National College of Ireland

Project Submission Sheet

Student Name:	Jeswanth Appasani		
Student ID:	x23284269		
Programme:	Cloud Computing	**Year:**	2024
Module:	Cloud Dev-opps		
Lecturer:	Vikas Sahni		
Submission Due Date:	9 December 2024		
Project Title:	Dynamic Cloud-Based Application with CI/CD Integration		
Word Count:	2543		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Jeswanth Appasani

Date: 09/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

AI Acknowledgement Supplement

[Cloud Dev-opps]

[Dynamic Cloud-Based Application with CI/CD Integration]

Your Name/Student Number	Course	Date
Jeswanth Appasani/x23284269	Cloud Computing	9 December 2024

This section is a supplement to the main assignment, to be used if AI was used in any capacity in the creation of your assignment; if you have queries about how to do this, please contact your lecturer. For an example of how to fill these sections out, please click here.

AI Acknowledgment

This section acknowledges the AI tools that were utilized in the process of completing this assignment.

Tool Name	Brief Description	Link to tool

Description of AI Usage

This section provides a more detailed description of how the AI tools were used in the assignment. It includes information about the prompts given to the AI tool, the responses received, and how these responses were utilized or modified in the assignment. **One table should be used for each tool used**.

[Insert Tool Name]	
[Insert Description of use]	
[Insert Sample prompt]	[Insert Sample response]

Evidence of AI Usage

This section includes evidence of significant prompts and responses used or generated through the AI tool. It should provide a clear understanding of the extent to which the AI tool was used in the assignment. Evidence may be attached via screenshots or text.

Additional Evidence:

[Place evidence here]

Additional Evidence:
[Place evidence here]

Dynamic Cloud-Based Application with CI/CD Pipeline on AWS EC2

ABSTRACT **-**This report presents a cloud-based application that is both dynamic and user-friendly, leveraging the capabilities of AWS EC2 for deployment and integrating a CI/CD pipeline for efficiency. The application showcases strong functionality, allowing for Create, Read, Update, and Delete (CRUD) operations while prioritizing user input validation and effective data management through SQLite. The frontend is designed with HTML and CSS, focusing on simplicity and responsiveness to provide a clean and intuitive user experience across various devices. On the backend, Flask is utilized as the primary framework, offering reliability and flexibility for application logic and interactions. Gunicorn is used for hosting, which boosts performance and scalability on AWS EC2 instances. To ensure the application's health and performance, AWS Cloud-watch is implemented for real-time monitoring, giving insights into system behavior and resource usage.

This report explores the application's architectural design, detailing the strategic integration of CI/CD practices for efficient updates and deployments. It also discusses deployment strategies and static code analysis, emphasizing the steps taken to maintain code quality and manageability. The report wraps up with a reflective discussion on the challenges faced during development and deployment, along with the key lessons learned. These insights are intended to guide future projects and highlight the significance of merging technical accuracy with user-centered design.

1.Introduction

Cloud computing has changed the way applications are created, deployed, and managed, offering scalable, flexible, and reliable platforms that meet contemporary demands. This project takes advantage of these innovations by developing a lightweight, cloud-based application designed to solve real-world problems. Hosted on AWS EC2, **[1]** the application showcases efficiency and functionality, featuring essential CRUD operations, secure data management, and smooth automated updates through an integrated CI/CD pipeline. The main objective of this project is to build a scalable, resilient application architecture that fully utilizes cloud-native tools like AWS EC2 for deployment and AWS CloudWatch for monitoring and performance enhancement. The design prioritizes security, ensuring strong data handling and compliance with best practices in application development. By incorporating these elements, the project seeks to provide a dependable and efficient solution while highlighting the advantages of cloud-based strategies. This report offers a detailed examination of the technical implementation, addressing every stage of the development life cycle—from establishing the cloud environment and deploying the application to integrating CI/CD pipelines for automated updates. Particular focus is placed on analyzing how these strategies not only streamline the development process but also greatly improve the application's operational efficiency. Beyond the technical aspects, this project exemplifies the transformative potential of cloud computing, illustrating its capacity to simplify complex workflows while ensuring scalability, security, and reliability. The insights and reflections presented in this report aim to deliver a thorough understanding of the challenges and opportunities associated with cloud-based application development. **[2]**

