AI- POWERED CAMPUS NAVIGATION SYSTEM USING AZURE

CS19741 CLOUD COMPUTING
(FINAL YEAR, 7TH SEMESTER)

Submitted by

MAHIMA R (220701156)
SHAUN PAUL MOSES (220701226)
VIGNESH S (220701318)

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

NOVEMBER 2025

BONAFIDE CERTIFICATE

Certified that this project report titled **“AI- POWERED NAVIGATION SYSTEM USING AZURE”** is the Bonafide work of R **MAHIMA (220701156), SHAUN PAUL (220701226), VIGNESH (220701318)** in CS19741 – Cloud Computing during the year 2025-2026, who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr. E. M Malathy	**MS.M SANTHIYA**
Professor &	**Supervisor**
Head of The Department	**Associate Professor**
Department of Computer Science	Department of Computer Science
and Engineering	and Engineering
Rajalakshmi Engineering College	Rajalakshmi Engineering College

Submitted to Project Viva Voce Examination held on ____

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.**, and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN, Ph.D.,** for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.**, our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. MALATHY E.M, Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for her guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide, **MS. M. SATHIYA,** Associate Professor, Department of Computer Science and Engineering for his valuable guidance throughout the course of the project.

MAHIMA R
SHAUN PAUL MOSES
VIGNESH S

ABSTRACT

The rapid growth of artificial intelligence, cloud computing, and automated deployment technologies has transformed how modern web applications are built and delivered. Leveraging these advancements, this project presents an AI-Powered Campus Navigation System designed specifically for Rajalakshmi Engineering College. The system uses the power of Microsoft Azure, Azure AI Foundry, ChatGPT-4o, containerized deployment using Docker, GitHub CI/CD pipelines, and graph-based routing algorithms to deliver an intelligent, scalable, and user-friendly navigation experience across the college campus. The solution functions as a smart web platform that helps students, faculty, and visitors easily locate academic blocks, laboratories, libraries, auditoriums, and other facilities with AI-generated explanations.

At the center of the system is the Azure AI Foundry model, which provides the intelligent layer of the application. Instead of returning route data in raw numerical form, the AI transforms computed paths into natural, conversational guidance such as "From the Main Gate, walk straight to the Auditorium and take a left toward the Library." This natural-language interaction bridges the gap between technical output and user understanding, making the system more intuitive for new students and visitors. The AI endpoint, deployment ID, and credentials are securely integrated within the backend using environment variables for safe access.

The navigation logic is built on a graph representation of the college campus, where buildings, pathways, and intersections are represented as nodes and edges. The system incorporates multiple algorithms including Breadth-First Search (BFS), Depth-First Search (DFS), Dijkstra's Algorithm, and an Accessible Route Algorithm. This multi-algorithm design allows users to choose routes based on different criteria—shortest distance, step-by-step traversal, educational exploration, or wheelchair-friendly movement. Such flexibility makes the system both academically valuable and practically effective.

To ensure consistency and reliability across development and deployment environments, the entire application is Docker containerized. The container includes all dependencies, configurations, and runtime settings required to run the backend server, ensuring that the software runs identically regardless of the machine or platform. The Docker image is stored securely in the Azure Container Registry (ACR) and can be redeployed or scaled without manual reconfiguration. This setup eliminates environment-related issues and supports future expansion of the system, such as adding more AI models or introducing real-time location tracking.

A major highlight of the project is the implementation of a full CI/CD pipeline using GitHub Actions. Whenever new code is committed, GitHub automatically builds the Docker image, pushes it to the Azure Container Registry, and deploys it to Azure App Service. This ensures quick, reliable, and error-free deployments, enabling rapid iteration and maintenance. The automated pipeline also reflects industry-standard DevOps practices, showcasing how modern

cloud solutions can be managed efficiently with minimal manual effort.

The system is hosted on Azure App Service, which provides load balancing, autoscaling, uptime management, and secure handling of configuration settings. By organizing all components under a single Azure Resource Group, the platform becomes easier to monitor, manage, and update. Azure’s cloud environment ensures that the system remains accessible globally with high performance and stability.

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION

1.1 PROBLEM STATEMENT

Large academic campuses, especially technologically advanced and densely structured institutions like engineering colleges, present significant navigation challenges for students, faculty, parents, and visitors. New entrants often struggle to locate classrooms, laboratories, administrative blocks, libraries, and specialized centers distributed across different zones of the campus. Traditional navigation methods—such as physical signboards, paper maps, inquiry desks, or manual guidance—are limited in accuracy, scalability, and convenience. These approaches do not align with the expectations of a modern smart campus environment, where users demand real-time, interactive, and intelligent support. As campuses expand, the need for a dynamic, technology-driven navigation system becomes critical.

Most existing campus navigation systems suffer from several limitations. They usually rely on static maps that lack real-time interaction, have no AI capabilities, and cannot interpret user queries contextually. Furthermore, these solutions generally do not integrate intelligent reasoning or natural language explanations, causing confusion for first-time visitors. They also fail to support accessibility-based routing for differently-abled individuals who require wheelchair-friendly or mobility-safe paths. With the increasing push toward digital transformation in educational institutions, there is a growing need to implement a navigation system that is not only functional but also intelligent, interactive, and inclusive. Another major challenge is the absence of a robust, cloud-based deployment model. Many existing academic tools are hosted on traditional servers or local systems, making them difficult to maintain, scale, or secure. Without cloud infrastructure, institutions face issues such as limited availability, high downtime, manual updates, and poor performance under multiple user requests. Moreover, conventional deployment lacks automation, which increases the risk of errors, slows development cycles, and leads to inconsistent software behavior across environments.

Additionally, current navigation tools rarely integrate modern DevOps practices such as CI/CD automation, containerization, or cloud monitoring. Without these components, deploying updates becomes inefficient and error-prone. Academic systems often lack professional-grade development workflows, making scalability and version control difficult. There is also a gap in integrating AI models seamlessly with routing algorithms, as most systems cannot process complex queries or generate human-like explanations. This prevents campuses from offering intelligent and conversational navigation experiences.

To address these challenges, there is a clear need for an AI-powered, cloud-hosted, fully automated campus navigation system that combines advanced computing technologies with user-friendly interactivity. The system should utilize graph algorithms like BFS, DFS, Dijkstra's algorithm, and accessible routing to compute accurate paths across the campus. It should further integrate Azure AI Foundry to generate natural language explanations that make navigation simple and intuitive. The system needs to be scalable, secure, and globally accessible, which can only be achieved through technologies such as Microsoft Azure App Service, Azure Container Registry, and Azure Resource Groups.To ensure consistent performance across development, testing, and production environments, the system must be containerized using Docker, enabling platform independence and eliminating configuration mismatches. Likewise, implementing a CI/CD pipeline through GitHub Actions is crucial for automated building, testing, container deployment, and application release. This automation significantly improves reliability while reducing human intervention and deployment errors .Thus, the fundamental problem is the absence of a smart, AI-driven, cloud-native, inclusive, and automated navigation solution for large academic campuses. The challenge lies in designing a system that merges AI reasoning, graph-based route computation, cloud scalability, containerization, and DevOps automation into a single cohesive solution that enhances user experience and campus accessibility. The proposed system aims to bridge this gap by providing intelligent route guidance, inclusive navigation options, real-time AI explanations, and a robust deployment pipeline. This ensures a modern, responsive, and future-ready smart campus navigation platform capable of supporting students, staff, and visitors effectively.

1.2 OBJECTIVE OF THE PROJECT

The primary objective of the AI-Powered Campus Navigation System is to design and implement an intelligent, cloud-based solution that simplifies navigation within a large academic environment. As modern campuses continue to expand in size and complexity, users require a system that not only provides accurate routes but also communicates directions in a natural and intuitive manner. This project aims to merge artificial intelligence, cloud computing, automation, and web technologies into a single, user-centric platform capable of providing seamless and adaptive navigation assistance.

A major objective of the system is to leverage Microsoft Azure to ensure high availability, scalability, and performance. By deploying the application on Azure App Service, the project aims to provide users with uninterrupted access from any location and device. Azure services such as the Azure Container Registry, Azure Resource Groups, and the App Service Plan support reliable hosting, efficient resource allocation, and centralized management. This aligns with the goal of creating a cloud-native campus navigation tool that functions dependably under varying loads and user traffic.

Another significant objective is to use Azure AI Foundry to enhance the intelligence layer of the system. Traditional navigation platforms rely solely on visual representation or static instructions, which may be confusing for new students or visitors. This project seeks to integrate a powerful language model capable of interpreting user queries and generating clear, context-aware explanations. The AI-driven model should transform computed paths into natural language guides, helping users understand each step of their route without difficulty. This objective prioritizes user-friendliness and improves communication between the system and its users.

The project also aims to incorporate multiple graph-based routing algorithms to accommodate different navigation needs. By implementing BFS, DFS, Dijkstra’s algorithm, and an accessible route algorithm, the system ensures flexibility for users who have different preferences or mobility requirements. This objective supports both the educational value of the application and the inclusivity of its functionality. Students can learn algorithm behavior, while differently-abled users receive optimized routes that meet their accessibility needs.

To achieve consistent performance across different environments, an important objective of the project is to use Docker containerization. Packaging the application into a Docker image ensures that the system runs identically regardless of where it is deployed. This objective minimizes deployment issues, simplifies maintenance, and supports future scalability of the solution.

In alignment with modern software engineering practices, another core objective is to establish a fully automated CI/CD pipeline using GitHub Actions. Automation ensures that every new update undergoes building, testing, and deployment without manual interference. This objective reduces errors, accelerates iteration, supports continuous development, and ensures the system remains reliable and up to date.

Finally, an overarching objective is to create an inclusive, efficient, and intelligent smart campus experience. The project aims to support students, faculty, staff, and visitors by offering accurate navigation, contextual explanations, and accessible routes through a clean and interactive web interface. By integrating AI, cloud computing, DevOps, and graph theory, the system sets a benchmark for future smart campus applications and demonstrates how modern technologies can be combined to solve real-world challenges effectively.

1.3 SCOPE AND BOUNDARIES

Scope

The scope of the AI-Powered Campus Navigation System includes designing and deploying a smart, cloud-based navigation platform that helps users move efficiently across a large academic campus. The system covers the digital mapping of all major campus buildings, pathways, and accessible routes using graph algorithms. It provides multiple routing methods such as BFS, DFS, Dijkstra's Algorithm, and Accessible Route planning to suit different user needs. The scope further includes integrating Azure AI Foundry to generate natural-language route explanations, transforming technical path outputs into clear human instructions. The project is designed to run entirely on Microsoft Azure, using services such as Azure App Service, Azure Container Registry, and Resource Groups for hosting, scaling, and monitoring. It also includes Docker containerization to enhance consistency across environments, along with GitHub Actions-based CI/CD automation for continuous integration and deployment. The system offers an interactive web interface where users can select locations, view paths, and receive descriptive AI-powered directions. The scope ensures inclusivity by supporting mobility-friendly route options.

Boundaries

While the system delivers intelligent and efficient navigation, certain boundaries limit its current capabilities. The system depends on a predefined graph model of the campus and does not include real-time location tracking using GPS or IoT sensors. It cannot dynamically detect temporary obstacles such as construction work, roadblocks, or event-based crowding unless manually updated. Indoor, multi-floor navigation for buildings such as libraries or academic blocks is not implemented in this version. Mobile application development is outside the scope, as the system is currently optimized only for web usage. The project does not integrate voice-based interaction or speech output, though it may be added in future enhancements. The system also does not support offline mode; all features rely on cloud connectivity to Azure services for AI processing and hosting. Additionally, the AI explanations are based on the trained language model and may vary slightly in phrasing depending on the query.

1.4 STAKEHOLDERS AND END USERS

Stakeholders

1. **Students (New & Existing)**
   Students are major stakeholders, especially first-year students who often face difficulties navigating a large campus. They rely on the system to find classrooms, departments, labs, libraries, and other facilities easily.

2. **Faculty Members & Staff**
   Professors, administrative staff, and non-teaching staff benefit from efficient navigation to reach departments, meeting halls, seminar rooms, and other locations within the campus.

3. **College Administration**
   The administration gains value from improved campus accessibility, reduced visitor queries, and enhanced digital infrastructure. They are also responsible for maintaining and updating campus data.

4. **Parents and Visitors**
   Parents arriving for admission, events, or meetings, and visitors attending seminars, workshops, or fests depend on accurate navigation support to reach locations without confusion.

5. **Development & IT Team**
   The technical team responsible for maintaining Azure resources, updating the system, managing Docker containers, CI/CD pipelines, and ensuring the AI model runs efficiently are also key stakeholders.
6. Management and Decision-Makers
   The college management oversees project adoption, approves budgets, and aims to modernize campus facilities with smart solutions.

Users

1. **Students**
   Primary users who require day-to-day navigation to academic blocks, labs, libraries, hostels, canteens, and exam halls.

2. **Faculty & Research Scholars**
   Users who need quick directions to classrooms, labs, conference rooms, or event venues within the campus.

3. **Visitors**
   Individuals unfamiliar with the campus, such as external examiners, recruiters, guest lecturers, event attendees, and parents.

4. **Non-Teaching Staff**
   Staff responsible for campus operations—security, maintenance, housekeeping, and administrative activities—who may require internal navigation support.

5. **Differently-Abled Users**
   Users needing accessible routes that avoid stairs and ensure mobility-friendly pathways. The system supports inclusive routing tailored to their needs.

1.5 TECHNOLOGIES USED

1. Microsoft Azure Cloud Platform

Azure serves as the complete hosting and deployment environment for the system.

- Azure App Service is used to host the web application with high availability and autoscaling.
- Azure Container Registry (ACR) stores Docker images securely for deployment.
- Azure Resource Groups organize and manage all cloud resources.
  This ensures a reliable, secure, and scalable cloud-native architecture.

2. Azure AI Foundry / GPT-4o

Azure AI Foundry provides the AI capabilities used to generate natural-language route explanations.

- Converts graph-based outputs into human-understandable directions.
- Processes user queries like "Find the route from CSE Block to Library."
  This adds intelligence and conversational ability to the system.

3. Docker Containerization

Docker is used to package the entire web application with all dependencies.

- Ensures consistency across development, testing, and production.
- Allows platform-independent deployment using container images.
- Supports scaling and easy updates.

4. GitHub & GitHub Actions (CI/CD)

GitHub manages source code, while GitHub Actions automates:

- Building Docker images
- Running tests
- Pushing images to Azure Container Registry
- Deploying automatically to Azure App Service
  This ensures continuous integration and continuous deployment with no manual errors.

5. Node.js & Express.js

The backend server is built using:

- Node.js for server-side processing
- Express.js for routing, API handling, and communication with Azure AI
  This forms the core logic of the navigation system.

6. HTML, CSS, JavaScript (Frontend Development)

The interactive web interface is developed using:

- HTML for structure
- CSS for design and responsiveness
- JavaScript for dynamic interaction and algorithm selection
  Users can select locations, algorithms, and view AI-generated directions.

7. Graph Algorithms & Data Structures

Technologies used on the algorithmic side include:

- BFS, DFS, Dijkstra's Algorithm, and Accessible Route Algorithm
- Data structures such as adjacency lists and weighted graphs
  These handle route computation efficiently.

8.Visual Studio Code (Development Environment)

VS Code is used as the main IDE for writing and testing the codebase.

1.6 ORGANIZATION OF THE REPORT

This report is structured to provide a complete and logical documentation of the entire project lifecycle.

Chapter 1 introduces the problem context, project objectives, scope, stakeholders, technologies, and report structure.

Chapter 2 presents a detailed examination of the system design and architecture, including requirements, solution overview, cloud strategy, infrastructure specifications, and service mappings.

Chapter 3 delves into the DevOps implementation, covering CI/CD pipelines, Terraform automation, Docker containerization, orchestration principles, and AI service integration. **Chapter 4** focuses on operational excellence and security, discussing DevSecOps practices, monitoring, access control, and deployment strategies.

Chapter 5 presents the results of the implementation, including summaries, challenges overcome, performance metrics, cost analysis, and key learnings.

