Text

Description automatically generated

**AIN SHAMS UNIVERSITY**

**FACULTY OF ENGINEERING**

**AIN SHAMS UNIVERSITY**

**FACULTY OF ENGINEERING**

**CREDIT HOURS ENG. PROGRAM**

**Computer engineering and software systems**

A picture containing logo

Description automatically generated

**Computer Networks: CSE 351**

**Project Proposal**

**Submitted to:**

**Prof. Ayman M. Bahaa-Eldin**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **ID** |
| **Mohamed Mostafa ElMaghraby** | **20p7732** |
| **Anass Zikry** | **20P6382** |
| **Tarek Khaled Ezzat** | **20P1087** |
| **Mohamed Hesham El Said** | **20P7579** |

Contents

[Project Plan 3](#_Toc164348616)

[Phase 1: Planning and Design (week 1) 3](#_Toc164348617)

[Phase 2: Development (week 2) 3](#_Toc164348618)

[1- Setting up Cloud Infrastructure 3](#_Toc164348619)

[2 - Implementing UI 3](#_Toc164348620)

[3 - Implementing Image Processing Layer 3](#_Toc164348621)

[Phase 3: Distribution infrastructure (week 3) 3](#_Toc164348622)

[Phase 4: Testing and Deployment (week 4) 3](#_Toc164348623)

[Project Scope: 3](#_Toc164348624)

[Objectives: 4](#_Toc164348625)

[Requirements: 4](#_Toc164348626)

[Functional Requirements: 4](#_Toc164348627)

[Non-Functional Requirements: 4](#_Toc164348628)

[System Architecture 5](#_Toc164348629)

[1- User Interface (UI): 5](#_Toc164348630)

[2- Application Layer: 5](#_Toc164348631)

[3- Image Processing Layer: 5](#_Toc164348632)

[4- Cloud Infrastructure: 5](#_Toc164348633)

[5- Fault Tolerance and Monitoring: 5](#_Toc164348634)

[User Stories 7](#_Toc164348635)

[UML Diagrams: 8](#_Toc164348636)

[Sequence Diagram: 8](#_Toc164348637)

[Class Diagram: 8](#_Toc164348638)

[Component Diagram: 8](#_Toc164348639)

[Network Diagram: 8](#_Toc164348640)

# Project Plan

## Phase 1: Planning and Design (week 1)

1. Discuss project scope, objectives, and requirements.
2. Assign team roles and responsibilities.
3. Review and finalize functional requirements.
4. Identify any additional non-functional requirements.
5. Design the high-level system architecture.
6. Determine the technology stack and tools to be used.
7. Document user stories and acceptance criteria.
8. Create diagrams (e.g., UML diagrams) to illustrate system interactions.

## Phase 2: Development (week 2)

### Setting up Cloud Infrastructure

* Provision cloud resources (VM instances, object storage).
* Set up monitoring and logging tools.

### 2 - Implementing UI

* Set up a framework for the web-based UI.
* Implement functionality for uploading images and selecting processing operations.
* Integrate UI with backend APIs for task submission and monitoring.

### 3 - Implementing Image Processing Layer

* Integrate OpenCV library for image processing algorithms.
* Implement parallel processing using OpenCL or MPI.
* Test and optimize image processing algorithms for performance.

## Phase 3: Distribution infrastructure (week 3)

* Implement task scheduler for distributing processing tasks.
* Develop load balancer for evenly distributing workload across VMs and making the system scalable.
* Implement error handling and retry mechanisms for fault tolerance.

## Phase 4: Testing and Deployment (week 4)

* Conduct thorough testing to ensure the system works as expected.
* Deploy the system to the cloud and ensure it is operational.
* Prepare end-user manual. (Readme file on GitHub)

# Project Scope:

The scope of the project encompasses the design, implementation, and testing of a distributed image processing system using cloud computing technologies. The system will allow users to upload images for various processing operations, such as filtering, edge detection, and color manipulation. It will distribute processing tasks across multiple virtual machines in the cloud to achieve scalability and improve performance. The system will also be designed to be fault-tolerant, capable of detecting and recovering from failures to ensure uninterrupted processing.

# Objectives:

* Develop a distributed image processing system that leverages cloud computing technologies for scalability and performance.
* Implement various image processing algorithms, including filtering, edge detection, and color manipulation, to provide a diverse range of processing options.
* Enable users to upload images, select processing operations, monitor task progress, and download processed images.
* Ensure the system is scalable, allowing for the addition of more virtual machines to handle increased workload.
* Design fault tolerance mechanisms to detect and recover from failures, ensuring uninterrupted processing even in the event of VM failures.

# Requirements:

## Functional Requirements:

* Users should be able to upload images for processing.
* The system should support various image processing operations, including filtering, edge detection, and color manipulation.
* Users should be able to select the desired processing operation for each uploaded image.
* The system should distribute image processing tasks across multiple virtual machines in the cloud.
* Progress monitoring functionality should be provided to allow users to track the status of their processing tasks.
* Processed images should be available for download once processing is complete.

## Non-Functional Requirements:

* The system should process image tasks within a reasonable timeframe.
* The system should be scalable.
* The system should be capable of recovering from failures (e.g., VM crashes, network outages) automatically without human intervention.
* Code should be well-documented and adhere to coding best practices to ease maintenance and troubleshooting efforts.
* The user interface should be intuitive and easy to navigate, with clear instructions provided for each functionality.

