CLOUD COMPUTING

Countdown Timer using Microsoft Azure

Building a dynamic website on Azure with Blob Storage, Functions, CosmosDB, and GitHub Actions and hosting using a custom domain

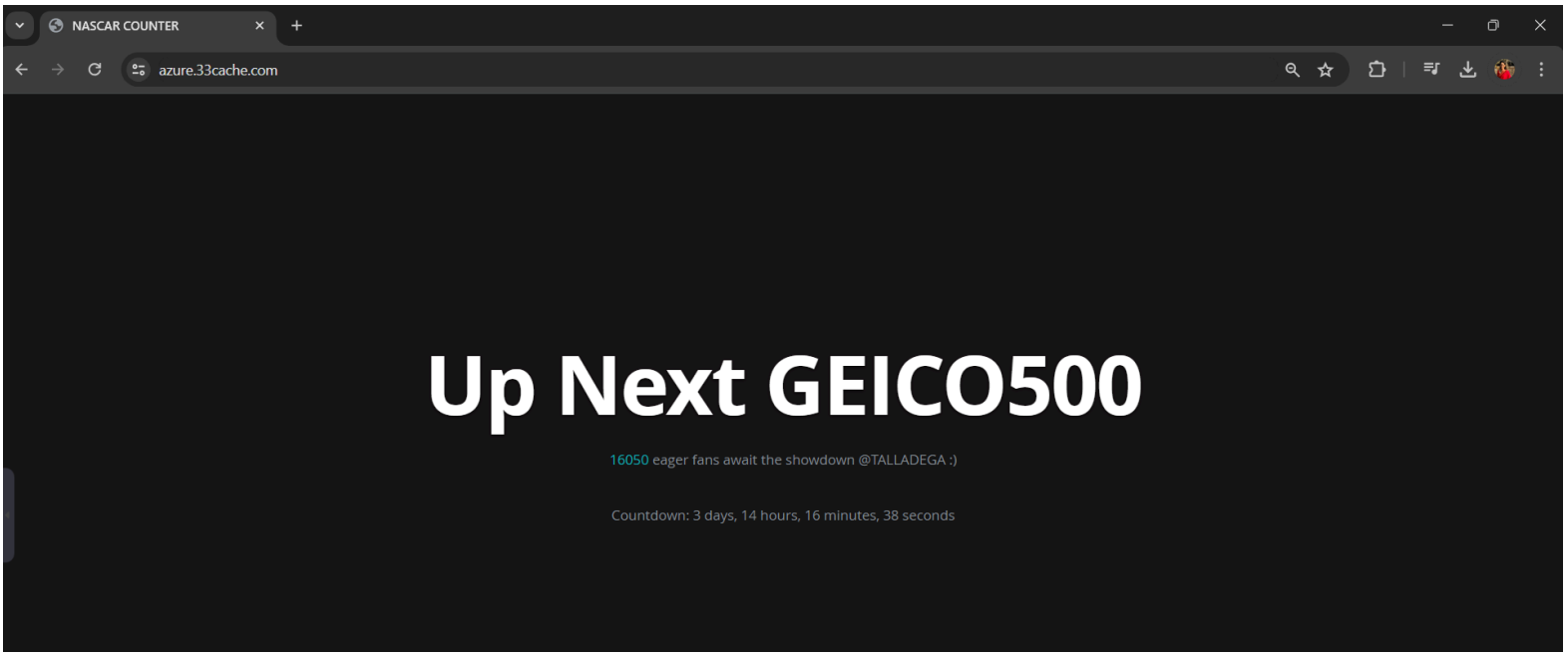
Name	Rashi Rajesh Shetty
Roll no.	36
UID	2309045
Class	MSc BDA Part I

