# Table of contents

**Table of Figures** 2

**Project Overview** 3

**Technical Architecture** 3

Cloud-Native 3

Load balancing and HTTP routing 3

Microservices 3

Cluster 3

Communication 4

CI/CD through Skaffold 4

**Microservices Overview** 5

User Account Management Microservice (UserMngService) 5

Storage Management Microservice (StorageMngService) 6

Usage Monitoring Microservice (UsageMngService) 6

Video Presentation Microservice (videoMngService) 6

Logging microService (loggingService) 7

Frontend microService (frontendServ) 7

**M-V-C Architecture** 7

Controller Service: 7

Model Service: 7

Complete Architecture Diagram 8

**Load Testing** 9

Tests conducted: 10

**UI Screenshots**

**11**

**References** 12

# Table of figures

| | |
|---|---|
| Figure 1 | Ingress routing example |
| Figure 2 | services within the cluster and their cluster-IPs |
| Figure 3 | skaffold build configuration for microservice |
| Figure 4 | logs from logging microservice |
| Figure 5 | MVC Architecture Diagram |
| Figure 6 | Complete architecture diagram of project |
| Figure 7 | interface of locust load testing |

| | |
|---|---|
| Figure 8 | logs of locust file |
| Figure 9 | performance testing graphs |
| Figure 10 | Homepage of vid-city |
| Figure 11 | User’s landing page |

# Project Overview

Vid-city is our semester project focused on developing a cloud-based, microservices-driven video streaming platform. The application is deployed as a software-as-a-service (SaaS) on google cloud platform. This platform will enable users to upload, view, and manage short videos, with account management, storage, usage monitoring, and alerting mechanisms integrated into the system. GCP acts as a Platform as a Service (PaaS) to host our system online.

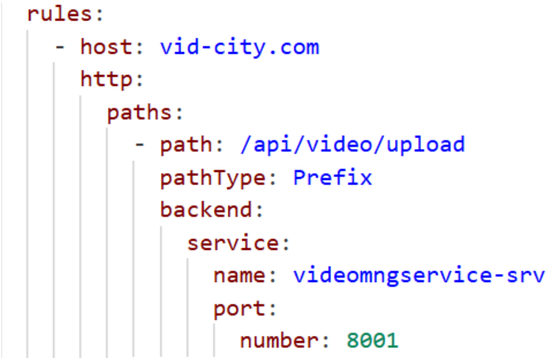
# Technical Architecture

## Cloud-Native

The solution is deployed on **Google Cloud Platform (GCP)**, utilizing PaaS services such as **Google Kubernetes Engine (GKE)** for orchestration and **Google Cloud Storage** for file storage.

## Load balancing and HTTP routing

Kubernetes ingress API object is used to manage external HTTP requests to Kubernetes cluster microservices. The ingress API also provides load balancing and name-based (vid-city in our project) virtual hosting. Refer to figure 1. (*Kubernetes.io*, 2024)



```yaml
rules:
  - host: vid-city.com
    http:
      paths:
        - path: /api/video/upload
          pathType: Prefix
          backend:
            service:
              name: videomngservice-srv
              port:
                number: 8001
```

Figure 1: example of Ingress routing:

The host *vid-city.com* is specified, meaning this Ingress rule applies to requests made to this domain.

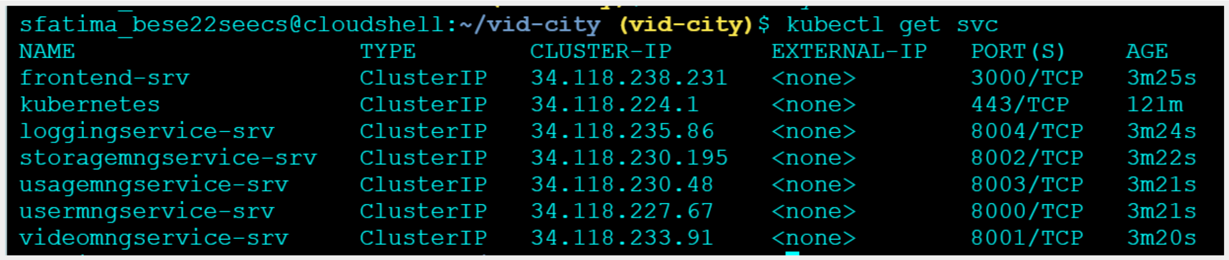
/api/video/upload: Routes traffic to the videomngservice-srv service on port 8001.

## Microservices

The application is divided into independent microservices, each performing a specific function (e.g., rendering front-end, user management, storage management, usage tracking, logging).

