



**Title of the Project:**

**QR Code Generator on Azure Function  
App with CI/CD Automation using  
Terraform & GitHub Actions :**

NAME- KHUSHI AGRAHARI

EMAIL- KHUSHIAGRARI05@GMAIL.COM

# **TABLE OF CONTENTS**

<b>SL. NO.</b>	<b>TITLE</b>	<b>PAGE NO</b>
<b>I</b>	<b>INTRODUCTION &amp; OBJECTIVE</b>	<b>2</b>
<b>II</b>	<b>HARDWARE &amp; SOFTWARE REQUIREMENTS</b>	<b>3</b>
<b>III</b>	<b>ANALYSIS</b>	<b>3</b>
<b>IV</b>	<b>SEQUENCE DIAGRAM</b>	<b>4</b>
<b>V</b>	<b>ER-DIAGRAM</b>	<b>5</b>
<b>VI</b>	<b>SYSTEM DESIGN</b>	<b>6-7</b>
<b>VII</b>	<b>IMPLEMENTATION DETAILS</b>	<b>8-10</b>
<b>VIII</b>	<b>OUTPUT SCREENSHOTS</b>	<b>11-24</b>
<b>IX</b>	<b>ADVANTAGES / FEATURES &amp; FUTURE SCOPE</b>	<b>25</b>
<b>X</b>	<b>CONCLUSION &amp; REFERENCES</b>	<b>26</b>

## **Introduction**

In today's fast-paced digital era, businesses and developers are increasingly shifting towards serverless architectures to simplify deployment, reduce costs, and improve scalability. Azure Functions Microsoft's serverless compute service offers a powerful platform for executing backend code without the need to manage infrastructure. At the same time, Infrastructure as Code (IaC) tools like Terraform allow for consistent, repeatable, and automated provisioning of cloud infrastructure.

This project focuses on developing a QR Code Generator using Python that will be deployed as an HTTP-triggered Azure Function. Users can send input data (like a URL or text), and the Azure Function will generate a corresponding QR code in image format, store it in azure storage account and return it as a response along with downloadable blob url link.

To enhance deployment efficiency, the project uses Terraform to define and provision the necessary cloud infrastructure automatically. Furthermore, GitHub Actions is used to implement a CI/CD (Continuous Integration/Continuous Deployment) pipeline, which ensures that any change pushed to the GitHub repository is automatically tested and deployed to Azure, reducing the chances of human error and enhancing productivity.

By integrating these technologies, the project demonstrates the real-world application of DevOps practices, cloud automation, and serverless computing for developing a modern, scalable, and reliable application.

## **Objective of the Project**

1. To develop a QR Code Generator using Python.
2. To build a serverless application using Azure Functions.
3. To automate cloud infrastructure provisioning using Terraform.
4. To implement Continuous Integration and Continuous Deployment using GitHub Actions.
5. To explore real-world DevOps practices and Infrastructure as Code principles.
6. To reduce human errors and manual deployment time.
7. To provide a secure, fast, and reusable cloud-based solution.

## **Project Category**

Cloud Computing / DevOps / Serverless Architecture / Python Programming

## **Hardware and Software Requirements**

### **Hardware Requirements:**

- Processor: Intel i3 or higher
- RAM: 4 GB or more
- Storage: 100 GB HDD/SSD- Reliable Internet Connection

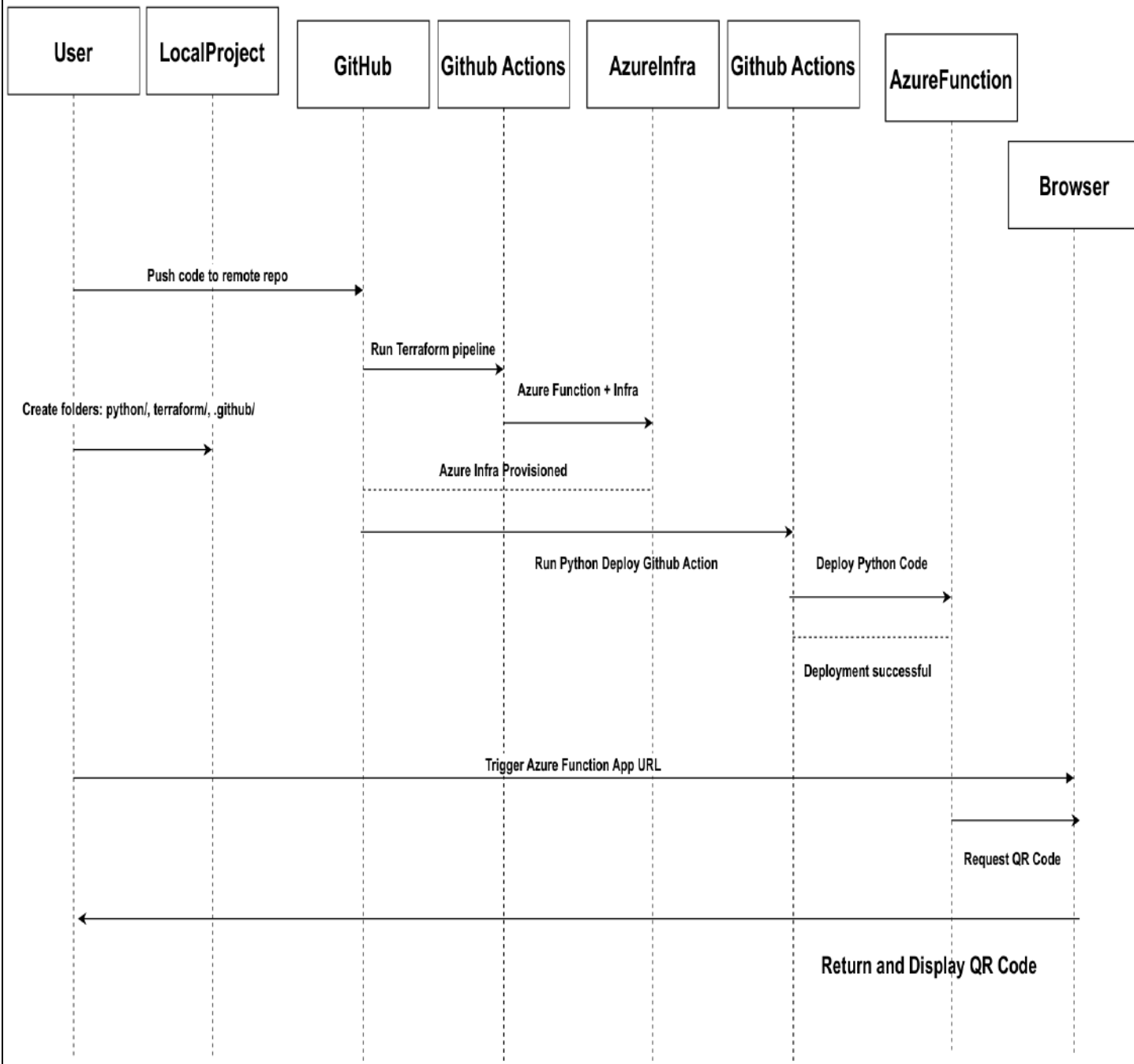
### **Software Requirements:**

- Operating System: Windows 10/11 or Linux
- Python 3.10+
- Visual Studio Code
- Azure CLI or Azure PowerShell
- Terraform
- Git and GitHub
- GitHub Actions
- Azure Portal Account

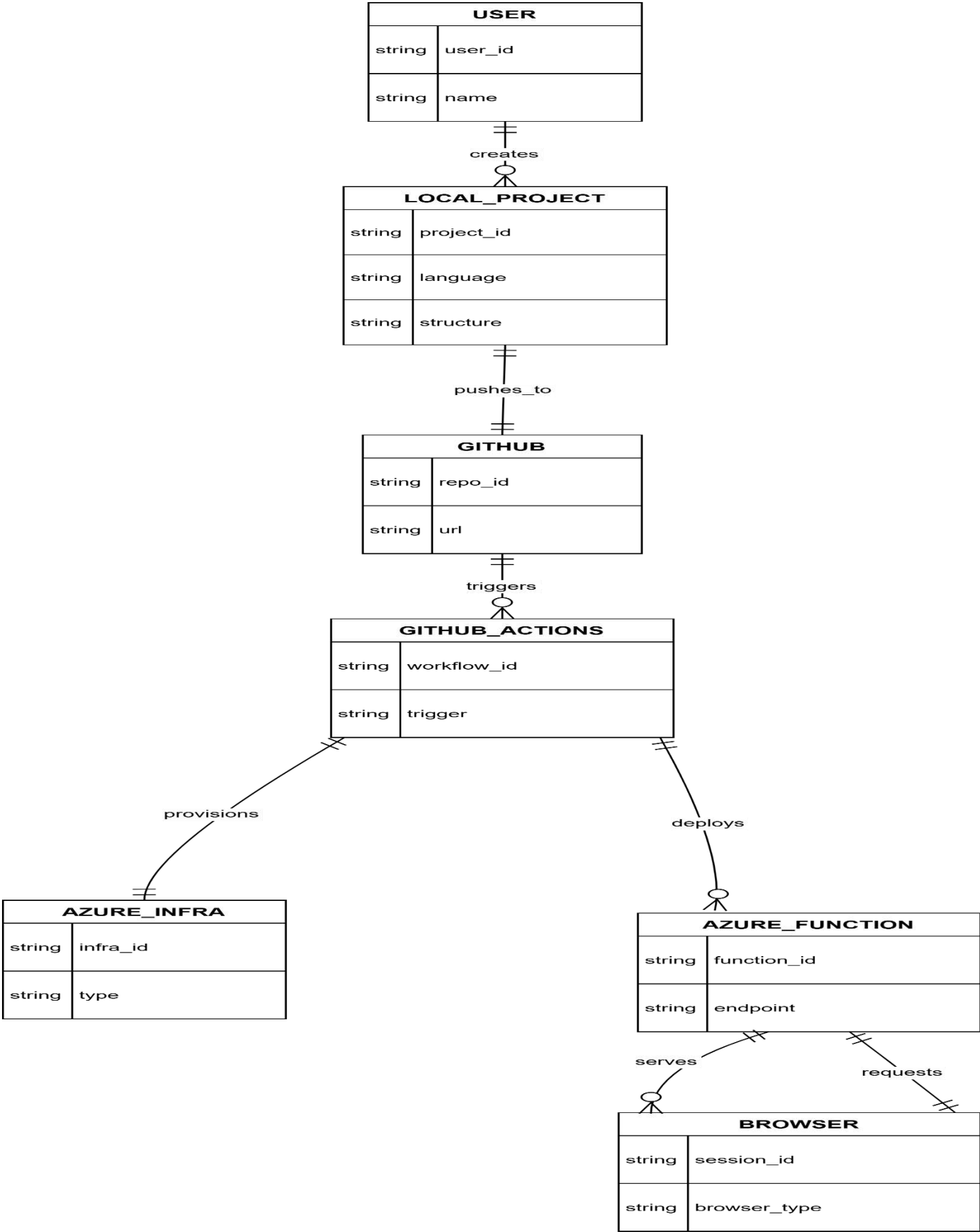
## **Analysis:**

1. **QR Code Generator Module (Python):** Uses Python libraries to create QR codes.
2. **Azure Function App:** Hosts the QR generator logic as a serverless function.
3. **Terraform Automation:** Automates Azure resource provisioning.
4. **GitHub Actions Workflow:** Manages CI/CD for automatic deployment to Azure.

## Sequence Diagram



# ER-DIAGRAM



# **SYSTEM DESIGN**

This project follows a modular and automated design to build and deploy a serverless QR Code Generator using Azure services. The complete system is divided into three key layers:

## **1. Application Layer (Azure Functions)**

- **The application contains two Azure Functions deployed under a single Function App.**
  - **static-ui:** This function serves a static HTML frontend where users can enter text.
  - **backend-api:** This function receives text input from the frontend, generates a QR code, store it in the Azure Storage Account and return with image as well as blob url link.
- The two functions interact using an HTTP trigger, where the frontend calls the backend function as an API.

## **2. Infrastructure Layer (Terraform Scripts)**

- Infrastructure provisioning is fully automated using Terraform.
- The following Azure resources are defined and managed using Infrastructure as Code:
  - **Resource Group**
  - **Storage Account**
  - **Function App**
  - **App Service Plan**
  - **Application Insight**
- All configuration files are stored under the Infra/ folder and include:
  - main.tf, provider.tf, variables.tf, and optionally an env/ for environment-specific values.

### **3. CI/CD Layer (GitHub Actions)**

- Continuous Integration and Deployment is implemented using GitHub Actions.
- **Two YAML workflows are defined in the .github/workflows directory:**
  - **deploy-azure-function-infra.yaml:** Deploys Azure infrastructure using Terraform.
  - **deploy-app-code-function-app.yaml:** Deploys the application code to the Function App as a zip package.
- Secrets and environment variables are securely configured in GitHub Settings under environment.

### **Execution Flow**

1. User visits the web interface provided by static-ui.
2. User enters a text input and submits.
3. The frontend sends a request to backend-api.
4. backend-api generates a QR code and saves the image to Azure Storage.
5. The QR code is displayed back to the user in the frontend along with downloadable blob url link.
6. All deployment steps are handled automatically through GitHub pipelines.

### **Benefits of This System Design**

- Secure, fast, and cost-effective deployment using serverless technology.
- Eliminates manual setup by using Terraform and GitHub Actions.
- Reusable code and infra setup for similar future projects.
- Adopts real-world DevOps and IaC principles.



## IMPLEMENTATION DETAILS

This section describes how the entire project was implemented step-by-step, covering code structure, technologies used, automation pipelines, and deployment processes. The main goal was to create a fully functional, serverless QR Code Generator using Azure, Python, Terraform, and GitHub Actions.

### 1. Tools & Technologies Used

Technology	Purpose
Python	Backend function logic and QR code generation
HTML/CSS	Static frontend UI for user input
Azure Functions	Hosting serverless functions (frontend + backend)
Azure Storage Account	Storing generated QR code images
Terraform	Automating Azure infrastructure (IaC)
GitHub Actions	CI/CD pipeline for infrastructure and code deployment
Git + GitHub	Version control and remote repository

### 2. Project Folder Structure

```

project-root/
|
├── .github/
|   └── workflows/
|       ├── deploy-azure-function-infra.yaml    # Deploys Azure infra via Terraform
|       └── deploy-app-code-function-app.yaml  # Deploys code to Function App
|
├── Infra/
|   ├── main.tf          # Defines resources
|   ├── provider.tf      # Azure provider config
|   ├── variables.tf     # Input variables
|   └── outputs.tf       # Output values (optional)
|
├── qr-generator/
|   ├── backend-api/      # Backend QR generator function
|   │   ├── __init__.py  # Python code to generate QR
|   │   └── function.json # Azure Function configuration
|   │
|   ├── static-ui/       # Frontend function serving HTML
|   │   ├── __init__.py  # Handles HTTP trigger (optional)
|   │   ├── function.json # Function config for HTTP
|   │   └── index.html    # HTML UI for user input
|   │
|   └── host.json         # Function App settings
|
├── requirements.txt      # Python dependencies
├── .funcignore          # Ignore files during deployment
├── scripts/
|   └── automation-setup.ps1 # Optional infra script (windows)
└── README.md            # Project overview

```

## 📁 .github/workflows/

Contains **GitHub Actions** YAML workflows to automate infrastructure and application deployment:

- **deploy-azure-function-infra.yaml**  
→ Automates the provisioning of Azure infrastructure using Terraform.
- **deploy-app-code-function-app.yaml**  
→ Packages the Azure Function App code into a ZIP archive and deploys it to Azure.

## 📁 Infra/

This folder holds all **Terraform configuration files** for provisioning the cloud infrastructure:

- **main.tf**  
→ Declares core resources such as Azure Function App, App Service Plan, Storage Account, and Resource Group.
- **provider.tf**  
→ Specifies the Azure provider configuration (like subscription ID, region, etc.).
- **variables.tf**  
→ Contains input variables to make the Terraform code reusable and flexible.
- **outputs.tf**  
→ Defines outputs like Function App URL after deployment.
- **env/ dev.tfvars**  
→ Stores environment-specific .tfvars files for staging, development, or production.

## 📁 qr-generator/

This is the main application layer, consisting of two Azure Functions:

### 📁 backend-api/

- **\_\_init\_\_.py**  
→ Contains Python code that receives user input, generates a QR code using the qrcode library, and stores it in Azure Blob Storage.
- **function.json**  
→ Defines function bindings, HTTP trigger settings, and route configuration.

### 📁 static-ui/

- **index.html**  
→ A simple HTML page with a form to enter user text input and view the generated QR code.

- **\_\_init\_\_.py**  
→ Handles HTTP GET/POST requests and acts as the interface between frontend and backend.
- **function.json**  
→ Configures HTTP **triggers and routing** for serving the frontend.

## Other Supporting Files

- **.funcignore**  
→ Specifies files to ignore when deploying the Azure Function.
- **host.json**  
→ Global configuration for the Azure Function App (e.g., logging, extensions).
- **requirements.txt**  
→ Lists all Python package dependencies, such as: `qrcode`, `azure.storage.blob`, etc.
- **scripts/automation-setup.ps1**  
→ PowerShell script to automate local setup or other deployment tasks part of pre-requisites.
- **.gitignore**  
→ Prevents unnecessary files (e.g., `.venv`, `.terraform`, cache files) from being committed to Git.
- **README.md**  
→ Provides a complete overview of the project, including usage, setup instructions, and architecture summary.

## Summary

### This modular design ensures:

- Clean separation between **infrastructure, backend logic, and UI**.
- Easy automation of deployment and provisioning.
- Clear code organization for collaboration, reusability, and debugging.

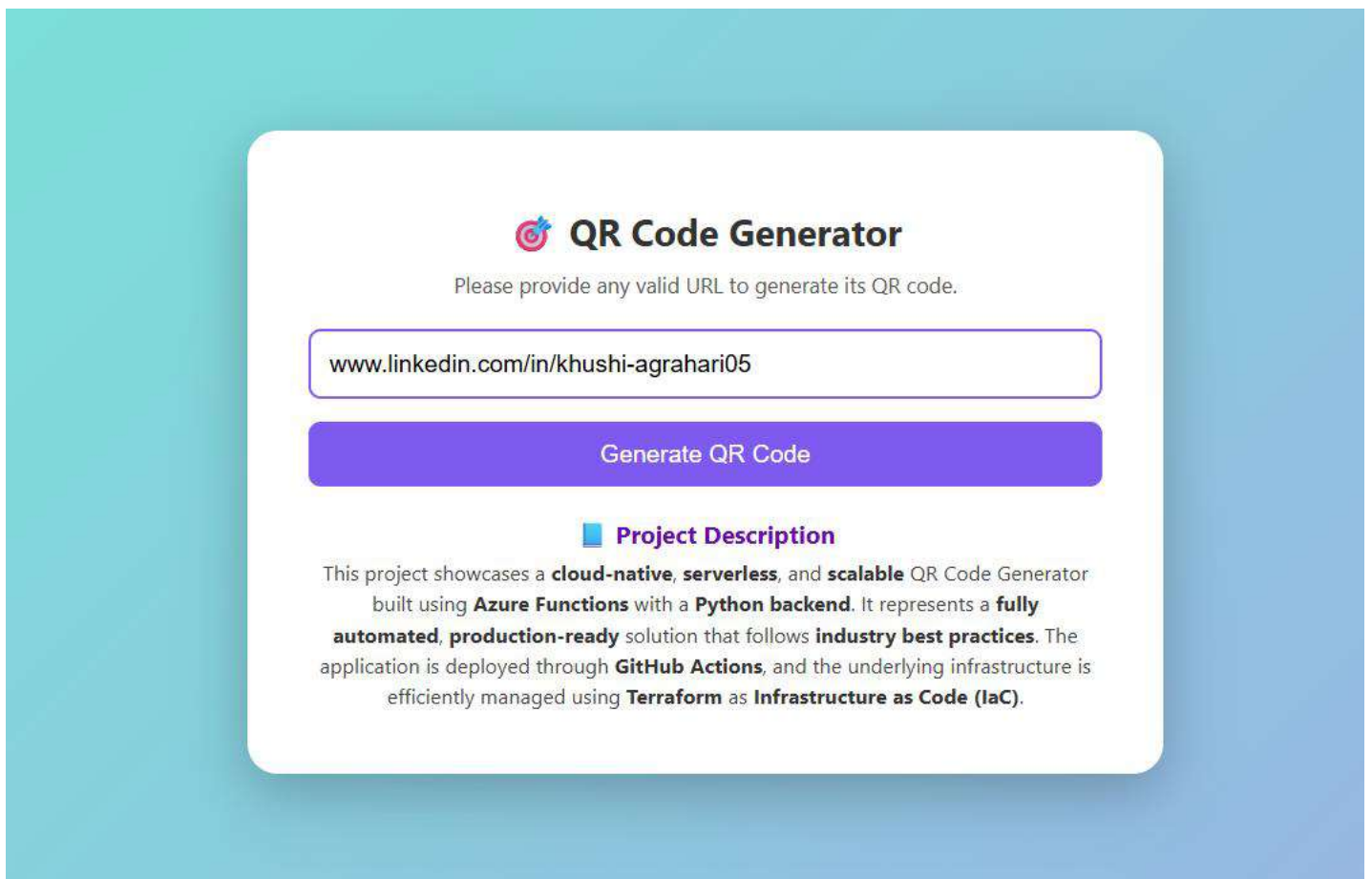
## OUTPUT SCREENSHOTS

This section provides visual proof of the project's successful execution, user interaction, and deployment across Azure and GitHub Actions.

**1. User Interface of QR Code Generator** The user interface is a simple HTML page served by the static-ui Azure Function. It includes:

- A text input field
- A "Generate" button to trigger QR code creation

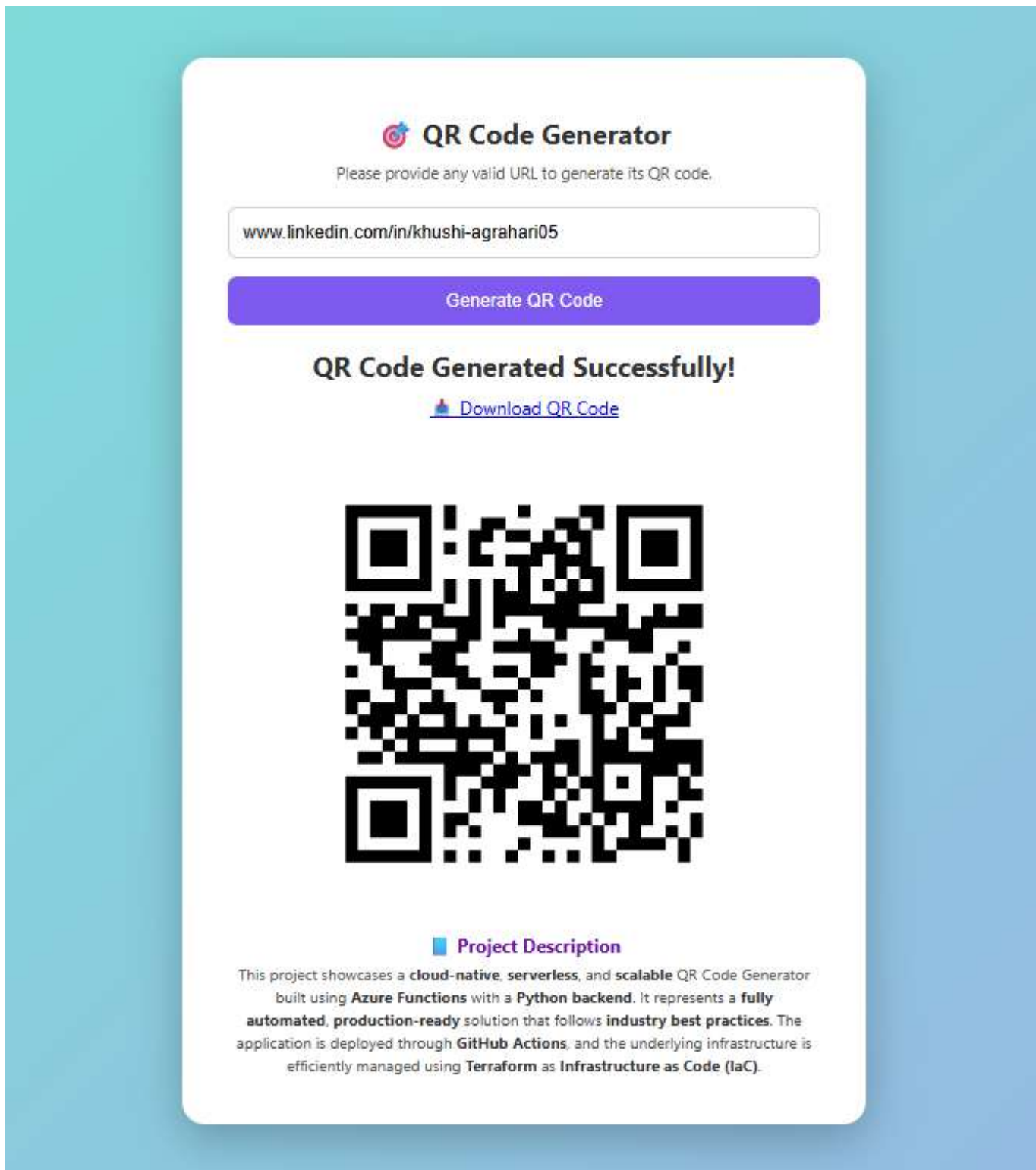
📷 Screenshot:



**2. QR Code Display After Generation** Upon submitting the input:

- The frontend calls the backend-api
- A QR code is generated and returned
- The QR code image is displayed/downloaded on the frontend

📷 Screenshot: Generated QR code displayed on the web page



### 3. Azure Portal: Function App Deployment The deployment confirmation includes:

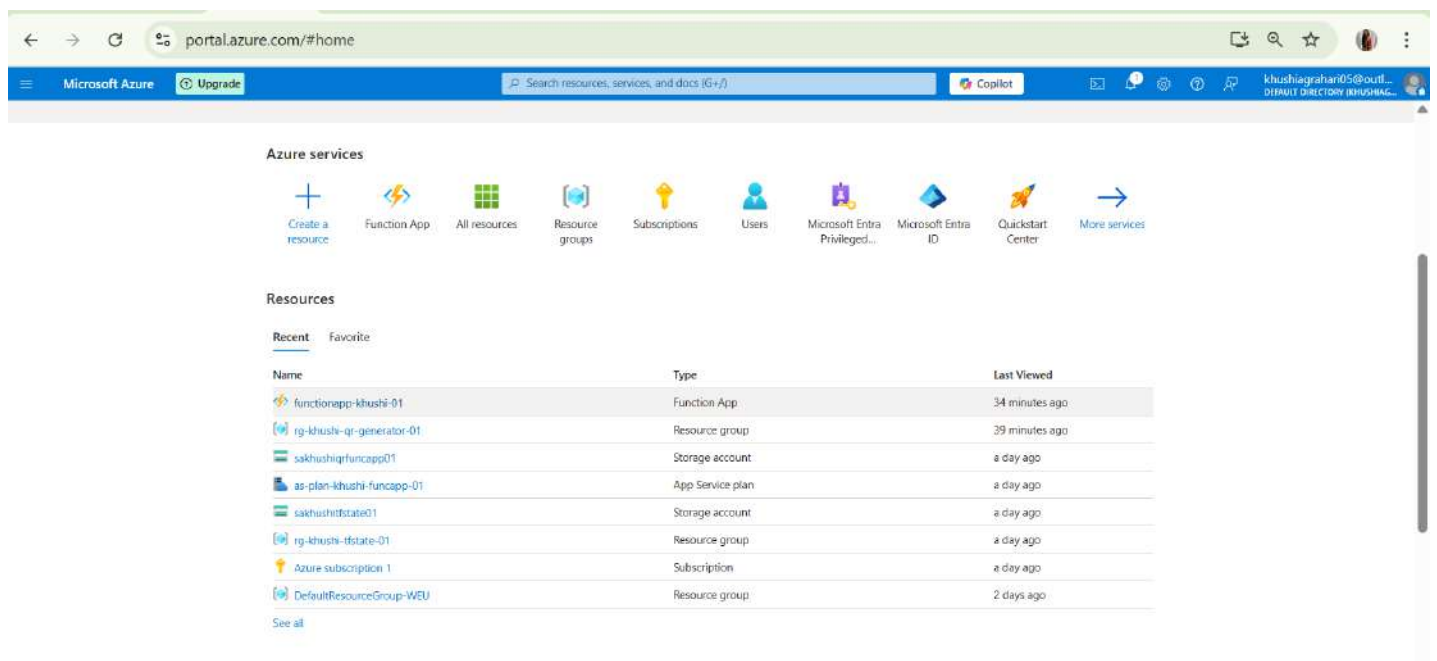
#### Description:

Below screenshots display the **Azure portal's home page**, showing all recently created resources. It confirms that the following components have been successfully deployed using Terraform:

- **Function App:** functionapp-khushi-01
- **Resource Group:** rg-khushi-qr-generator-01
- **Storage Account:** sakshukhushifuncapp01 ( app ), sakshukhushitfstate01 (tf state file)
- **App Service Plan:** as-plan-khushi-funcapp-01
- **Application insights:** appinsights-khushi-01
- **Terraform State Resource Group:** rg-khushi-tfstate-01

It validates that the infrastructure provisioning was done correctly and that all resources are listed under the Azure account, showing their types and recent activity.