Table of Contents

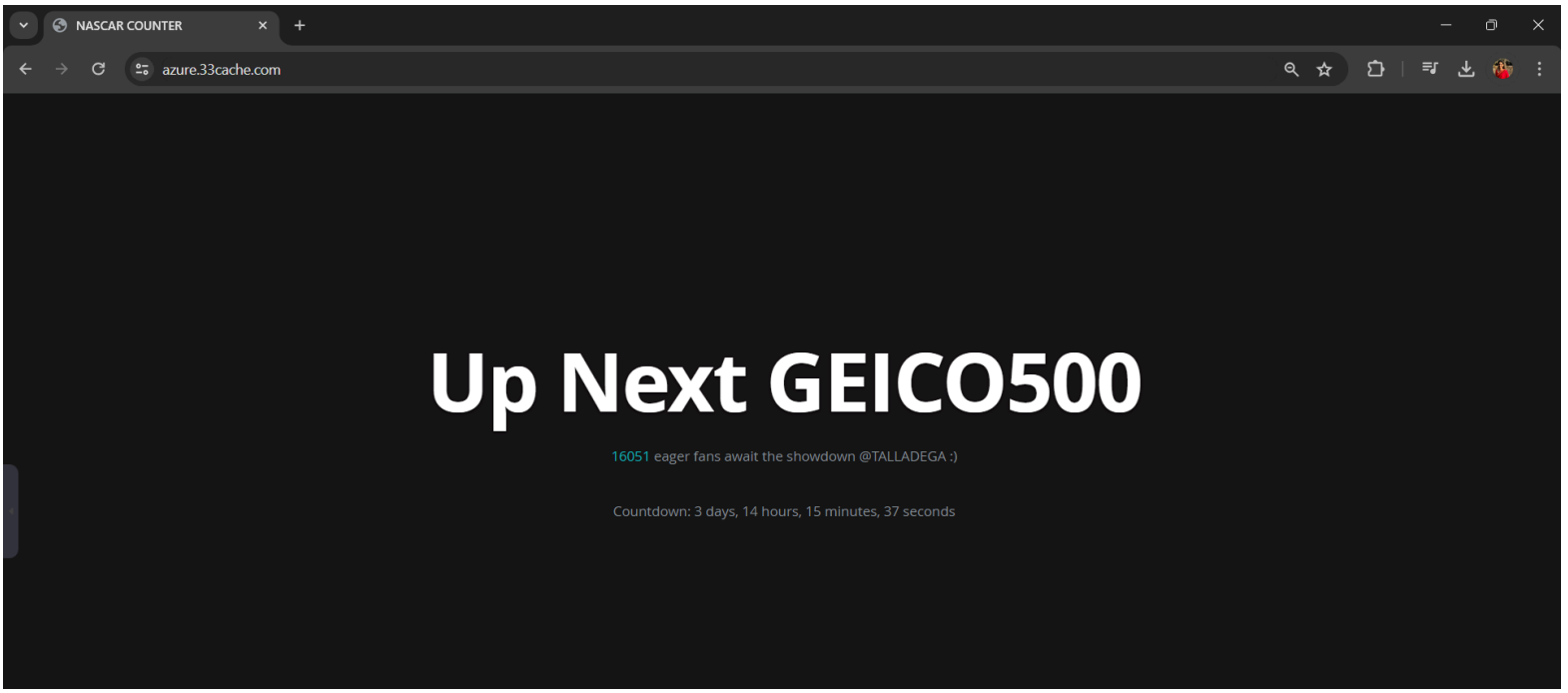
Building a dynamic website on Azure with Blob Storage, Functions, CosmosDB, and GitHub Actions and hosting using a custom domain