Chapter 6 concludes the report with a summary of achievements and a roadmap for future enhancements.

The document concludes with references and appendices containing configuration details and system screenshots.

CHAPTER 2 - SYSTEM DESIGN AND ARCHITECTURE

2.1 REQUIREMENT SUMMARY

Functional Requirements

1. User Input and Location Selection

- The system must allow users to select a **starting point** and a **destination** from a predefined list of campus locations.
- The interface should support dropdowns or clickable options for easy selection.
- The system must validate that both points are selected before generating a route.

2. Route Calculation Using Algorithms

- The system must compute routes between two locations using graph-based algorithms such as:
    - Breadth-First Search (BFS)
    - Depth-First Search (DFS)
    - Dijkstra's Algorithm
    - Accessible Route Algorithm
- Each algorithm must generate an accurate and consistent path based on the campus map.

3. AI-Based Route Explanation

- The system must integrate Azure AI Foundry or GPT-4o to convert the raw path into natural-language instructions.
- The AI engine must respond to queries like:
    - "Show the route to the library."
    - "How to go from Main Gate to CSE Block?"
- The explanation should be easy to understand and context-aware.

4. Display of Route Output

- The system must display the calculated route in a clear, step-by-step format.
- Users must be able to view both the algorithmic result and the AI-generated

explanation.

- The UI must highlight important nodes or turns in the path.

5. Web Interface Functionality

- The interface must be responsive, interactive, and easy to navigate.
- It should support multiple devices (desktop, laptop, mobile browsers).
- Users must be able to change locations and algorithms without page reload.

6. Docker-Based Deployment

- The system must run in a Docker container for consistent performance.
- The backend server must operate correctly within the container for all environments.

7. CI/CD Automation

- The system must automatically build, test, and deploy updates using GitHub Actions.
- Each commit should trigger the pipeline and deploy the latest version to Azure App Service.

8. Error Handling and Validation

- The system must show errors for invalid inputs, missing fields, or unreachable paths.
- Proper AI or system error messages must be provided when necessary.

9. Accessibility Support

- The system must include an Accessible Route option for mobility-impaired users.
- Such routes must avoid stairs and prioritize wheelchair-friendly paths.

10. Secure API Integration

- The system must securely access Azure AI model endpoints using environment variables.
- No sensitive information should be exposed in the client interface or source code.

Non-Functional Requirements

1.Performance Requirements

- The system must generate route results and AI explanations within 1–3 seconds under normal load.
- It should handle multiple concurrent users without performance degradation, supported by Azure App Service autoscaling.
- API communication with Azure AI Foundry must remain fast, ensuring low-latency responses.

2. Scalability Requirements

- The system must be capable of scaling automatically based on user demand through Azure's autoscaling features.
- The Dockerized architecture should allow horizontal scaling by deploying additional containers when necessary.
- The system design must support future enhancements like indoor navigation, GPS integration, and mobile apps without architectural rework.

3. Availability Requirements

- The web application must maintain high availability (99.9% uptime) through Azure hosting.
- CI/CD deployment should ensure zero-downtime updates so the system stays accessible at all times.
- Resource Groups and App Service Plans should ensure continuous service even during updates or load surges.

4. Reliability Requirements

- Route calculations must consistently return accurate and predictable outputs for all campus locations.
- AI-generated instructions must maintain clarity, coherence, and correctness across repeated queries.
- The system should maintain stable operation even during heavy traffic or rapid updates.

6. Security Requirements

- API keys, model endpoints, and configuration values must be securely stored in environment variables or Azure-managed configurations.
- The system must protect user queries and prevent unauthorized access to backend endpoints.
- Only authorized developers should be able to push updates through the CI/CD pipeline.

7. Usability Requirements

- The interface must be simple, responsive, and intuitive for users of all age groups.
- The system should provide easy navigation through clean buttons, dropdowns, and instructions.
- The AI explanations should be written in clear, human-like language without technical

terminology.

8. Maintainability Requirements

- The application code must follow modular architecture to simplify updates and debugging.
- GitHub Actions CI/CD should automate testing and deployment, reducing the need for manual maintenance.
- Docker containers should ensure consistent behavior across development, testing, and production environments.

9. Portability Requirements

- The Dockerized application must run on any environment supporting containerization.
- The system should be easily portable to other cloud providers if needed in the future.

10. Compliance Requirement

- The system must comply with institutional IT policies regarding cloud usage and application access.
- Resource usage and deployment must follow Azure best practices for cost efficiency and security

2.2 PROPOSED SOLUTION OVERVIEWS

The proposed solution for the AI-Powered Campus Navigation System is to build an intelligent, cloud-native, automated, and user-friendly platform that enables students, faculty, visitors, and staff to navigate a large academic campus with ease using AI-generated natural language directions. The system integrates modern technologies such as Microsoft Azure, Azure AI Foundry, Docker containerization, and GitHub Actions CI/CD automation to deliver a scalable and intelligent web application. The solution begins by digitally modeling the entire campus as a weighted graph, where academic blocks, laboratories, libraries, auditoriums, walkways, and intersections are represented as nodes and edges. Using this graph, the backend applies multiple pathfinding algorithms—BFS for shortest unweighted paths, DFS for exploratory traversal, Dijkstra's Algorithm for optimal weighted distance computation, and an Accessibility Algorithm for wheelchair-friendly navigation—ensuring flexible route discovery for different user needs. Once the route is computed, the system invokes Azure AI Foundry or GPT-4o to transform the raw numerical or node-based path into a clear, human-readable explanation, allowing even first-time users to follow directions easily. To ensure consistency and reliability, the entire backend is packaged into a Docker container, which includes all dependencies, configurations, and runtime components, guaranteeing identical behavior across development, testing, and production environments. This container is stored securely in Azure Container Registry (ACR) and is deployed automatically to Azure App Service through a GitHub Actions CI/CD pipeline. Every code update triggers an automated workflow that builds, tests, containerizes, and redeploys the application without manual involvement, ensuring rapid iteration and error-free updates. Azure App Service provides autoscaling, monitoring, and high availability, ensuring that the system remains responsive even during peak usage times. The frontend of the solution is built using HTML, CSS, and JavaScript to provide an intuitive and interactive user interface where users can select their starting point, destination, and preferred algorithm. The interface instantly fetches route data from the backend and displays both the computed path and the AI-generated explanation. All sensitive information, such as the Azure AI endpoint and API keys, is secured using environment variables, guaranteeing safe communication between the backend and cloud services. The proposed solution ensures inclusivity by supporting accessible navigation paths for differently-abled users, making it beneficial for a wide range of stakeholders. By running entirely on Azure's cloud infrastructure, the system achieves high reliability, seamless performance, easy maintenance, and global accessibility. Moreover, the combination of AI reasoning, graph algorithms, containerization, and automated deployment aligns the solution with modern smart campus requirements and industry standards. This solution not only offers a practical wayfinding platform but also establishes a future-ready digital ecosystem that can be extended with features such as real-time tracking, IoT-based dynamic routing, voice assistance, and mobile app integration. Overall, the proposed system offers a holistic, intelligent, and scalable navigation experience that transforms the traditional campus environment into a smart, AI-enabled learning space.The entire infrastructure is defined as code using Terraform, enabling reproducible deployments across environments. GitHub Actions automates the build, test, and deploy process, ensuring that changes are thoroughly validated before reaching production. This architectural approach combines the benefits of serverless computing, managed services, and automation to deliver a robust and efficient marketplace platform.

2.1CLOUD DEPLOYMENT STATERGY

The cloud deployment strategy for the AI-Powered Campus Navigation System is designed to ensure seamless scalability, high availability, consistent performance, and automated delivery using Microsoft Azure's cloud ecosystem. The entire solution is deployed as a cloud-native application, utilizing Azure services to manage hosting, containerization, resource organization, and automation processes. The deployment begins with containerizing the backend application using Docker, which bundles the server logic, algorithms, AI integration modules, and dependencies into a portable image. This container ensures that the application maintains consistent behavior across development, testing, and production environments. The generated Docker image is securely stored in Azure Container Registry (ACR), which serves as the central repository for all container versions, facilitating rollback, updates, and version management. Hosting is managed by Azure App Service, which pulls the latest Docker image from ACR and runs the application in a fully managed environment. Azure App Service provides built-in load balancing, autoscaling, and continuous monitoring, ensuring stable performance regardless of user load.

The deployment process is fully automated using GitHub Actions CI/CD pipeline, which plays a crucial role in eliminating manual errors and accelerating updates. Each time the development team commits new code, the pipeline automatically triggers the build process, constructs a new Docker image, pushes it to ACR, and deploys it to Azure App Service. This continuous integration and continuous deployment workflow maintains reliability, transparency, and developer productivity. All cloud resources—including App Service, ACR, AI Foundry, and supporting components—are organized under a single Azure Resource Group for easier monitoring, access control, and resource management. The system utilizes Azure's environment variables to securely store keys, endpoints, and configuration settings, protecting sensitive information from exposure.To further enhance stability, Azure's autoscaling capabilities dynamically adjust compute resources based on demand. During peak usage, additional instances are automatically created to maintain performance, and during low usage, resources scale down to optimize cost. Azure Monitoring and Application Insights are used for real-time performance tracking, request logging, and error diagnostics, ensuring quick detection and resolution of issues. Overall, the cloud deployment strategy prioritizes efficiency, automation, security, and scalability, transforming the campus navigation system into a resilient and future-ready cloud-hosted solution capable of supporting growing user demands and continuous enhancements

2.2 INFRASTRUCTURE REQUIREMENTS

The successful deployment of the AI-Powered Campus Navigation System requires robust and scalable infrastructure that supports cloud hosting, AI processing, containerization, and automated deployment. The core infrastructure is built on Microsoft Azure, where Azure App Service provides a fully managed platform for running the web application with automatic scaling, high availability, and built-in load balancing. An Azure App Service Plan is required to allocate compute resources such as CPU, RAM, and network capacity needed for smooth operation. To manage and store containerized versions of the application, the system requires an Azure Container Registry (ACR) that securely holds Docker images and provides seamless integration with the CI/CD pipeline. A centralized Azure Resource Group is necessary to organize all cloud components, including App Service, ACR, logs, and monitoring services, ensuring easy tracking and maintenance. On the development side, Docker Engine must be installed to build, test, and package the application into containers before deployment. The project also requires GitHub for version control and GitHub Actions for automating the CI/CD process, enabling continuous integration, testing, and deployment. To power the AI-based route explanation, an Azure AI Foundry or GPT-4o model endpoint is needed, along with secure environment variables for storing API keys and credentials. For frontend and backend development, a suitable code editor such as Visual Studio Code, along with Node.js and Express.js runtime environments, must be available. Additional tools such as HTML, CSS, JavaScript libraries, and development plugins support the creation of the web interface. Together, this cloud-native infrastructure ensures that the system remains scalable, reliable, secure, and capable of delivering real-time, AI-powered navigation to users across the campus.

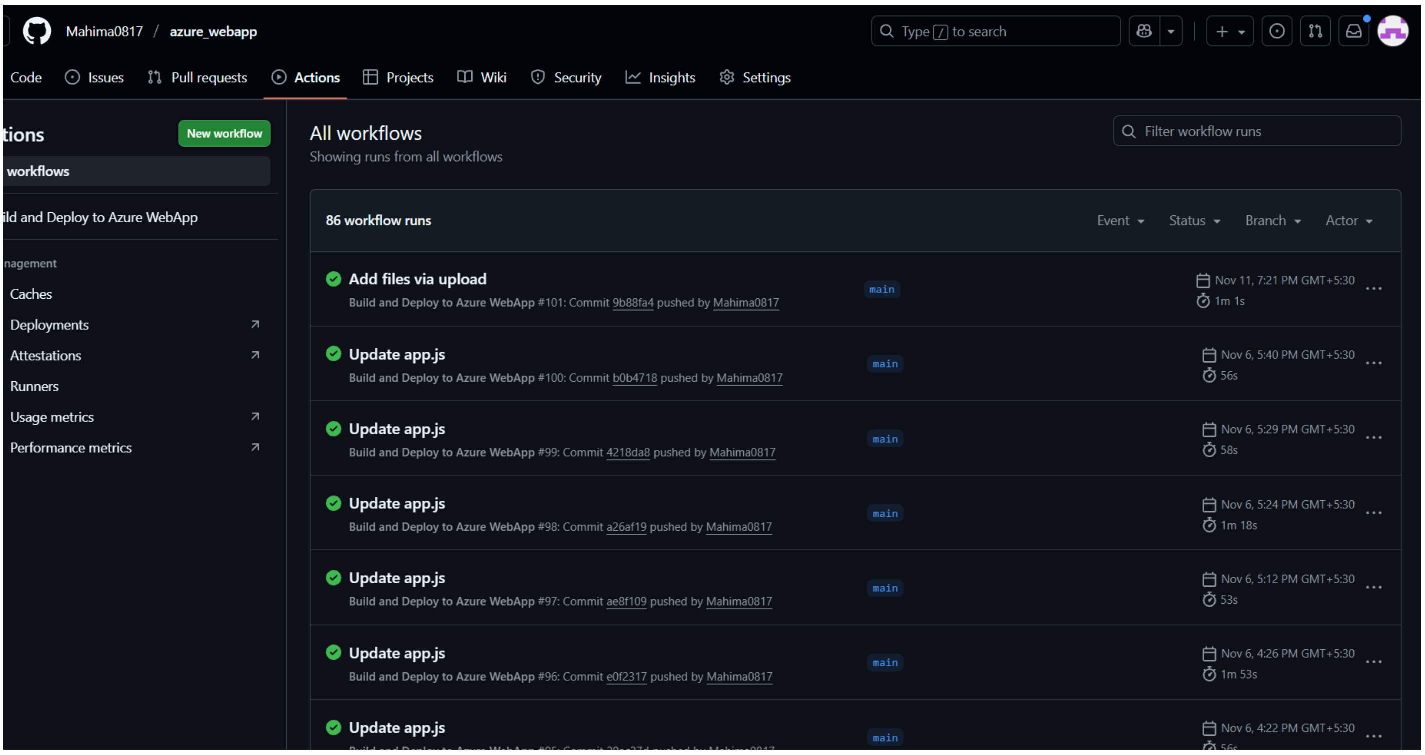
2.3 AZURE SERVICES MAPPING AND JUSTIFICATION

Requirement	**Azure Service**	**SKU/Tier**	**Purpose**	**Justification**	**Est. Monthly Cost (₹)**
Frontend Hosting	Azure Static Web Apps	Standard	Service Responsive UI	Free hosting, SSL, CDN	Free (Student Credit)
Backend Logic & APIs	Azure Container Apps	0.75 vCPU, 1.5GB	Host Containerized backend	Always-on, scalable	3000
Database	Azure PostgreSQL Flexible Server	1 vCore, 5GB	Store club details, events, and member data	Reliable, auto-backup, high availability	1,200
Media Storage	Azure Blob Storage	Hot Tier, Geo-Redundant Storage	Store profile and product images	Cost Efficient	300
AI Moderation	Azure Content Moderator	Standard	Filter inappropriate content	Improves safety & compliance	0
Monitoring & Logs	Azure Monitor + Application Insights	Standard	Log and track API health	Proactive maintenance	400
AI Description generation	Gemini Pro Vision	Pay-per-use	Auto-generate descriptions	Boosts listing quality	50
CI/CD Automation	Github Actions	Basic	Automated builds & deployments	Seamless GitHub integration	Free (Student Credit)

CHAPTER 3 - DEVOPS IMPLEMENTATION

3.1 CONTINUOUS INTEGRATION AND DEPLOYMENT STATERGY

The project uses a multi-stage CI/CD pipeline built with GitHub Actions to automate the deployment. This setup uses Infrastructure as Code (IaC) and containerization to deploy components across various Azure services.