## Cluster

Microservices have a private IP address for internal communication within the cluster. To interact with the outside world, Ingress controller is used.



```
sfatima_bese22seecs@cloudshell:~/vid-city (vid-city)$ kubectl get svc
NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
frontend-srv            ClusterIP   34.118.238.231   <none>        3000/TCP   3m25s
kubernetes              ClusterIP   34.118.224.1     <none>        443/TCP    121m
loggingservice-srv      ClusterIP   34.118.235.86    <none>        8004/TCP   3m24s
storagemngservice-srv   ClusterIP   34.118.230.195   <none>        8002/TCP   3m22s
usagemngservice-srv     ClusterIP   34.118.230.48    <none>        8003/TCP   3m21s
usermngservice-srv      ClusterIP   34.118.227.67    <none>        8000/TCP   3m21s
videomngservice-srv     ClusterIP   34.118.233.91    <none>        8001/TCP   3m20s
```

*Figure 2:* microservices within the cluster. They do not have an external-ip address.

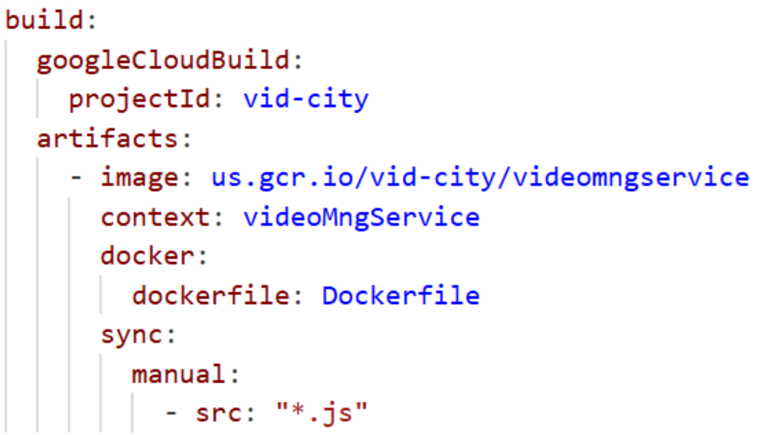
## Communication

All the Microservices communicate within the cluster over private IP addresses. i.e., the services send logs to the logging API by using it’s private IP.

## CI/CD through Skaffold

The project uses **skaffold** for Continuous Integration and Continuous Deployment (CI/CD), a CLI-based tool for automating the deployment of microservices.
skaffold.yaml is used to define the build and deployment process for all the microservices. (*Skaffold Documentation*, 2024)

build configuration for each microservice through skaffold:
Each service in the project (e.g., videomngservice, loggingservice, frontend, etc.) is defined as an artifact in the build section of the Skaffold configuration.



```
build:
  googleCloudBuild:
    projectId: vid-city
  artifacts:
    - image: us.gcr.io/vid-city/videomngservice
      context: videoMngService
      docker:
        dockerfile: Dockerfile
      sync:
        manual:
          - src: "*.js"
```

*Figure 3:* example of build configuration for a microservice

---

- **image:** Specifies the name and location of the Docker image in Google Container Registry (GCR). For instance, **us.gcr.io/vid-city/videomngservice** indicates that the image will be stored under this path in GCR.
- **context:** Points to the directory containing the service's source code, such as videoMngService, which holds the Dockerfile and other relevant files for the service.
- **docker:** Defines the location of the Dockerfile to be used for building the image. Each microservice uses the Dockerfile located in its respective directory.

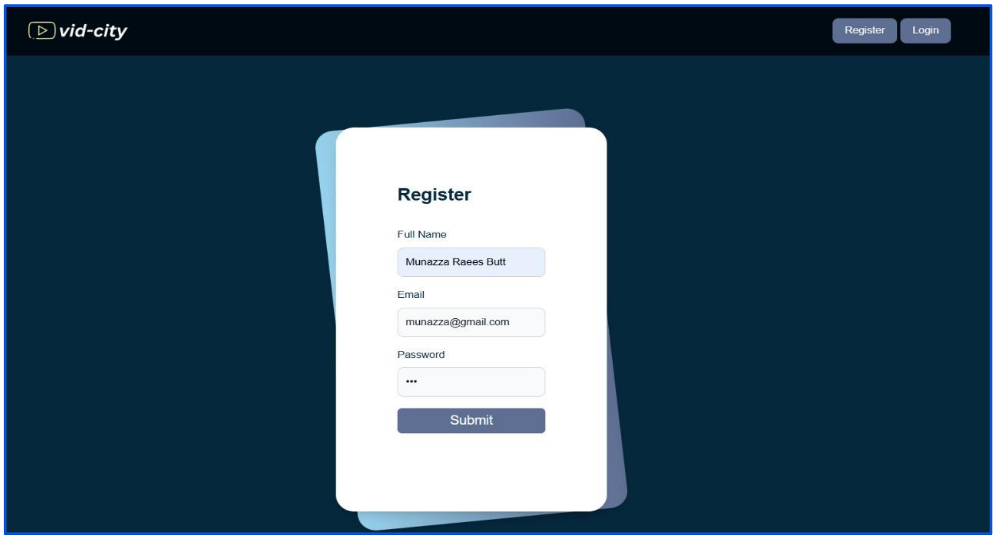
- **sync:** Controls file synchronization between the local environment and the Docker build process. It specifies that any changes to JavaScript files (*.js) should be manually synchronized, ensuring that updates to these files are reflected during the build without needing to rebuild the entire image.