I. INTRODUCTION:

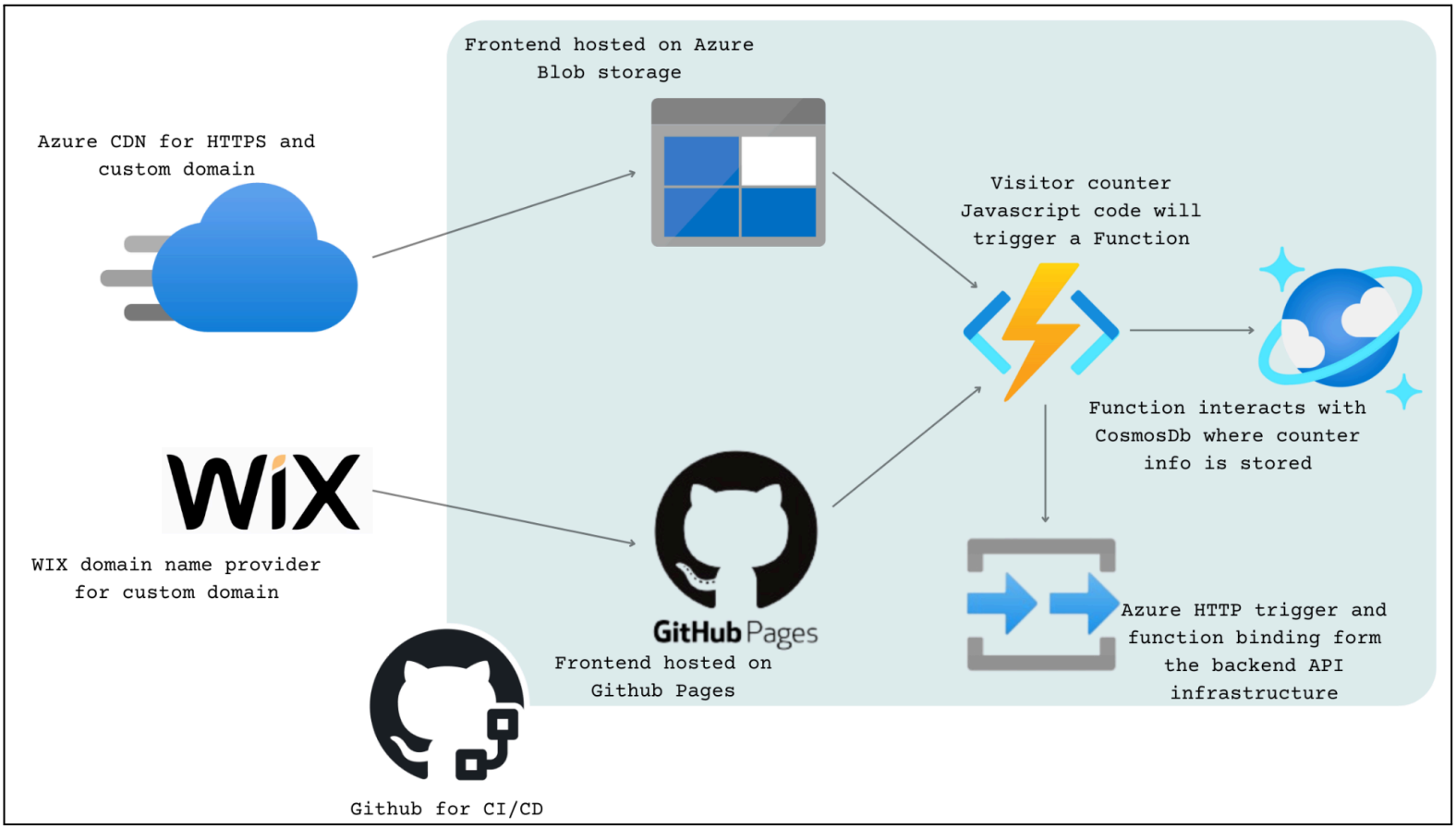
Website: https://azure.33cache.com/
The website serves as a countdown timer for the upcoming NASCAR (motorsports) race. The core feature of the website is a visitor counter, symbolizing the number of fans anticipating the upcoming race. Each visitor to the website increments the counter by one.



On refresh:



II. ARCHITECTURE:



III. PROJECT OBJECTIVE:

The primary goal of this project is to create a dynamic website hosted on the Azure platform. Leveraging a combination of Azure services including Blob Storage, Azure Functions, and Cosmos DB. GitHub Pages will serve as the frontend host for the website, facilitating easy deployment and version control.

The website's frontend will be developed using HTML, CSS, and JavaScript. A key feature of the website will be a visitor counter, implemented using JavaScript. This counter will interact with an API deployed on Azure Functions, which will serve as an HTTP trigger. The Azure Function will connect with Cosmos DB, acting as the serverless database to store and manage visitor counter information securely.

Alternatively, instead of GitHub Pages, the project can explore the option of utilizing Azure CDN (Content Delivery Network) for custom domain support for website deployed on Azure blob storage. Overall, the project aims to deliver a high-quality, scalable website hosted on Azure, with a focus on functionality, performance, and reliability.

IV. PRE-REQUISITES:

1. Microsoft Azure Account:

to deploy our functions and set up Azure CosmosDB. Acquire an active Azure subscription to create and manage Azure resources

2. .NET Core 6.0.421 SDK

cross-platform open source framework to develop Azure Functions

3. Azure Functions Core Tools:

for local development and debugging of Azure Functions

4. Visual Studio Code:

code editor for frontend and backend programming

5. Azure CLI:

run commands to create and manage Azure resources

6. C# Extension:

VS Code extension that provides tools and support for programming in C#

7. Azure Functions, Storage, Resources Extension:

to run the website locally and then deploy to the Azure portal using VS Code

8. GitHub Account:

for version control and hosting the frontend on GitHub Pages

V. WORKING:

1. Creation of Azure Function:

- An Azure Function is created to serve as the backend logic for the visitor counter functionality.
- Configured as an HTTP trigger, the function is invoked upon receiving HTTP requests from users visiting the website.

2. Connection to Cosmos DB:

- The Azure Function is connected to Cosmos DB using bindings, facilitating seamless data exchange between the function and the database.

3. HTTP Trigger:

- As an HTTP trigger, the Azure Function exposes an API endpoint accessible via HTTP requests initiated by users visiting the website.

4. Fetching Visitor Count from Cosmos DB:

- Upon receiving an HTTP request, the Azure Function queries Cosmos DB to retrieve the current visitor count stored in the database.

