SE-315 Cloud Computing

Semester Project Report

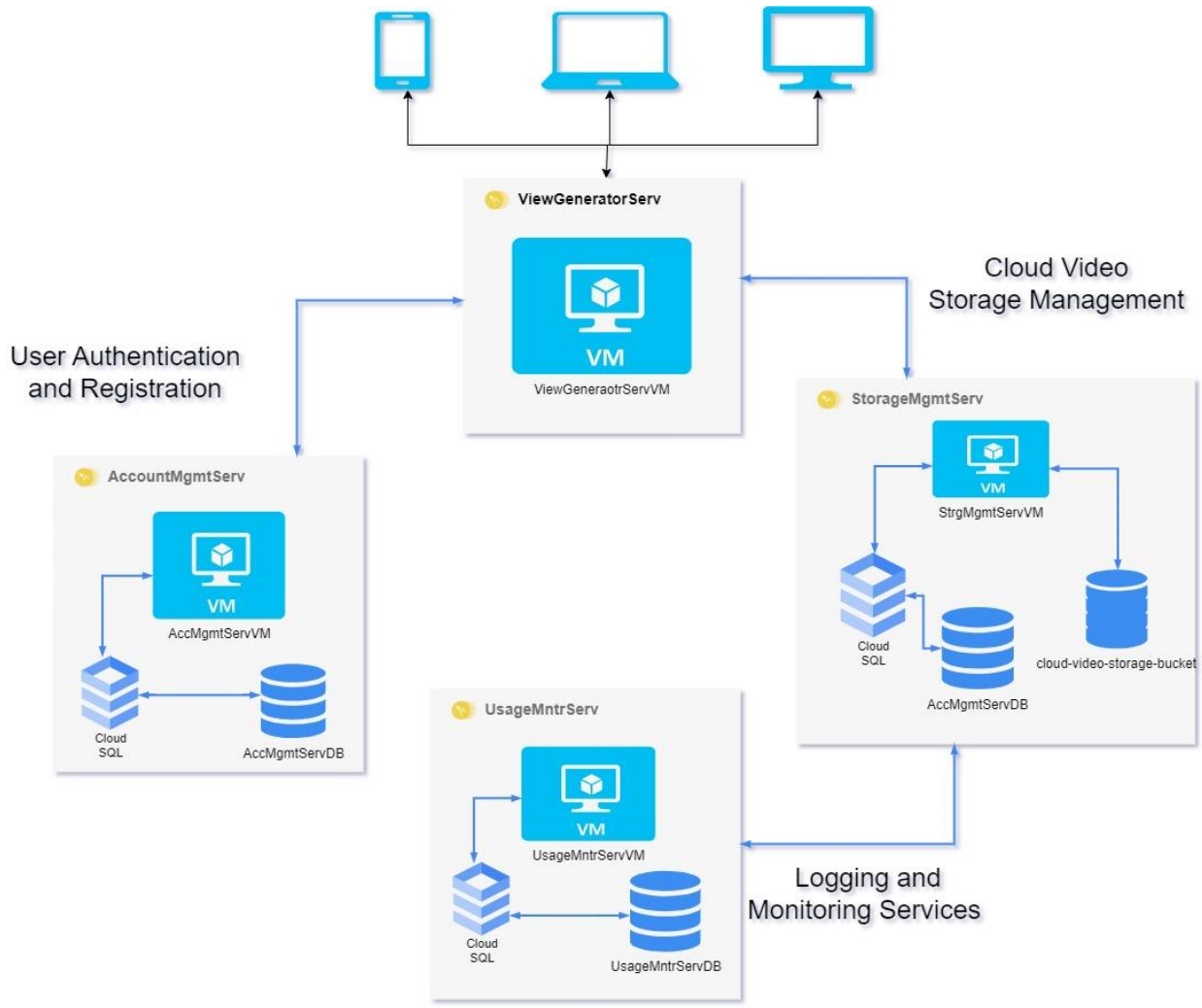
BESE 13 B

Submitted To: Dr. Qaiser Riaz

NAME	CMS ID
Asna Maqsood	426990
Jaweria Manahil	419118
Muhammad Owais Khan	404262
Sara Adnan Ghori	411228
Umar Farooq	406481

Contents

Architecture diagram:



API Specifications

The application comprises four key microservices, each offering distinct functionalities to support user operations, video storage, and monitoring. Below is a comprehensive breakdown of the API specifications:

1. Storage Management Service
This service manages video uploads, deletions, user storage, and bandwidth allocation.

- **File Upload**
  Uploads a video file to Google Cloud Storage. Ensures that user quotas for storage and bandwidth are not exceeded. If successful, updates the user's storage and bandwidth usage in the database and logs the operation.