# Microservices Overview

## User Account Management Microservice (UserMngService)

- It handles account creation and authentication of exiting users.
- MongoDB Atlas is used for managing user data such as login credentials.

### *Registration*



Register

Full Name

Munazza Raees Butt

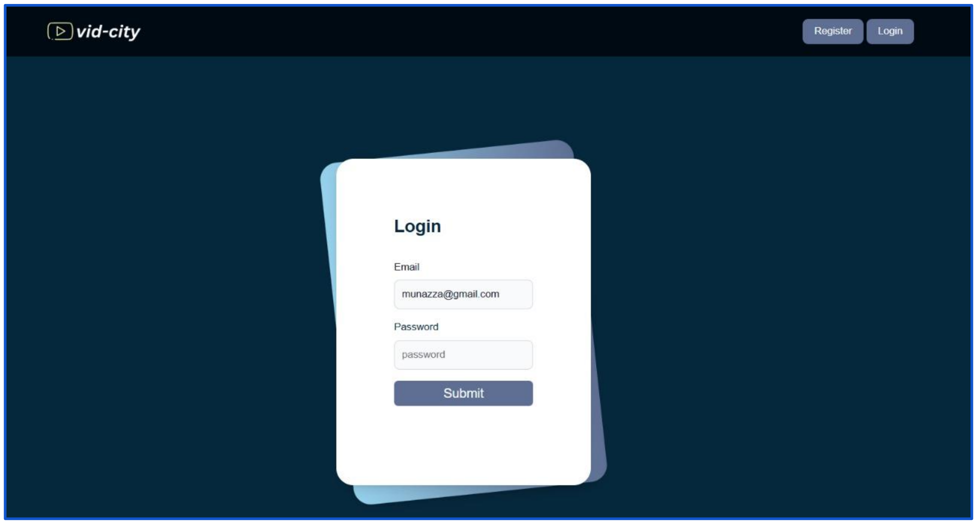
Email

munazza@gmail.com

Password

•••

### *Login*



Login

Email

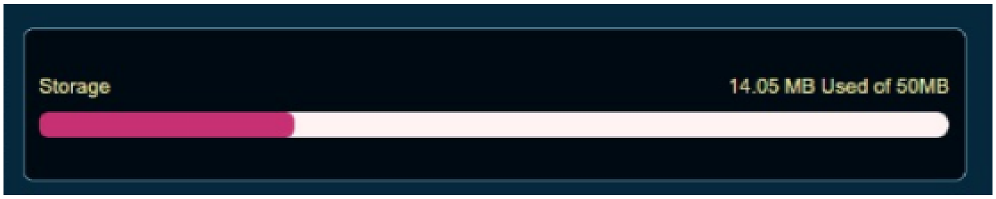
munazza@gmail.com

Password

password

## Storage Management Microservice (StorageMngService)

50MB of cloud storage is allocated per user for video uploads and monitors storage usage.

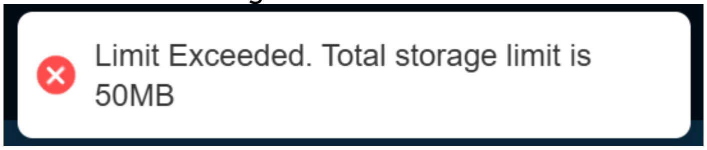


Storage

14.05 MB Used of 50MB

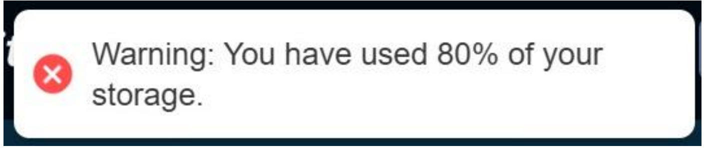
**Storage Limitation:**

Upload is restricted after 50 mb of storage is consumed.



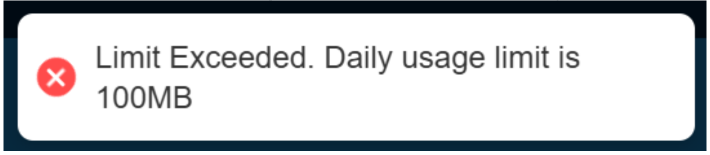
Limit Exceeded. Total storage limit is 50MB

**80% storage Alert:** Sends alerts to users when 80% of their storage quota is consumed.



Warning: You have used 80% of your storage.