5. Displaying Visitor Count on Frontend:

- Upon retrieving the count value, the Azure Function returns it as a response to the HTTP request.
- The frontend JavaScript code fetches this count value from the provided API endpoint URL and updates the visitor counter displayed on the website.

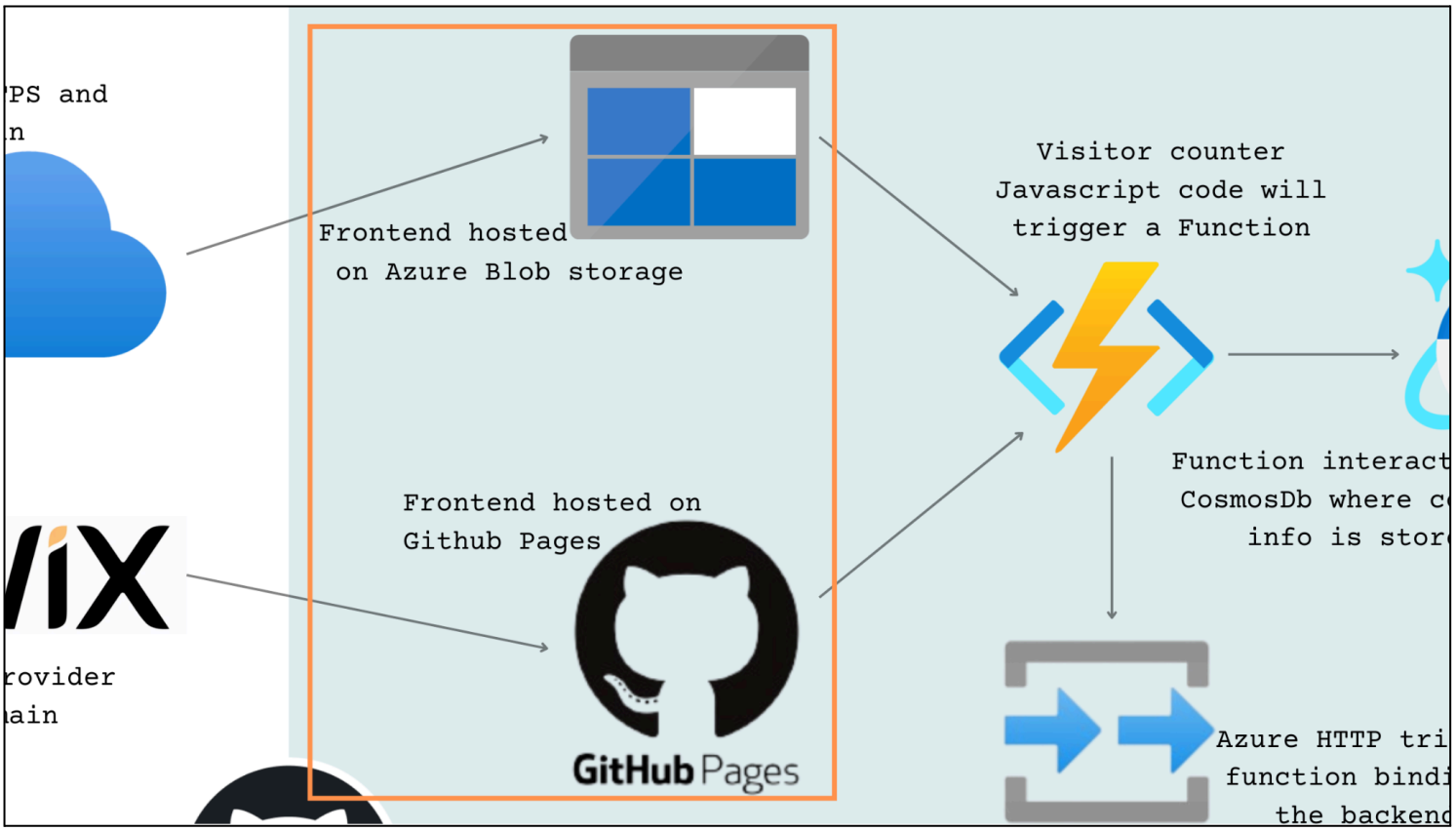
6. Incrementing Visitor Count and Updating Cosmos DB:

- Following the retrieval of the current count value, the Azure Function increments it by one to reflect the new visitor.
- The updated count value is then stored back into Cosmos DB, replacing the previous count value.

7. Persistence of Count Value:

- The updated count value remains stored in Cosmos DB until the next HTTP trigger.
- With each user visit triggering the Azure Function, the count value is fetched, incremented, and updated in Cosmos DB, ensuring the continuous accuracy of the visitor count.

VI. BUILDING THE FRONTEND:



Setting up version control

Create a GitHub repository, clone the starter code, and understand the project structure.

