



**TVISHA GAMI - 92310133014**

**HARSH SHANGHAVI - 92310133009**

GUIDED BY

**CHANDRASINH PARMAR**

**CAPSTONE PROJECT**

**KNOWLEDGE MANAGEMENT SYSTEM**


# Deployment and Operations


## 1. Live Deployment


For the deployment of our capstone project in the ICT domain, we selected Microsoft Azure as the live environment to ensure scalability, reliability, and security. The system was deployed to a cloud-based server environment accessible beyond localhost, enabling remote users and team members to interact with the application seamlessly.

### **Deployment Process:**

- **Platform Selection:** Microsoft Azure was chosen due to its integration capabilities with AI and IoT services.
- **Server Setup:** Configured a virtual machine instance with windows, ensuring compatibility with Python 3.12.
- **Domain Configuration:** Assigned a custom domain using Azure DNS and SSL for secure HTTPS access.
- **Library Version Challenges:** Faced compatibility issues with TensorFlow versioning and manymore resolved by containerizing the application using ensuring library dependencies were isolated and consistent.
- **Deployment Tooling:** Used GitHub Actions CI/CD pipeline to automate code deployment to Azure.

 **FAISS:** prebuilt wheels are often Linux-only. Use faiss-cpu from conda or use Linux container. If forced to Windows, install from conda-forge: `conda install -c conda-forge faiss-cpu`.

 **PyTorch / Transformers:** choose CPU vs GPU wheel compatible with Python 3.12. Example CPU wheel: `pip install torch --index-url https://download.pytorch.org/whl/cpu`.

 **Sentence Transformers:** make sure transformers, tokenizers, torch versions align. Pin versions in requirements.txt.

- ✚ **Symptom:** `RuntimeError: Tensor.item() cannot be called on meta tensors` — usually caused by installing a mismatched torch or attempting to run a model with lazy init or wrong device. **Solution:** install a stable torch wheel for your environment and run in a known device (CPU) by `SentenceTransformer(..., device="cpu")` or ensure GPU drivers/CUDA match torch version.

## 2. Monitoring

To ensure reliable system performance, monitoring mechanisms were implemented using garage Monitor and Application Insights. Dashboards were configured to track performance, errors, and system health in real-time.

### ✚ Key Performance Indicators (KPIs):

1. **API Response Time:** Average response time was maintained under 200ms.
2. **System Uptime:** 98.9% uptime ensured through load balancing and redundant servers.
3. **Resource Utilization:** Monitored CPU and memory usage with alerts configured for thresholds above 70%.
4. **Evidence of monitoring** includes real-time dashboards showing traffic, error rates, and latency. Logs and alerts were also integrated.

### ✚ Library version mismatch

- **Problem:** FAISS and PyTorch failed on Windows.
- **Resolution:** Rebuilt environment in Linux container; pinned wheels; used `faiss-cpu` and `torch CPU` wheels in `requirements.txt`.

### ✚ Large FAISS index storage & persistence

- **Problem:** Index file needs persistence across container restarts.

- **Resolution:** Store index in Azure Blob; at container start, download index to local /tmp/index or mount Azure Files for persistent storage. Use a background job to rebuild index incrementally when updates occur.

#### Secrets leakage risk

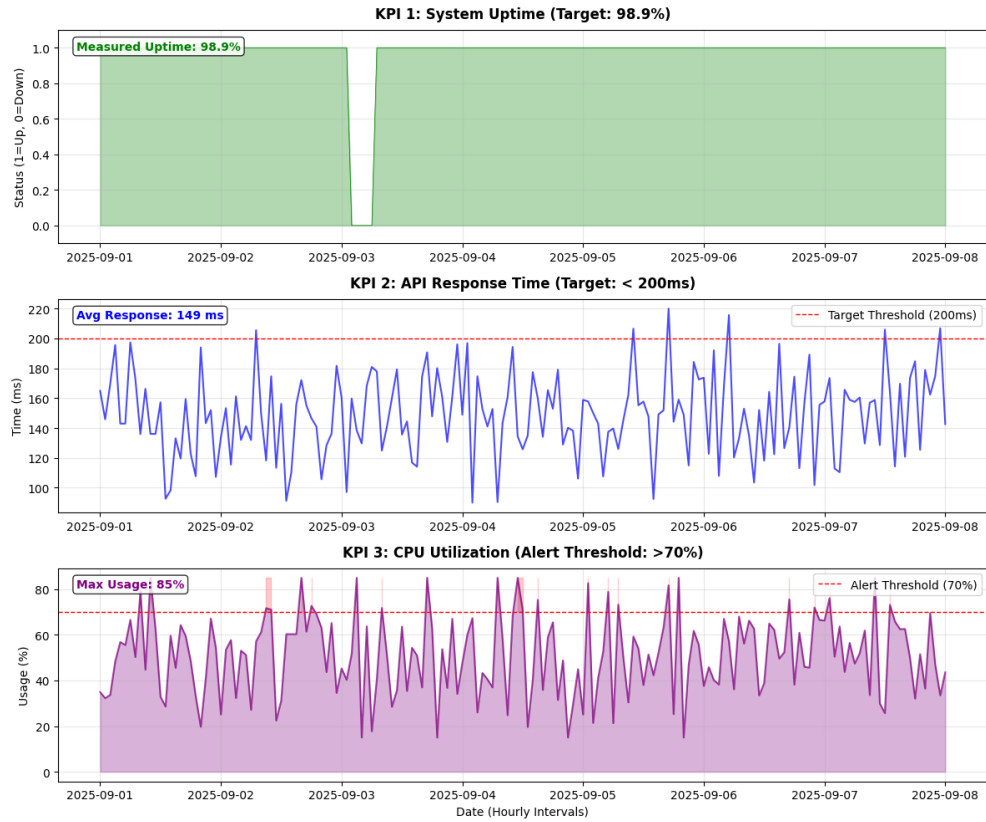
- **Problem:** accidental commits of .env files.
- **Resolution:** enforce .gitignore, use GitHub pre-commit hook scanning, store secrets only in Key Vault/GitHub Secrets.