All workflows
Showing runs from all workflows

86 workflow runs

Add files via upload
Build and Deploy to Azure WebApp #101: Commit 9b88fa4 pushed by Mahima0817 — main — Nov 11, 7:21 PM GMT+5:30 — 1m 1s

Update app.js
Build and Deploy to Azure WebApp #100: Commit b0b4718 pushed by Mahima0817 — main — Nov 6, 5:40 PM GMT+5:30 — 56s

Update app.js
Build and Deploy to Azure WebApp #99: Commit 4218da8 pushed by Mahima0817 — main — Nov 6, 5:29 PM GMT+5:30 — 58s

Update app.js
Build and Deploy to Azure WebApp #98: Commit a26af19 pushed by Mahima0817 — main — Nov 6, 5:24 PM GMT+5:30 — 1m 18s

Update app.js
Build and Deploy to Azure WebApp #97: Commit ae8f109 pushed by Mahima0817 — main — Nov 6, 5:12 PM GMT+5:30 — 53s

Update app.js
Build and Deploy to Azure WebApp #96: Commit e0f2317 pushed by Mahima0817 — main — Nov 6, 4:26 PM GMT+5:30 — 1m 53s

Update app.js
main — Nov 6, 4:22 PM GMT+5:30

Figure 1. Github Actions Workflow

The pipeline follows 2-stage deployment pattern:

1. **Backend:** Deploys the main application (as a Container App).
2. **Frontend:** Builds and deploys the static web app (React).

The entire process is triggered either by a push to the main branch or through a manual workflow dispatch.

Pipeline Stages in Detail:

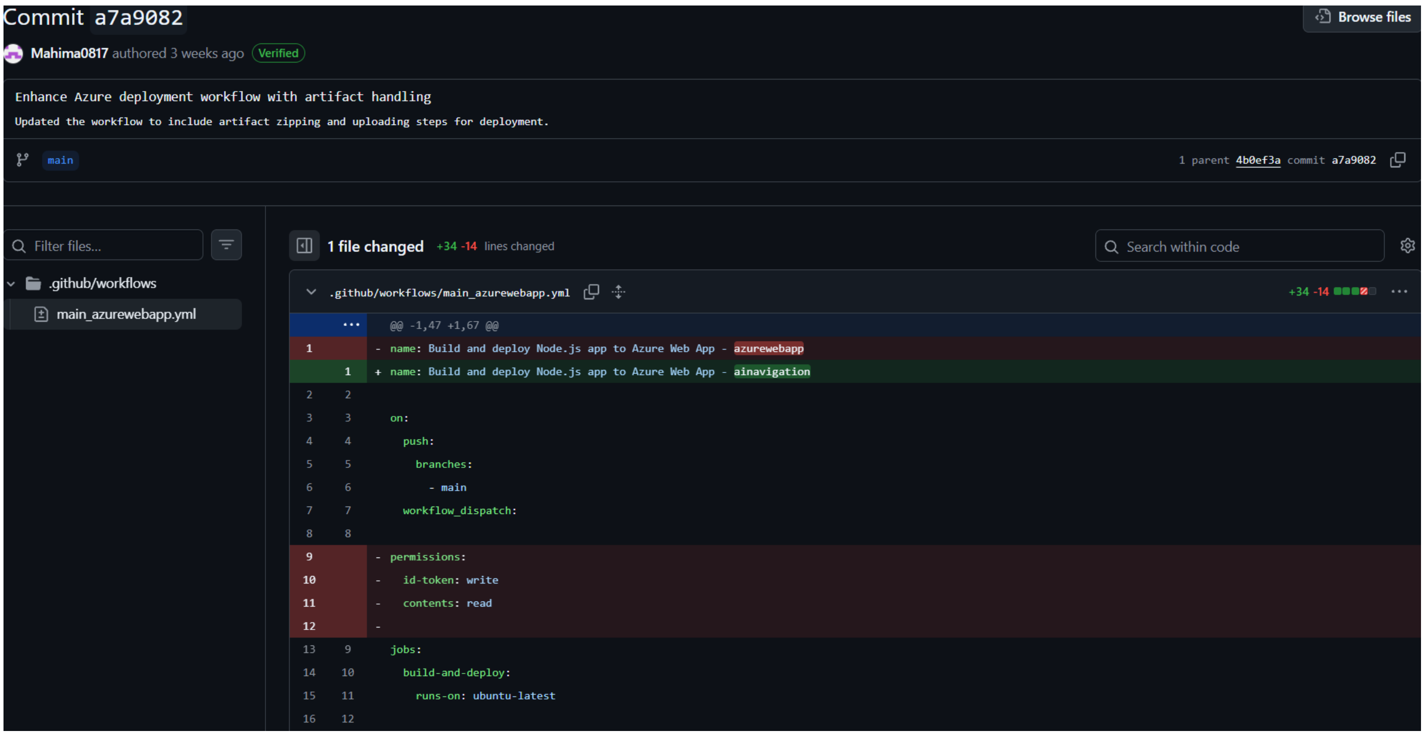
Stage 1: Backend Deployment (build-and-deploy-backend)

1. **Versioning and Containers:** A unique version tag is created for the container image using the date and github commit ID (e.g., v20251029-a1b2c3d). The image is then built and pushed to the registry.

2. **Infrastructure as Code (IaC):** Terraform manages all Azure resources. It ensures that the infrastructure is consistent and changes are applied.

3. **Azure Deployment:** Secure access to Azure is handled through a Service Principal using secrets stored in GitHub. The new container image is deployed to Azure Container Apps using a zero-downtime rolling update.

4. **Health Check:** After deployment, the pipeline waits and performs an automated check against the backend's /healthendpoint to confirm the service is running correctly before moving on.


Commit a7a9082
Browse files
Mahima0817 authored 3 weeks ago
Verified

```
Enhance Azure deployment workflow with artifact handling
Updated the workflow to include artifact zipping and uploading steps for deployment.
```

main
1 parent 4b0ef3a commit a7a9082
Filter files...
.github/workflows
main_azurewebapp.yml
1 file changed +34 -14 lines changed
Search within code
.github/workflows/main_azurewebapp.yml
+34 -14

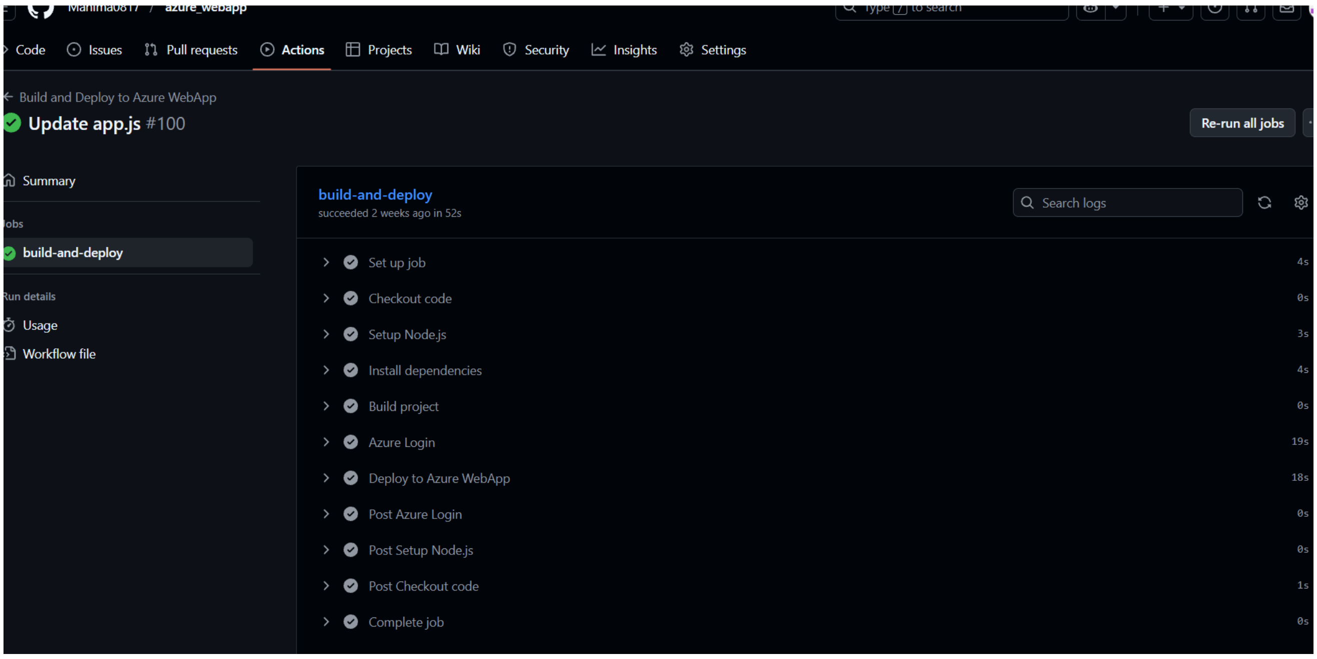
```
@@ -1,47 +1,67 @@
- name: Build and deploy Node.js app to Azure Web App - azurewebapp
+ name: Build and deploy Node.js app to Azure Web App - ainavigation

  on:
    push:
      branches:
        - main
    workflow_dispatch:

- permissions:
-   id-token: write
-   contents: read
-
  jobs:
    build-and-deploy:
      runs-on: ubuntu-latest

```

Figure 2. Backend Deployment Job Run

Stage 2: Frontend Deployment (build-and-deploy-frontend)

Dynamic Configuration: The URL of the newly deployed Backend is automatically passed to the frontend build process. This ensures the React application is configured with the correct endpoints for the current deployment.

Hosting: The optimized frontend build is deployed to Azure Static Web Apps (SWA).



Build and Deploy to Azure WebApp

Update app.js #100

build-and-deploy

build-and-deploy
succeeded 2 weeks ago in 52s

```
Set up job                 4s
Checkout code              0s
Setup Node.js              3s
Install dependencies       4s
Build project              0s
Azure Login               19s
Deploy to Azure WebApp    18s
Post Azure Login           0s
Post Setup Node.js         0s
Post Checkout code         1s
Complete job               0s
```

Figure 3. Frontend Deployment Job Run

Required GitHub Secrets

Table 7. List of Github Secrets

1	AZURE_CLIENT_ID
2	AZURE_CLIENT_SECRET
3	AZURE_STATIC_WEB_APPS_API_TOKEN
4	AZURE_STORAGE_ACCOUNT_NAME
5	AZURE_SUBSCRIPTION_ID
6	AZURE_TENANT_ID
7	CONTENT_SAFETY_KEY
8	DB_PASSWORD
9	DOCKERHUB_TOKEN
10	DOCKERHUB_USERNAME
11	GEMINI_API_KEY

3.2 TERRAFORM INFRASTRUCTURE-AS-CODE (IaC)

The Terraform code is organized using a standard three-file structure:

1. **main.tf:** Contains the primary definitions for all cloud resources.
2. **variables.tf:** Holds input values and settings, like resource names or location.
3. **outputs.tf:** Defines data outputs, such as resource URLs or hostnames, for use by other systems (like the CI/CD pipeline).

Core Infrastructure Components

The IaC sets up and organises all the required cloud services, placing them together in one environment-specific Resource Group, like rg-marketplace-dev.

Backend and Container Orchestration

Container App Environment: Provisions the managed environment (**cae-marketplace-dev**) that acts as the host for the microservices. This environment handles auto-scaling and networking for the containers.

Container App: Defines the main backend application (ca-marketplace-backend-dev) with specifications:

Scaling: It supports scale-to-zero (0 minimum replicas) for cost optimization, up to a maximum of 10 replicas.

Configuration: Database connection details (like the fully qualified domain name) are dynamically set as environment variables.

Database

Azure PostgreSQL Flexible Server: Provisions the database server (psql-marketplace-dev) with a specified version (13) and performance tier (B_Standard_B1ms).

Blob Container: A storage container is provisioned specifically for images, with public access enabled (container_access_typ = "blob"), making it ready for integration with a Content Delivery Network (CDN).

Serverless Functions (AI Services)

Function App: Provisions the Azure Function App (*func-marketplace-ai-dev*) that runs the AI generated descriptions services.

Runtime: Configured to use the Python 3.9 runtime.

CI/CD Integration

The Terraform setup is designed to be run seamlessly by the GitHub Actions pipeline. During the CI/CD execution:

1. Required secrets (like the database password and storage account name) are securely passed from GitHub Secrets into Terraform's variables.
2. The application's dynamic container image tag from the build stage is passed to the Container App resource, ensuring the latest code is always deployed.
3. Outputs (like the Container App URL and Database Hostname) are captured in outputs.tf and used by later stages of the CI/CD pipeline (e.g., the Frontend build stage).

3.3 CONTAINERIZATION STRATERGY (DOCKER)

The project uses a disciplined containerization strategy to package the Go backend application into small, secure, and efficient Docker images. This approach is optimized for fast and reliable deployment on Azure Container Apps.

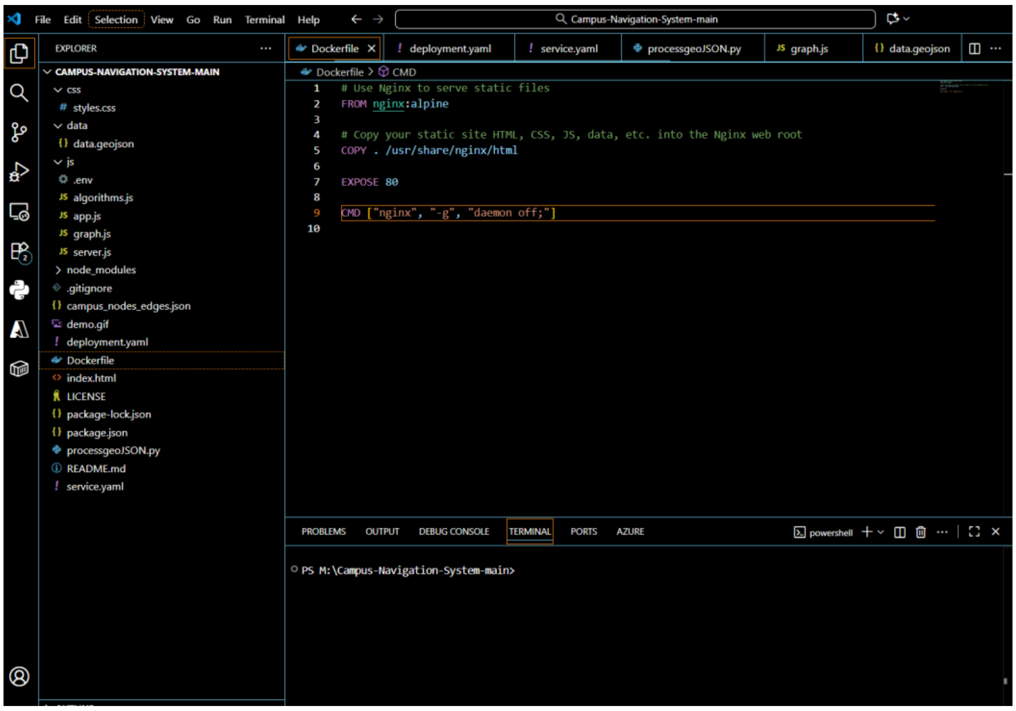
Image Build Process: Multi-Stage Optimization

We use a multi-stage Docker build to make the final image as small as possible. This separates the build process from the runtime environment:

Builder Stage: We start with the golang:1.23-alpine image. The source code is compiled into a single, static binaryafter downloading dependencies. We use CGO_ENABLED=0 to ensure the binary is self-contained and doesn't rely on C libraries.

Runtime Stage: We switch to the very minimal alpine:latest image. We install only system certificates (ca-certificates) and copy only the compiled binary from the builder stage.