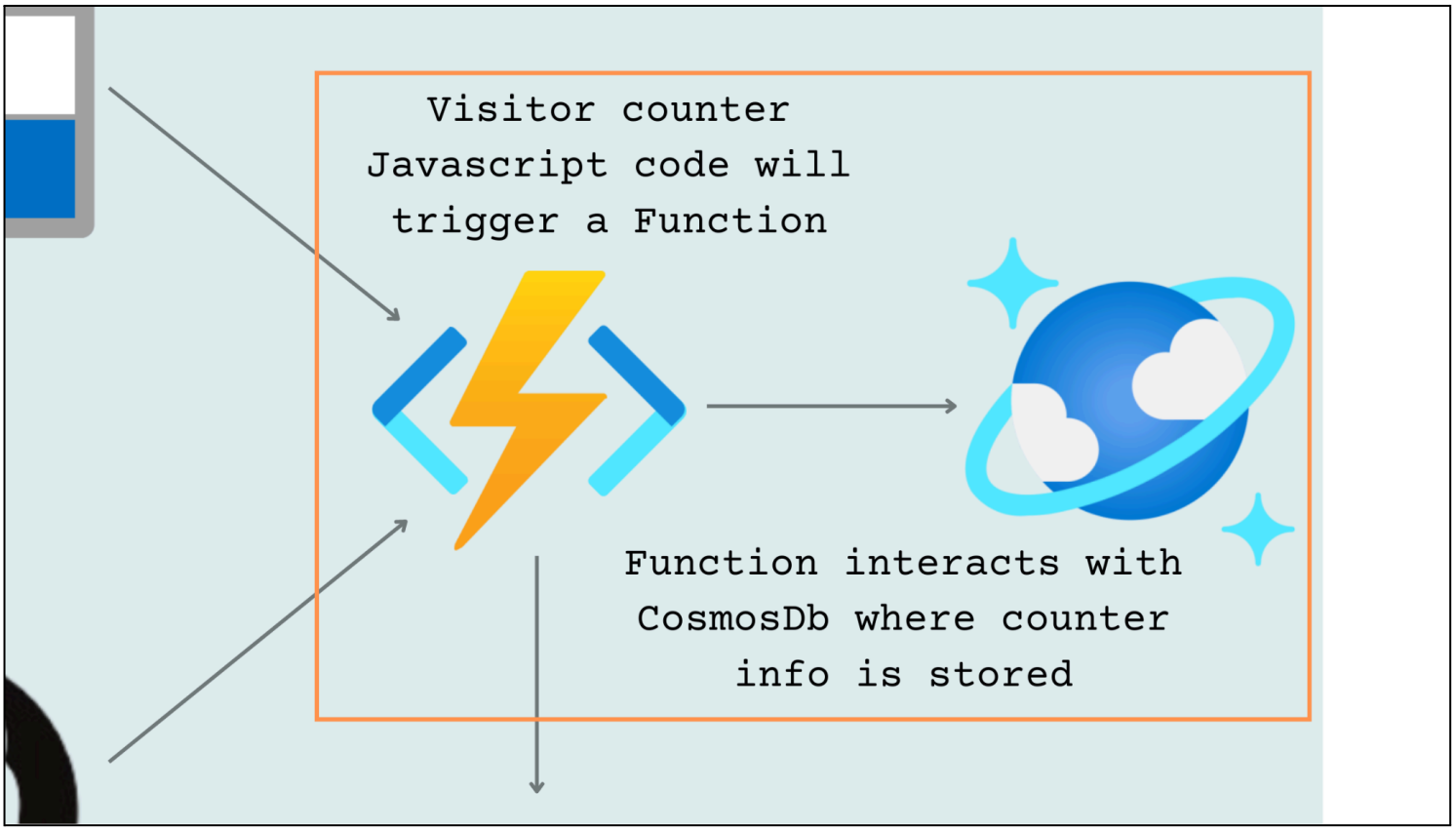
Update the HTML and implement a counter

Update index.html with countdown functionality and write the JavaScript code (main.js) for the visitor counter.

Test locally and push changes to GitHub

View the website locally and push all changes to GitHub.

VII. BUILDING THE BACKEND:



Setting up Cosmos DB resources

Create CosmosDB account, database, container, database, and container, and data.