#### Screenshot: Azure Portal – Resource Overview



#### Description:

Below screenshot shows the **detailed overview of the deployed Azure Function App** named functionapp-khushi-01. It confirms that:

- **Resource Group:** rg-khushi-qr-generator-01

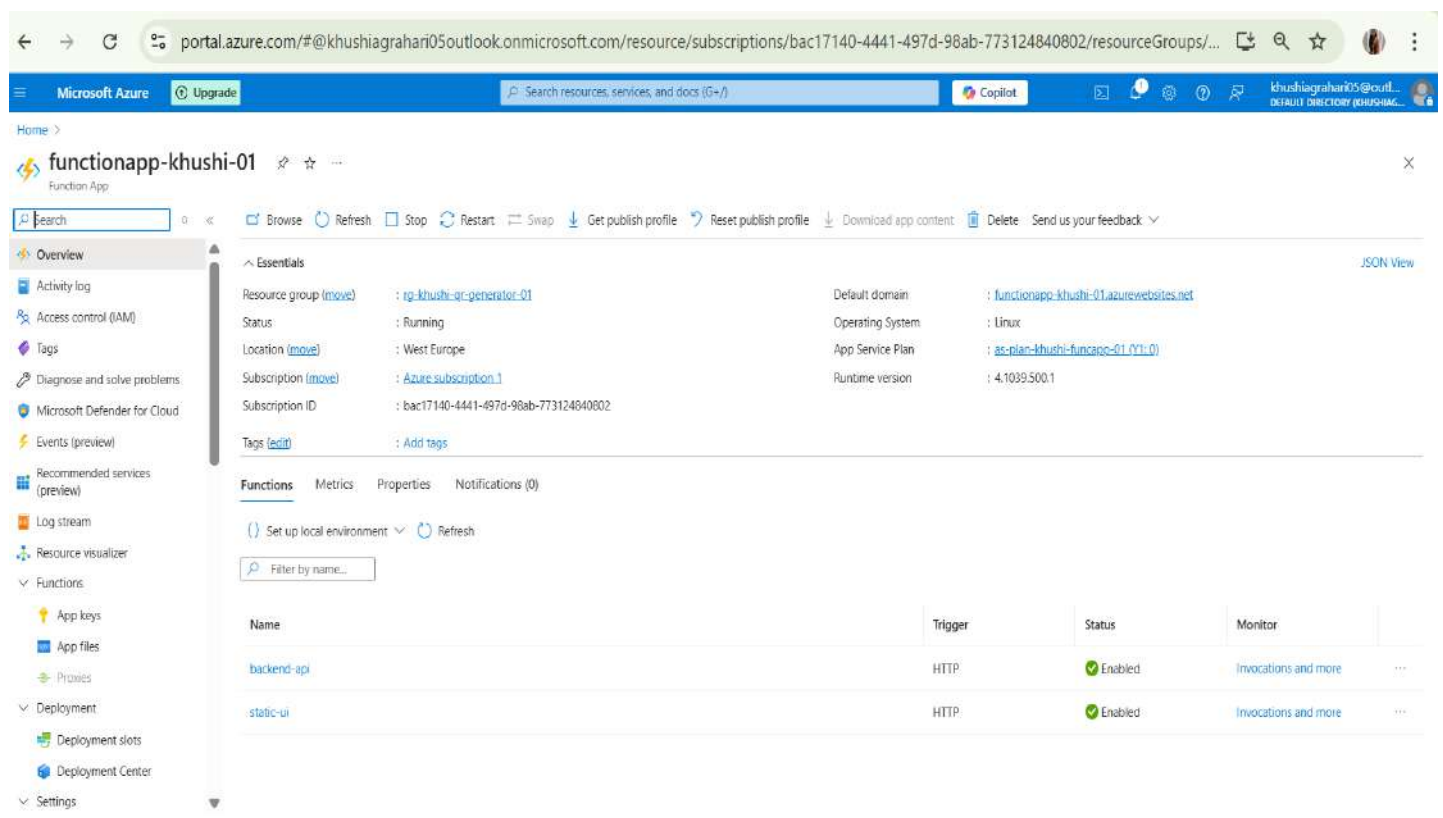
- **Status:** Running
- **Location:** West Europe
- **Operating System:** Linux
- **App Service Plan:** as-plan-khushi-funcapp-01
- **Runtime Version:** v4.1039.500.1

Additionally, it shows the **list of deployed Azure Functions**, which include:

- backend-api
- static-ui

Both functions are **HTTP triggered** and their statuses are **Enabled**, indicating successful deployment and readiness for invocation.

### Screenshot: Azure Function App Deployment Details



The screenshot displays the Azure Portal interface for the Function App 'functionapp-khushi-01'. The 'Overview' tab is selected, showing the following details:

- Resource group:** rg-khushi-qr-generator-01
- Status:** Running
- Location:** West Europe
- Subscription:** Azure subscription 1
- Subscription ID:** bac17140-4441-497d-98ab-773124840802
- Default domain:** functionapp-khushi-01.azurewebsites.net
- Operating System:** Linux
- App Service Plan:** as-plan-khushi-funcapp-01 (Y1:0)
- Runtime version:** 4.1039.500.1

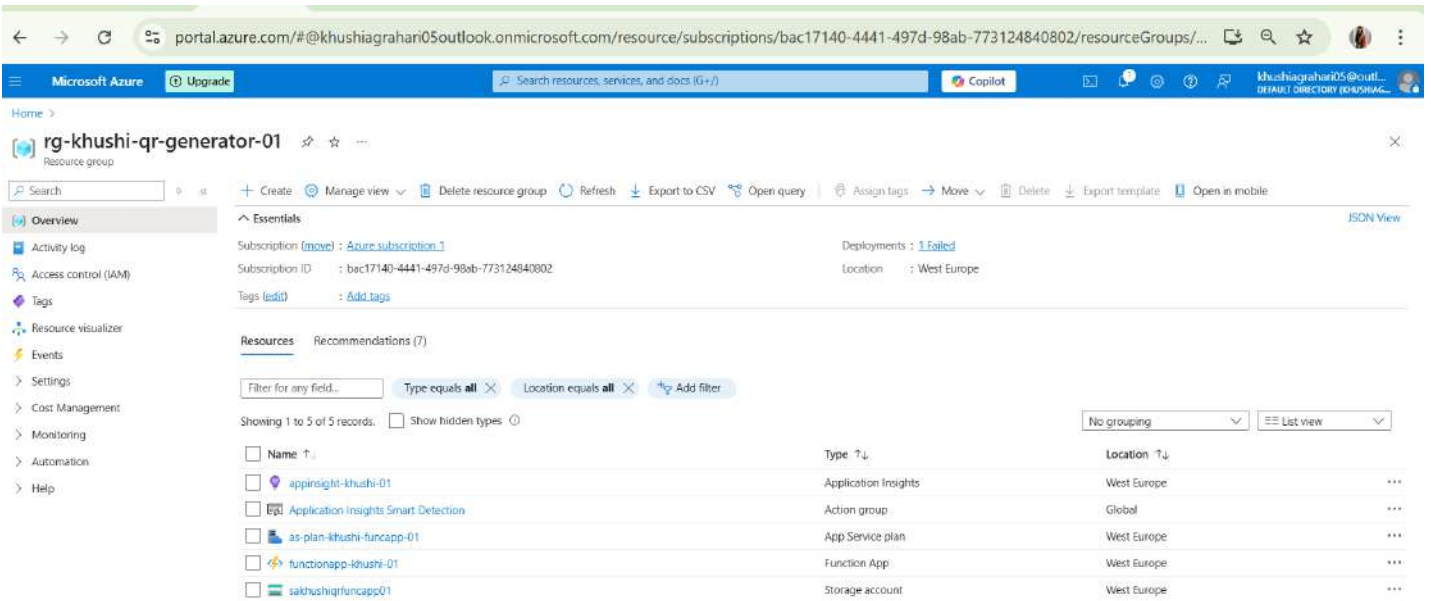
Below the overview, the 'Functions' tab is active, displaying a table of deployed functions:

Name	Trigger	Status	Monitor
backend-api	HTTP	Enabled	Invocations and more
static-ui	HTTP	Enabled	Invocations and more

### Description:

This below screenshot shows the Azure Portal homepage listing all the created resources. It includes the Function App, Resource Groups, Storage Accounts, and App Service Plan associated with the QR Code Generator project. The Function App named functionapp-khushi-01 is clearly visible, confirming its deployment.

## Screenshot : Azure Portal - Resources Overview



The screenshot shows the Azure Portal interface for the resource group 'rg-khushi-qr-generator-01'. The left sidebar contains navigation options like Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, Cost Management, Monitoring, Automation, and Help. The main area displays the 'Essentials' section with subscription details (Subscription ID: bac17140-4441-497d-98ab-773124840802, Location: West Europe) and a table of resources. The resources table lists the following items:

Name	Type	Location
appinsight-khushi-01	Application Insights	West Europe
Application Insights Smart Detection	Action group	Global
as-plan-khushi-funcapp-01	App Service plan	West Europe
functionapp-khushi-01	Function App	West Europe
sakushiqfuncapp01	Storage account	West Europe