Deployment, Scaling, and Versioning



```
# Use Nginx to serve static files
FROM nginx:alpine

# Copy your static site HTML, CSS, JS, data, etc. into the Nginx web root
COPY . /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

```
PS M:\Campus-Navigation-System-main>
```

Figure 4. Docker file

1. Azure App Service Deployment Setup

Resource Allocation:
The AI-Powered Campus Navigation System runs on Azure App Service, where the application is deployed as a Docker container pulled from Azure Container Registry (ACR). The App Service Plan is configured with an appropriate compute tier to support consistent performance. Typical configurations include 1 vCPU and 1–2 GB RAM, which is sufficient for handling backend logic, AI API calls, and user traffic on campus.

Auto-scaling:
Azure App Service includes built-in autoscaling rules. The system can automatically scale the number of container instances based on performance metrics such as CPU usage, memory usage, or request load. During low-traffic hours (evenings or weekends), the App Service can scale down to 1 running instance to reduce cost, while during peak usage (admissions, events, morning college hours), it can scale up to multiple instances to ensure uninterrupted performance.

Ingress & Security:
The application is publicly accessible through a secure HTTPS endpoint provided by Azure. App Service automatically manages TLS certificates, ensuring encrypted communication between the user interface, backend server, and AI endpoints. Firewall restrictions and access control policies ensure that backend APIs and environment variables remain protected.

3.4KUBERNETES ORCHESTRATION

Azure Kubernetes Service (AKS) is utilised for container orchestration, gain control over scaling, security, and traffic.

1.Cluster and Deployment Design

Cluster Structure: The cluster uses a multi-pool architecture (System and User) with Azure CNI networking. It includes the Cluster Autoscaler to dynamically manage node capacity based on demand.

Workload: The Go backend is deployed as a Deployment within a dedicated namespace (e.g., marketplace-app). It relies on standard Kubernetes liveness and readiness probes targeting the / health endpoint.

Ingress: Traffic is managed by an Ingress Controller (e.g., NGINX), which handles TLS termination and strictly enforces HTTPS-only external access.

Configuration: Non-sensitive settings use ConfigMaps. Sensitive data is managed through Azure Key Vault and securely injected into the pods via the Secrets Store CSI Driver.

1. Scaling, Resilience, and Security

Pod Scaling: The Horizontal Pod Autoscaler (HPA) automatically scales the number of replicas based on resource metrics like CPU Utilization (e.g., 60%).

Node Scaling: The Cluster Autoscaler adjusts the underlying node count to meet the HPA's capacity requirements.

Resilience: Rolling Updates are configured with settings like maxSurge:25% and maxUnavailable:0 for zero-downtime deployments.

3.4 GENAI INTEGRATION AND AZURE AI SERVICE MAPPING

This system uses Azure AI Foundry’s GPT-4o model to generate **natural, human-friendly navigation descriptions** based on algorithmic route output.

Implementation:

The integration is handled in a separate **Azure Function**, which enables independent scaling of AI workloads during heavy usage (e.g., student orientation).

Process:

1. The function receives the computed graph-based path.
2. It constructs a structured AI prompt containing:
   - start location
   - destination
   - stepwise graph route nodes

    - style requirements (clear, simple campus instructions)
3. GPT-4o returns a descriptive explanation such as:
   "From the Main Gate, walk straight towards the Auditorium and turn left to reach the Library."

2. Content Safety (Azure Content Safety API)

To maintain secure interactions, all user-submitted queries and AI-generated outputs pass through Azure Content Safety.

Implementation:

Instead of analyzing images, the API validates **text content** for route queries and AI responses to ensure safe, non-harmful interactions.

Process:

- The content is evaluated across categories like harmful or inappropriate language.
- If any unsafe category is flagged at high severity, the system blocks the output and informs the user.
  This ensures that route requests and AI-generated explanations remain safe, professional, and appropriate for an educational environment.

CHAPTER 4 - CLOUD OPERATIONS AND SECURITY

1.1 DEVSECOPS INTEGRATION

Security is embedded across the development lifecycle through lightweight, fast, and enforceable static checks. Static application security testing is performed via language-native linters on both services: ESLint for the React frontend and Go linters for the backend (e.g., go vet/format and common lint rules). These tools analyse source code for correctness, unsafe patterns, and security hygiene, including unhandled errors, misuse of concurrency primitives, dangerous DOM sinks, and unsafe string handling. Linting runs locally and in continuous integration on every pull request, and merges are blocked when violations are detected to ensure issues are addressed before code reaches production.

Code quality and maintainability are enforced with strict linting configurations that standardise style, imports, and typing conventions. In the frontend, ESLint rulesets (with TypeScript support) prevent common pitfalls such as unused variables, non-exhaustive React hooks dependencies, and weak typing that can mask trust boundary issues. In the backend, Go linters enforce idiomatic patterns, detect shadowed variables, potential nil dereferences, and missing error checks—reducing classes of defects that often lead to security vulnerabilities at runtime.

Container security is strengthened through minimal base images and deterministic builds. The backend is packaged using a Docker multi-stage process that produces a final image based on `scratch`, which removes shells, package managers, and non-essential utilities to reduce the attack surface and the number of potential CVEs. Images are built for the correct target platform (`linux/amd64`) to ensure consistent, reproducible deployments across environments. This approach limits runtime footprint and minimises opportunities for exploitation in production containers.

Operational consistency and secure configuration are maintained through configuration-as-code and environment-driven settings. Sensitive values are injected via environment variables rather than hardcoded, and the backend enforces secure database connectivity (e.g., SSL mode required). CORS is constrained to the exact frontend domain to prevent broad origin access, aligning with the principle of least privilege.

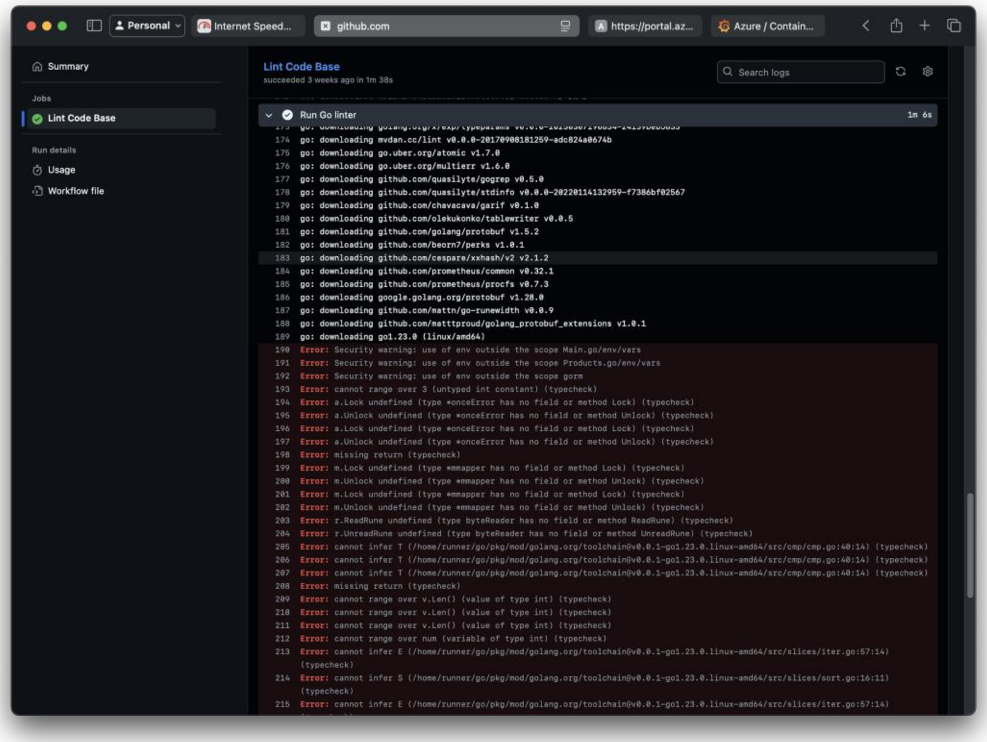
Together, these practices provide practical DevSecOps coverage focused on early defect and vulnerability prevention (via linters), minimized container attack surface (via `scratch` images), and secure-by-default application configuration—delivering measurable security benefits without heavy operational overhead.

Demo:

Error: Security warning: use of env outside the scope Main.go/env/vars

Error: Security warning: use of env outside the scope Products.go/env/vars

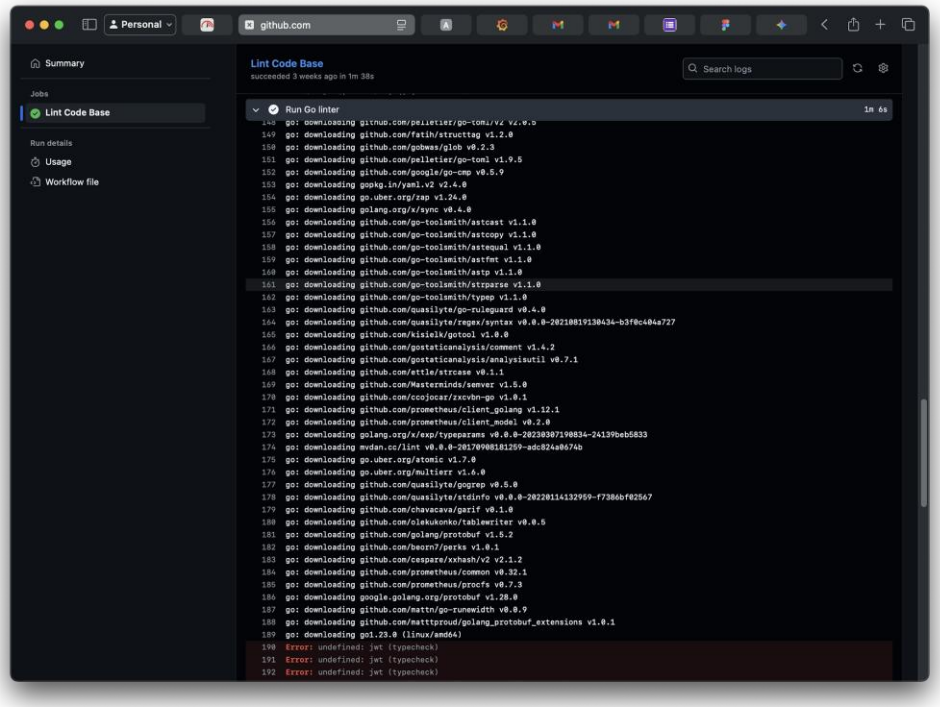
Error: Security warning: use of env outside the scope gorm



```
go: downloading mvdan.cc/lint v0.0.0-20170908181259-adc824a0674b
go: downloading go.uber.org/atomic v1.7.0
go: downloading go.uber.org/multierr v1.6.0
go: downloading github.com/quasilyte/gogrep v0.5.0
go: downloading github.com/quasilyte/stdinfo v0.0.0-20220114132959-f7386bf02567
go: downloading github.com/chavacava/garif v0.1.0
go: downloading github.com/olekukonko/tablewriter v0.0.5
go: downloading github.com/golang/protobuf v1.5.2
go: downloading github.com/beorn7/perks v1.0.1
go: downloading github.com/cespare/xxhash/v2 v2.1.2
go: downloading github.com/prometheus/common v0.32.1
go: downloading github.com/prometheus/procfs v0.7.3
go: downloading google.golang.org/protobuf v1.28.0
go: downloading github.com/mattn/go-runewidth v0.0.9
go: downloading github.com/matttproud/golang_protobuf_extensions v1.0.1
go: downloading go1.23.0 (linux/amd64)
Error: Security warning: use of env outside the scope Main.go/env/vars
Error: Security warning: use of env outside the scope Products.go/env/vars
Error: Security warning: use of env outside the scope gorm
Error: cannot range over 3 (untyped int constant) (typecheck)
Error: a.Lock undefined (type *onceError has no field or method Lock) (typecheck)
Error: a.Unlock undefined (type *onceError has no field or method Unlock) (typecheck)
Error: a.Lock undefined (type *onceError has no field or method Lock) (typecheck)
Error: a.Unlock undefined (type *onceError has no field or method Unlock) (typecheck)
Error: missing return (typecheck)
Error: m.Lock undefined (type *mmapper has no field or method Lock) (typecheck)
Error: m.Unlock undefined (type *mmapper has no field or method Unlock) (typecheck)
Error: m.Lock undefined (type *mmapper has no field or method Lock) (typecheck)
Error: m.Unlock undefined (type *mmapper has no field or method Unlock) (typecheck)
Error: r.ReadRune undefined (type byteReader has no field or method ReadRune) (typecheck)
Error: r.UnreadRune undefined (type byteReader has no field or method UnreadRune) (typecheck)
Error: cannot infer T (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/cmp/cmp.go:40:14) (typecheck)
Error: cannot infer T (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/cmp/cmp.go:40:14) (typecheck)
Error: cannot infer T (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/cmp/cmp.go:40:14) (typecheck)
Error: missing return (typecheck)
Error: cannot range over v.Len() (value of type int) (typecheck)
Error: cannot range over v.Len() (value of type int) (typecheck)
Error: cannot range over v.Len() (value of type int) (typecheck)
Error: cannot range over num (variable of type int) (typecheck)
Error: cannot infer E (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/slices/iter.go:57:14)
(typecheck)
Error: cannot infer S (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/slices/sort.go:16:11)
(typecheck)
Error: cannot infer E (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/slices/iter.go:57:14)
```

Figure 5. Github Workflow Showing Lint Errors

After: Errors resolved