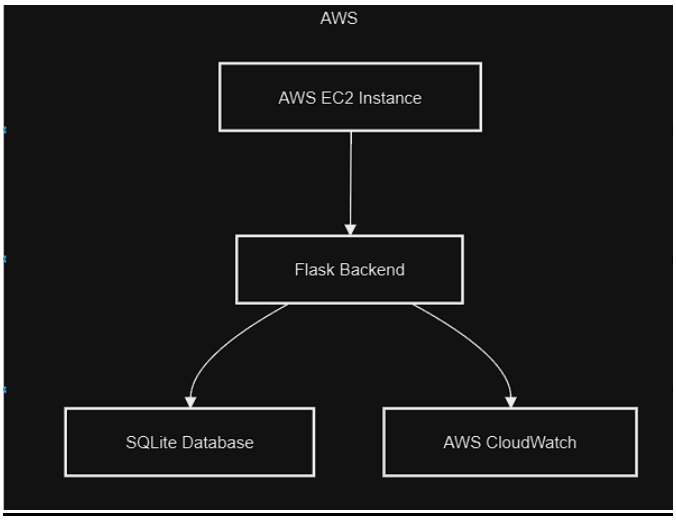
2. *Architectural Design*

2.1 Overview

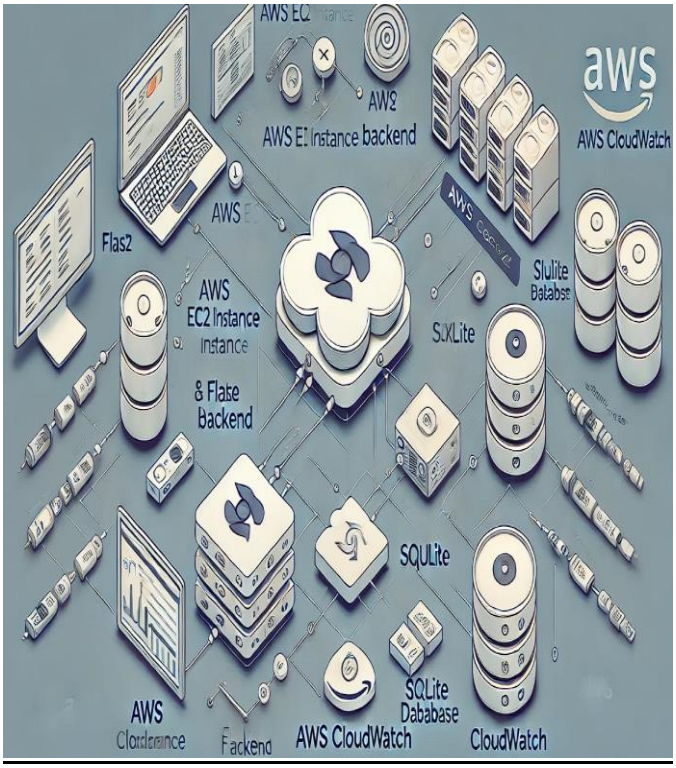
The architecture of this project has been thoughtfully crafted to use AWS EC2 as the hosting platform, ensuring a scalable and dependable environment for deployment. Gunicorn serves as the application server, effectively managing backend requests and facilitating smooth interactions between users and the server. This combination creates a strong infrastructure capable of supporting the application's functionality across various workloads. A significant element of the design is the clear

division of components. The frontend, developed with HTML and CSS, emphasizes simplicity and responsiveness, providing an intuitive user experience. The backend, driven by Flask, handles the application logic and enables seamless communication between the user interface and the database. For data storage and management, SQLite has been incorporated, delivering lightweight yet efficient database capabilities. This modular strategy enhances scalability, making it simpler to maintain and expand the application as necessary. To guarantee reliability and performance, AWS CloudWatch has been set up to monitor the application's health and performance metrics. CloudWatch offers real-time insights into resource usage, application performance, and potential issues. By logging errors and providing actionable alerts, it supports proactive troubleshooting and ensures that the application remains accessible and efficient. This architecture strikes a balance between functionality, scalability, and maintainability while utilizing cloud-native tools to enhance performance and ensure a positive user experience. It lays the groundwork for future improvements, reflecting a careful approach to modern application development.

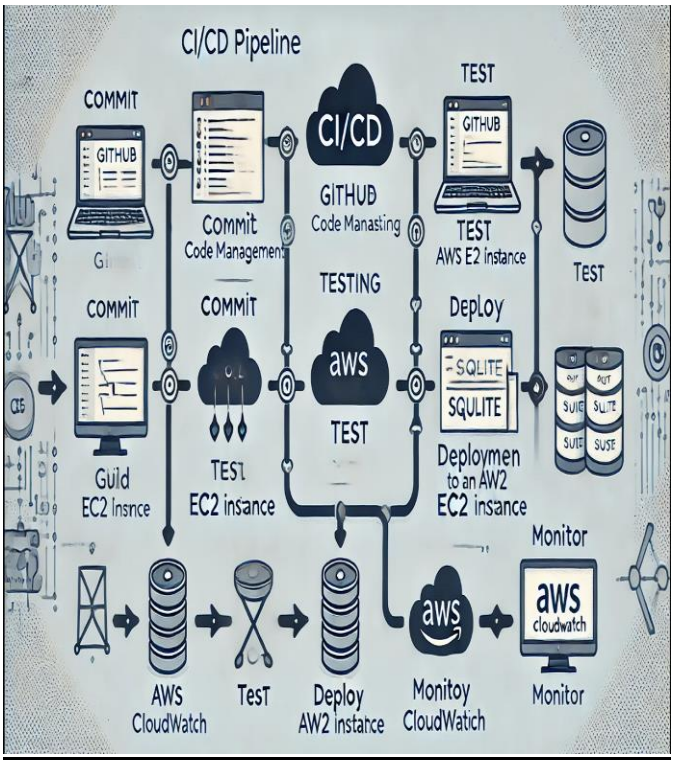
2.2 Architecture Diagram



[Architecture Diagram: Highlighting components like EC2 instance, Flask backend, SQLite database, and monitoring with AWS CloudWatch.**]**



Architecture Diagram: Showcasing components like EC2 instance, Flask backend, SQLite database, and AWS CloudWatch monitoring.



CI/CD PIPELINE

2.3 Components

Frontend Design :

The user interface has been carefully crafted using HTML and CSS, focusing on both visual appeal and user-friendliness. Its responsive design guarantees a smooth experience on various devices and screen sizes, making the application accessible to a broader audience. To ensure reliability, input validation has been put in place, effectively stopping users from submitting incomplete or incorrect data. This strategy not only enhances functionality but also helps preserve data integrity.

Backend Design

The backend utilizes Flask, a lightweight and flexible web framework. Flask acts as the core of the application, managing API requests and efficiently handling Create, Read, Update, and Delete (CRUD) operations. It also creates a seamless connection with the SQLite database, ensuring that the data flow between the user interface and the database is consistent and error-free. This backend setup is designed to be both strong and adaptable, ready to scale with future needs.

Database SQLite

has been selected as the database solution due to its lightweight yet dependable features, making it perfect for this application. It allows for efficient data handling during runtime, enabling quick and responsive operations. Its simplicity and ease of integration with Flask ensure that the application's data management processes are streamlined, contributing to overall system reliability.

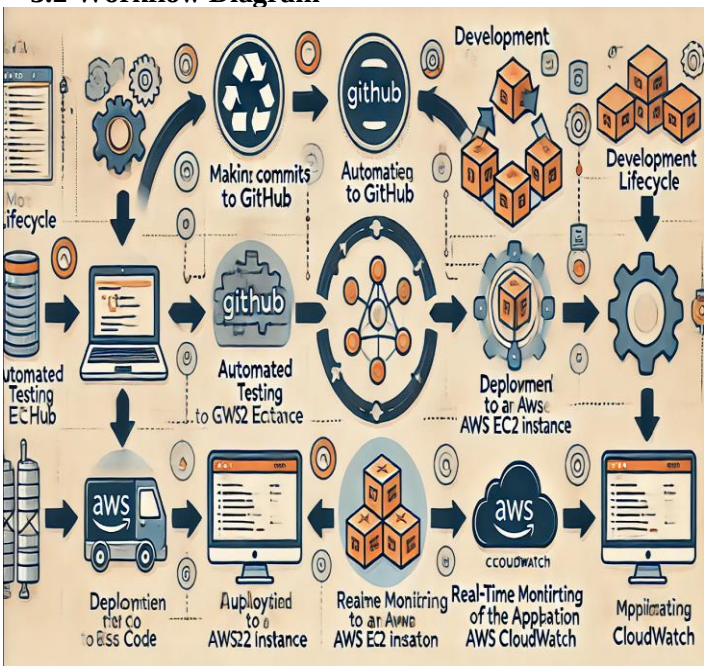
Deployment and Hosting The application is hosted on an AWS EC2 instance, a powerful and scalable platform that ensures reliable hosting. Gunicorn is used as the WSGI **HTTP** server to manage server-side operations. This combination of **AWS EC2** and **Gunicorn** guarantees high availability, strong performance, and the ability to handle multiple requests efficiently. This deployment approach not only enhances performance but also sets the stage for future improvements and scalability.

3. *Continuous Integration and Deployment (CI/CD)*

3.1 Pipeline Overview

The CI/CD pipeline is a fundamental part of this project, automating the transition from development to deployment and ensuring a smooth workflow. This automation removes the need for manual interventions, greatly minimizing the risk of human error and speeding up the deployment process. Version control is effectively handled through GitHub, which serves as a central platform for tracking code changes, collaborating with team members, and keeping a clear and organized record of the application's development history.**[3]** This guarantees that every update is traceable and can be assessed for quality. AWS tools are incorporated into the pipeline to manage the deployment process with an emphasis on security and efficiency. These tools ensure that updates are deployed safely to the live environment, preserving the application's integrity and performance even as changes are made. By merging GitHub's strong version control with AWS's robust deployment capabilities, the CI/CD pipeline guarantees that the application stays current, dependable, and secure, while also allowing developers to implement enhancements more swiftly and effectively. This streamlined approach underscores the significance of automation in contemporary software development.

3.2 Workflow Diagram



[Workflow Diagram: Showing stages like GitHub commits, testing, deployment to AWS EC2, and monitoring using CloudWatch.**]**

3.3 Pipeline Implementation

Source Code Management [5]

GitHub maintains version control and tracks changes.

Build and Test

Local tests validate code integrity before deployment.**[4]**

Deployment

Application files are securely transferred to the AWS EC2 instance via SSH. Gunicorn is used to host the application. **[3].**

Monitoring

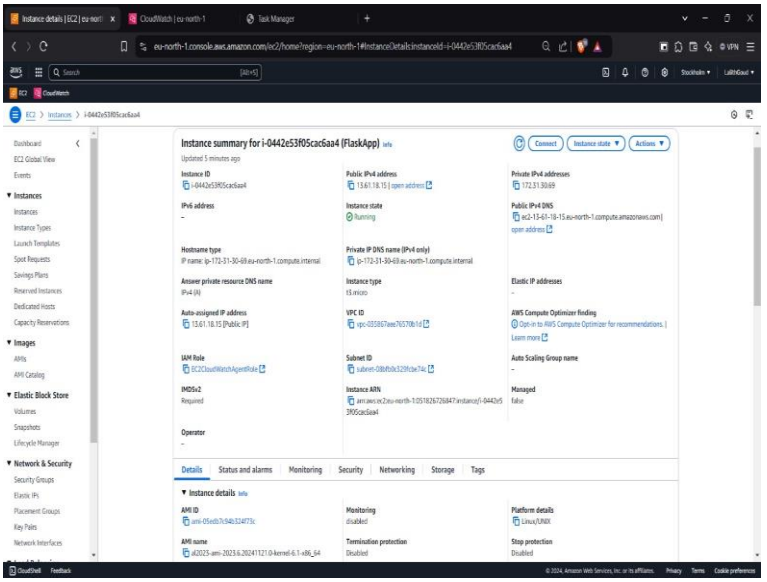
AWS CloudWatch logs provide insights into application performance, aiding in error detection and resolution. **[1]**

4. ***Deployment on AWS EC2***

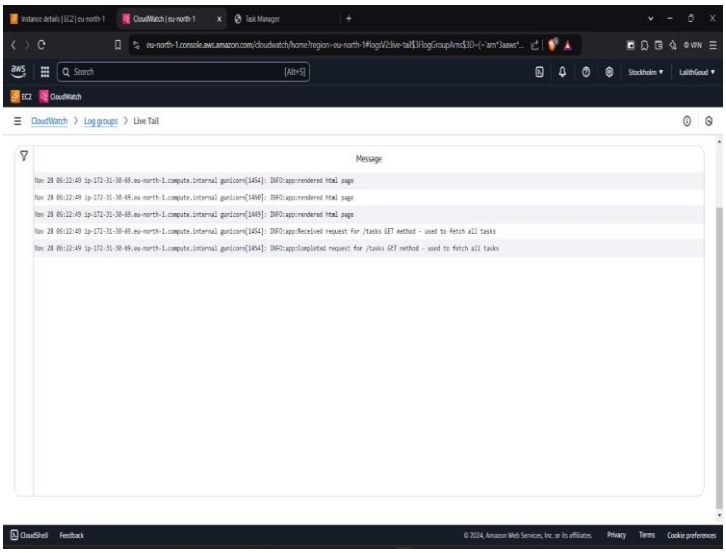
4.1 Deployment Steps:

Configuring the EC2 Instance **[1]** The deployment process starts with setting up an AWS EC2 instance to host the application. An Ubuntu-based t2.micro instance is chosen for its cost-effectiveness and performance within the AWS free tier. After launching the instance, the environment is prepared by installing Python, Flask, and Gunicorn**[3]**. These tools are essential for the application's backend, allowing it to handle requests and provide responses efficiently. Deploying the Application Once the instance is set up, the application files are securely transferred to the EC2 instance using Secure Copy Protocol (SCP). This step ensures that the codebase is moved to the server safely, maintaining data integrity. After the files are transferred, the application is deployed using Gunicorn, a powerful WSGI HTTP server that guarantees efficient request handling and optimal performance. This configuration enables the application to manage multiple user requests simultaneously without issues. Monitoring and Logs To ensure the application's health and quickly resolve any problems, AWS CloudWatch is set up for real-time monitoring. CloudWatch offers valuable insights into the application's performance, including resource usage and error logs.**[1]**These metrics are crucial for spotting potential bottlenecks or failures, allowing for proactive troubleshooting and ensuring the application operates smoothly at all times. This deployment process highlights a careful approach to establishing a cloud-based application, integrating thorough configuration, secure file transfer, and ongoing monitoring to provide a reliable and high-performing solution.

4.2 Images for Deployment



[EC2 Instance Configuration]



[Placeholder: CloudWatch Logs]

5. ***Static Code and Security Vulnerabilities Analysis***

5.1 Methodology

Static Code Analysis To ensure that the application meets established coding standards and best practices, Pylint is employed for static code analysis. Pylint examines the codebase for potential issues such as syntax errors, inconsistent naming conventions, or deviations from standard coding guidelines. By identifying these issues early in the development process, Pylint helps maintain clean, readable, and maintainable code. This proactive approach reduces the risk of technical debt and ensures that the application remains scalable and easy to update in the future.**[2]**Security

Vulnerability Assessment Security is a vital aspect of any application, and this project prioritizes it. To identify and address potential vulnerabilities, tools like OWASP ZAP are utilized to conduct security scans. These scans specifically target common threats such as Cross-Site Scripting (XSS) and SQL Injection **[4]** which can jeopardize user data and application integrity. OWASP ZAP aids in uncovering these vulnerabilities, allowing for the implementation of necessary safeguards to protect against malicious attacks. By integrating static code analysis with thorough security testing, the project not only guarantees high-quality code but also establishes a secure foundation, fostering confidence in both developers and end users. This dual emphasis on quality and security underscores the commitment to delivering a reliable and secure application.

5.2 Findings and Fixes

Findings: Exposed credentials and potential SQL injection vulnerabilities.

Fixes: Implemented environment variables and enhanced validation mechanisms. **[5].**

6. *Code Implementation Highlights*

6.1 Code Snippets

```
// Add a new task
addTaskForm.addEventListener('submit', event => {
  event.preventDefault();
  const taskName = document.getElementById('taskName').value;
  const taskDescription = document.getElementById('taskDescription').value;
  fetch('/tasks', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name: taskName, description: taskDescription })
  }).then(() => {
    fetchTasks();
    addTaskForm.reset();
  });
});

// Delete a task
function deleteTask(taskId) {
  fetch(`/tasks/${taskId}`, { method: 'DELETE' })
    .then(() => fetchTasks());
}
```

[Add Task and Delete Task Code Snippet (JS)]

```
# Update an existing task
@app.route('/tasks/<int:task_id>', methods=['PUT'])
def update_task(task_id):
    logger.info('Received request for /tasks PUT method - used to update a task')
    data = request.json
    if not data or 'name' not in data:
        return jsonify({'error': 'Task name is required'}), 400
    result = query_db('UPDATE tasks SET name = ?, description = ? WHERE id = ?',
                      (data['name'], data.get('description', ''), task_id))
    if result:
        logger.info('Completed request for /tasks PUT method - used to update a task')
        return jsonify({'message': 'Task updated'})
    return jsonify({'error': 'Task not found'}), 404

# Delete a task
@app.route('/tasks/<int:task_id>', methods=['DELETE'])
def delete_task(task_id):
    logger.info('Received request for /tasks DELETE method - used to delete a task')
    query_db('DELETE FROM tasks WHERE id = ?', (task_id,))
    logger.info('Completed request for /tasks DELETE method - used to delete a task')
    return jsonify({'message': f'Task with id {task_id} deleted'})
```

[Update/Delete Task Code Snippet (JS)]

```
// Fetch tasks and display them
function fetchTasks() {
  fetch('/tasks')
    .then(response => response.json())
    .then(tasks => {
      taskTableBody.innerHTML = '';
      tasks.forEach(task => {
        const row = document.createElement('tr');
        row.innerHTML = `
          <td>${task.id}</td>
          <td>${task.name}</td>
          <td>${task.description || ''}</td>
          <td>
            <button class="btn btn-warning btn-sm" onclick="editTask(${task.id}, '${task.name}', '${task.description || ''}')">
            <button class="btn btn-danger btn-sm" onclick="deleteTask(${task.id})">Delete</button>
          </td>
        `;
        taskTableBody.appendChild(row);
      });
    });
}
```

[Fetch Task and Display Code Snippet (JS)]

```
# Add a new task
@app.route('/tasks', methods=['POST'])
def add_task():
    logger.info('Received request for /tasks POST method - used to add new task')
    data = request.json
    if not data or 'name' not in data:
        return jsonify({'error': 'Task name is required'}), 400
    query_db('INSERT INTO tasks (name, description) VALUES (?, ?)',
             (data['name'], data.get('description', '')))
    logger.info('Completed request for /tasks POST method - used to add new task')
    return jsonify({'message': 'Task added'}), 201

# View all tasks
@app.route('/tasks', methods=['GET'])
def get_tasks():
    logger.info('Received request for /tasks GET method - used to fetch all tasks')
    tasks = query_db('SELECT * FROM tasks')
    logger.info('Completed request for /tasks GET method - used to fetch all tasks')
    return jsonify([dict(task) for task in tasks])
```

[Viewing all tasks]

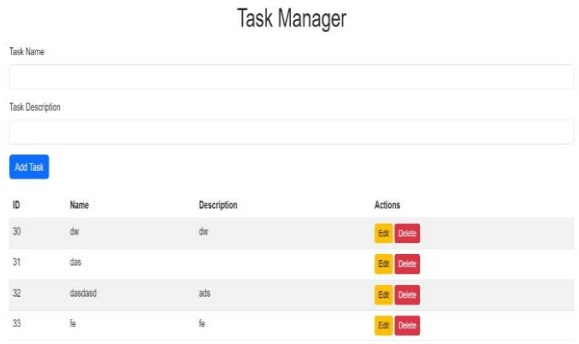
```javascript
// Edit a task
function editTask(taskId, name, description) {
  editingTaskId = taskId;
  editTaskName.value = name;
  editTaskDescription.value = description;
  editTaskForm.style.display = 'block';
}

// Update the task
editTaskForm.addEventListener('submit', event => {
  event.preventDefault();
  fetch(`/tasks/${editingTaskId}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      name: editTaskName.value,
      description: editTaskDescription.value
    })
  }).then(() => {
    fetchTasks();
    editTaskForm.reset();
    editTaskForm.style.display = 'none';
  });
});
```

[Edit task and Update task]

```javascript
// Fetch tasks and display them
function fetchTasks() {
  fetch('/tasks')
    .then(response => response.json())
    .then(tasks => {
      taskTableBody.innerHTML = '';
      tasks.forEach(task => {
        const row = document.createElement('tr');
        row.innerHTML = `
          <td>${task.id}</td>
          <td>${task.name}</td>
          <td>${task.description || ''}</td>
          <td>
            <button class="btn btn-warning btn-sm" onclick="editTask(${task.id}, '${task.name}', '${task.description || ''}')">
            <button class="btn btn-danger btn-sm" onclick="deleteTask(${task.id})">Delete</button>
          </td>
        `;
        taskTableBody.appendChild(row);
      });
    });
}
```

[Backend View – CRUD Operations Workflow]

6.2 Frontend and Backend Views:



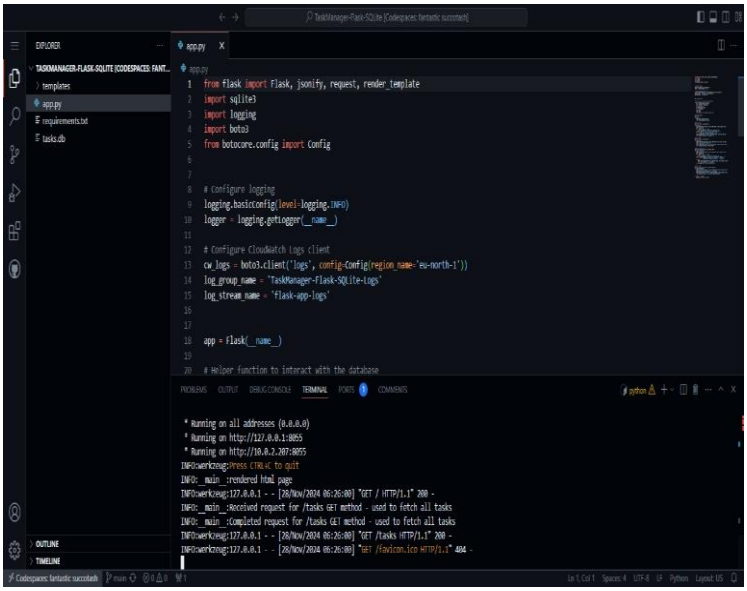
[Frontend View – Task Management Interface]

[DATABASE VIEW]

6.3 Logs

[Edit task frontend view]

[Local Terminal HTTPS Call Logs]



[CloudWatch Configuration for Flask Application]

7. Conclusion

This project demonstrates the successful implementation of a scalable and dynamic cloud-based application, leveraging the power of AWS EC2 for deployment and hosting. By integrating a CI/CD pipeline, the application achieves automated updates and seamless delivery, ensuring a reliable and efficient development workflow. Effective monitoring through AWS CloudWatch further enhances reliability by providing real-time insights into performance and system health, allowing for quick identification and resolution of potential issues.

Future Scope

The project opens up numerous opportunities for future improvement and expansion, including:

Expanding Database Support [4].

As the application grows, integrating support for more robust and scalable databases, such as PostgreSQL or Amazon RDS, would enable it to handle larger datasets and accommodate higher traffic demands. This enhancement would ensure continued performance and reliability as user requirements increase. **Integrating Advanced Logging and Monitoring [1].**

While AWS CloudWatch provides comprehensive monitoring, future iterations could incorporate advanced logging tools, such as ELK Stack (Elasticsearch, Logstash, Kibana) or AWS X-Ray, to gain deeper insights into application performance and user behavior.

8. *References :*

1. AWS, "EC2 User Guide," 2024.
2. Pylint Documentation, "Static Code Analysis with Pylint," 2024.
3. Gunicorn Documentation, "Gunicorn HTTP Server," 2024.
4. SQLite, "SQLite Documentation," 2024.
5. Flask Documentation, "Flask API Reference," 2024.