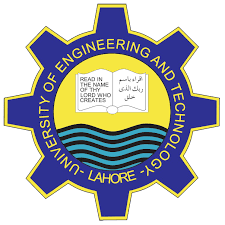
**Semester Project Report**

**ADVANCE DATABASE MANAGEMENT**

**SYSTEM LAB**



**SUBMITTED BY:**

Muhammad Saad Cheema (2023-CS-729)

Malik Muhammad Shakeel (2023-CS-736)

**SUBMITTED TO:**

MAAM RIDA SAEED

**PROJECT TITLE:**

**Dual Key Value Store Engine**

**DEPARTMENT OF COMPUTER SCIENCE**

**UNIVERSITY OF ENGINEERING AND TECHNOLOGY LAHORE, NEW CAMPUS, KSK**

**Table of Contents**

[1. Introduction: 2](#_Toc196625878)

[2. Project Overview: 2](#_Toc196625879)

[2.1 Objectives 2](#_Toc196625880)

[2.2 Technologies Used 2](#_Toc196625881)

[3. System Architecture: 3](#_Toc196625882)

[3.1 Component Diagram 3](#_Toc196625883)

[3.2 Key Components 3](#_Toc196625884)

[4. Implementation Details: 4](#_Toc196625885)

[4.1 B+ Tree Implementation 4](#_Toc196625886)

[4.2 LSM Tree Implementation 4](#_Toc196625887)

[4.3 Benchmarking System 4](#_Toc196625888)

[5. Performance Analysis: 5](#_Toc196625889)

[5.1 Expected Performance Characteristics 5](#_Toc196625890)

[5.2 Benchmark Results 5](#_Toc196625891)

[6. User Interface: 6](#_Toc196625892)

[6.1 API Endpoints 6](#_Toc196625893)

[7. Security Considerations: 6](#_Toc196625894)

[8. Future Enhancements: 7](#_Toc196625895)

[9. Conclusion: 7](#_Toc196625896)

**Dual Key Value Store Engine**

# **Introduction:**

This project implements *DualStore*, a Python-based key-value store featuring dual storage engines: a **B+ Tree** for fast reads and an **LSM Tree** for write-optimized workloads. Designed as a benchmarking platform, it evaluates **CRUD** operation performance under simulated disk I/O conditions (5ms latency) while integrating user authentication via MongoDB. The system quantifies fundamental architecture tradeoffs, providing actionable insights for storage system selection.

# **Project Overview:**

## **Objectives**

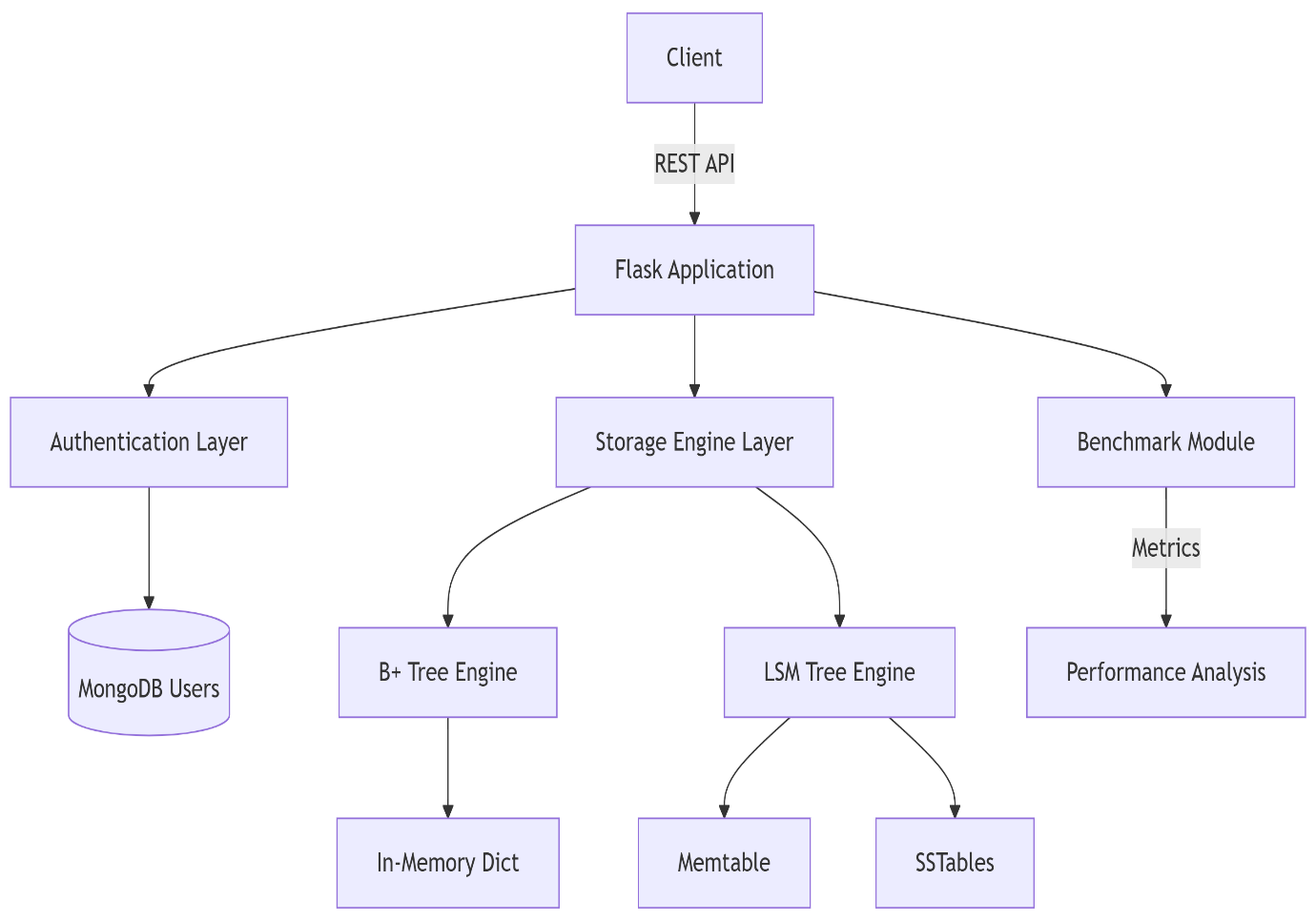
1. Implement **B+** Tree and **LSM** Tree data structures
2. Create a web interface for database operations
3. Develop **benchmarking** capabilities
4. Provide visual representations of engine structures
5. Implement user authentication

## **2.2 Technologies Used**

1. **Backend**: Python with Flask framework
2. **Database**: MongoDB for user authentication storage
3. **Security**: JWT for authentication, password hashing
4. **Data Structures**: Custom implementations of B+ Tree and LSM Tree

# **System Architecture:**

## **3.1 Component Diagram**



## **3.2 Key Components**

1. **Authentication System**
   * User signup/login with password hashing
   * JWT token-based session management
   * MongoDB-backed user storage
2. **Database Engines**
   * **B+ Tree Implementation**: Optimized for fast lookups and range queries
   * **LSM Tree Implementation**: Optimized for write-heavy workloads
3. **API Endpoints**
   * **CRUD** operations (insert, get, range query, delete)
   * **Engine switching**
   * **Benchmarking**
   * Image serving for visual representations

# **Implementation Details:**

## **4.1 B+ Tree Implementation**

* Degree parameter controls tree fan-out
* Supports:
  + O(log n) insertions
  + O(log n) lookups
  + O(log n + k) range queries (k = number of results)
  + O(log n) deletions

## **4.2 LSM Tree Implementation**

* Memtable for in-memory writes
* SSTables for disk persistence simulation
* Supports:
  + O(1) in-memory inserts
  + O(k) lookups (k = number of SSTables)
  + Range queries with merging across levels
  + Tombstone-based deletions

## **4.3 Benchmarking System**

* Tests operations at different scales (10, 50, 100 records)
* Measures:
  + Average insert time
  + Average get time
  + Average range query time
  + Average delete time
* Reports results in milliseconds for easy comparison

# **Performance Analysis:**

## **Expected Performance Characteristics**

|  |  |  |
| --- | --- | --- |
| **Operation** | **B+ Tree Performance** | **LSM Tree Performnce** |
| Insert | Moderate | Very Fast |
| Get | Very Fast | Moderate |
| Range Query | Very Fast | Moderate |
| Delete | Moderate | Fast |

## **5.2 Benchmark Results**

|  |  |
| --- | --- |
| **Parameter** | **Values** |
| Dataset Sizes | 10, 50, 100 records |
| Operations | Insert/Get/Range/Delete |
| Disk Simulation | 5ms latency penalty |

# **User Interface:**

## **API Endpoints**

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Method** | **Description** |
| Signup | POST | User registration |
| Login | POST | User authentication |
| Engine | POST | Switch database engine |
| Insert | POST | Insert key-value pair |
| Get/<key> | GET | Retrieve value by key |
| Range/<s>/<e> | GET | Range query |
| Delete/<key> | DELETE | Delete by key |
| Benchmark | GET | Run performance benchmarks |
| Engine-image/<eng> | GET | Get engine structure visualization |

# **Security Considerations:**

1. Password hashing with Werkzeug security
2. JWT tokens with expiration
3. CORS configuration for controlled API access
4. Input validation on all endpoints

# **Future Enhancements:**

1. **Persistent Storage**
   * Add actual disk-based storage for LSM Tree SSTables
   * Implement B+ Tree persistence
2. **Enhanced Benchmarking**

* Add concurrent operation testing
* Include more complex query patterns

1. **Additional Engines**
   * Implement other database structures (e.g., Hash Indexes, Fractal Trees)
2. **Improved Visualization**
   * Dynamic visualization of tree structures
   * Real-time operation animation

# **9. Conclusion:**

This project successfully implements and compares two fundamental database engine architectures through a web-accessible interface. The system demonstrates clear performance trade-offs between B+ Trees and LSM Trees, providing valuable insights for database selection in different application scenarios. The modular design allows for easy extension with additional engines and features.