# PROG8850 Assignment 3 — Database Automation Report

**Student Name:** Prem Chander Jebastian  
**Course:** PROG8850 - Database Automation

## Table of Contents

1. Project Overview
2. Technology Stack
3. System Architecture
4. Web Application Development
5. Database Setup
6. Docker Containerization
7. Selenium Integration and Testing
8. Security Considerations
9. Setup and Deployment Instructions
10. Code Implementation Highlights
11. Testing and Verification
12. References

## 1. Project Overview

This project implements a user registration web application using Flask integrated with a MySQL database backend. The system is containerized with Docker to ensure consistent deployment environments. Selenium WebDriver automates end-to-end testing by interacting with the frontend and verifying database persistence.

The project demonstrates key DevOps and database automation concepts, including:

* Full-stack web development
* Database schema design and automation
* Container orchestration with Docker Compose
* Automated UI and database testing
* Best practices in security, error handling, and logging

## 2. Technology Stack

* **Backend:** Python 3, Flask
* **Database:** MySQL 8.0
* **Frontend:** HTML5, Bootstrap 5, Font Awesome
* **Containerization:** Docker, Docker Compose
* **Testing:** Selenium WebDriver with ChromeDriver
* **Development Tools:** Git, Visual Studio Code
* **Operating System:** Cross-platform (tested on Windows 10, Linux)

## 3. System Architecture

The application consists of three main components:

| User Browser <----> | Flask Web App | <----> | MySQL Database |  
 (HTTP) (REST API) (SQL Queries)

All components are containerized using Docker:

* The Flask app runs inside a flask-web container
* MySQL runs inside a flask-mysql container with persistent volume
* Containers communicate over a Docker network to allow seamless connectivity

This architecture ensures environment consistency, isolation, and easy deployment.

## 4. Web Application Development

### 4.1 Features

* User registration form with username and password input
* Client-side HTML5 validation with clear placeholders and icons
* Password masking for security
* On successful registration, user redirected to a confirmation page showing assigned User ID
* Responsive UI using Bootstrap with gradient background and professional styling

### 4.2 UI Screenshots

* **Figure 1:** Login form  
  A screenshot of a computer

  AI-generated content may be incorrect.
* **Figure 2:** Login form filled with sample data  
  A screenshot of a computer

  AI-generated content may be incorrect.
* **Figure 3:** Registration success confirmation page  
  A screenshot of a computer

  AI-generated content may be incorrect.

## 5. Database Setup

### 5.1 Schema Design

CREATE TABLE IF NOT EXISTS users (  
 id INT AUTO\_INCREMENT PRIMARY KEY,  
 username VARCHAR(50) NOT NULL UNIQUE,  
 password VARCHAR(255) NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 INDEX idx\_username (username),  
 INDEX idx\_created\_at (created\_at)  
);

**Design rationale:**

* Unique usernames enforced at database level
* Indexed for query optimization
* Automatic timestamp for audit purposes

### 5.2 Initialization

Sample user data is inserted automatically on first run if the table is empty.

## 6. Docker Containerization

### 6.1 Docker Compose Configuration

* flask-mysql container running MySQL 8.0
* flask-web container running Flask app
* Volumes for persistent database storage
* Environment variables securely stored and injected
* Dependency ordering ensures DB ready before app starts

### 6.2 Docker Screenshot

* **Figure 4:** Docker Compose building and starting MySQL and Flask containers  
  A screenshot of a computer program

  AI-generated content may be incorrect.

**Benefits:**

* Easy setup across dev and prod
* Isolation of services
* Data persists despite container restarts

## 7. Selenium Integration and Testing

### 7.1 Automated UI Testing

* Tests fill registration form with unique timestamp-based usernames
* Submit the form and wait for success page
* Verify user presence in MySQL database via direct query

### 7.2 Test Architecture

* Headless Chrome WebDriver for CI compatibility
* Parameterized CSS selectors for reliable element identification
* Comprehensive error handling and reporting

### 7.3 Test Execution Screenshot

* **Figure 5:** Selenium tests executing successfully with database verification  
  A screenshot of a computer program

  AI-generated content may be incorrect.

7.4 Sample Test Code

def test\_registration\_form(self):  
 self.driver.get(self.app\_url)  
 timestamp = int(time.time())  
 test\_username = f"test\_user\_{timestamp}"  
 test\_password = "test\_password\_123"  
  
 username\_field = self.driver.find\_element(By.CSS\_SELECTOR, '[data-testid="username-input"]')  
 password\_field = self.driver.find\_element(By.CSS\_SELECTOR, '[data-testid="password-input"]')  
 submit\_button = self.driver.find\_element(By.CSS\_SELECTOR, '[data-testid="submit-button"]')  
  
 username\_field.send\_keys(test\_username)  
 password\_field.send\_keys(test\_password)  
 submit\_button.click()  
 time.sleep(3)  
 self.verify\_database(test\_username)

## 8. Security Considerations

* Database credentials handled via environment variables
* SQL queries use parameterized inputs to prevent injection
* Passwords currently stored in plaintext in sample data (for demo only); hashing with bcrypt or similar recommended for production
* Error messages avoid exposing sensitive information
* Selenium tests run in sandboxed headless mode

## 9. Setup and Deployment Instructions

### Prerequisites:

* Install Docker and Docker Compose
* Python 3 and pip for running Selenium tests
* Chrome browser and ChromeDriver compatible with installed Chrome version

### Steps:

# Clone repo  
git clone https://github.com/jpremchander/PROG8850-25S-Sec1-DatabaseAutomation.git  
cd Assignment3  
  
# Start containers  
docker-compose up -d  
  
# Access app at http://localhost:5000  
  
# Run Selenium tests  
python selenium\_test.py  
  
# View latest users in database  
docker-compose exec mysql mysql -u loginappuser -pLoginAppDbPwd@2025 -D loginapp -e "SELECT \* FROM users ORDER BY created\_at DESC LIMIT 5;"

## 10. Code Implementation Highlights

### Flask Route Example

@app.route('/register', methods=['POST'])  
def register():  
 username = request.form.get('username', '').strip()  
 password = request.form.get('password', '').strip()  
  
 if not username or not password:  
 flash('Username and password are required', 'error')  
 return redirect(url\_for('index'))  
  
 # Database insertion logic here...

### Database Connection

def get\_db\_connection():  
 try:  
 connection = mysql.connector.connect(DB\_CONFIG)  
 return connection  
 except Error as e:  
 logger.error(f"Error connecting to MySQL: {e}")  
 return None

## 11. Testing and Verification

### Manual Testing

* Registration form loads and validates inputs
* Success page displays user ID after submission
* Duplicate username entry shows error
* Database verified for inserted users

### Automated Testing

* Selenium test suite covers UI form submission
* Verifies database insertion directly
* Reports success or failure with clear logs

## 12. References

* [Flask Official Documentation](https://flask.palletsprojects.com/en/latest/)
* [MySQL Documentation](https://dev.mysql.com/doc/)
* [Docker Documentation](https://docs.docker.com/)
* [Selenium with Python](https://selenium-python.readthedocs.io/)
* [Bootstrap 5 Documentation](https://getbootstrap.com/docs/5.0/getting-started/introduction/)