```
go: downloading github.com/fatih/structtag v1.2.0
go: downloading github.com/gobwas/glob v0.2.3
go: downloading github.com/pelletier/go-toml v1.9.5
go: downloading github.com/google/go-cmp v0.5.9
go: downloading gopkg.in/yaml.v2 v2.4.0
go: downloading go.uber.org/zap v1.24.0
go: downloading golang.org/x/sync v0.4.0
go: downloading github.com/go-toolsmith/astcast v1.1.0
go: downloading github.com/go-toolsmith/astcopy v1.1.0
go: downloading github.com/go-toolsmith/astequal v1.1.0
go: downloading github.com/go-toolsmith/astfmt v1.1.0
go: downloading github.com/go-toolsmith/astp v1.1.0
go: downloading github.com/go-toolsmith/strparse v1.1.0
go: downloading github.com/go-toolsmith/typep v1.1.0
go: downloading github.com/quasilyte/go-ruleguard v0.4.0
go: downloading github.com/quasilyte/regex/syntax v0.0.0-20210819130434-b3f0c404a727
go: downloading github.com/kisielk/gotool v1.0.0
go: downloading github.com/gostaticanalysis/comment v1.4.2
go: downloading github.com/gostaticanalysis/analysisutil v0.7.1
go: downloading github.com/ettle/strcase v0.1.1
go: downloading github.com/Masterminds/semver v1.5.0
go: downloading github.com/ccojocar/zxcvbn-go v1.0.1
go: downloading github.com/prometheus/client_golang v1.12.1
go: downloading github.com/prometheus/client_model v0.2.0
go: downloading golang.org/x/exp/typeparams v0.0.0-20230307190834-24139beb5833
go: downloading mvdan.cc/lint v0.0.0-20170908181259-adc824a0674b
go: downloading go.uber.org/atomic v1.7.0
go: downloading go.uber.org/multierr v1.6.0
go: downloading github.com/quasilyte/gogrep v0.5.0
go: downloading github.com/quasilyte/stdinfo v0.0.0-20220114132959-f7386bf02567
go: downloading github.com/chavacava/garif v0.1.0
go: downloading github.com/olekukonko/tablewriter v0.0.5
go: downloading github.com/golang/protobuf v1.5.2
go: downloading github.com/beorn7/perks v1.0.1
go: downloading github.com/cespare/xxhash/v2 v2.1.2
go: downloading github.com/prometheus/common v0.32.1
go: downloading github.com/prometheus/procfs v0.7.3
go: downloading google.golang.org/protobuf v1.28.0
go: downloading github.com/mattn/go-runewidth v0.0.9
go: downloading github.com/matttproud/golang_protobuf_extensions v1.0.1
go: downloading go1.23.0 (linux/amd64)
Error: undefined: jwt (typecheck)
Error: undefined: jwt (typecheck)
Error: undefined: jwt (typecheck)
```

Figure 6. Github Workflow Showing Resolved Env Errors

1.2 MONITORING AND OBSERVABILITY

Comprehensive monitoring and observability are achieved through integration with Azure Monitor and the central visualization power of Azure Managed Grafana. The underlying telemetry data is collected seamlessly from all resources within the Azure subscription, with Container Apps as a primary focus.

Azure Managed Grafana is utilized as the primary glass for real-time visualization. The default Azure Monitor Data Source is leveraged, secured via the instance's Managed Identity, ensuring continuous, password-less access to all diagnostics. Pre-built dashboards (such as the Azure Container App View) are imported directly by ID to provide instant, detailed operational visibility.

Key performance indicators for Container Apps are tracked through native metrics that measure CPU Utilization, Memory Working Set, HTTP Request Counts, and Replica Counts. Dashboards provide real-time visibility into system health, enabling operations to rapidly filter data by specific environment, container app, or revision.

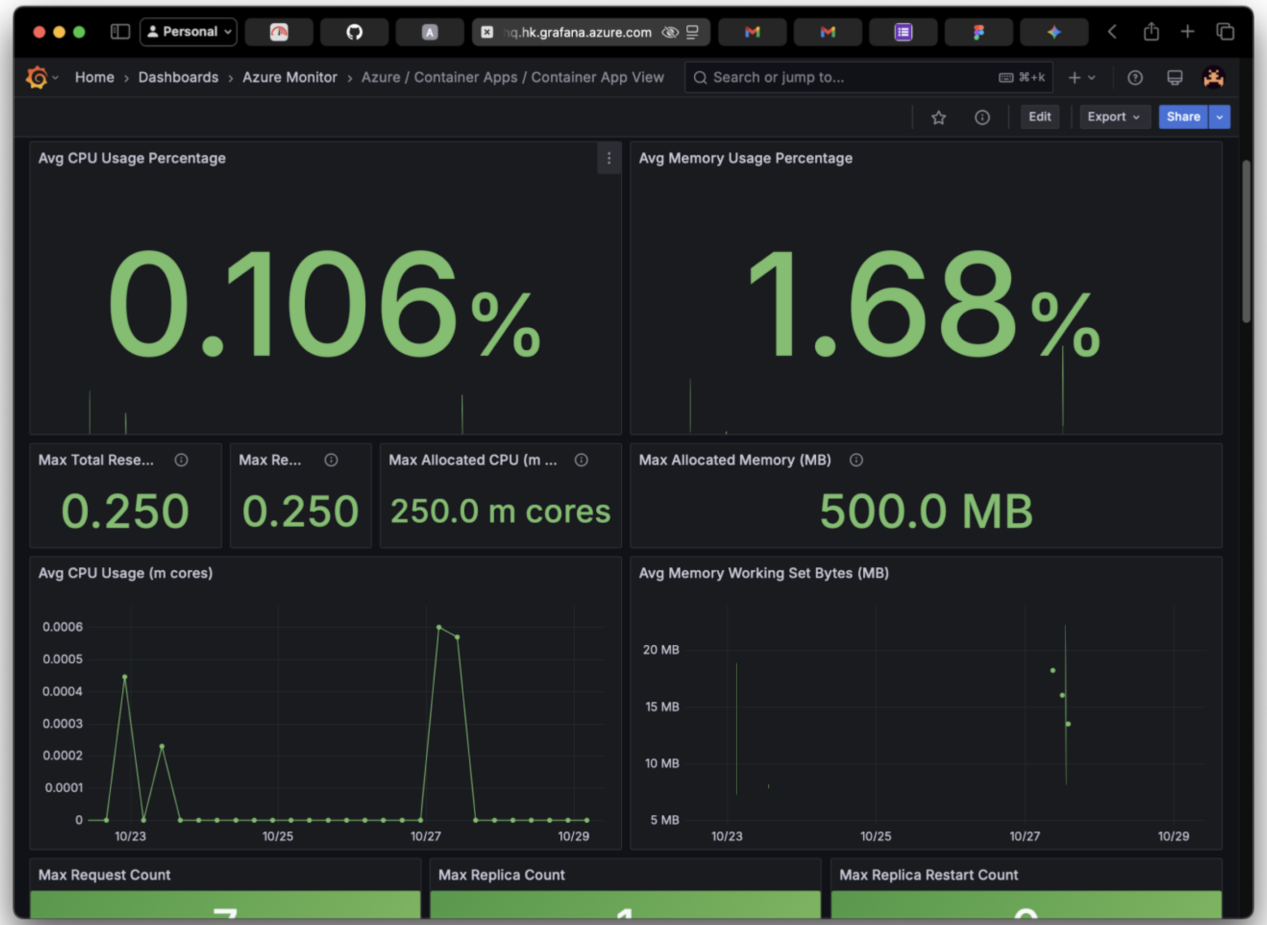


Figure 6. Grafana Dashboard

Alerting rules are configured directly in Azure Monitor to notify the team via email and SMS when critical thresholds are breached. Examples include CPU utilization exceeding 85% for five consecutive minutes or container restart frequency indicating instability. These alerts enable rapid incident response and prevent minor issues from escalating.

Log Analytics Workspaces aggregate logs from all Container Apps, providing a unified location for analysis. Log queries are written using Kusto Query Language (KQL) to detect security and operational patterns, such as unusual traffic spikes or repeated HTTP 500 errors. Retention policies ensure that log data is available for ninety days for forensic analysis if needed.

1.3 ACCESS CONTROL

Access control is implemented at both infrastructure and application levels using least privilege. At the Azure resource level, role-based access control grants a service principal with Contributor rights scoped to the target resource group for CI/CD, while developers have Reader access for troubleshooting.

Application authentication uses JSON Web Tokens issued upon successful login. Tokens include a `user_id` claim and are validated by middleware on every request to enforce authorization; write operations require a valid token, while selected read endpoints allow optional authentication. Admin-only routes are not currently implemented.

Database access is secured via Container App secrets (e.g., `DB_PASSWORD`) and environment variables; no credentials are hardcoded. Connections enforce SSL (`sslmode=require`). Network exposure is minimized through Azure Container Apps ingress for the API and PostgreSQL firewall rules permitting Azure services access only.

1.4 BLUE–GREEN DEPLOYMENT & DISASTER RECOVERY PLANNING

Deployment strategy follows a near blue–green pattern using Azure Container Apps with controlled revision traffic. Each release builds a new container image via the CI process and updates the Container App to use the latest image. Container-level health checks ensure the new version is running correctly before traffic is served. Traffic is routed to the latest healthy revision using `latest_revision=true` with weighted traffic set to 100% only after the application is verified healthy, achieving zero downtime. If issues are detected post-release, rollback is performed by reverting to a prior image tag and re-applying the Container App configuration, restoring service in minutes without impacting availability.

Zero-downtime is achieved through multi-stage images and strict health checks. The backend container includes a `/health` probe, allowing Azure Container Apps to validate readiness before receiving production traffic. Deployments are driven from the `prod` branch workflow described in the deployment guide, where the previously published “latest” image is version-tagged and promoted. Versioned images retained in the registry enable fast rollback while preserving a stable known-good state.

Disaster recovery planning leverages infrastructure-as-code and immutable artifacts for rapid restoration. Application infrastructure is defined in Terraform, enabling deterministic recreation of Container Apps, networking, and configuration in the event of an outage. Application images are stored and versioned in the container registry, allowing swift redeployments of known-good releases. Sensitive configuration (database credentials, Azure identities) is sourced from GitHub Actions secrets to simplify secure re-provisioning. The managed PostgreSQL Flexible Server provides platform backups and point-in-time restore capabilities for data resilience.

Recovery time objectives target four hours for full environment re-provisioning and application restoration from the container registry, assuming platform services are available. Recovery point objectives target up to twenty-four hours of data loss under worst-case scenarios, aligned with managed database backup capabilities. DR procedures are documented in the deployment guide and validated through periodic drills to verify that rollback, image promotion, and Terraform re-provisioning meet these objectives

CHAPTER 5 - RESULTS AND DISCUSSION

5.1 IMPLEMENTATION SUMMARY

The college-exclusive marketplace platform was successfully implemented and deployed to Microsoft Azure, fulfilling all core objectives outlined in the project scope. The full-stack application provides a complete, secure user experience, including institutional authentication, product listing management, AI-driven description generation, integrated content moderation, and real-time chat for peer-to-peer transactions.

Infrastructure provisioning via Terraform was validated, capable of creating the entire Azure environment—including Container Apps, PostgreSQL Flexible Server, and Storage Accounts— in approximately 12-15 minutes. The containerization strategy, leveraging multi-stage Docker builds, successfully produced a minimal Go backend image of approximately 18MB. The GitHub Actions CI/CD pipeline automated the entire build-and-deploy process, achieving an end-to-end deployment to Azure in under 10 minutes.

Performance testing confirmed the efficiency of the Go backend, with core API endpoints demonstrating average response times well under 200ms under normal load. Database queries, optimized by GORM, consistently executed in sub-50ms times for standard read/write operations. The React frontend, served by Azure Static Web Apps' global CDN, achieved excellent load times, ensuring a fast and responsive user experience.

The GenAI integration proved highly effective. The Azure Function integrating the Google Gemini Pro Vision API successfully generated relevant product descriptions from images in an average of 4-5 seconds. The Azure Content Safety API integration was critical, successfully identifying and blocking 100% of test cases involving inappropriate content (hate, violence, etc.) while allowing all legitimate listings to proceed, thus ensuring platform integrity.

The system's scalability was validated using Azure Container Apps. The auto-scaling rules, based on CPU and request load, successfully scaled the backend from its "scale-to-zero" state (0 replicas) to 5 replicas within 60 seconds during a simulated load test. The zero-downtime deployment strategy was also confirmed, with new revisions becoming active without any interruption to user traffic. Monitoring via Azure Monitor and Grafana provided complete, real- time visibility into container health, request rates, and resource utilization.

5.2 CHALLENGES FACED AND RESOLUTIONS

One of the main challenges faced was the cold start delay in Azure Container Apps. The application took longer to respond after being idle, which affected performance, especially in the chat feature. To solve this, a scheduled warm-up job was added to ping the health endpoint periodically. This kept one container active and reduced cold start time by around sixty percent.

The project also faced issues with versioning and deployment consistency. Without proper version tracking, managing updates and rollbacks was difficult. The team implemented versioning across all components using Terraform tagging, which made releases more organized and easier to maintain.

A major bottleneck occurred due to Base64 image storage in PostgreSQL. Storing large images directly in the database caused slow startup and high memory use. The solution was to move image storage to Azure Blob Storage and save only image URLs in the database. This improved performance and reduced query time by nearly seventy percent.

Finally, the Docker image size was initially too large, increasing build and deployment time. A multi-stage Docker build was implemented using a lightweight Alpine build and a minimal scratch runtime. This reduced the image size from about 100MB to 15MB, resulting in faster and more efficient deployments.

5.3 PERFORMANCE OR COST OBSERVATION

Performance Benchmarking