**100% Restriction:** Prevents additional uploads once the storage limit is reached.



Limit Exceeded. Daily usage limit is 100MB

This service Interacts with Cloudinary for storing videos, while metadata related to storage usage is tracked in MongoDB Atlas.

## Usage Monitoring Microservice (UsageMngService)

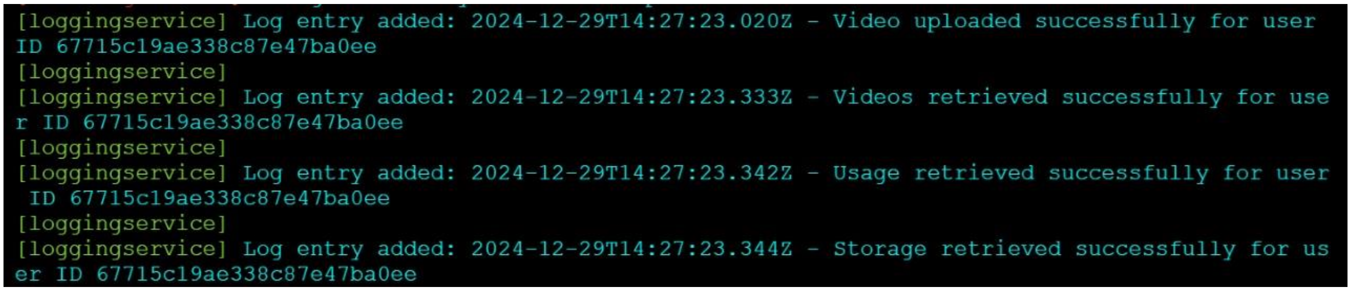
- It tracks data volume usage, including uploads and deletions, and applies daily bandwidth limits.
- Daily Bandwidth Limit:
    - Sets a threshold of 100MB/day per user.
    - Alerts users when the limit is exceeded and restricts uploads for the day.
- MongoDB Atlas is used to log and track usage metrics.

## Video Presentation Microservice (videoMngService)

- Video streaming services like downloading, setting playback speed, viewing in full screen is handled by this service.
- MongoDB Atlas stores the metadata for videos, **cloudinary** is used to store and render the actual videos.

## Logging microService (loggingService)

All other microservices communicate with the logging service through its private address. The logging service can not be accessed from outside the cluster. The logging service then prints logs onto the console and saves it in logs.txt



```
[loggingservice] Log entry added: 2024-12-29T14:27:23.020Z - Video uploaded successfully for user
ID 67715c19ae338c87e47ba0ee
[loggingservice]
[loggingservice] Log entry added: 2024-12-29T14:27:23.333Z - Videos retrieved successfully for use
r ID 67715c19ae338c87e47ba0ee
[loggingservice]
[loggingservice] Log entry added: 2024-12-29T14:27:23.342Z - Usage retrieved successfully for user
 ID 67715c19ae338c87e47ba0ee
[loggingservice]
[loggingservice] Log entry added: 2024-12-29T14:27:23.344Z - Storage retrieved successfully for us
er ID 67715c19ae338c87e47ba0ee
```

*Figure 4:* Logs generated by the logging microservice

## Frontend microService (frontendServ)

- The FrontendService is responsible for rendering the user interface of the application.
- It also manages environment-specific configurations for deployment and local testing.
- It provides dependencies and scripts required for building and running the frontend.

# M-V-C Architecture

Our project follows an MVC architecture in which each microservice has its own model, view (interface) and controller.

### Controller Service:

The controller service defines the functionality of each microservice. The functionality also includes methods for inter-cluster communication.

### Model Service:

- Business logic for services like user authentication, storage management, and usage monitoring is encapsulated.
- Mongodb is used as the database.