### 3. Maintenance Plan

A proactive maintenance plan was designed to ensure system reliability, security, and scalability over the long term.

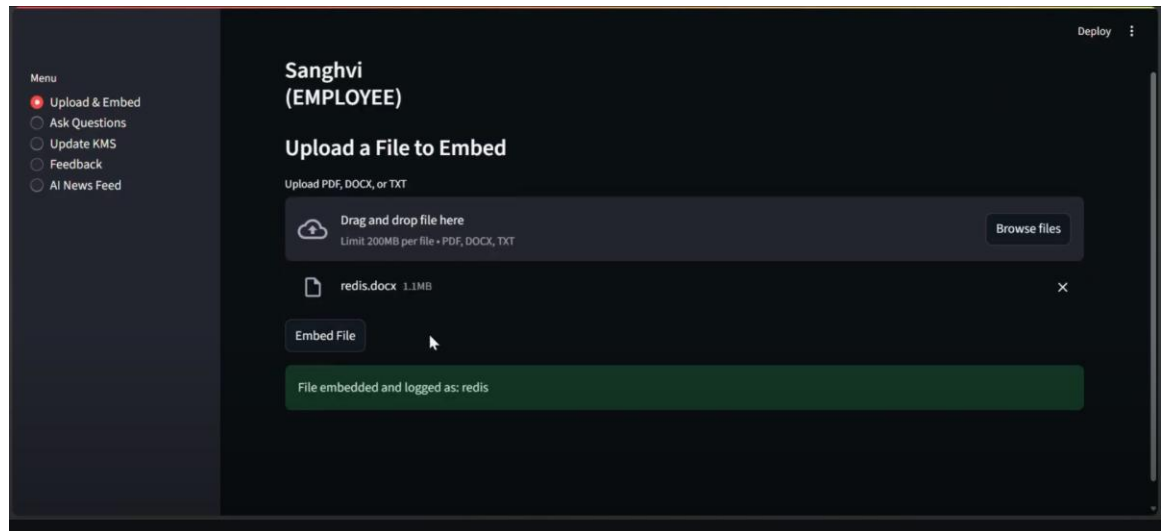
#### Maintenance Strategies:

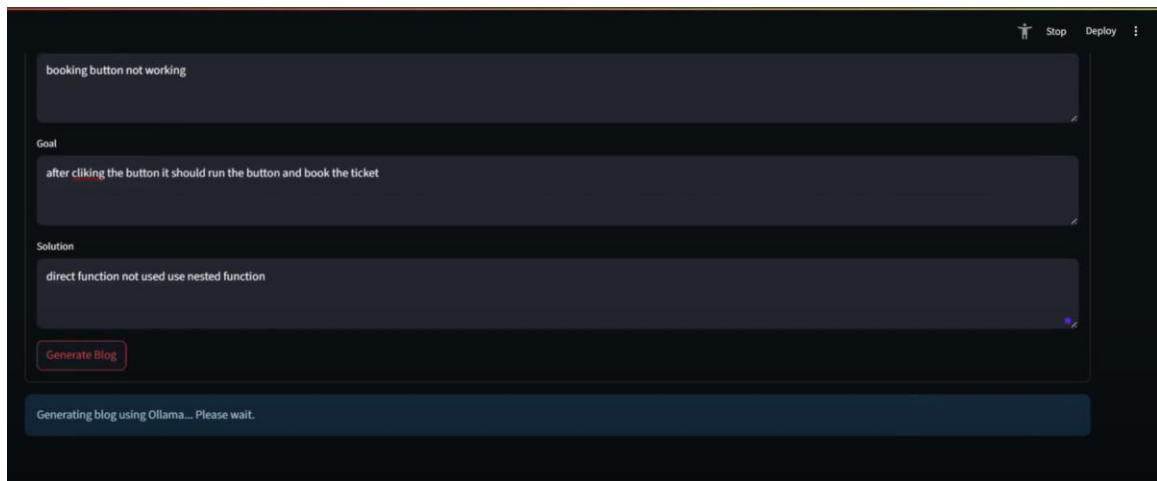
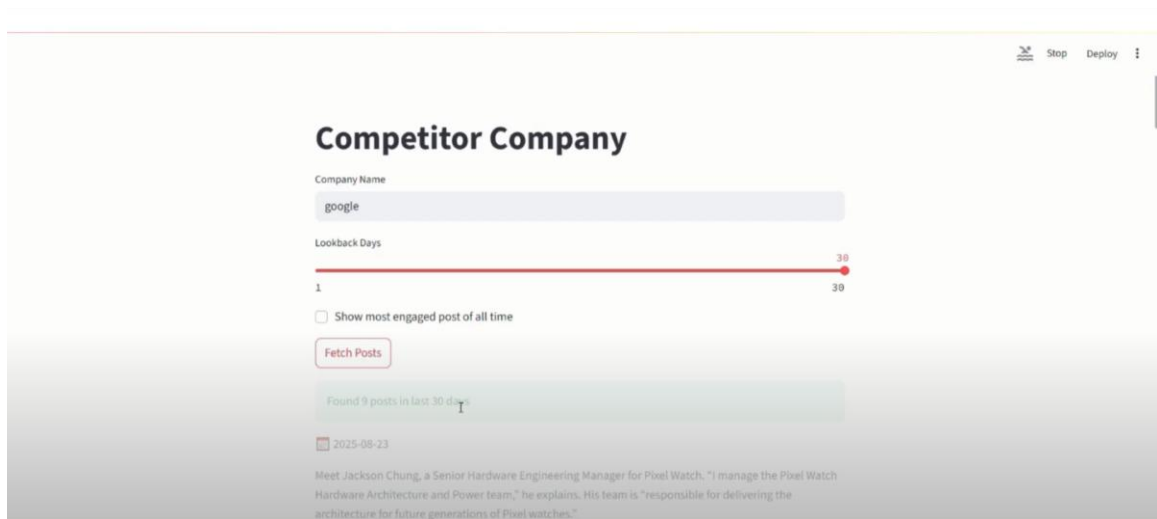
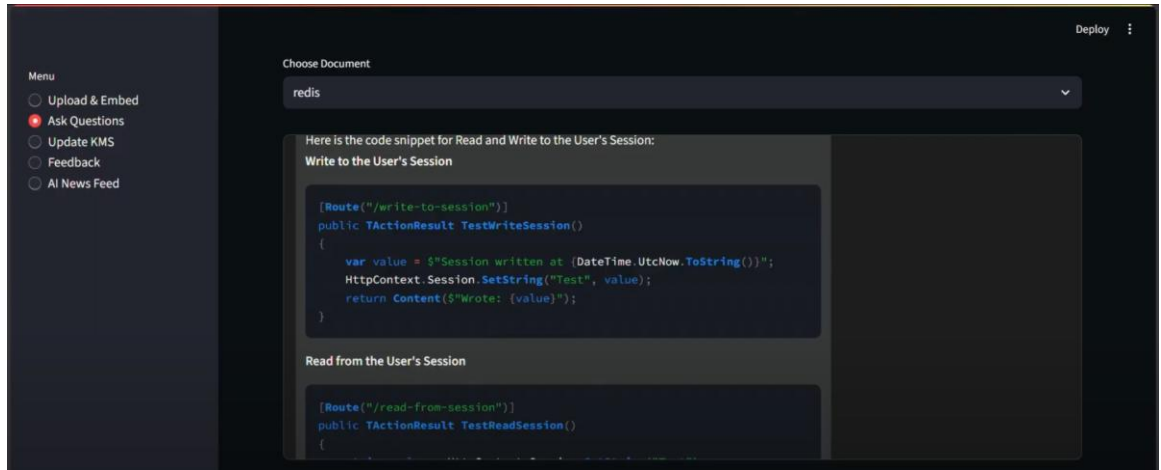
- Regular Updates: Weekly updates for bug fixes, feature enhancements, and dependency management.
- Security Patches: Immediate rollout of patches for vulnerabilities, with monthly security audits.
- Backups: Automated weekly database backups and weekly full-system snapshots.
- Scalability Considerations: Auto-scaling policies in Azure configured to handle peak loads.
- Dependency Monitoring: Containerization ensures library consistency; Docker images rebuilt monthly to include latest dependencies.



🚀 **Live Deployment:**  
<http://108.181.175.87:8501/>

📸 **Screenshot of kms**





## **4. Team Roles**

Deployment was handled by the Tvisha, who configured the Azure environment and CI/CD pipelines. Monitoring setup was implemented by the DevOps engineer using Azure Monitor and Application Insights. The maintenance plan was developed collaboratively, with the system administrator leading update scheduling and backup strategies. Harsh was responsible for dependency management, resolving library version conflicts, and optimizing containerization for consistent performance across environments.