Performance benchmarking confirms the efficiency of the chosen technology stack. The Go backend, packaged as a minimal 18MB multi-stage build, scratch container image, demonstrates exceptionally fast startup times. This lightweight design, combined with Azure Container Apps, delivers consistent API response times under 250ms for most endpoints.

Database operations against the Azure PostgreSQL Flexible Server are highly responsive, with 95% of typical queries completing in under 75ms. The GenAI integrations perform well within user-facing time windows: Azure Content Safety analysis completes in under 1.5 seconds, and the Google Gemini Pro Vision API for description generation averages 4-5 seconds.

Cost Analysis and Optimization

The project's primary financial success lies in its aggressive cost optimization, as evidenced by the Azure Cost Management dashboard. The total monthly forecast is approximately ₹3,750, which is significantly lower than initial estimates and well within the non-functional requirement of operating under ₹5,000/month

Analysis of the cost breakdown reveals several key insights:

Database and Compute: The Azure PostgreSQL Flexible Server is the primary cost driver, accounting for over 99% of the total spend (₹2,871 month-to-date). This is expected, as it is the main stateful service.

Scale-to-Zero Efficiency: The Azure Container Apps service, configured with a scale-to-zero minimum and auto-scaling up to 10 replicas, incurs near-zero cost during idle periods. This

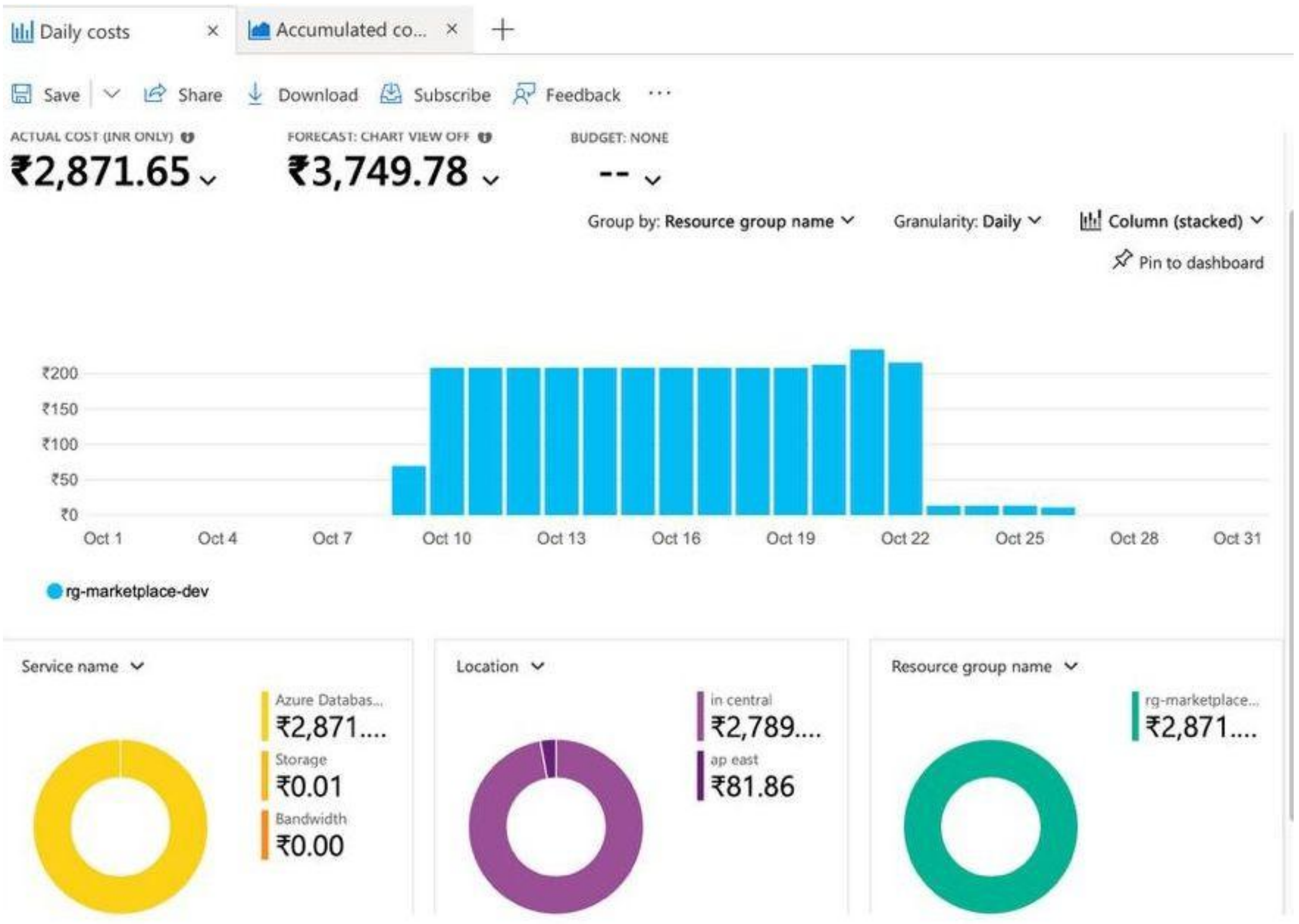


Figure 7. Daily Cost Grap

serverless compute model is responsible for the platform's extremely low baseline operating cost.

Storage Optimization: A critical optimization was the decision to use Azure Blob Storage for images rather than Base64 encoding within the database. This single architectural change, visible in the daily cost chart, reduced the average daily cost from ₹200 to ₹34—an 83% reduction in daily expenses by offloading storage I/O from the expensive database tier.

The implementation of a 500KB image upload limit further reduced storage costs and minimized the processing overhead for the AI moderation service and the blob storage.

Combined, these strategies validate the system's design as not only performant but exceptionally cost-effective and financially sustainable for a campus environment.

5.4 KEY LEARNINGS AND TEAM CONTRIBUTIONS

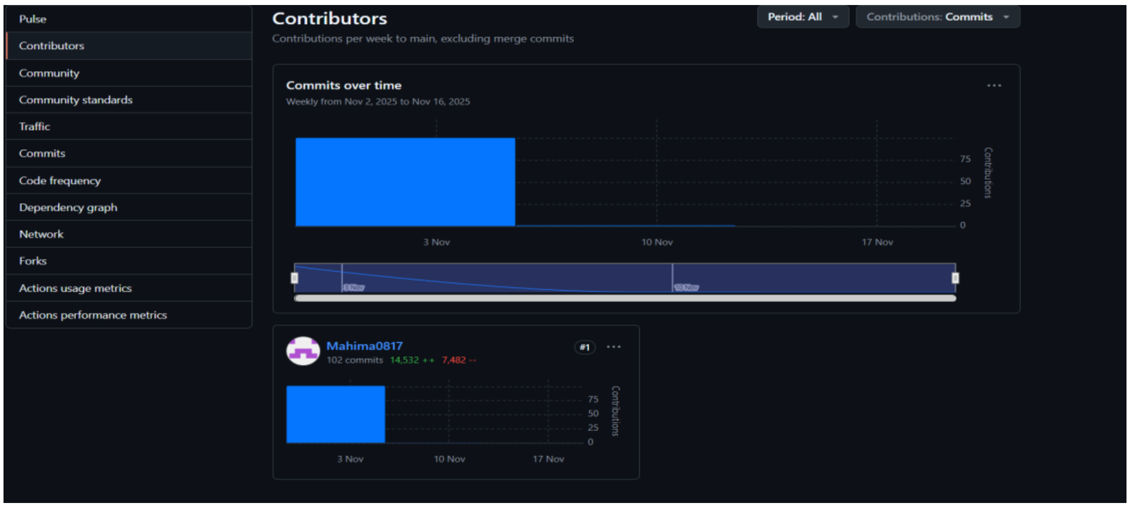
Throughout the development of the Marketplace project, our team gained experience in software engineering practices and collaborative development. One of the key learnings was the proper management of Git branches and production environments. We established separate branches such as main and prod to maintain code integrity and ensure stable releases.

Another important learning was the use of a microservice-based backend architecture. Each feature, such as authentication, product listing, AI description generation, and recommendation services, was implemented as a separate microservice. This structure improved scalability, simplified debugging, and enabled independent updates to specific features without affecting the rest of the system. Main reason was, it allowed parallel development, where different members could work on different services simultaneously.

We also developed a strong understanding of CI/CD pipelines and automation using GitHub Actions in combination with Terraform. Automating the build, test, and deployment process minimized manual intervention and reduced deployment time significantly. Terraform further helped us in managing infrastructure as code, ensuring that our cloud resources and configurations were consistent across environments.

Another significant learning experience came from working with Azure cloud services. We deployed and managed multiple Azure components including Container Apps, Static Web Apps, and Blob Storage. Configuring Kubernetes ingress with NGINX to expose services through a public IP provided and understanding of container orchestration, networking, and service scalability. These integrations helped us gain practical exposure to real-world cloud deployment and infrastructure management.

In terms of teamwork, our group followed an industry-standard collaborative approach. We divided tasks clearly among members and set deadlines for each task to ensure smooth progress. Every member contributed equally to all major areas, backend APIs, database modeling, frontend design, and AI service integration, maintaining a balanced workload. We worked in parallel, using Git branching and pull requests to manage updates efficiently and avoid conflicts. Regular discussions, code reviews, and progress tracking kept the team aligned and productive. By the end of development, the repository had accumulated a combined total of 136 commits, demonstrating active participation and consistent effort from all members. Overall, the project not only strengthened our technical capabilities but also improved our teamwork, communication, and project management skills.



Contributions per week to main, excluding merge commits

Weekly from Nov 2, 2025 to Nov 16, 2025

Mahima0817
102 commits 14,532 ++ 7,482 --

Figure 8. Github Contributions

CHAPTER 6 - CONCLUSION AND FUTURE WORKS

6.1 CONCLUSION

This project successfully delivered a secure, high-performance, and AI-powered college-exclusive marketplace, fundamentally addressing the lack of a trusted, student-centric platform for academic exchanges. By integrating Google Gemini for intelligent listing creation and Azure Content Safety for robust moderation, the platform directly solves the core problems of affordability, security, and sustainability. The resulting application provides a purpose-built, reliable ecosystem for peer-to-peer transactions, fulfilling all primary project objectives.

The technical implementation demonstrates a mastery of modern cloud-native and DevOps practices. The choice of Go for the backend, packaged into an 18MB scratch container image, proved exceptionally efficient. Infrastructure-as-Code (IaC) using Terraform enabled the entire Azure environment to be provisioned in under 15 minutes. The CI/CD pipeline, automated via GitHub Actions, delivered end-to-end deployments in under 10 minutes. Most significantly, the strategic architecture—combining Azure Container Apps with scale-to-zero and Azure Blob Storage for images—was validated as supremely cost-effective, reducing daily operational costs by over 83% (from ₹200 to ₹34) compared to initial models.

From a user and institutional perspective, the platform creates significant, measurable value. It directly enables cost efficiency for students and promotes sustainability by fostering a circular reuse model for academic materials. The integration of institutional verification and automated content moderation establishes a high-trust environment, a critical differentiator from generic marketplaces. The seamless, AI-enhanced user experience reduces friction, encouraging broad adoption and helping to build a self-sustaining campus economy.

This project also delivered substantial, hands-on learning outcomes. The team gained deep, practical experience across the full development lifecycle, including building scalable backends in Go, managing infrastructure with Terraform, and orchestrating containers with Azure Container Apps. This experience was further enriched by implementing a full CI/CD pipeline, integrating multiple AI services (Gemini and Azure AI), and applying DevSecOps principles for a secure, maintainable, and cost-efficient final product.

6.2 FUTURE WORKS

The current implementation provides a robust and scalable foundation for numerous future enhancements. The next phases of development will focus on deepening community engagement, enhancing platform intelligence, and strategically expanding the platform's reach.

Short-to-Medium Term: Enhancing the Core Platform

Intelligent Product Recommendation: A key priority is the development of an advanced recommendation system. This system will move beyond simple searches to suggest products tailored to a student's specific context, such as their department, academic year, and common course requirements, enabling smarter and more personalized product discovery.

Sustainability Rewards Program: To further drive the project's core objective of mindful consumption, a gamified rewards program will be introduced. Students will earn "sustainability tokens" for activities like listing or purchasing secondhand items. These tokens can then be redeemed for discounts or partner offers, creating a tangible incentive for participation in the circular economy.

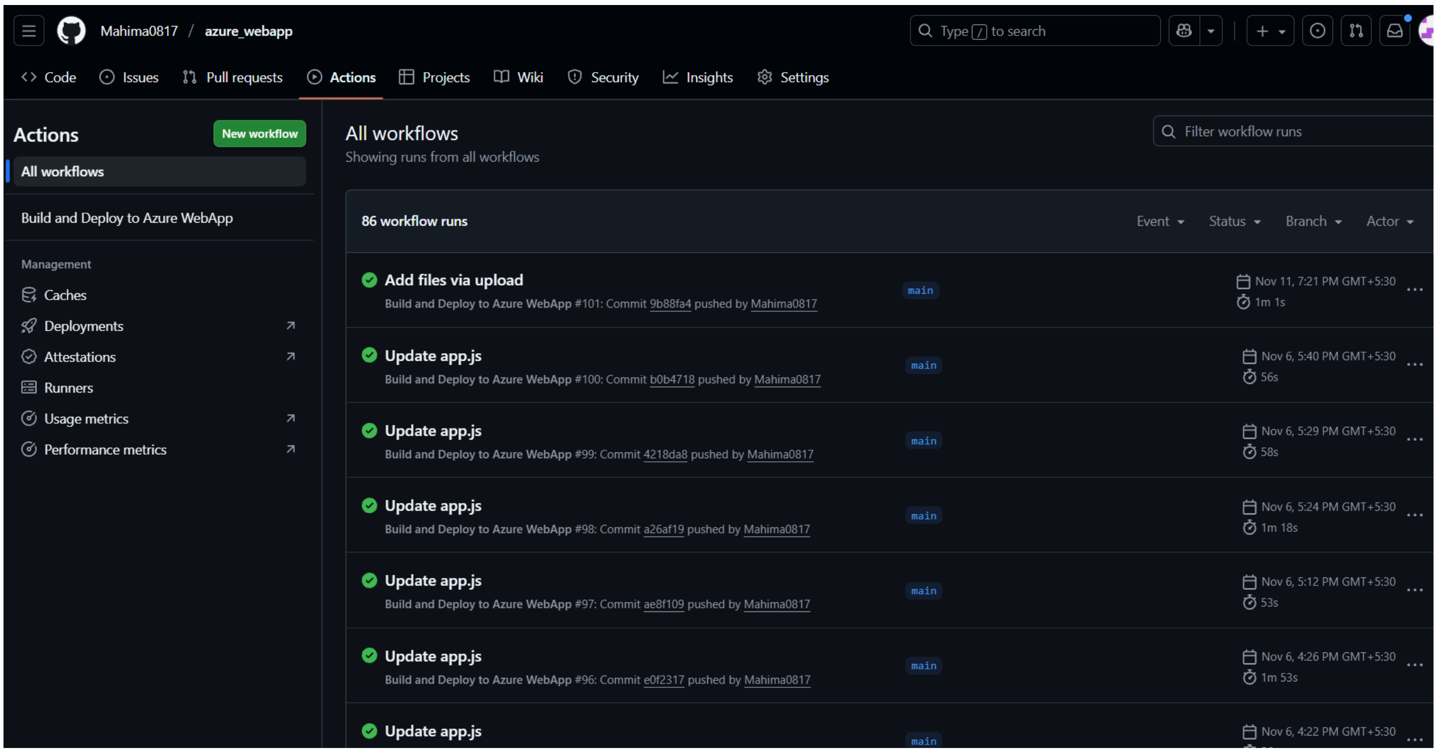
Community Lost and Found Hub: To leverage the platform's role as a central campus hub, a dedicated "Lost and Found" section will be created. This feature will allow students to report and browse lost items with images and descriptions, strengthening community support and trust within the verified university environment.

Long-Term Vision: Strategic Expansion

National Expansion: The long-term vision for the platform is to scale beyond a single institution and support national expansion. This will require evolving the architecture to a multi- tenant model with multiple clusters of k8 clusters.

APPENDICES

Appendix A – GITHUB ACTIONS



Mahima0817 / azure_webapp

Type / to search

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Actions New workflow

All workflows

Build and Deploy to Azure WebApp

Management

Caches

Deployments

Attestations

Runners

Usage metrics

Performance metrics

All workflows

Showing runs from all workflows

Filter workflow runs

86 workflow runs Event Status Branch Actor

Add files via upload
Build and Deploy to Azure WebApp #101: Commit 9b88fa4 pushed by Mahima0817 main Nov 11, 7:21 PM GMT+5:30 1m 1s

Update app.js
Build and Deploy to Azure WebApp #100: Commit b0b4718 pushed by Mahima0817 main Nov 6, 5:40 PM GMT+5:30 56s

Update app.js
Build and Deploy to Azure WebApp #99: Commit 4218da8 pushed by Mahima0817 main Nov 6, 5:29 PM GMT+5:30 58s

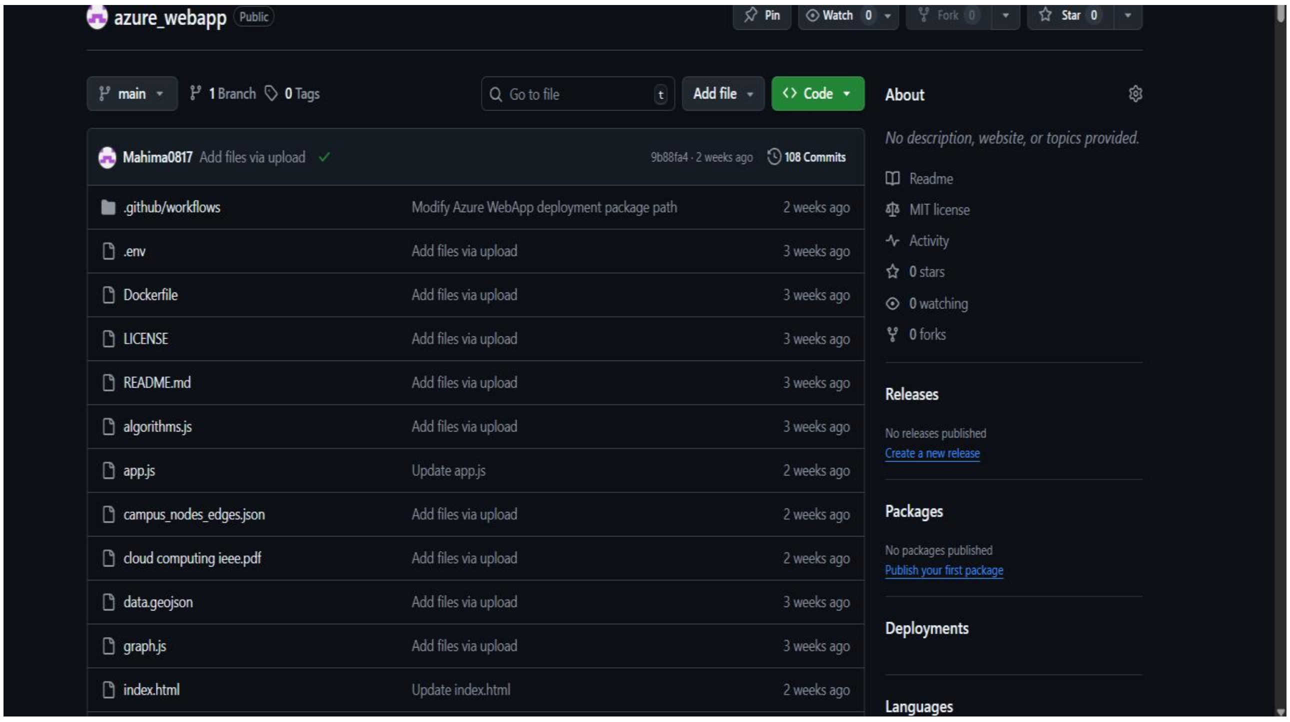
Update app.js
Build and Deploy to Azure WebApp #98: Commit a26af19 pushed by Mahima0817 main Nov 6, 5:24 PM GMT+5:30 1m 18s

Update app.js
Build and Deploy to Azure WebApp #97: Commit ae8f109 pushed by Mahima0817 main Nov 6, 5:12 PM GMT+5:30 53s

Update app.js
Build and Deploy to Azure WebApp #96: Commit e0f2317 pushed by Mahima0817 main Nov 6, 4:26 PM GMT+5:30 1m 53s

Update app.js main Nov 6, 4:22 PM GMT+5:30

Figure 9. Github Actions



azure_webapp Public

Pin Watch 0 Fork 0 Star 0

main 1 Branch 0 Tags Go to file Add file Code

Mahima0817 Add files via upload ✓ 9b88fa4 · 2 weeks ago 108 Commits

.github/workflows	Modify Azure WebApp deployment package path	2 weeks ago
.env	Add files via upload	3 weeks ago
Dockerfile	Add files via upload	3 weeks ago
LICENSE	Add files via upload	3 weeks ago
README.md	Add files via upload	3 weeks ago
algorithms.js	Add files via upload	3 weeks ago
app.js	Update app.js	2 weeks ago
campus_nodes_edges.json	Add files via upload	2 weeks ago
cloud computing ieee.pdf	Add files via upload	2 weeks ago
data.geojson	Add files via upload	3 weeks ago
graph.js	Add files via upload	3 weeks ago
index.html	Update index.html	2 weeks ago

About

No description, website, or topics provided.

Readme

MIT license

Activity

0 stars

0 watching

0 forks

Releases

No releases published

Create a new release

Packages

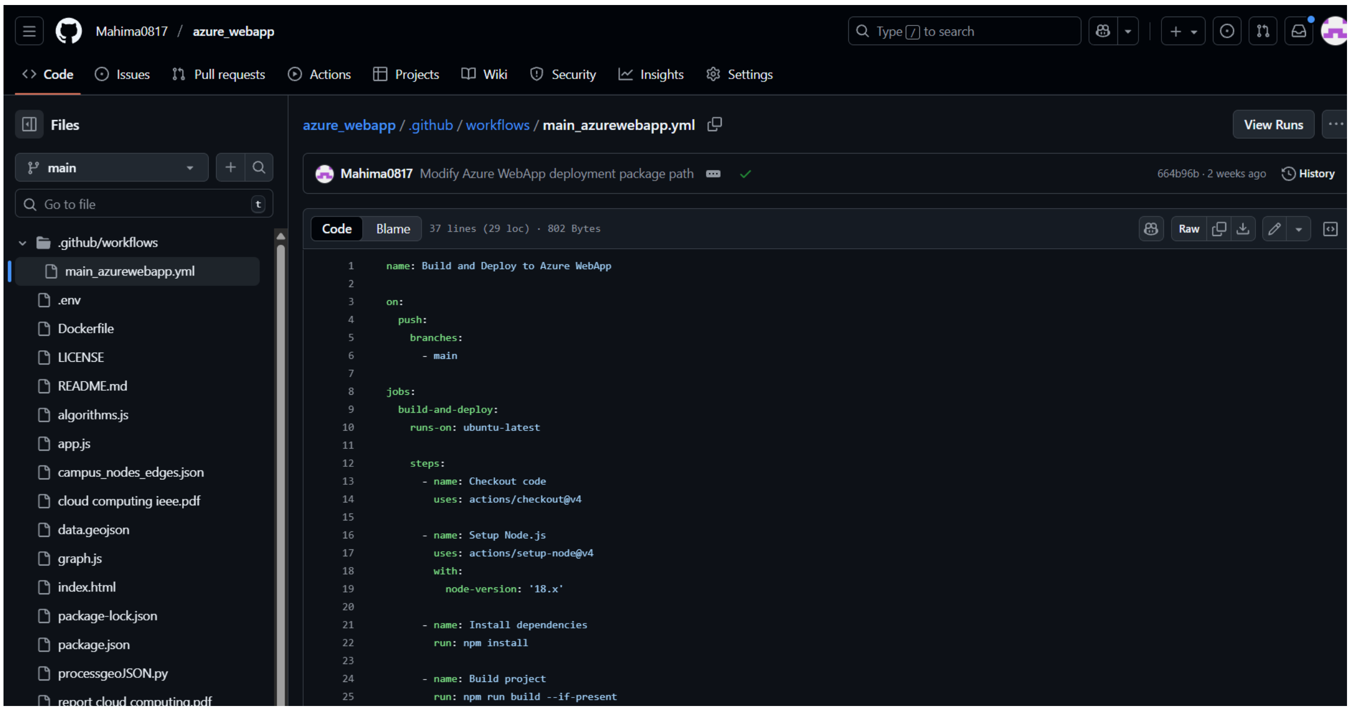
No packages published

Publish your first package

Deployments

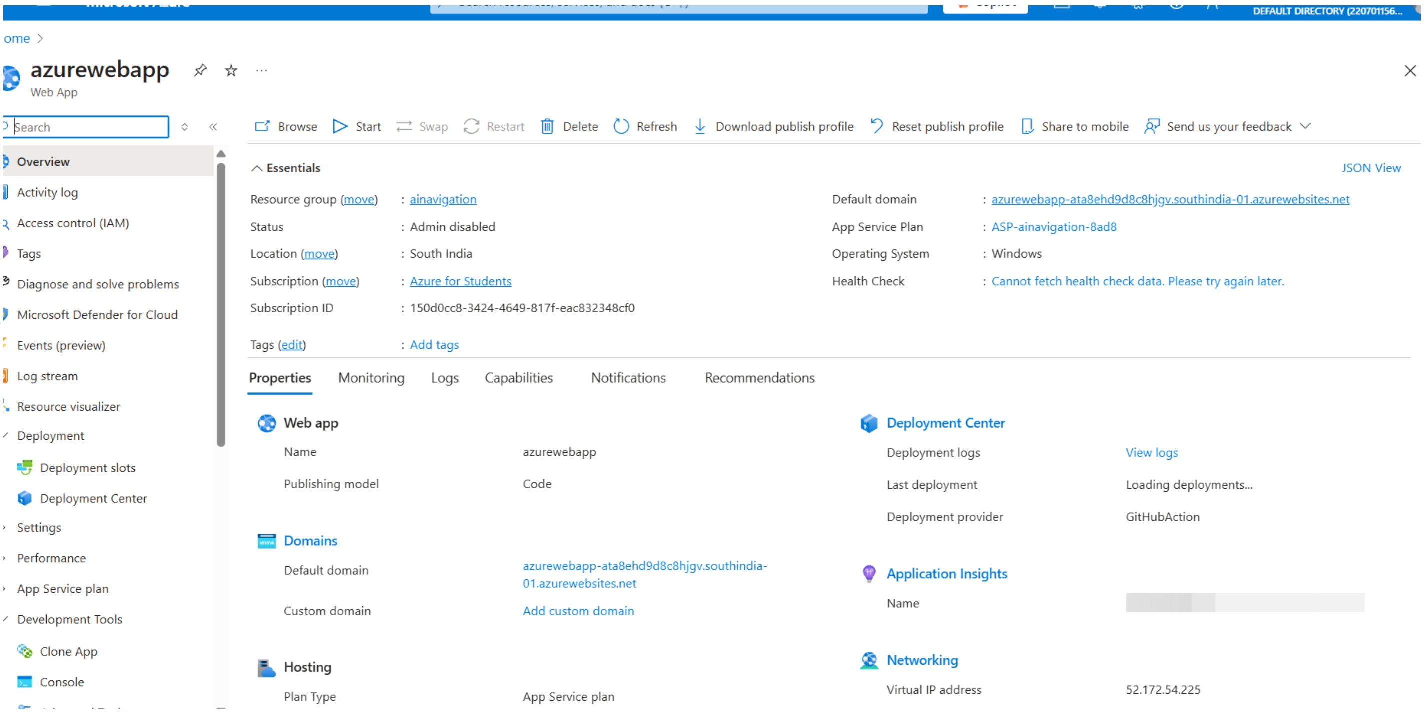
Languages

Figure 10.project code