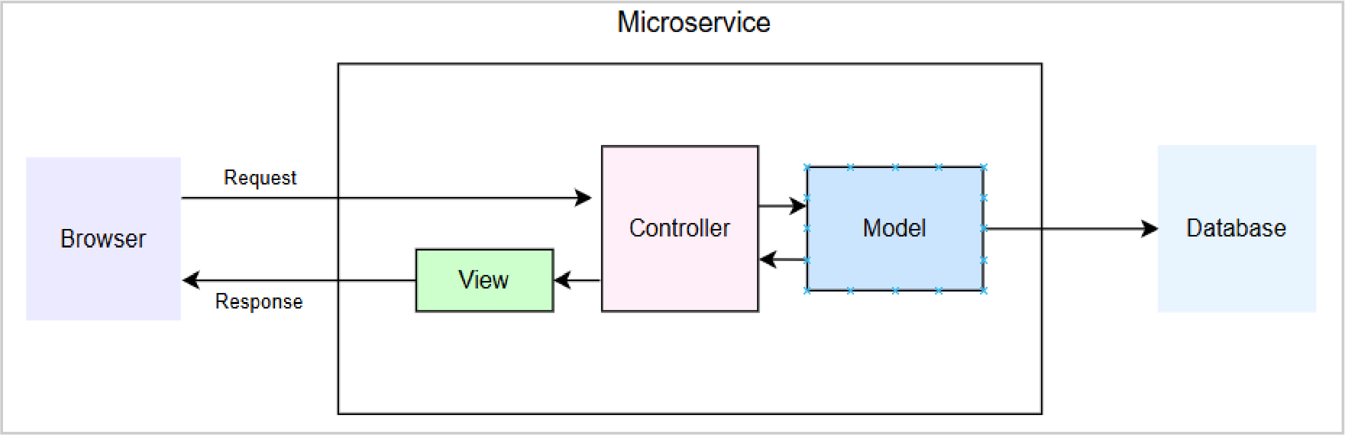


*Figure 5:* The browser sends a request, the Controller processes it using the Model (interacting with the database), and the View generates a response for the browser.

## Complete Architecture Diagram

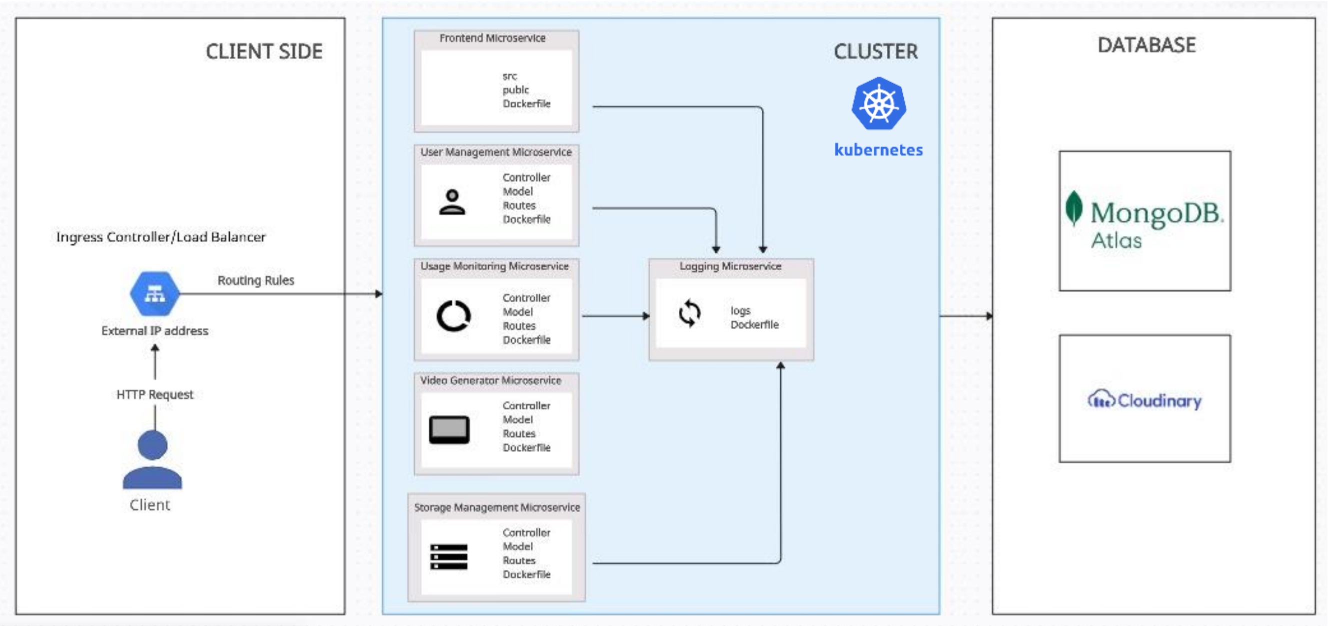
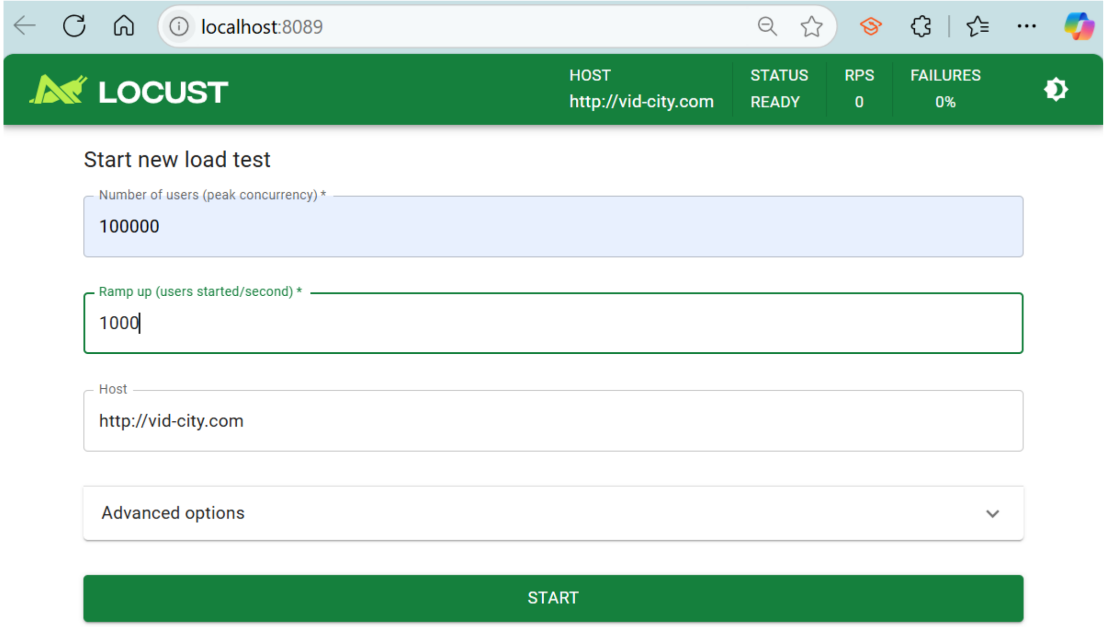