Endpoint: POST /video
Request Body:

```
{
  "user_id": "string",
  "vname": "string",
  "bandwidth": "float",
  "video": "file"
}
```

Response:

- Success: Returns the video URL and status.
- Failure: Returns an error message indicating quota limits or other issues.

- **Delete File**
  Deletes a video from cloud storage and updates the user's storage record in the database. Logs the operation for monitoring purposes.
  Endpoint: DELETE /video
  Request Body:

```
{
  "user_id": "string",
  "vname": "string",
  "storage": "float"
}
```

Response:

- Success: Confirms the deletion with status OK.
- Failure: Returns an error message.

- **Fetch User Info**
  Retrieves the storage and bandwidth usage of a user by their ID.
  Endpoint: GET /userinfo/:user_id
  Response:

```
{
  "id": "string",
```

```
  "storage_used": "float",
  "bandwidth": "float"
}
```


- **Video Size Retrieval**
  Fetches the size of a specific video in MB by querying the cloud storage metadata.
  Endpoint: POST /videosize
  Request Body:


```
{
  "vname": "string"
}
```


  Response:


```
  "status": "OK",
  "vsize": "float"
```


- **Fetch All Videos**
  Returns a list of all videos uploaded by a user along with their signed URLs for access.
  Endpoint: GET /fetch-videos/:user_id
  Response:


```
{
  "videos": [
    {
      "name": "string",
      "url": "string"
    }
  ]
}
```


2. Usage Monitoring Service
This service logs user actions (e.g., upload, delete) for monitoring and analytics purposes.

- **Log User Actions**
  Records details of user actions, including the operation performed, the video involved, and a timestamp.

Endpoint: POST /logit
Request Body:


```
{
  "user_id": "string",
  "video": "string",
  "info": "string"
}
```


Response:

- Success: Acknowledges the log entry.
- Failure: Returns an error message.

3. User Account Management Service
This service handles user authentication, registration, and profile retrieval.

- **Signup**
  Registers a new user in the system. If successful, returns the user's profile details.
  Endpoint: POST /signup
  Request Body:


```
{
  "username": "string",
  "name": "string",
  "email": "string",
  "password": "string",
  "contact": "string"
}
```


Response:


```
{
  "status": "OK",
  "info": {
    "id": "string",
    "username": "string",
```

```
    "name": "string",
    "email": "string",
    "contact": "string"
  }
}
```


- **Login**
  Authenticates user credentials and returns profile details on success.
  Endpoint: POST /login
  Request Body:


```
{
  "username": "string",
  "password": "string"
}
```


  Response:


```
{
  "status": "OK",
  "info": {
    "id": "string",
    "username": "string",
    "name": "string",
    "email": "string",
    "contact": "string"
  }
}
```


- **Fetch User Details**
  Retrieves a user's basic profile details by their ID.
  Endpoint: GET /fetch-user/:user_id
  Response:


```
{
```

```
  "status": "OK",
  "info": {
    "id": "string",
    "username": "string",
    "name": "string",
    "email": "string",
    "contact": "string"
  }
}
```


- **Fetch User ID by Username**
  Retrieves the user ID corresponding to a given username.
  Endpoint: GET /id/:username
  Response:


```
{
  "status": "OK",
  "id": "string"
}
```


4. View Generator Service
This service delivers the static front-end pages, enabling users to interact with the system.

- **Static Pages**
  Serves static files for key user operations.
  Endpoints:
    - / (Homepage)
    - /login (Login page)
    - /signup (Signup page)
    - /home?id=:user_id (User dashboard)

Response:
HTML content of the respective page.

Key Features and Design Decisions

1. **Scalability and Modularity**:
   The system follows a microservices architecture, separating concerns into distinct services (e.g., storage management, user account handling). This modularity facilitates scaling individual components independently based on load.

2. **Google Cloud Integration**:
   Leveraged Google Cloud Storage for efficient and scalable video storage. Functions for uploading, deleting, and streaming are optimized with features like resumable uploads and signed URLs.

3. **User Storage and Bandwidth Management**:
   Implemented stringent checks to ensure users stay within their allocated storage (50MB) and bandwidth limits. Dynamic updates to user records in MySQL ensure accuracy.

4. **Frontend Integration**:
   Built a user-friendly frontend in the View Generator Service to enable non-technical users to interact seamlessly with backend features. The interface includes pages for login, signup, and video management.

5. **Logging and Monitoring**:
   A dedicated Usage Monitoring Service logs user actions, providing data for usage analysis and debugging.

6. **Error Handling and Validation**:
   Comprehensive error handling for API endpoints ensures robust functionality. Each service validates inputs to prevent invalid data from disrupting the workflow.