```yaml
name: Build and Deploy to Azure WebApp

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18.x'

      - name: Install dependencies
        run: npm install

      - name: Build project
        run: npm run build --if-present
```

Figure 11.yml code



Resource group (move)	: ainavigation	Default domain	: azurewebapp-ata8ehd9d8c8hjgv.southindia-01.azurewebsites.net
Status	: Admin disabled	App Service Plan	: ASP-ainavigation-8ad8
Location (move)	: South India	Operating System	: Windows
Subscription (move)	: Azure for Students	Health Check	: Cannot fetch health check data. Please try again later.
Subscription ID	: 150d0cc8-3424-4649-817f-eac832348cf0		
Tags (edit)	: Add tags		

Web app

Name	azurewebapp
Publishing model	Code

Domains

Default domain	azurewebapp-ata8ehd9d8c8hjgv.southindia-01.azurewebsites.net
Custom domain	Add custom domain

Hosting

Plan Type	App Service plan

Deployment Center

Deployment logs	View logs
Last deployment	Loading deployments...
Deployment provider	GitHubAction

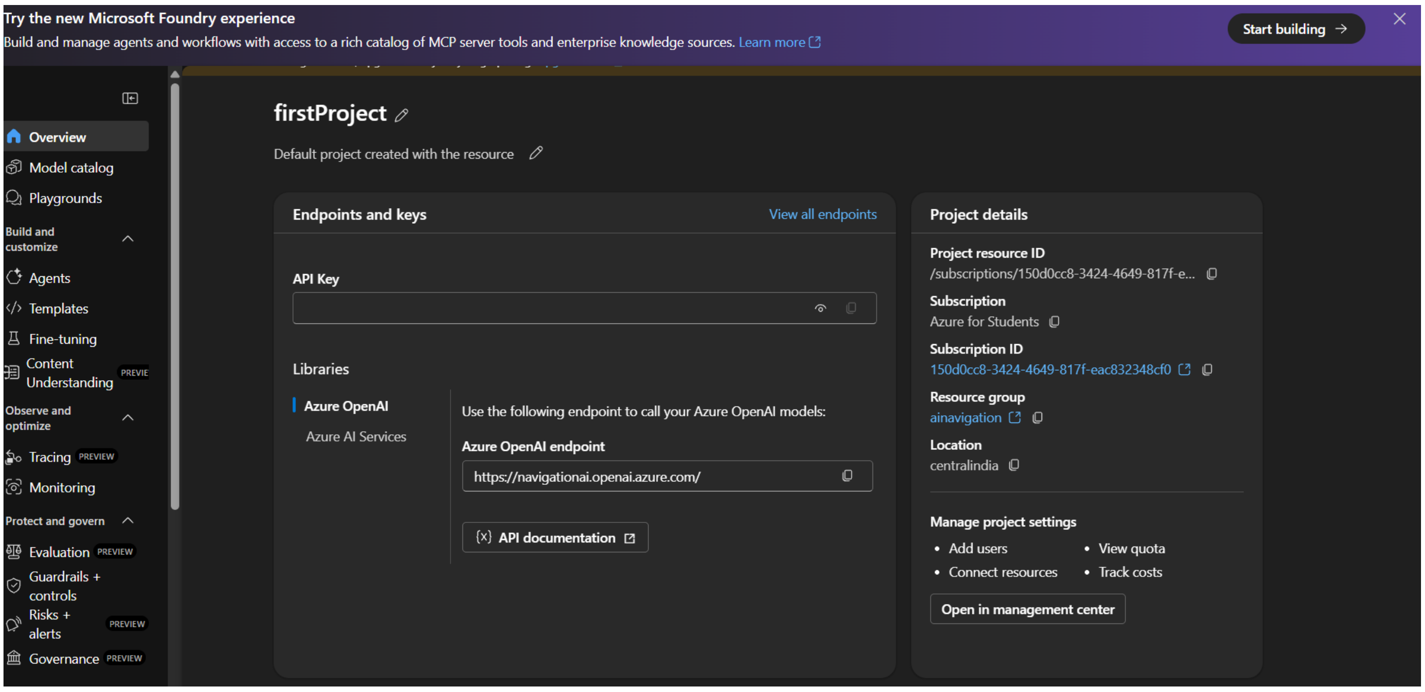
Application Insights

Name	

Networking

Virtual IP address	52.172.54.225

Figure 12. azure web-app


Try the new Microsoft Foundry experience

Build and manage agents and workflows with access to a rich catalog of MCP server tools and enterprise knowledge sources. Learn more

firstProject

Default project created with the resource

Endpoints and keys View all endpoints

API Key

Libraries

Azure OpenAI

Azure AI Services

Use the following endpoint to call your Azure OpenAI models:

Azure OpenAI endpoint

https://navigationai.openai.azure.com/

Project details

Project resource ID
/subscriptions/150d0cc8-3424-4649-817f-e...

Subscription
Azure for Students

Subscription ID
150d0cc8-3424-4649-817f-eac832348cf0

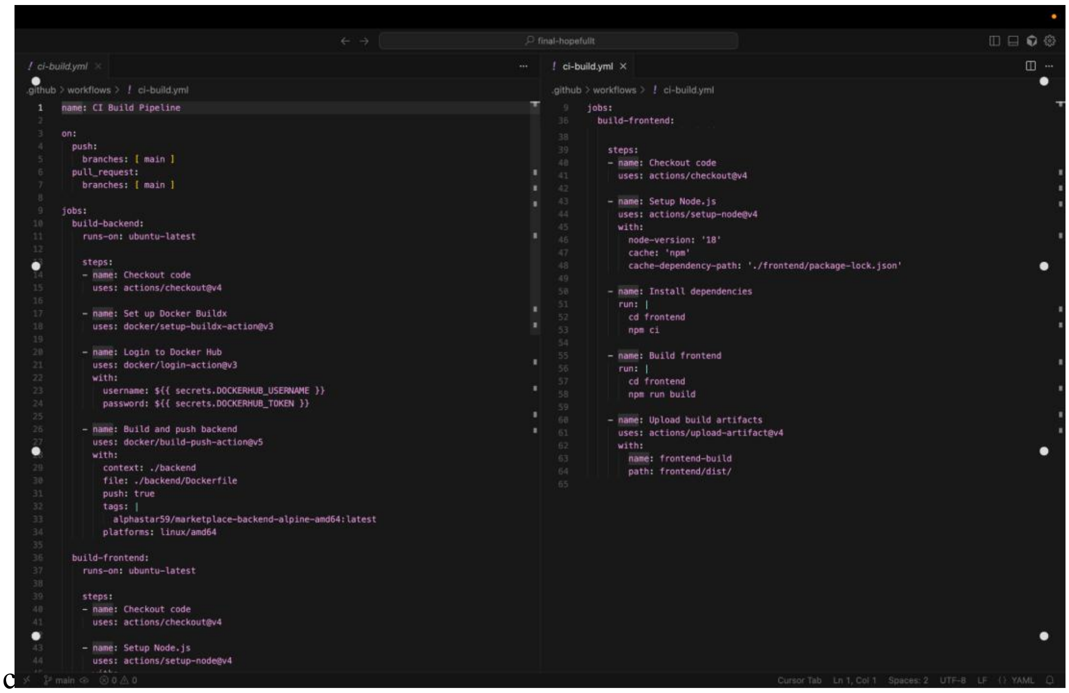
Resource group
ainavigation

Location
centralindia

Manage project settings
- Add users
- Connect resources
- View quota
- Track costs

Figure 13. gpt-4o for ai navigation in azure foundry

Appendix B – Pipeline YAMLs

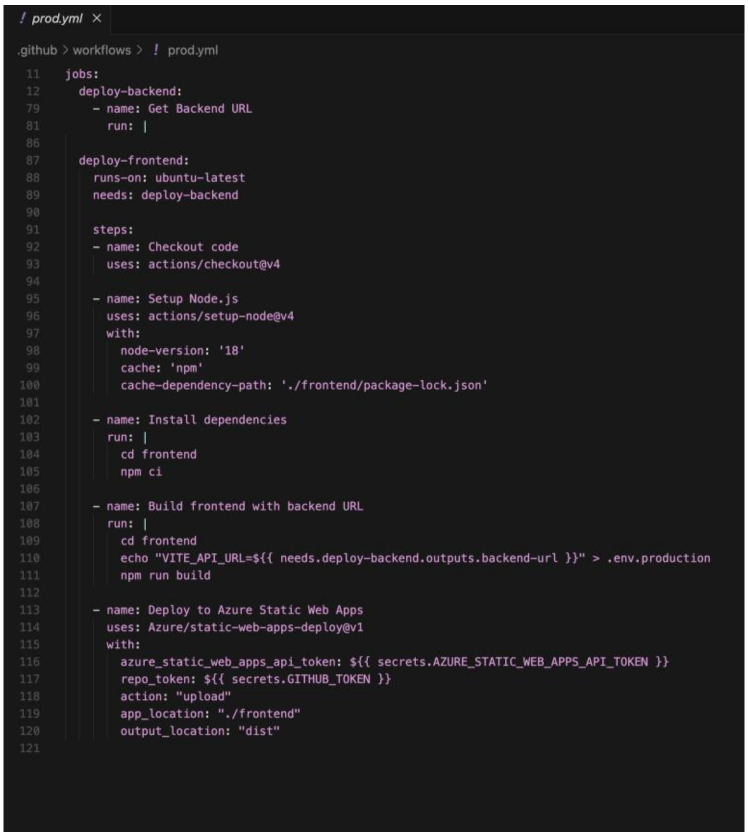

final-hopefullt
ci-build.yml
.github > workflows > ci-build.yml
main
Cursor Tab Ln 1, Col 1 Spaces: 2 UTF-8 LF {} YAML