*Figure 6:* The client sends a HTTP request, which is routed through Ingress. Ingress handles HTTP routing, load balancing, and directs the request to the appropriate microservice. All microservices include a controller, model, routes, and are connected to a centralized logging microservice for live updates.

# Load Testing

locust is run on the local machine and a load test is conducted by specifiying 100,000 users and 1000 users per second.



HOST http://vid-city.com | STATUS READY | RPS 0 | FAILURES 0%

Start new load test

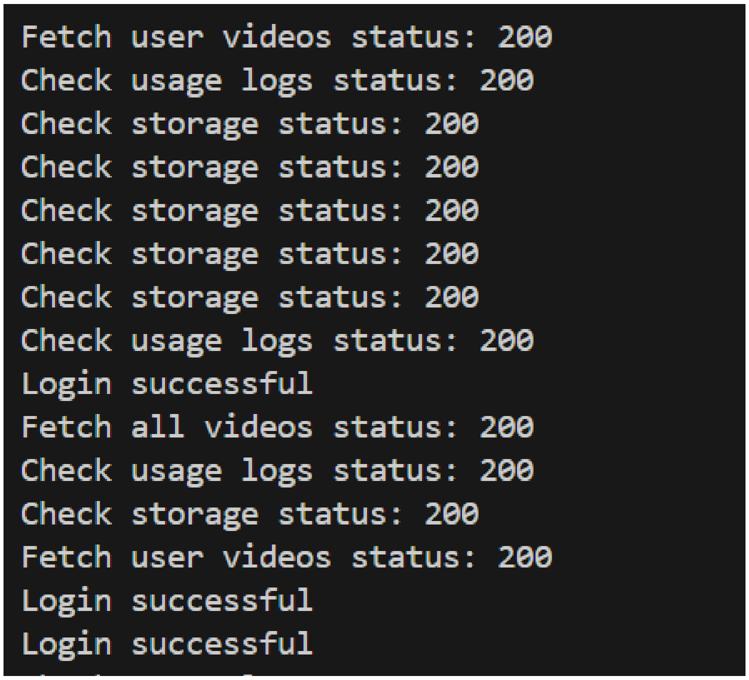
Number of users (peak concurrency) *: 100000

Ramp up (users started/second) *: 1000

Host: http://vid-city.com

*Figure 7:* Conducting a locust load test

The locust file returns logs of successful tests:



```
Fetch user videos status: 200
Check usage logs status: 200
Check storage status: 200
Check storage status: 200
Check storage status: 200
Check storage status: 200
Check storage status: 200
Check usage logs status: 200
Login successful
Fetch all videos status: 200
Check usage logs status: 200
Check storage status: 200
Fetch user videos status: 200
Login successful
Login successful
```

*Figure 8:* Logs returned by running the locust file

The results are as shown:

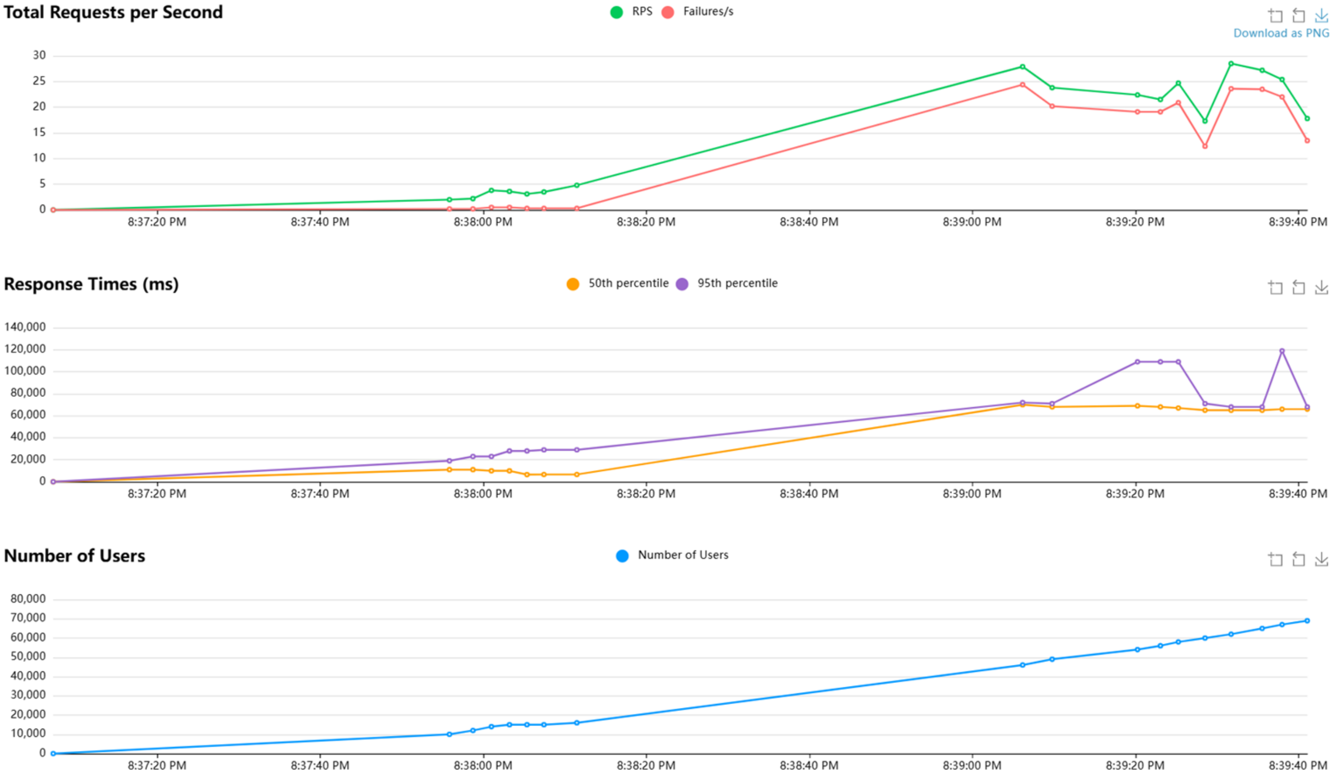


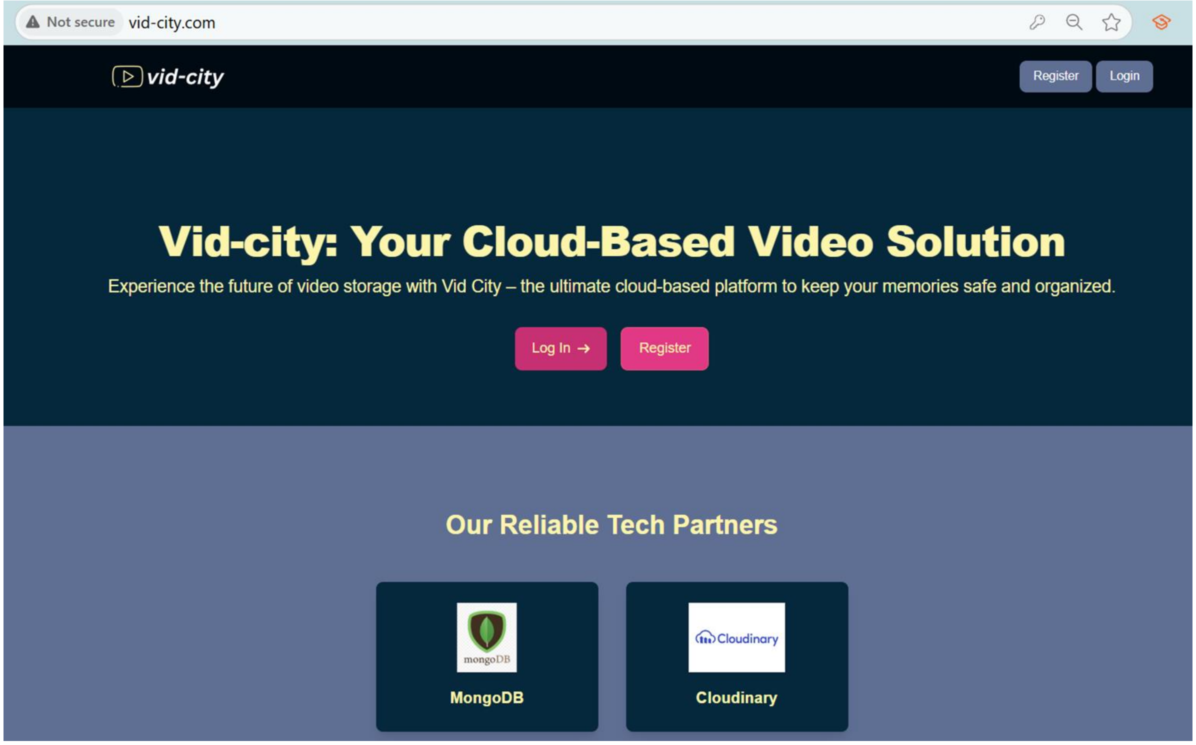
*Figure 9:* graphs represent the application’s performance after simulating user activity.