7. **Technology Stack**:
   - **Backend**: Node.js, Express.js
   - **Database**: MySQL
   - **Frontend**: Static HTML served via Node.js
   - **Cloud**: Google Cloud Storage for file storage

8. **Security Considerations**:
   - Restricted user actions with authentication and unique user IDs.
   - Signed URLs ensure secure video access with time-bound constraints.

Load Testing:

Locust Test Report

During: 12/29/2024, 5:52:34 PM - 12/29/2024, 5:57:31 PM (4 minutes and 57 seconds)
Target Host: http://34.47.196.178:3000
Script: locustfile.py

Request Statistics

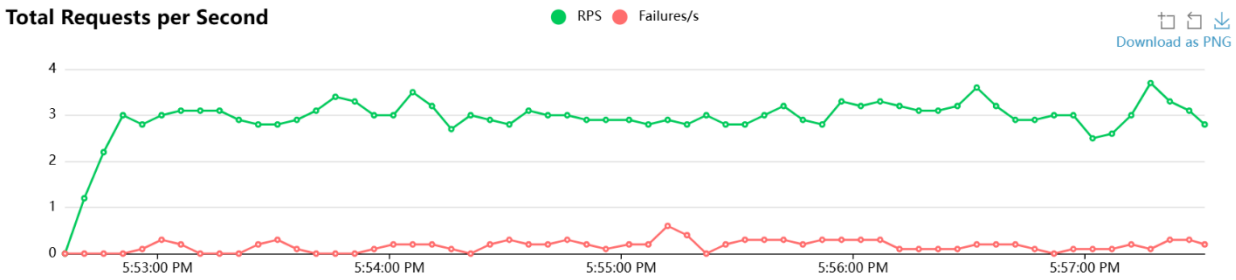
Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/signup	889	50	307.43	8	760	6282.59	2.99	0.17
	Aggregated	889	50	307.43	8	760	6282.59	2.99	0.17

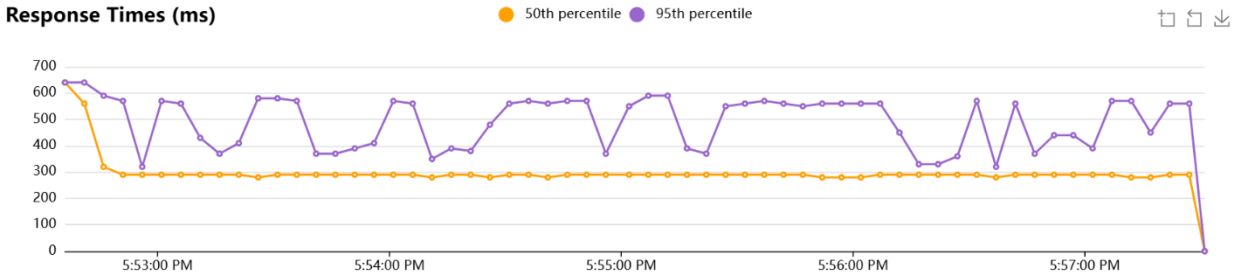
Response Time Statistics

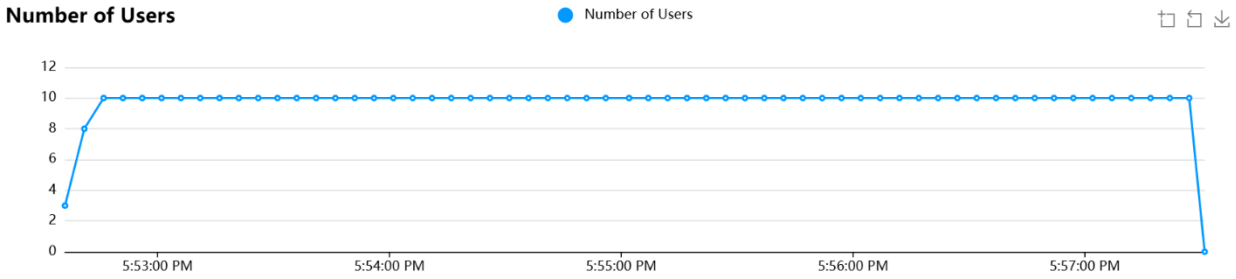
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/signup	290	290	290	300	380	560	590	760
	Aggregated	290	290	290	300	380	560	590	760

Failures Statistics

# Failures	Method	Name	Message
50	GET	/signup	RemoteDisconnected('Remote end closed connection without response')







Final ratio

Ratio Per Class

- 100.0% VideoStorageUser
  - 100.0% signupAndUploadVideo

Total Ratio

- 100.0% VideoStorageUser
  - 100.0% signupAndUploadVideo

The working URL:

http://34.47.196.178:3000

GITHUB LINK:

https://github.com/Umar-Farooq-2112/Cloud-Video-App.git