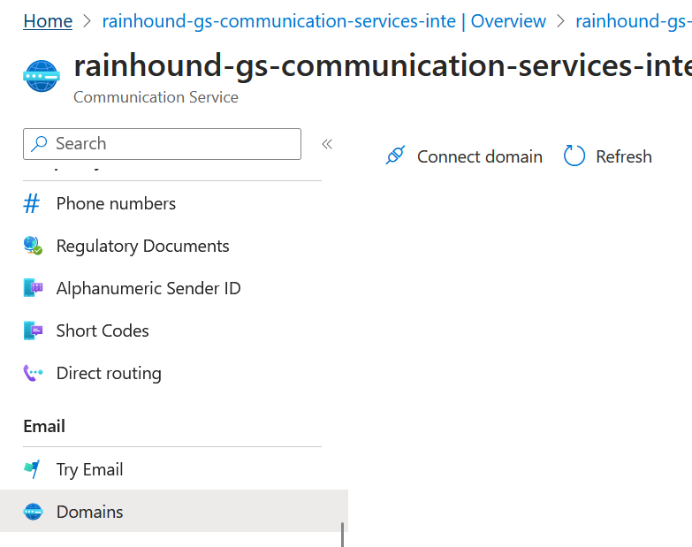
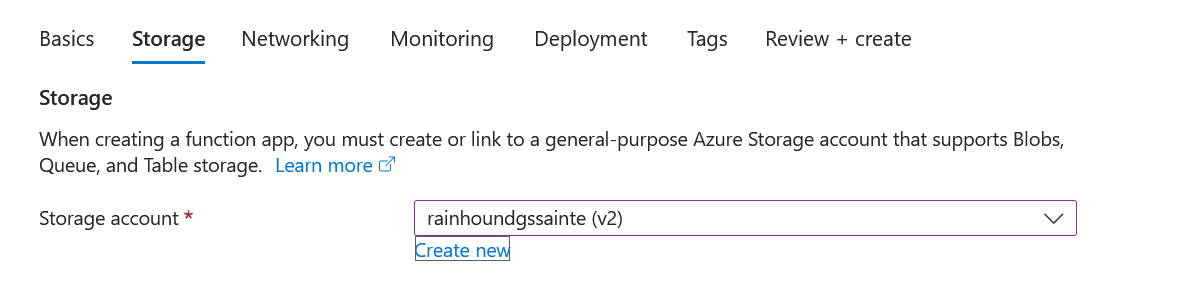
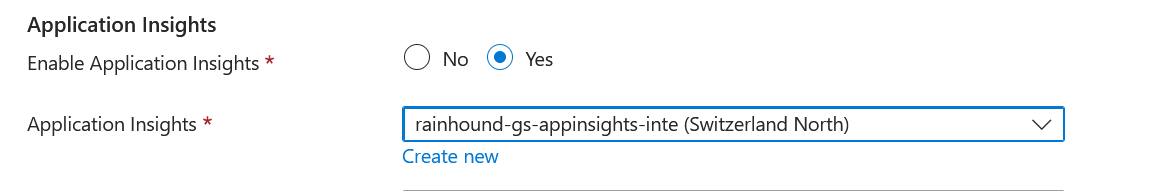


The user can change the view and see forecast data by clicking the cloud icon in the navbar:

Here the user sees the temperature, precipitations volume, humidity and chances of rain for the next few days. To configure the city, alert thresholds (minimum/maximum temperature, minimum precipitations chances), the email to be alerted at and the number of days for the forecast data, the user clicks the gear icon on the navbar. The form is validated (required values, minimum temperature should be blow the high temperature, maximum 3 days for forecast data): 

1. **Deployment**

The deployment is done using a CI/CD pipeline built on GitHub actions. The integration environment deploy is triggered by any push on the **integration** branch, while the production environment is triggered by any push on the **master** branch. The actions files (see under github/workflows in the repository) contain descriptive comments for each task/instruction. The following steps were taken for initializing the resources in Azure (screenshots taken from integration, the same steps were applied in production):

* 1. Create AppInsights instance
  2. Create Email Communication Services
     1. Add Azure/Custom domain to it
  3. Create Communication Services
     1. Connect email domain from step 2a
  4. Create storage account
     1. Create Alerts & Emails tables
  5. Create Function App
     1. Use the storage account from step 4
     2. Use Application Insights from step 1
     3. Set other configurations in Configuration -> Application Settings

"Values": {

    "AzureWebJobsStorage": "connection string of storage account from step 4",

    "FUNCTIONS\_WORKER\_RUNTIME": "dotnet-isolated",

    "Alerts:ScheduleCron": "0 0 \* \* \* \*",

    "Alerts:FromEmail": "email domain from steps 2 & 3",

    "Alerts:Subject": "Weather Alert",

    "Alerts:ConnectionString": "connection string of communication services from step 2"

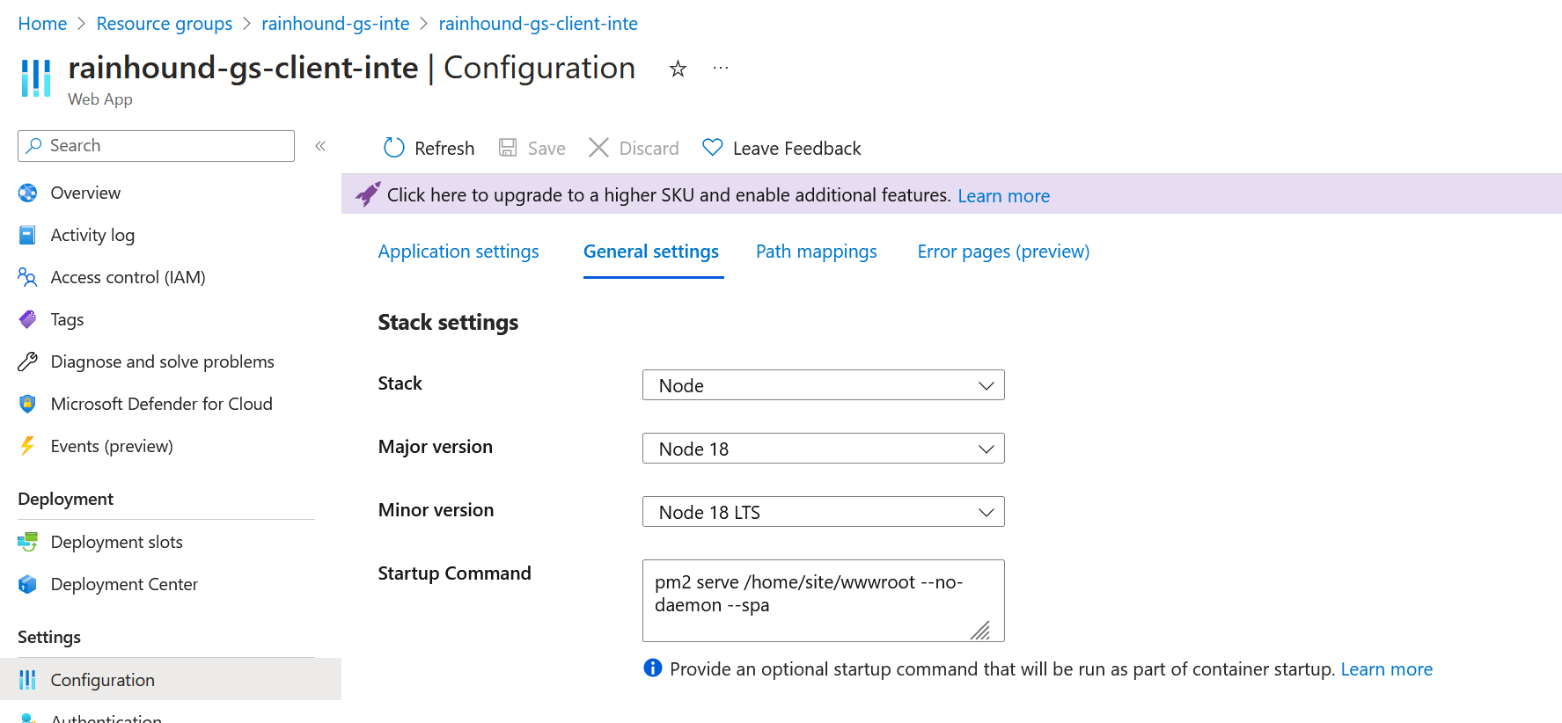
    "TableStorage:ConnectionString": "connection string of storage account from step 4",

    "TableStorage:AlertsTable": "Alerts",

    "TableStorage:EmailsTable": "Emails",

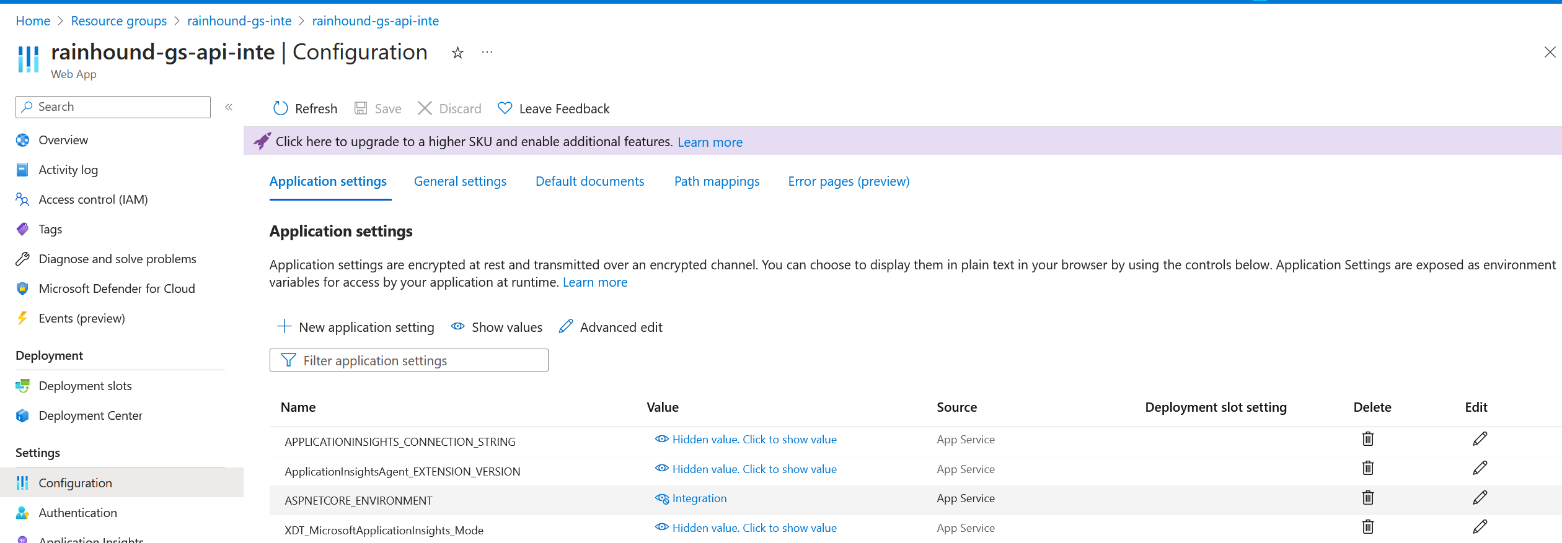
    "WeatherApi:Url": "url of weather api"

  },

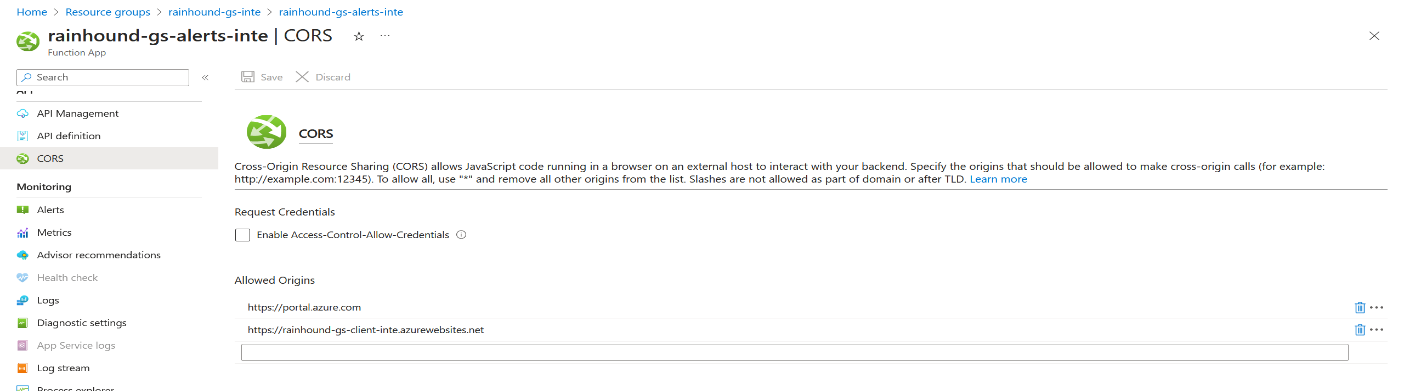
* 1. Create web app for frontend application.
     1. Connect the Web App to the Application Insights instance from step 1
     2. Set startup command in Configuration -> General settings:

(step taken from this article: <https://nicolgit.github.io/how-deploy-angular-app-to-azure-appservice-running-linux-from-github/>)

**pm2 serve /home/site/wwwroot --no-daemon –spa**

* 1. Create web app for backend application.
     1. Connect the Web Application to the Application Insights instance from step 1
     2. Set environment in Configuration:

**ASPNETCORE\_ENVIRONMENT : Integration / Production**

* 1. Set CORS for function app to allow calls from client app:

1. **Important notions**

* Make use of environments in Angular application:
  + Run **ng g environments** in terminal
  + Create the **environment.development.ts**, **environment.integration.ts** and **environment.ts** files, then complete the configurations for each environment:

export const environment = {

  production: false,

  weatherUrl: 'base api url for backend application',

  alertsUrl: 'base api url for setting alerts function',

setAlertEndpoint: 'SetAlert', // The name of the alert setting function

  appInsightsInstrumentationKey: 'instrumentation key for app insights, step 1',

  environment: 'Development' // or Integration or Production

};

* + Inside **angular.json**, under **build/configurations**:

"development": {

              "buildOptimizer": false,

              "optimization": false,

              "vendorChunk": true,

              "extractLicenses": false,

              "sourceMap": true,

              "namedChunks": true,

              "fileReplacements": [

                {

                  "replace": "src/environments/environment.ts",

                  "with": "src/environments/environment.development.ts"

                }

              ]

            },

            "integration": {

              "buildOptimizer": false,

              "optimization": false,

              "vendorChunk": true,

              "extractLicenses": false,

              "sourceMap": true,

              "namedChunks": true,

              "fileReplacements": [

                {

                  "replace": "src/environments/environment.ts",

                  "with": "src/environments/environment.integration.ts"

                }

              ]

            }

* + Inside **angular.json**, under **serve/configurations**:

"serve": {

          "builder": "@angular-devkit/build-angular:dev-server",

          "configurations": {

            "production": {

              "browserTarget": "client:build:production"

            },

            "development": {

              "browserTarget": "client:build:development"

            },

            "integration": {

              "browserTarget": "client:build:integration"

            }

          },

          "defaultConfiguration": "development"

        },

* + Inside **package.json**, under **scripts**: (these will map the **npm run build:environment** to the Angular CLI instruction for building with the desired configuration):

"scripts": {

    "ng": "ng",

    "start": "ng serve",

    "build": "ng build",

    "build:integration": "ng build --configuration=integration",

    "build:production": "ng build --configuration=production",

    "watch": "ng build --watch --configuration development",

    "test": "ng test"

  },

* Make use of configurations in backend API:
  + Complete the **appsettings.Development.json**, **appsettings.Development.json**, or **appsettings.Development.json** file with the required configurations:

{

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft.AspNetCore": "Warning"

    }

  },

  "WeatherApi": {

    "BaseUrl": "base api url for weather external service",

    "ApiKey": "key for external weather api",

    "CacheDurationMinutes": 60

  },

  "Environment": {

    "Name": "Integration"

  },

  "Client": {

    "BaseUrl": "base url for client, used for CORS"

  },

  "AlertsFunction": {

    "BaseUrl": "base url for function app, used for CORS"

  }

}

* Make use of configurations in function app: manually set the application settings in each environment, directly in Azure. See 4.5.c.

1. **Future improvements**

As of future improvements, the next features would be added:

* Allow user to unsubscribe from alerts for given city and thresholds
* Make the application responsive (mobile-friendly)
* Move the backend to Docker container, deploy as Container Registry + Container Instance (integration) and Container Apps (production)
* Send SMS instead of emails (for better reach)
* Move credentials to encrypted services, remove the configuration files from project and Git