As the load increases (users and requests), the system's response times slightly degrade, and failures occur more frequently, indicating the system may have reached or exceeded its capacity.

## Tests conducted:

- **login:** Logs in the user before tasks are performed.
- **upload_video:** Simulates uploading a video file.
- **fetch_user_videos:** Fetches videos uploaded by a specific user.
- **fetch_all_videos:** Retrieves a list of all available videos.
- **register_user:** Simulates user registration.
- **check_user_storage:** Checks storage details for a specific user.
- **check_usage_logs:** Retrieves usage logs for a specific user.

# UI Screenshots

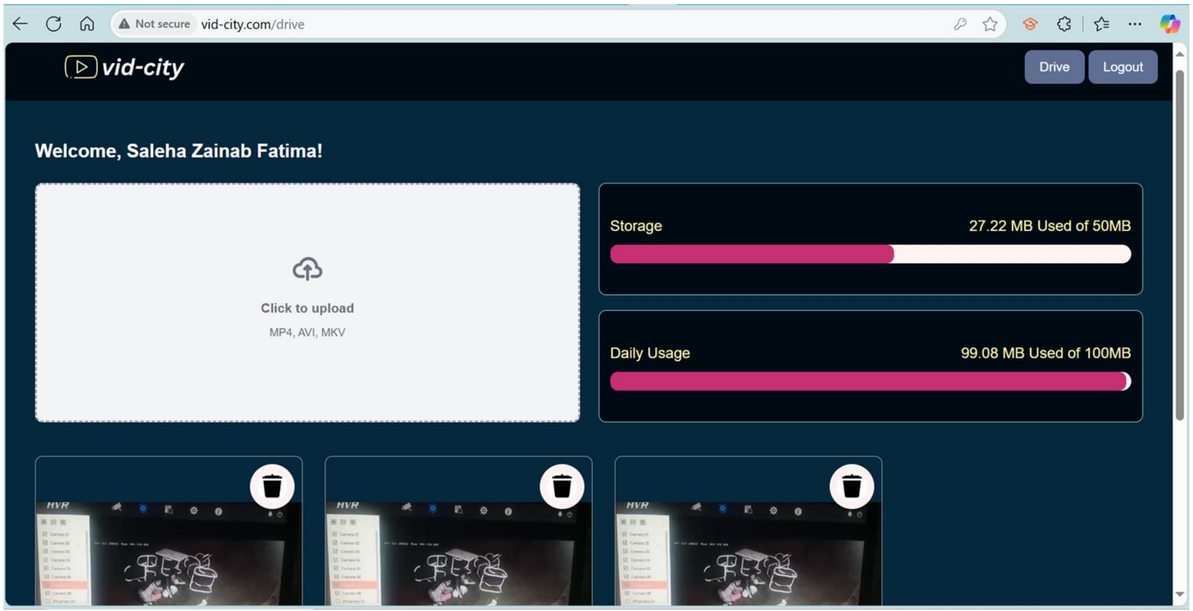


Vid-city: Your Cloud-Based Video Solution

Experience the future of video storage with Vid City – the ultimate cloud-based platform to keep your memories safe and organized.

Our Reliable Tech Partners

*Figure 10:* Homepage of vid-city



Welcome, Saleha Zainab Fatima!

Storage 27.22 MB Used of 50MB

Daily Usage 99.08 MB Used of 100MB

*Figure 11:* User’s landing page where they can see their storage,usage and uploaded videos

# References

*Ingress controllers*. (2024, April 23). Kubernetes.

https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/

*Kubernetes cheat sheet*. (n.d.). Coursera.

https://www.coursera.org/collections/kubernetes-cheat-sheet

*Skaffold.yaml*. (n.d.). Skaffold. https://skaffold.dev/docs/references/yaml/