**Course Code:** CSE 3513

**Course Name:** NoSQL Data Management

**Course Instructor**: Sunil Sahoo

**Lab Session**: 05

**Activity**: **Authentication in MongoDB** : In this lab, you will learn how to manage database users, roles, and passwords in MongoDB. Authentication is a key component of database security, ensuring that only authorized users can perform certain actions. We will work on the **Student** database to create, modify, and delete users, as well as update their passwords and roles.

# Learning Objectives (Los)

# LO1: Understand the concept and importance of authentication in MongoDB.

**LO2:** Create, update, and delete users with specific roles and privileges.

**LO3:** Apply role-based access control to manage database security.

**LO4:** Perform password management operations like change and update securely.

### **Authentication in MongoDB**

Authentication in MongoDB is the process of verifying the identity of users attempting to connect to the database. It works hand-in-hand with **authorization**, which determines what authenticated users are allowed to do. Using authentication ensures that only verified users can read, write, or modify data, protecting sensitive information from unauthorized access.

#### **1. Purpose and Power of Authentication**

* **User Identity Verification:** Ensures only legitimate users gain access to the database.
* **Role-Based Access Control (RBAC):** Assign specific permissions like read, readWrite, or dbAdmin to each user.
* **Security Compliance:** Meets requirements for data protection and audit logging.

#### **2. Key Advantages**

* **Data Security:** Protects sensitive information from unauthorized users.
* **Granular Permissions:** Control exactly what each user can do.
* **Operational Safety:** Reduces the risk of accidental or malicious changes.

#### **3. Common Authentication Operations**

**Switch to database**

use Student

**Create user with readWrite role**

db.createUser({

user: "studentUser",

pwd: "student123",

roles: [{ role: "readWrite", db: "Student" }]

})

**List all users**

db.getUsers()

**Get a specific user**

db.getUser("studentUser")

**Update user’s password and role**

db.updateUser("studentUser", {

pwd: "newstudent123",

roles: [{ role: "read", db: "Student" }]

})

**Change only the password**

db.changeUserPassword("studentUser", "superSecure123")

**Delete user**

db.dropUser("studentUser")

#### **4. Real-World Applications**

* **University System:** Only faculty can update grades; students can view but not modify them.
* **Corporate HR:** Payroll staff can edit salary data; other employees can view only their own pay slip.
* **E-Commerce Platform:** Admins manage inventory and prices; customers can only browse and purchase.

#### **5. Integration with Business Logic**

Authentication ensures that:

* Business rules are enforced at the database level.
* Sensitive data remains protected even if the application layer is compromised.
* Only authorized workflows can trigger database changes.

**Summary:**  
Authentication in MongoDB safeguards your data by ensuring that only verified and authorized users can access or modify it. Combined with role-based permissions, it provides fine-grained control over database operations, enabling secure and compliant data handling in real-world applications.

**CREATE Operation in MongoDB**

**Step 1: Switch to lab4 Database**

Use the use command followed by the name of the database you want to switch to (or create).

* + If "**lab4**" doesn't exist, MongoDB won't create it immediately, but it will switch the context to that database in the shell.

**Step 2: Create a user with readWrite role on the database**

db.createUser({

user: "studentUser",

pwd: "student123",

roles: [{ role: "readWrite", db: "Student" }]

});

**Step 3: List all users in Student DB.**

db.getUsers();

**Step 4: Get details of a specific user:**

db.getUser("studentUser");

**Step 5: Update the user’s password and role:**

db.updateUser("studentUser", {

pwd: "newstudent123",

roles: [{ role: "read", db: "Student" }]

});

**Step 6: Change the password:**

db.changeUserPassword("studentUser", "superSecure123");

**Step 7: Delete (Drop) the user**

Db.dropUser(“lab4”); // don’t try

**Lab challenge 5: University & Corporate Security Setup**

**Background**: A university and corporate firm share the same MongoDB cluster. The university stores **student academic records**, while the firm manages **employee payroll**. They require secure access control so each team only accesses its own data.**Requirements:**

**Requirements:**

1. Create a facultyUser with **readWrite** on the University database but no access to payroll.
2. Create an hrManager with **readWrite** on the Payroll database but no access to student data.
3. Test authentication by logging in with each user’s credentials and verifying access restrictions.
4. Change facultyUser’s password after first login for security.
5. Remove a user when they leave the organization.

**Real-Time Scenario Examples:**

* **Dean:** Logs in as facultyUser to update student grades without touching payroll records.
* **HR Manager:** Logs in as hrManager to approve salary revisions but cannot see student records.
* **Security Officer:** Enforces password updates and disables accounts of departed staff.

Happy Learning