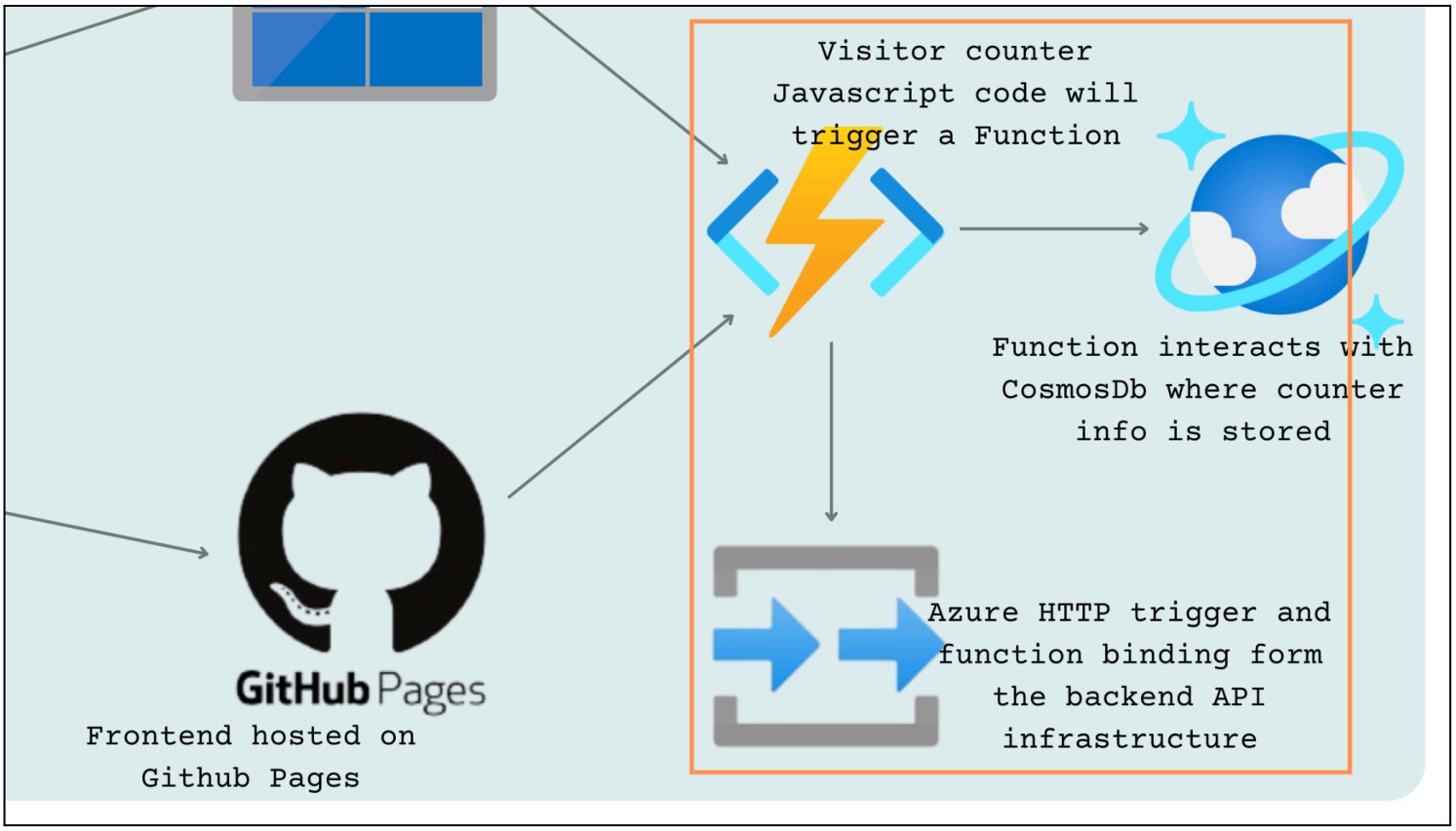
Setting up Azure Function

Create Azure Function to interact with our Cosmos DB counter data.

Test locally

Deploy the Function locally and make sure the counter data in the browser and website is visible locally.

VIII. AZURE FUNCTIONS:



Azure Functions is a serverless solution that helps develop event-driven pieces of code without worrying about the architecture behind the pieces of code. It plays a crucial role in implementing the backend logic for the visitor counter functionality.

HTTP Trigger

- An HTTP trigger in Azure Functions is used to create an API endpoint that increments the visitor counter every time a user visits the website.
- This trigger listens for HTTP requests and executes the specified function in response.

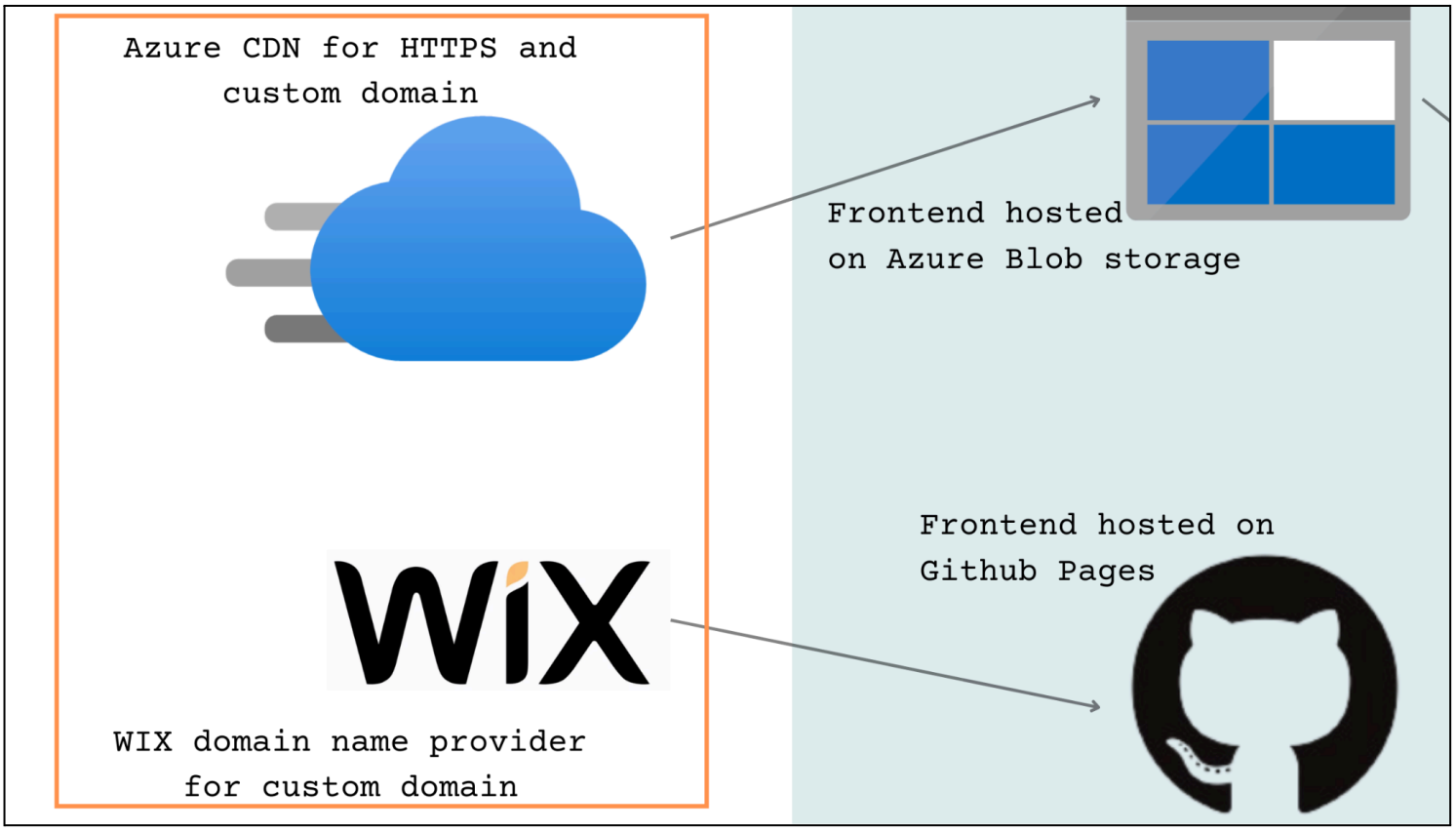
Integration with Cosmos DB

- Azure Functions interacts with Cosmos DB to retrieve and update the visitor counter data securely.
- Upon receiving a request from the frontend, the function fetches the current visitor count from Cosmos DB, increments it, and then updates the database with the new count.

Scalability and Cost-Efficiency

- Azure Functions automatically scale based on demand, allowing the website to handle varying levels of traffic without manual intervention.
- With the serverless model, the amount is charged only for the resources consumed during function execution, making it a cost-effective solution for our project.

IX. DEPLOYING TO AZURE:



Deploy Azure Function

The locally created Azure Function is deployed to the Azure FunctionApp in the portal, fetches the Function URL, and updates the JavaScript code (main.js) with it.

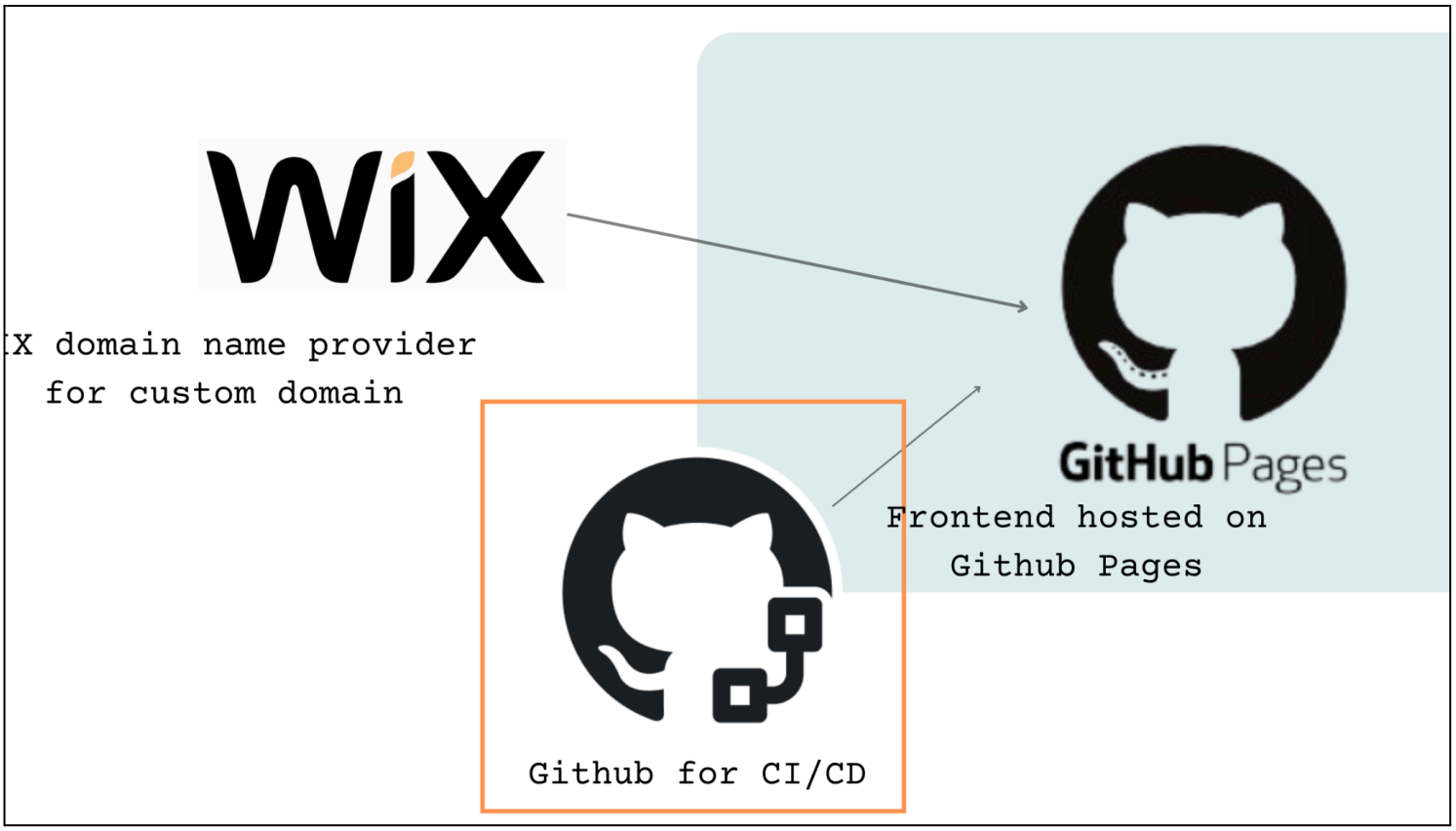
Deploy to Blob Container / Github Pages

Deploy our frontend of the website to the newly created Azure blob container or push it into the GitHub repository.

Host the website successfully online

Setup Azure CDN for HTTPS and custom domain support or setup Github pages and redirect the default GitHub domain (github.io) to personal custom domain (33cache.com).

X. BUILDING CI/CD PIPELINE:



Create frontend workflow

Develop GitHub workflow responsible for deploying the website frontend.

Implement unit testing

Create unit tests to test Azure Functions code as part of its deployment workflow.

Create our backend workflow

Create the GitHub workflow responsible for deploying the website backend.

XI. CONCLUSION:

In conclusion, the development of the dynamic website for displaying a countdown to NASCAR races implementing a visitor counter function leverages Azure services such as Blob Storage, Azure Functions, and Cosmos DB, creating a scalable and reliable solution.

The implementation of the visitor counter adds an interactive and dynamic element to the website. Additionally, the integration of GitHub for version control, workflow, and deployment streamlines the development process and ensures the maintainability of the codebase.

Looking ahead, there are ample opportunities for further enhancement and expansion of the website. Future iterations could focus on enhancing the user experience, implementing real-time updates, integrating analytics for insights, exploring mobile app integration, and exploring monetization opportunities.