# System Architecture

## User Interface (UI):

* Flask or Django (Python web frameworks): For building a web-based UI where users can upload images, select processing operations, monitor task progress, and download processed images.

## Application Layer:

* Python: As the primary language for implementing the application logic.
* Task Scheduler: Orchestrates the distribution of image processing tasks among virtual machines (VMs) in the cloud.
* Load Balancer: Distributes incoming user requests across multiple VMs to ensure even workload distribution.

## Image Processing Layer:

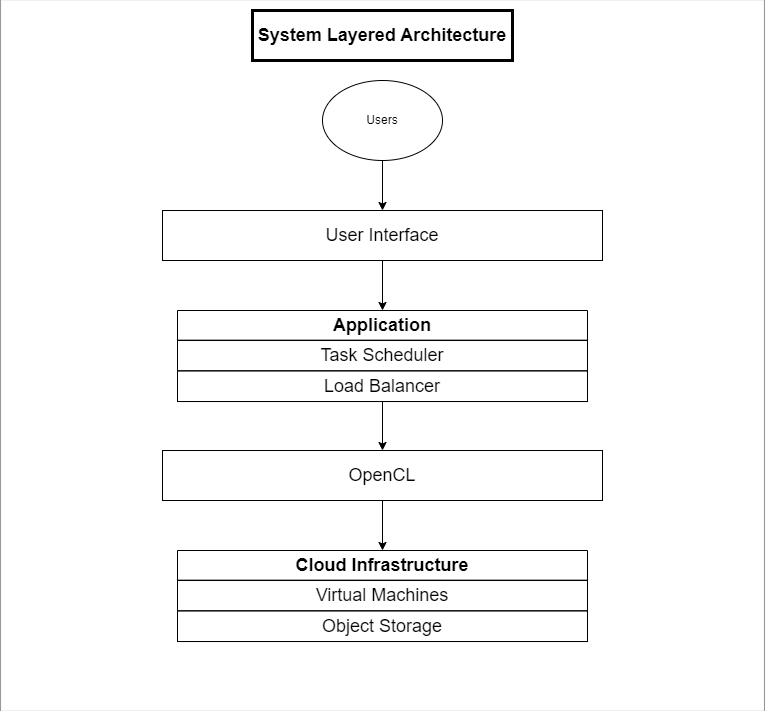
* OpenCV: A popular library for image processing in Python, providing functions for various operations such as filtering, edge detection, and color manipulation.
* OpenCL or MPI: For parallel processing of image data across multiple VMs. OpenCL is suitable for heterogeneous computing environments, while MPI is commonly used for distributed memory systems.

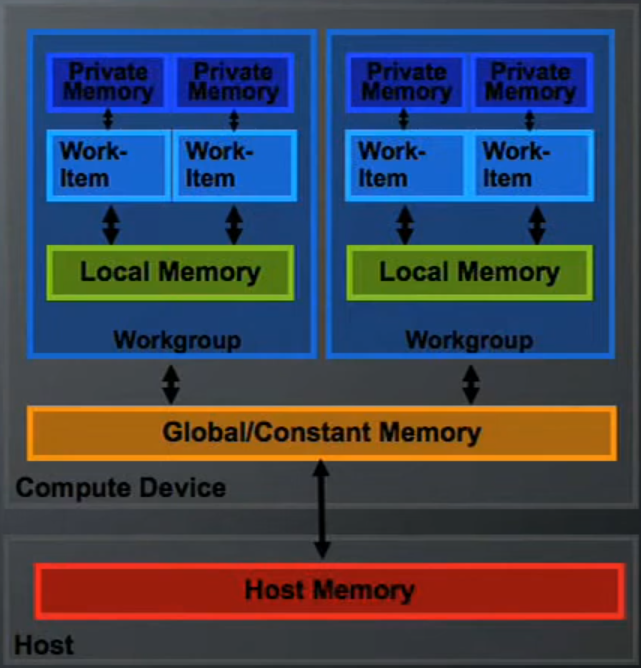
## Cloud Infrastructure:

* Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure: For provisioning virtual machines and managing the cloud infrastructure.
* Virtual Machines (VMs): Multiple VM instances will be created to distribute image processing tasks. VM instances should be scalable based on workload demands.
* Object Storage: Store uploaded images and processed images. AWS S3, Google Cloud Storage, or Azure Blob Storage can be used for this purpose.

## Fault Tolerance and Monitoring:

* Kubernetes: For container orchestration and managing VMs as containers. Kubernetes provides features like automatic scaling and self-healing, improving fault tolerance.
* Prometheus and Grafana: For monitoring the health and performance of VMs, containers, and application components.





# User Stories

* As a user, I want to upload an image to the system for processing.
* As a user, I want to select the type of image processing operation to be performed.
* As a user, I want to download the processed image once the operation is complete.
* As a user, I want to monitor the progress of the image processing task.
* As a team, we need to implement an efficient architecture for distributing image processing tasks across multiple virtual machines.
* As a team, we need to ensure the system’s scalability and fault tolerance.
* As a team, we need to consider image size, processing complexity, and available resources when splitting tasks into smaller units that can be distributed among available resources.
* As a team, we need to implement functionalities to monitor and track progress of tasks and inform the user of any change.
* As a team, we need to design and implement an easy to use and simple user interface to facilitate the user’s interaction with the app.
* As a team, we need to handle task failures through retrying, error logging, and automated recovery of the system.

# UML Diagrams:

## Sequence Diagram:

## Class Diagram:

## Component Diagram:

## Network Diagram: