MAJOR PROJECT-1

SYNOPSIS REPORT

For

Ransomware Resilient Backup System

Submitted By

Harsh Sharma	Satvik Verma	Swayam Prakash Mohanty	Abhay Pratap Topwal
R2142211093	R2142211356	R2142211200	R2142211187

Cloud Software Operations Cluster
School Of Computer Science
UNIVERSITY OF PETROLEUM & ENERGY STUDIES,
DEHRADUN- 248007. Uttarakhand

Under the guidance of-
Dr. Narendra Dewangan

Table of Contents

1. Introduction

1.1 Purpose of the Project

The **Ransomware Resilient Backup System** is designed to provide robust backup and recovery solutions against ransomware attacks. It ensures the safety and integrity of critical data through advanced backup mechanisms, real-time monitoring, and secure storage methods. By using **MongoDB, Percona Backup for MongoDB, and Docker**, this system provides an effective defence strategy to recover from data breaches or encryptions caused by ransomware.

1.2 Target Beneficiary

The primary beneficiaries of this project include:

- Small to medium enterprises requiring secure data backup.
- IT administrators managing critical infrastructure.
- Educational institutions and individual developers seeking ransomware-proof solutions.

1.3 Project Scope

The project involves creating a distributed backup solution featuring:

- Real-time database backups using **Percona Backup for MongoDB**.
- User-friendly interface built using **React.js** and **Flask** for managing backups, restores, and timestamped recovery options.
- Secure and portable deployment through **Docker containers**.

1.4 References APA format

[1] Docker, I. (2020). Docker. *lınea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker*.
[2] Mongo, D. B. (2015). *Mongodb*.
[3] Petrov, P., Kuyumdzhiev, I., Dimitrov, G., & Kremenska, A. (2022). Relative Performance of Various Types of Repositories for MySQL Archive Backup and Restore Operations. *iJOE, 18*(13), 153.
[4] Ghimire, D. (2020). Comparative study on Python web frameworks: Flask and Django.
[5] Bui, M. (2020). Implementing cluster backup solution to build resilient cloud architecture.
[6] Cybersecurity, C. I. (2018). Framework for improving critical infrastructure cybersecurity. *URL: https://nvlpubs. nist. gov/nistpubs/CSWP/NIST. CSWP, 4162018*, 7.
[7] Gillies, A. (2011). Improving the quality of information security management systems with ISO27000. *The TQM Journal, 23*(4), 367-376.

2. PROJECT DESCRIPTION

2.1 Data structure

The system leverages the following data structures:

i. **JSON Objects**: To store backup metadata (e.g., timestamps, backup IDs).
ii. **MongoDB Collections**: For database entries and recovery.
iii. **Log Files**: To track backup and restore operations.

2.2 SWOT Analysis

Strengths	Weaknesses
1. Provides a reliable and resilient backup system against ransomware attacks. 2. User-friendly GUI for ease of use. 3. Open-source tools reduce project cost.	1. Relies heavily on Docker and MongoDB; may require compatibility checks. 2. Requires manual setup for Docker and MongoDB initially. 3. No integration with cloud platforms (e.g., AWS, Azure) in the current scope.
Opportunities	**Threats**
1. Can be extended to support cloud backups. 2. Integration with AI/ML for predictive analytics on backup integrity.	1. New ransomware methods may target backup systems. 2. Security vulnerabilities in underlying libraries (e.g., Docker, MongoDB).

2.3 Project Features

The features of this project are:

i. **Backup Data:** Allows users to create backups of a MongoDB database.
ii. **Restore Data:** Enables restoration from backup based on user-selected timestamps.
iii. **Enter Data:** Provides a user-friendly GUI to add or modify database entries.
iv. **Display Backups:** Displays a list of all available backup timestamps.
v. **Dockerized Environment:** Ensures portability and reproducibility.

2.4 Design and Implementation Constraints

i. **Hardware Constraints:** Requires Docker-compatible systems.
ii. **Software Constraints:** Dependent on MongoDB and Percona Backup for MongoDB.
iii. **Environmental Constraints:** Requires a secure and isolated environment for deployment.

2.5 Design diagram

Figure 1 represents diagram of the **Ransomware Resilient Backup System** illustrates the system's architecture and the flow of data between various components. The **User** interacts with the **React Frontend**, where they can perform actions, such as creating backups, restoring data, or entering new data. The **React Frontend** sends API requests to the **Flask API**, which processes these requests and manages communication with the **MongoDB Database** to store metadata like backup timestamps and data details. The **Flask API** also interfaces with the **File Storage** system, either locally or in the cloud, where the actual backup files are stored for retrieval during restoration. This flow ensures that the system is both user-friendly and resilient, offering seamless backup and restore operations while securely storing data.

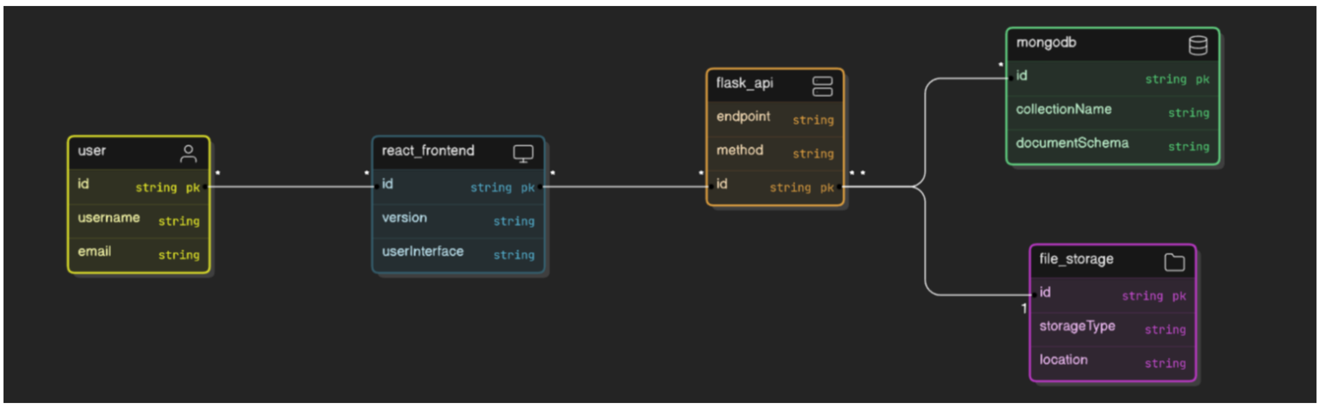

Fig 1: Design diagram

3. SYSTEM REQUIREMENTS

3.1 User Interface

i. **Interactive Dashboard:** Built with React.js, showing options for backup, restore, and data entry.
ii. **Real-time Updates:** Displays live database and backup statuses.
iii. **Error Alerts**: Notifies users of failed backups or restores.

3.2 Technologies and Tools Used

i. **Frontend**: React.js with Material-UI for styling.
ii. **Backend**: Flask (Python).
iii. **Database**: MongoDB.
iv. **Backup Tool**: Percona Backup for MongoDB.
v. **Containerization**: Docker.

4. NON-FUNCTIONAL REQUIREMENTS

4.1 Performance Requirements

i. **Real-time Backups:** Backup operations should execute within seconds for medium-sized datasets.
ii. **Scalability:** The system should handle increasing data sizes effectively**.**

4.2 Security Requirements

i. **Data Encryption:** Backup files are stored securely to prevent unauthorized access.
ii. **Role-Based Access Control:** Prevents unauthorized users from modifying backups.

4.3 Software Quality Attributes

i. **Usability:** GUI simplifies backup and restore operations.
ii. **Portability**: Docker ensures cross-platform deployment.

5 CODE IMPLEMENTATION

5.1 Backend (Flask API implementation)

```
@app.route('/backup', methods=['POST'])
def create_backup():
    # Backup logic here
    return jsonify({"message": "Backup created successfully"})
```

Fig 2: Flask route for creating a backup

```
@app.route('/restore', methods=['POST'])
def restore_backup():
    # Restore logic here
    return jsonify({"message": "Backup restored successfully"})
```

Fig 3: Flask route for restoring data

```python
@app.route('/data', methods=['POST'])
def enter_data():
    # Logic for adding data to the database
    return jsonify({"message": "Data added successfully"})

@app.route('/data', methods=['GET'])
def get_data():
    # Logic for retrieving data from the database
    return jsonify({"data": fetched_data})
```

Fig 4: Flask routes for data management

5.2 Frontend (React.js code)

```javascript
import React from 'react';

const BackupButton = () => {
    const handleBackup = () => {
        fetch('/backup', { method: 'POST' })
            .then(response => response.json())
            .then(data => alert(data.message))
            .catch(error => console.error('Error:', error));
    };

    return (
        <button onClick={handleBackup}>Create Backup</button>
    );
};

export default BackupButton;
```

Fig 5: React.js component for the Backup Button

```
import React from 'react';

const RestoreDataButton = () => {
    const handleRestore = () => {
        fetch('/restore', { method: 'POST' })
            .then(response => response.json())
            .then(data => alert(data.message))
            .catch(error => console.error('Error:', error));
    };

    return (
        <button onClick={handleRestore}>Restore Data</button>
    );
};

export default RestoreDataButton;
```

Figure 6: React.js component for the Restore Data Button

```
import React, { useState, useEffect } from 'react';

const TimestampDisplay = () => {
    const [timestamps, setTimestamps] = useState([]);

    useEffect(() => {
        fetch('/backup-timestamps')
            .then(response => response.json())
            .then(data => setTimestamps(data.timestamps))
            .catch(error => console.error('Error:', error));
    }, []);

    return (
        <div>
            <h3>Backup Timestamps</h3>
            <ul>
                {timestamps.map((timestamp, index) => (
                    <li key={index}>{timestamp}</li>
                ))}
            </ul>
        </div>
    );
};

export default TimestampDisplay;
```

Figure 7: React.js component for Timestamp Display

6 Conclusion

The **Ransomware Resilient Backup System** successfully demonstrates a reliable and scalable solution for protecting critical data from ransomware attacks. With an intuitive user interface and secure backend, this project offers a practical approach to ensuring data safety. Future enhancements include cloud integration and predictive failure analytics.