### Description:

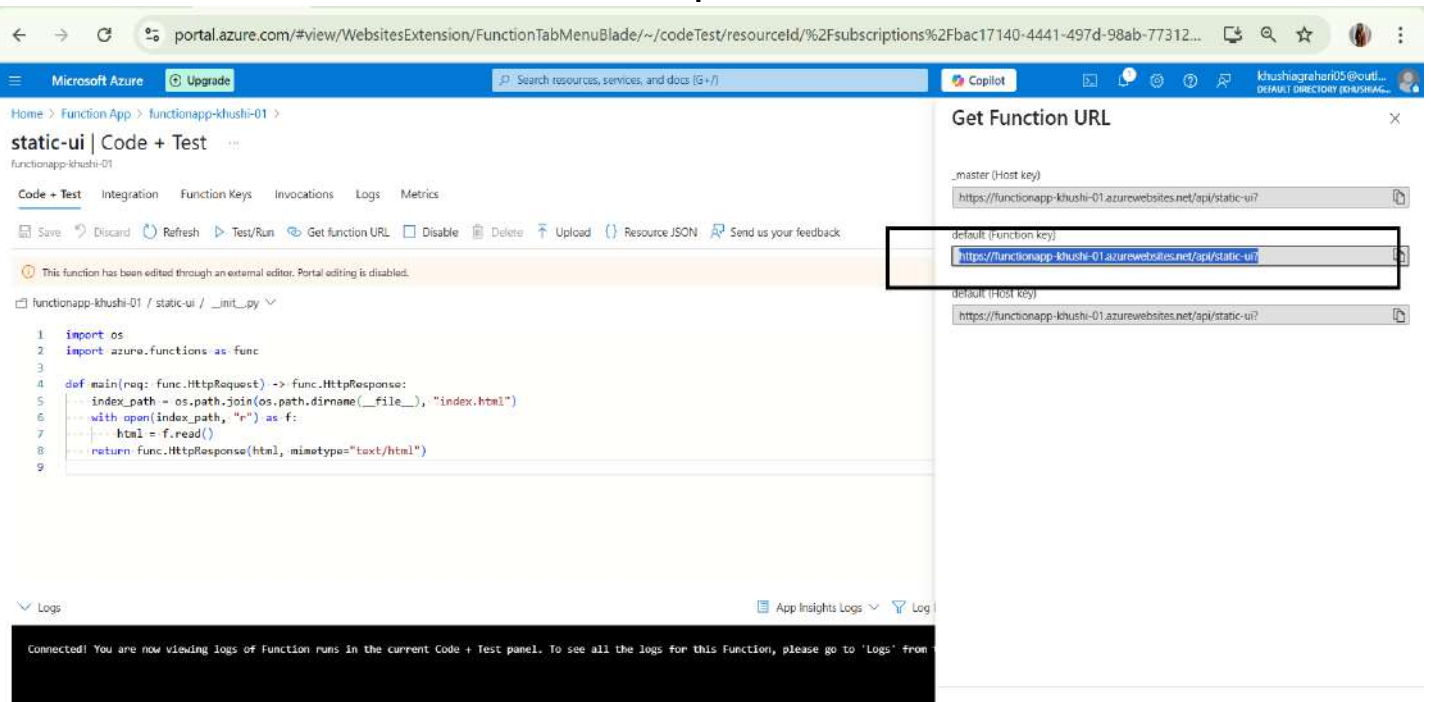
Below screenshot displays the **static-ui function code** within the Azure Portal. The function is written in Python and serves the index.html file as an HTTP response using Azure Functions. On the right side, the

**Function URL** is shown:

**<https://functionapp-khushi-01.azurewebsites.net/api/static-ui>**

This confirms that the function is publicly accessible and is used to host the front-end UI of the QR Code Generator project.

## Screenshot 5: Azure Function – Code and Endpoint URL



The screenshot displays the 'static-ui | Code + Test' view for the function 'functionapp-khushi-01'. The left pane shows the Python code for the function, which reads the 'index.html' file and returns it as an HTTP response. The right pane shows the 'Get Function URL' dialog, which lists the function keys and their corresponding URLs. The URL for the 'default (function key)' is highlighted:

```

1 import os
2 import azure.functions as func
3
4 def main(req: func.HttpRequest) -> func.HttpResponse:
5     index_path = os.path.join(os.path.dirname(__file__), "index.html")
6     with open(index_path, "r") as f:
7         html = f.read()
8     return func.HttpResponse(html, mimetype="text/html")
9

```

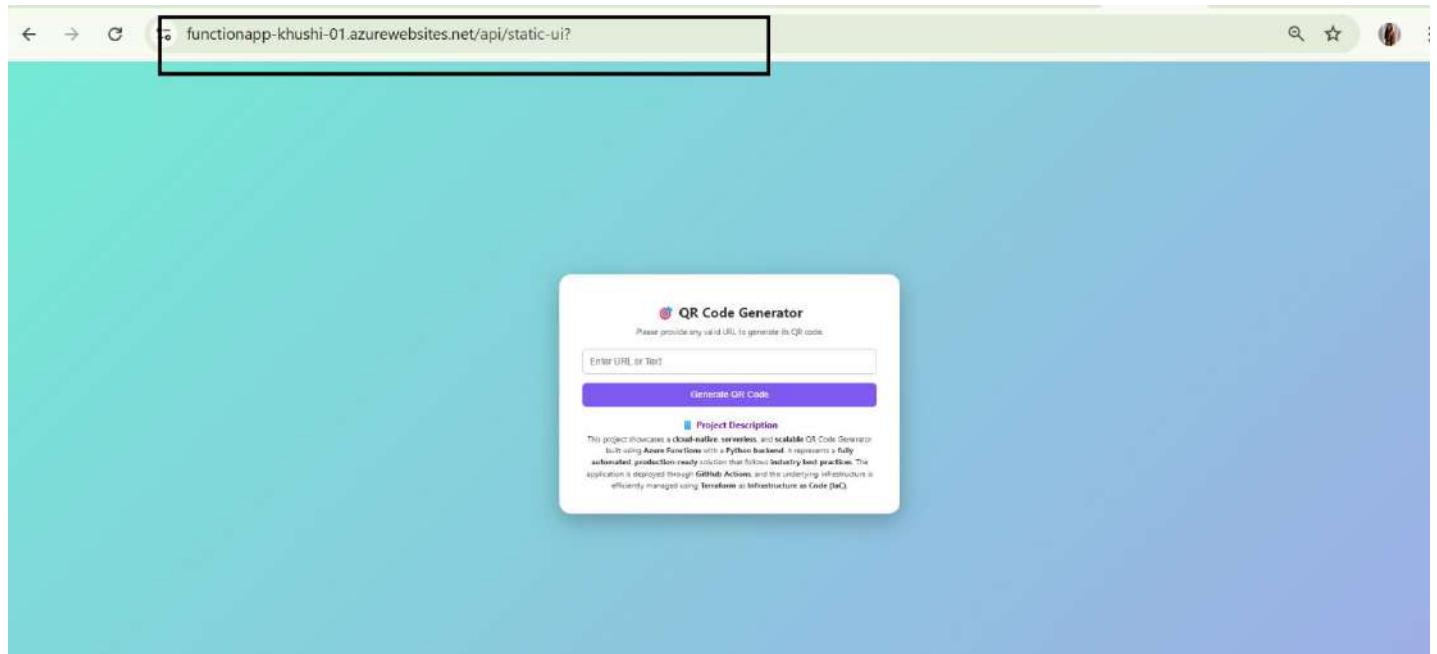
The URL shown in the dialog is: **<https://functionapp-khushi-01.azurewebsites.net/api/static-ui>**



**Description:**

This screenshot shows the hosted QR Code Generator web interface accessible through the URL: <https://functionapp-khushi-01.azurewebsites.net/api/static-ui>

The UI allows users to input a URL or text and generate a QR code. This interface proves that the static-ui function was deployed and is working as expected.

**📸 Screenshot 4: Running QR Code Generator Web UI****Description:**

Below screenshot captures the complete lifecycle of the backend-api Azure Function in action:

**Top Section (Azure Portal):**

The Code + Test tab shows the Python code of the **backend-api function**. This function takes a query parameter (data) from the HTTP request, generates a QR code image using the qrcode library, and returns it as a PNG image in the response.

**Function URL Panel:**

On the right, the Function URL dialog is open, displaying the live endpoint of the function:

**<https://functionapp-khushi-01.azurewebsites.net/api/backend-api>**

**Bottom Section (Browser Output):**

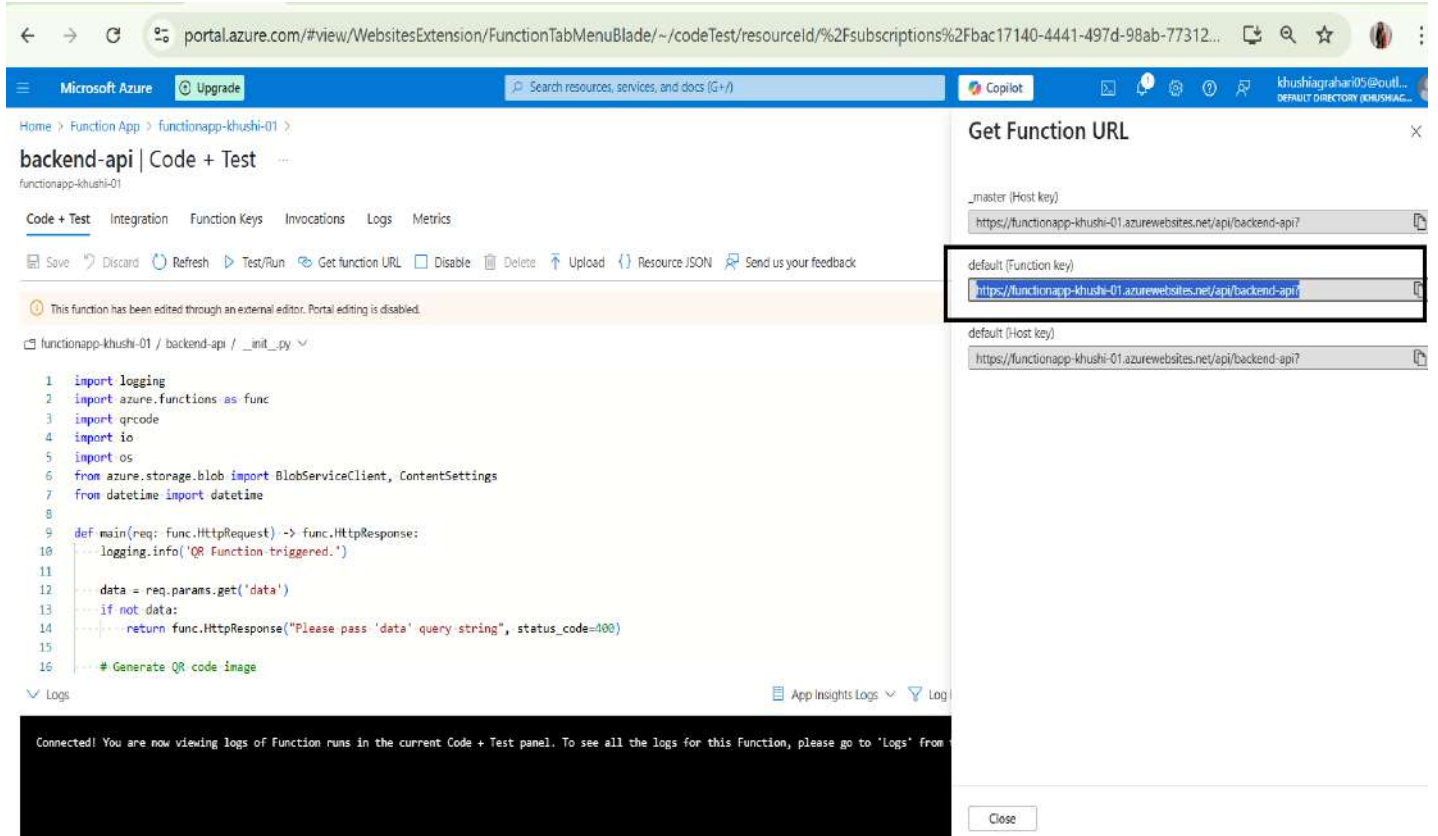
The backend function is triggered using a query string:

**[?data=https://khushi.com](https://khushi.com)**

It successfully returns a QR code image with the message **"QR Code Generated Successfully!"** and an option to download it.

This proves that the backend logic works independently without the UI (static-ui) and can directly serve QR codes via API — confirming a clean separation between backend and frontend in this serverless application.

### Screenshot 6: Azure Function – backend-api (Code Execution + QR Output)



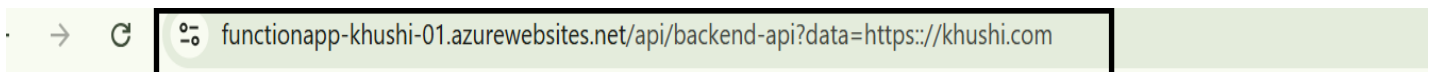
The screenshot shows the Azure Portal interface for the 'backend-api' function. The code editor displays the following Python code:

```

1 import logging
2 import azure.functions as func
3 import qrcode
4 import io
5 import os
6 from azure.storage.blob import BlobServiceClient, ContentSettings
7 from datetime import datetime
8
9 def main(req: func.HttpRequest) -> func.HttpResponse:
10     logging.info('QR Function triggered.')
11
12     data = req.params.get('data')
13     if not data:
14         return func.HttpResponse("Please pass 'data' query string", status_code=400)
15
16     # Generate QR code image
  
```

The 'Get Function URL' panel on the right shows the function URL: `https://functionapp-khushi-01.azurewebsites.net/api/backend-api?`. The 'default (Function key)' and 'default (Host key)' sections both show the same URL.

The logs panel at the bottom shows a message: "Connected! You are now viewing logs of Function runs in the current Code + Test panel. To see all the logs for this function, please go to 'Logs' from..."



The screenshot shows a web browser address bar with the URL: `functionapp-khushi-01.azurewebsites.net/api/backend-api?data=https://khushi.com`. The URL is highlighted with a black box.

**QR Code Generated Successfully!**

[Download QR Code](#)



## GitHub Repository View

### Description:

This screenshot displays the public GitHub repository containing the entire codebase for the QR Code Generator Function App. It includes:

- Infrastructure as Code (Infra/) written using Terraform
- Azure deployment workflows under .github/workflows/ using GitHub Actions
- Function app source code (qr-generator/) including backend Python function and frontend UI
- Deployment automation script under scripts/

## Version Control using Git & GitHub

To manage code changes efficiently and maintain a clean version history, we used **Git** as the version control system and **GitHub** as the remote repository. This helped in seamless collaboration, tracking every update made to the infrastructure and application code.

Below are the main commands used:

---

### 1. `git add .`

This command stages all the modified and new files in the working directory for commit.

```
git add .
```

◆ *Purpose:* Prepare files for committing.

◆ *In our case,* it staged `Infra/main.tf` and `qr-generator/static-ui/index.html`.

---

### 2. `git commit -m "add source and version in infra and modify ui"`

This command commits the staged changes with a meaningful message describing what was changed.

```
git commit -m "add source and version in infra and modify ui"
```

◆ *Purpose:* Create a snapshot of staged changes.

◆ *In our case,* it logged changes made to the Terraform infra and UI files.

---

### 3. `git push`

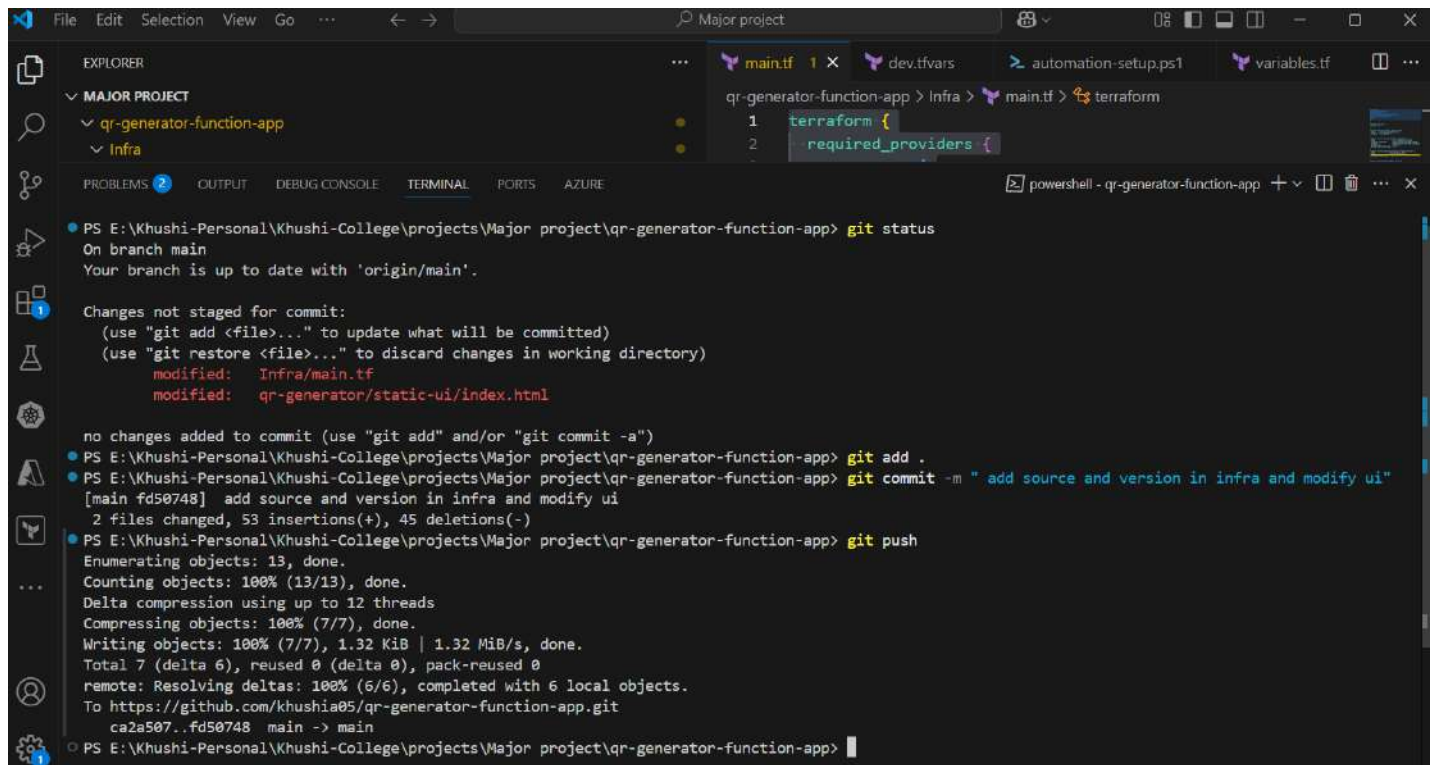
This command pushes the local commits to the remote repository on GitHub.

```
git push
```

- ◆ *Purpose:* Synchronize local commits with GitHub so the latest code is available online.
- ◆ *In our case,* the changes were pushed to the main branch of the repo:

<https://github.com/khushia05/qr-generator-function-app>

✧ **Figure:** *Git terminal showing successful commit and push. This confirms version control was properly used during the project to manage code changes.*



```

PS E:\Khushi-Personal\Khushi-College\projects\Major project\qr-generator-function-app> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Infra/main.tf
        modified:   qr-generator/static-ui/index.html

no changes added to commit (use "git add" and/or "git commit -a")
PS E:\Khushi-Personal\Khushi-College\projects\Major project\qr-generator-function-app> git add .
PS E:\Khushi-Personal\Khushi-College\projects\Major project\qr-generator-function-app> git commit -m "add source and version in infra and modify ui"
[main fd50748] add source and version in infra and modify ui
 2 files changed, 53 insertions(+), 45 deletions(-)
PS E:\Khushi-Personal\Khushi-College\projects\Major project\qr-generator-function-app> git push
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 12 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 1.32 KiB | 1.32 MiB/s, done.
Total 7 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), completed with 6 local objects.
To https://github.com/khushia05/qr-generator-function-app.git
 ca2a507..fd50748  main -> main
PS E:\Khushi-Personal\Khushi-College\projects\Major project\qr-generator-function-app>
  
```

- **GitHub Repo URL:**  
<https://github.com/khushia05/qr-generator-function-app>

### Description:

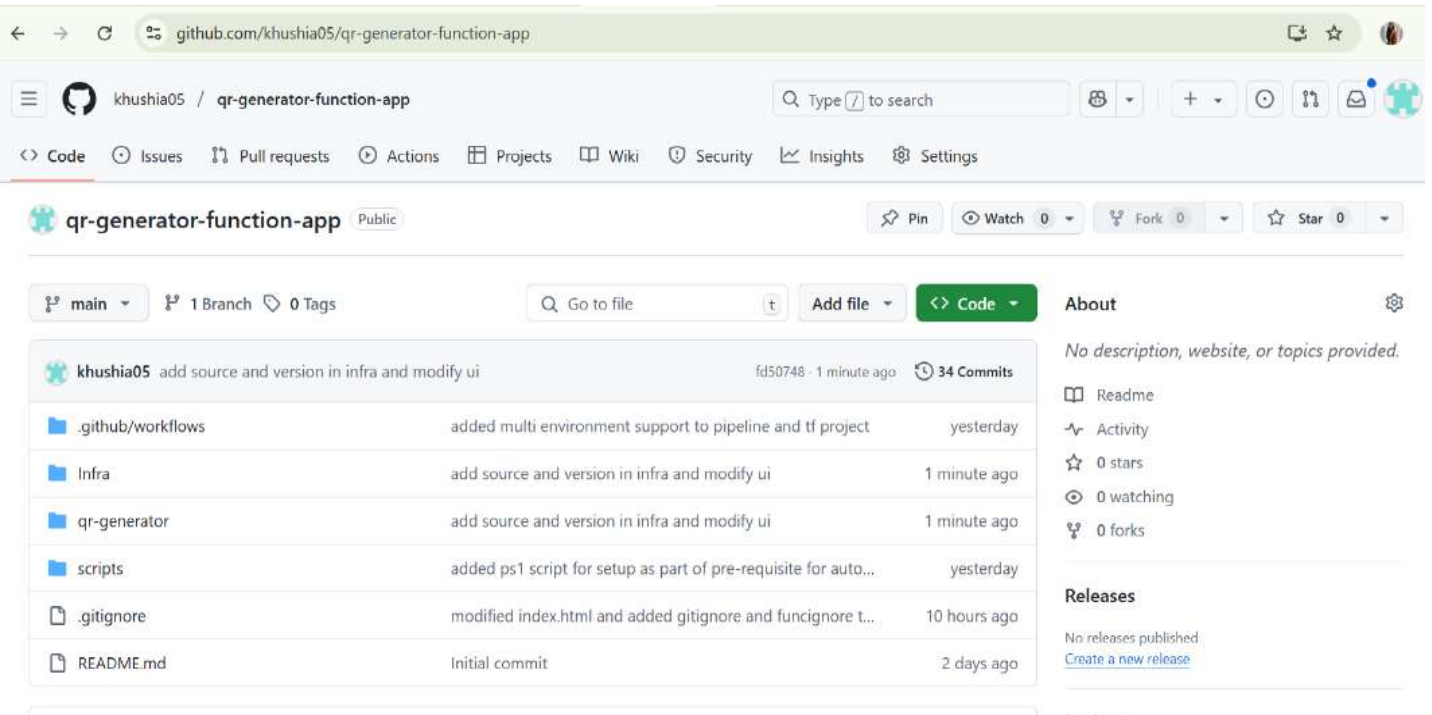
Below screenshot displays the public GitHub repository qr-generator-function-app owned by user **khushia05**. It contains the complete source code, automation scripts, and infrastructure-as-code for the Azure Function-based QR Code Generator project.

### Key repository contents:

- .github/workflows: Contains CI/CD pipeline files for GitHub Actions.
- Infra: Holds Terraform configuration files used to provision Azure resources.
- qr-generator: Contains backend and frontend Azure Function code (backend-api and static-ui).
- scripts: Includes helper PowerShell scripts for automation and setup.
- README.md: Project's initial documentation file.

.gitignore: Ensures irrelevant or sensitive files are excluded from version control.

### Screenshot 7: GitHub Repository – qr-generator-function-app



#### 4. GitHub Actions CI/CD Workflow Successful pipeline execution includes:

- Terraform deployment of infrastructure
- Python app code deployment as ZIP to Azure

#### Description:

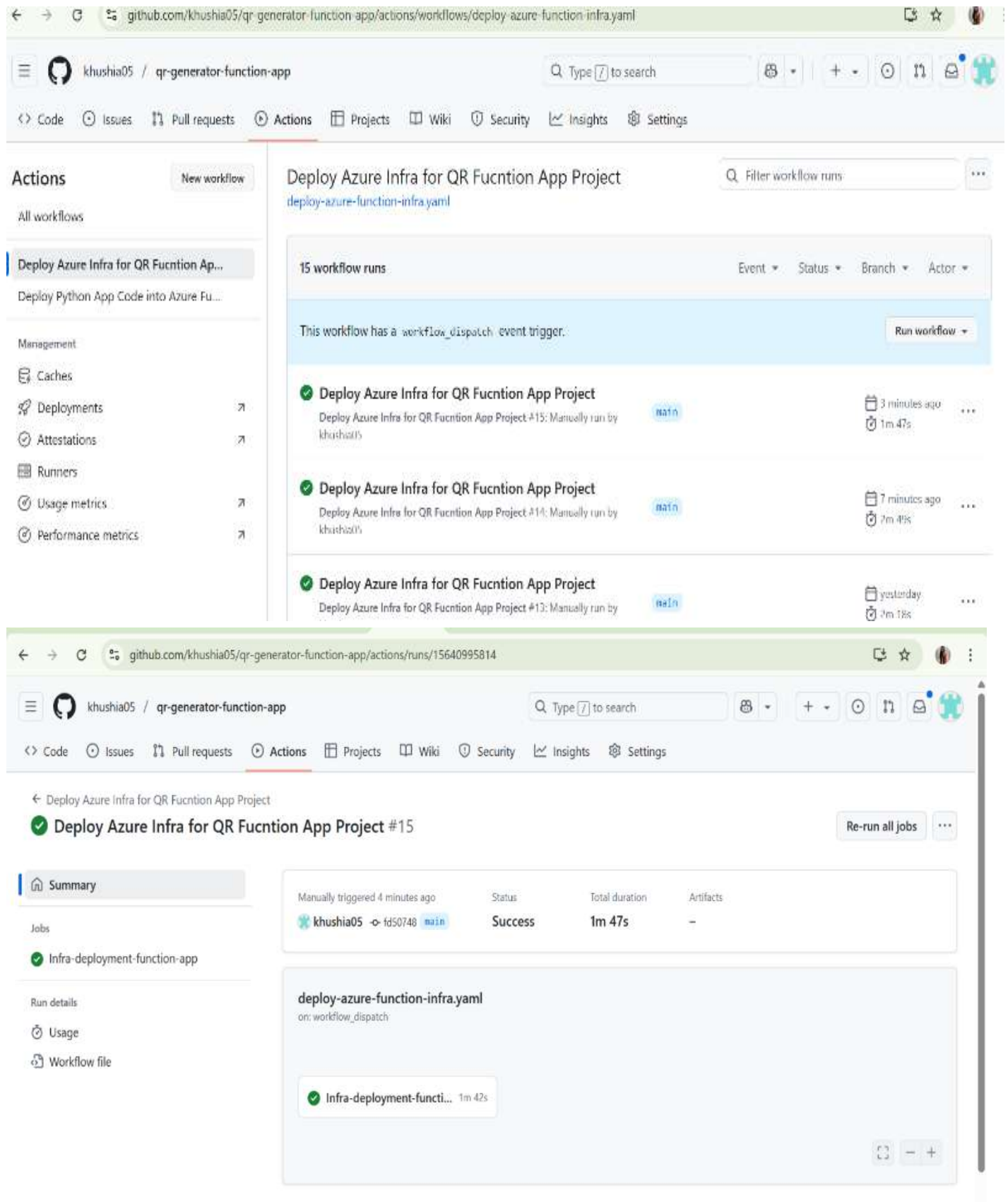
Below screenshot shows the GitHub Actions workflow panel for the project **qr-generator-function-app**, specifically the workflow named: **Deploy Azure Infra for QR Function App Project**.

#### Highlights:

- The workflow is defined in the YAML file: **deploy-azure-function-infra.yaml**.
- It uses the **workflow\_dispatch** trigger, meaning it is executed manually by the developer as needed.
- The screenshot displays successful workflow runs on the **main branch**, manually triggered by the user **khushia05**.
- Each run takes approximately **1–3 minutes**, showing efficient automation and provisioning of Azure infrastructure using **Terraform** or related tools.

- This confirms that **Continuous Deployment (CD)** is integrated into the project using **GitHub Actions** for infrastructure provisioning—making the project **cloud-ready** and **automatable** from code to infrastructure.

### Screenshot: GitHub Actions – Deploy Azure Infra for QR Function App Project



The screenshot displays the GitHub Actions interface for the repository `khushia05 / qr-generator-function-app`. The workflow `deploy-azure-function-infra.yaml` is selected, showing 15 workflow runs. The interface includes a sidebar with navigation options like Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main content area shows the workflow details, including a list of runs with their status (Success), duration (1m 47s), and trigger (Manually run by khushia05).

**Workflow Runs Summary:**

Run ID	Status	Duration	Trigger
15640995814	Success	1m 47s	Manually run by khushia05
15640995813	Success	1m 47s	Manually run by khushia05
15640995812	Success	1m 47s	Manually run by khushia05

**Workflow Details:**

The workflow `deploy-azure-function-infra.yaml` is triggered by `workflow_dispatch`. It includes a job `infra-deployment-function-app` with a duration of 1m 42s.

**Summary:**

- Manually triggered 4 minutes ago
- Status: Success
- Total duration: 1m 47s
- Artifacts: -

**Jobs:**

- infra-deployment-function-app

**Run details:**

- Usage
- Workflow file



**Description:** Below screenshot displays the **qr-generator-function-app** repository's GitHub Actions workflow panel, specifically showcasing the **Deploy Python App Code into Azure Function** workflow.

### Highlights:

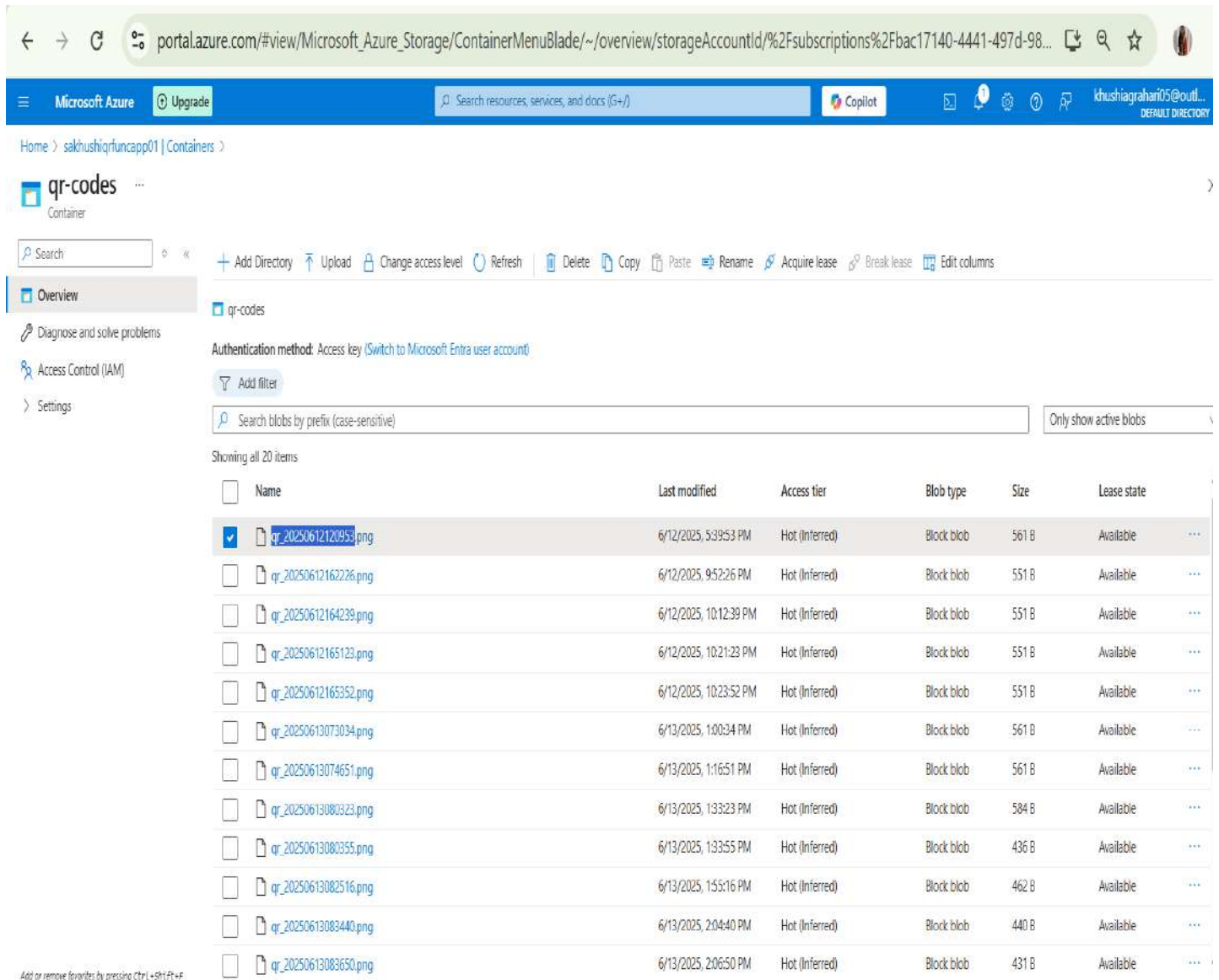
- The workflow is defined in the **deploy-app-code-function-app.yaml** file.
- It is triggered using **workflow\_dispatch**, meaning deployment is manually executed by the developer.
- The screenshot shows successful workflow runs manually triggered by **khushia05**.
- The workflow dropdown allows the developer to **select a branch** and specify the Azure Function App name.
- Each deployment completes within **1–3 minutes**, demonstrating efficient automation.
- **GitHub Actions** automates the deployment of the Python application into **Azure Functions**, ensuring a **cloud-ready** and **scalable** solution.

The screenshot displays the GitHub Actions interface for the repository **khushia05 / qr-generator-function-app**. The **Actions** tab is selected, showing a list of workflows on the left. The workflow **Deploy Python App Code into Azure Function** (defined in `deploy-app-code-function-app.yaml`) is highlighted. The main panel shows a list of 21 workflow runs, with the most recent run (#22) being successful. A modal window is open, allowing the user to select a branch (currently `main`) and specify the **Azure Function App name** (currently `functionapp-khushi-01`). Below the workflow list, the details for run #22 are shown, indicating it was manually triggered 3 minutes ago, succeeded, and took 49 seconds. The **Summary** tab is selected, showing the workflow file `deploy-app-code-function-app.yaml` and the job `build-and-deploy` which completed successfully in 45 seconds.

## Azure Storage Verification

- QR code image stored in Azure Blob Storage after API call
- Verified by checking Storage Account's container

### Screenshot: Azure > Storage Account > Blob Container showing QR image



The screenshot shows the Azure Portal interface for a storage account. The breadcrumb navigation indicates the path: Home > sakshuqifuncapp01 | Containers > qr-codes. The container name 'qr-codes' is displayed at the top left. Below the container name, there is a search bar and a list of actions: Add Directory, Upload, Change access level, Refresh, Delete, Copy, Paste, Rename, Acquire lease, Break lease, and Edit columns. The 'Overview' tab is selected, showing the authentication method as 'Access key' and an 'Add filter' button. A search bar for blobs is present, with the text 'Search blobs by prefix (case-sensitive)'. The table below shows 20 items, all of which are QR code images stored as block blobs. The first item is selected.

Name	Last modified	Access tier	Blob type	Size	Lease state
<input checked="" type="checkbox"/> qr_20250612120953.png	6/12/2025, 5:39:53 PM	Hot (Inferred)	Block blob	561 B	Available
<input type="checkbox"/> qr_20250612162226.png	6/12/2025, 9:52:26 PM	Hot (Inferred)	Block blob	551 B	Available
<input type="checkbox"/> qr_20250612164239.png	6/12/2025, 10:12:39 PM	Hot (Inferred)	Block blob	551 B	Available
<input type="checkbox"/> qr_20250612165123.png	6/12/2025, 10:21:23 PM	Hot (Inferred)	Block blob	551 B	Available
<input type="checkbox"/> qr_20250612165352.png	6/12/2025, 10:23:52 PM	Hot (Inferred)	Block blob	551 B	Available
<input type="checkbox"/> qr_20250613073034.png	6/13/2025, 1:00:34 PM	Hot (Inferred)	Block blob	561 B	Available
<input type="checkbox"/> qr_20250613074651.png	6/13/2025, 1:16:51 PM	Hot (Inferred)	Block blob	561 B	Available
<input type="checkbox"/> qr_20250613080323.png	6/13/2025, 1:33:23 PM	Hot (Inferred)	Block blob	584 B	Available
<input type="checkbox"/> qr_20250613080355.png	6/13/2025, 1:33:55 PM	Hot (Inferred)	Block blob	436 B	Available
<input type="checkbox"/> qr_20250613082516.png	6/13/2025, 1:55:16 PM	Hot (Inferred)	Block blob	462 B	Available
<input type="checkbox"/> qr_20250613083440.png	6/13/2025, 2:04:40 PM	Hot (Inferred)	Block blob	440 B	Available
<input type="checkbox"/> qr_20250613083650.png	6/13/2025, 2:06:50 PM	Hot (Inferred)	Block blob	431 B	Available

Add or remove favorites by pressing Ctrl+Shift+F



✔ Result: QR code stored as expected

3. Error Handling Tests Scenarios:

- Blank input → proper validation alert shown
- Invalid/corrupt request → error handled gracefully

✔ Result: System handled all common errors reliably

4. Deployment Testing with GitHub Actions

- Every push triggered the CI/CD pipelines
- Infra deployed using Terraform
- App code zipped and deployed using Azure CLI

📸 Screenshot: GitHub Actions pipeline showing successful run






✔ Result: Automated deployment succeeded without manual intervention

Test Summary Table:

Test Area	Status	Remarks
UI Load Test	✔	Passed HTML page rendered successfully
Backend API Test	✔	Passed QR code returned for valid input
Azure Storage Output	✔	Passed Image saved in blob storage
Input Validation	✔	Passed Blank input handled gracefully
GitHub CI/CD Workflow	✔	Passed Pipelines ran without any errors






## ADVANTAGES / FEATURES

This project offers the following key advantages:

-  **Fully Automated Deployment:** The entire infrastructure and application deployment is automated using GitHub Actions and Terraform, eliminating manual steps.
-  **Reusable Infrastructure:** Infrastructure as Code (IaC) ensures the same setup can be reused or replicated across environments (e.g., dev, test, prod).
-  **Low Human Error:** Automation minimizes the chances of configuration errors or manual mistakes.
-  **Real-world DevOps Implementation:** Implements CI/CD pipelines and cloud provisioning similar to industry-grade DevOps workflows.
-  **Serverless & Cost-Efficient:** Uses Azure Functions, which scale automatically and cost only per execution — perfect for lightweight applications.

## FUTURE SCOPE

There is significant potential to enhance the project further in the future:

-  **Database Integration:** Store previously generated QR codes along with timestamps and user info for analytics or user history.
-  **User Authentication:** Add login/signup functionality to provide personalized features like saved QR code history.
-  **Custom Branding:** Allow users to upload a logo or customize QR code colors/styles before generation.
-  **Mobile Responsive UI:** Improve the frontend interface to make it fully responsive and mobile-friendly.
-  **Download Option:** Add a "Download QR Code" button to save the generated image locally with one click.

## **CONCLUSION**

This project successfully demonstrates how a modern web application can be deployed using serverless technology, with full automation through Terraform and GitHub Actions. From generating QR codes in real time to provisioning all necessary Azure infrastructure, the entire system follows best practices in DevOps and Infrastructure as Code (IaC). The learning journey included hands-on experience in writing Terraform scripts, managing GitHub workflows, working with cloud services like Azure Functions and Storage, and building modular backend/frontend logic using Python and HTML.

## **REFERENCES**

The following official documentation and resources were referred to during the project:

<https://learn.microsoft.com/en-us/azure/azure-functions/>

<https://learn.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-python>

<https://developer.hashicorp.com/terraform/docs>

<https://docs.github.com/en/actions>

Azure Free Trial Account:

<https://go.microsoft.com/fwlink/?linkid=2227353&clcid=0x4009&l=en-in&icid=free-search>