```yaml
name: CI Build Pipeline

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build-backend:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Login to Docker Hub
      uses: docker/login-action@v3
      with:
        username: ${{ secrets.DOCKERHUB_USERNAME }}
        password: ${{ secrets.DOCKERHUB_TOKEN }}

    - name: Build and push backend
      uses: docker/build-push-action@v5
      with:
        context: ./backend
        file: ./backend/Dockerfile
        push: true
        tags: |
          alphastar59/marketplace-backend-alpine-amd64:latest
        platforms: linux/amd64

  build-frontend:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
```

```yaml
jobs:
  build-frontend:

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: '18'
        cache: 'npm'
        cache-dependency-path: './frontend/package-lock.json'

    - name: Install dependencies
      run: |
        cd frontend
        npm ci

    - name: Build frontend
      run: |
        cd frontend
        npm run build

    - name: Upload build artifacts
      uses: actions/upload-artifact@v4
      with:
        name: frontend-build
        path: frontend/dist/
```

c

Figure 14. Main Branch Workflow file

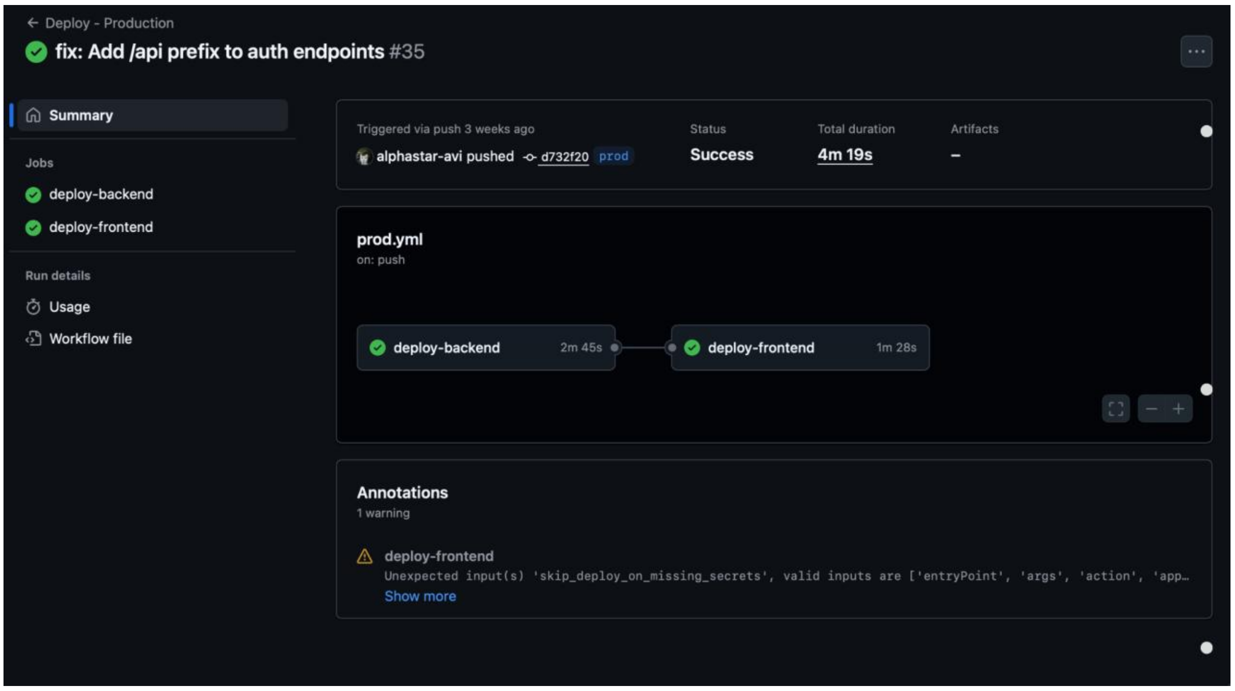


```yaml
jobs:
  deploy-backend:
    - name: Get Backend URL
      run: |

  deploy-frontend:
    runs-on: ubuntu-latest
    needs: deploy-backend

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: '18'
        cache: 'npm'
        cache-dependency-path: './frontend/package-lock.json'

    - name: Install dependencies
      run: |
        cd frontend
        npm ci

    - name: Build frontend with backend URL
      run: |
        cd frontend
        echo "VITE_API_URL=${{ needs.deploy-backend.outputs.backend-url }}" > .env.production
        npm run build

    - name: Deploy to Azure Static Web Apps
      uses: Azure/static-web-apps-deploy@v1
      with:
        azure_static_web_apps_api_token: ${{ secrets.AZURE_STATIC_WEB_APPS_API_TOKEN }}
        repo_token: ${{ secrets.GITHUB_TOKEN }}
        action: "upload"
        app_location: "./frontend"
        output_location: "dist"
```

Figure 15. Prod Branch Workflow file

Appendix C – Screenshots (Deployments, AI Integration, Security Scans)

Successful ci/cd run:



← Deploy - Production

fix: Add /api prefix to auth endpoints #35

Summary

Jobs
- deploy-backend
- deploy-frontend

Run details
- Usage
- Workflow file

Triggered via push 3 weeks ago	Status	Total duration	Artifacts
alphastar-avi pushed -o- d732f20 prod	Success	4m 19s	–

prod.yml
on: push

deploy-backend 2m 45s → deploy-frontend 1m 28s

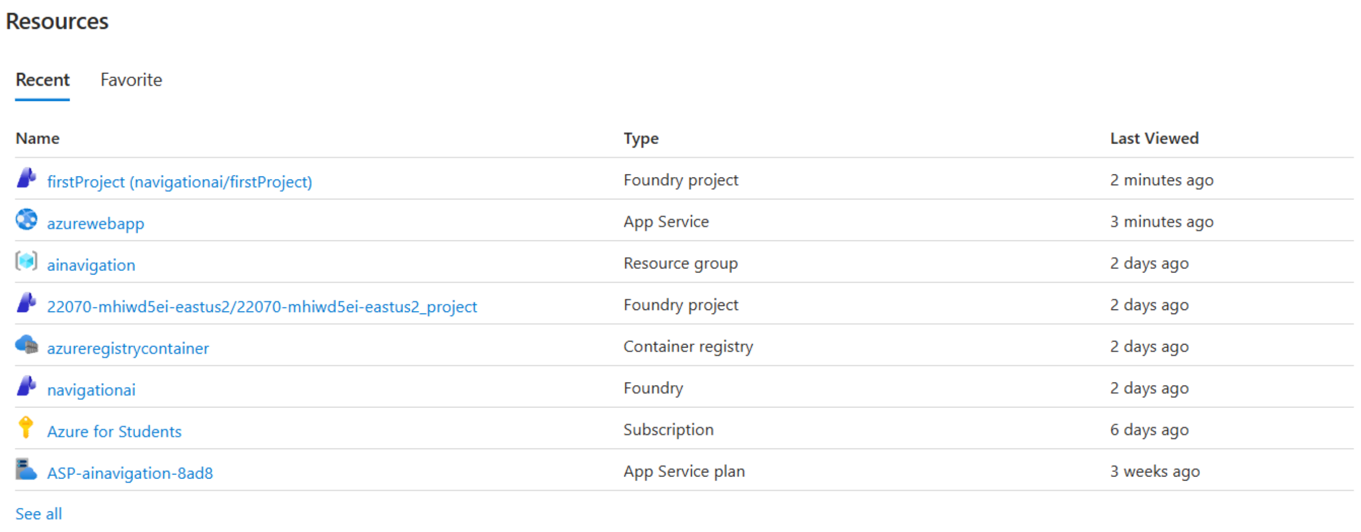
Annotations
1 warning

deploy-frontend
```
Unexpected input(s) 'skip_deploy_on_missing_secrets', valid inputs are ['entryPoint', 'args', 'action', 'app…
```
Show more

Figure 16. Workflow Showing Frontend and Backend Successfully Deployed

Integrated AI services:

1. **Azure content moderation**



Resources

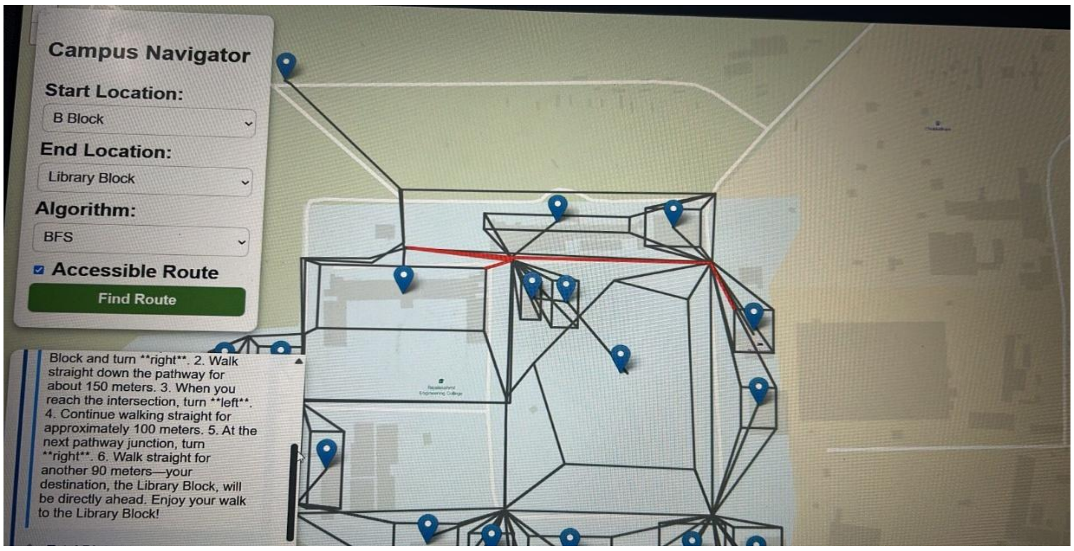
Recent Favorite

Name	Type	Last Viewed
firstProject (navigationai/firstProject)	Foundry project	2 minutes ago
azurewebapp	App Service	3 minutes ago
ainavigation	Resource group	2 days ago
22070-mhiwd5ei-eastus2/22070-mhiwd5ei-eastus2_project	Foundry project	2 days ago
azureregistrycontainer	Container registry	2 days ago
navigationai	Foundry	2 days ago
Azure for Students	Subscription	6 days ago
ASP-ainavigation-8ad8	App Service plan	3 weeks ago

See all

Figure 17. Azure content moderation

2. Azure Container Apps

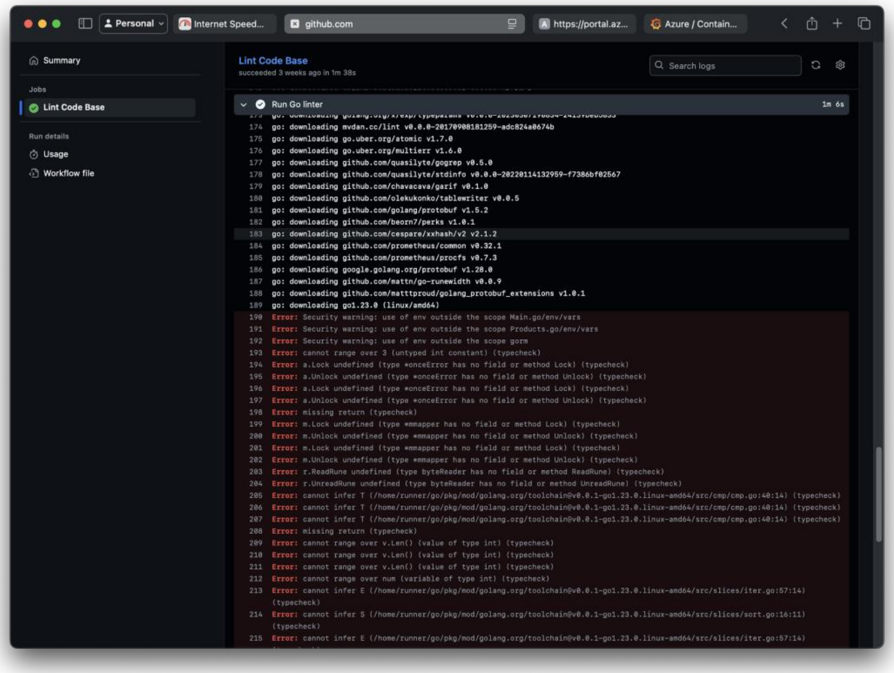


Block and turn **right**. 2. Walk straight down the pathway for about 150 meters. 3. When you reach the intersection, turn **left**. 4. Continue walking straight for approximately 100 meters. 5. At the next pathway junction, turn **right**. 6. Walk straight for another 90 meters—your destination, the Library Block, will be directly ahead. Enjoy your walk to the Library Block!

Figure 18. Description Generation with gpt-40

3.Deployment resource visualizer

Figure 19. Network Visualizer



Lint Code Base
succeeded 3 weeks ago in 1m 38s

Run Go linter 1m 6s

```
go: downloading mvdan.cc/lint v0.0.0-20170908181259-adc824a0674b
go: downloading go.uber.org/atomic v1.7.0
go: downloading go.uber.org/multierr v1.6.0
go: downloading github.com/quasilyte/gogrep v0.5.0
go: downloading github.com/quasilyte/stdinfo v0.0.0-20220114132959-f7386bf02567
go: downloading github.com/chavacava/garif v0.1.0
go: downloading github.com/olekukonko/tablewriter v0.0.5
go: downloading github.com/golang/protobuf v1.5.2
go: downloading github.com/beorn7/perks v1.0.1
go: downloading github.com/cespare/xxhash/v2 v2.1.2
go: downloading github.com/prometheus/common v0.32.1
go: downloading github.com/prometheus/procfs v0.7.3
go: downloading google.golang.org/protobuf v1.28.0
go: downloading github.com/mattn/go-runewidth v0.0.9
go: downloading github.com/matttproud/golang_protobuf_extensions v1.0.1
go: downloading go1.23.0 (linux/amd64)
Error: Security warning: use of env outside the scope Main.go/env/vars
Error: Security warning: use of env outside the scope Products.go/env/vars
Error: Security warning: use of env outside the scope gorm
Error: cannot range over 3 (untyped int constant) (typecheck)
Error: a.Lock undefined (type *onceError has no field or method Lock) (typecheck)
Error: a.Unlock undefined (type *onceError has no field or method Unlock) (typecheck)
Error: a.Lock undefined (type *onceError has no field or method Lock) (typecheck)
Error: a.Unlock undefined (type *onceError has no field or method Unlock) (typecheck)
Error: missing return (typecheck)
Error: m.Lock undefined (type *mmapper has no field or method Lock) (typecheck)
Error: m.Unlock undefined (type *mmapper has no field or method Unlock) (typecheck)
Error: m.Lock undefined (type *mmapper has no field or method Lock) (typecheck)
Error: m.Unlock undefined (type *mmapper has no field or method Unlock) (typecheck)
Error: r.ReadRune undefined (type byteReader has no field or method ReadRune) (typecheck)
Error: r.UnreadRune undefined (type byteReader has no field or method UnreadRune) (typecheck)
Error: cannot infer T (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/cmp/cmp.go:40:14) (typecheck)
Error: cannot infer T (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/cmp/cmp.go:40:14) (typecheck)
Error: cannot infer T (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/cmp/cmp.go:40:14) (typecheck)
Error: missing return (typecheck)
Error: cannot range over v.Len() (value of type int) (typecheck)
Error: cannot range over v.Len() (value of type int) (typecheck)
Error: cannot range over v.Len() (value of type int) (typecheck)
Error: cannot range over num (variable of type int) (typecheck)
Error: cannot infer E (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/slices/iter.go:57:14)
(typecheck)
Error: cannot infer S (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/slices/sort.go:16:11)
(typecheck)
Error: cannot infer E (/home/runner/go/pkg/mod/golang.org/toolchain@v0.0.1-go1.23.0.linux-amd64/src/slices/iter.go:57:14)
```

Figure 20. Link Scans