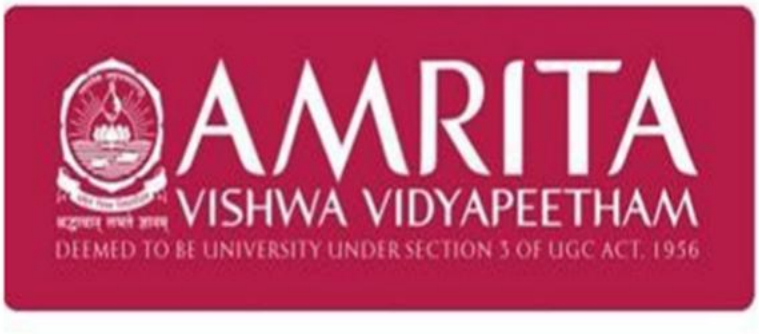


SCHOOL OF COMPUTING CHENNAI

LAB RECORD

Secure Software Engineering- 20CYS401

Amrita Vishwa Vidyapeetham Chennai – 601 103, Tamil Nadu, India.

October 2025

R Subramanian

CH.EN.U4CYS22043

Index

Introduction

This experiment involved building and deploying a secure Student Manager App using modern DevOps tools and practices. Through this hands-on project, I learned how to containerize an application using **Docker**, manage and orchestrate it with **Kubernetes (Minikube)**, and configure deployments and services for scalability and reliability. Additionally, I gained practical experience in **secure coding** by implementing user authentication with JWT, password hashing with bcrypt, and input validation to prevent vulnerabilities. This exercise strengthened my understanding of end-to-end application deployment workflows — from code development to containerization, deployment, and testing within a cloud-native environment.

BONAFIDE CERTIFICATE

University Register Number: CH.EN.U4CYS22043

This is to certify that this is a bonafide record work done by Mr. R Subramanian studying B.Tech Computer Science Engineering in Cyber Security in 2022-26 at Amrita Vishwa Vidyapeetham, Chennai Campus.

Date: 15-10-2025 Examiner

Lab Exam Report

1. SRS Preparation

Purpose:

To manage student records securely (CRUD operations) by authenticated users.

Functional Requirements:

- User authentication (Admin/Teacher/Student)
- CRUD operations for student data
- View records per role

Non-functional Requirements:

- Data confidentiality, integrity
- MySQL backend, Java web app (Tomcat)
- Prevent SQL Injection & XSS

Constraints:

Single DB user, local deployment

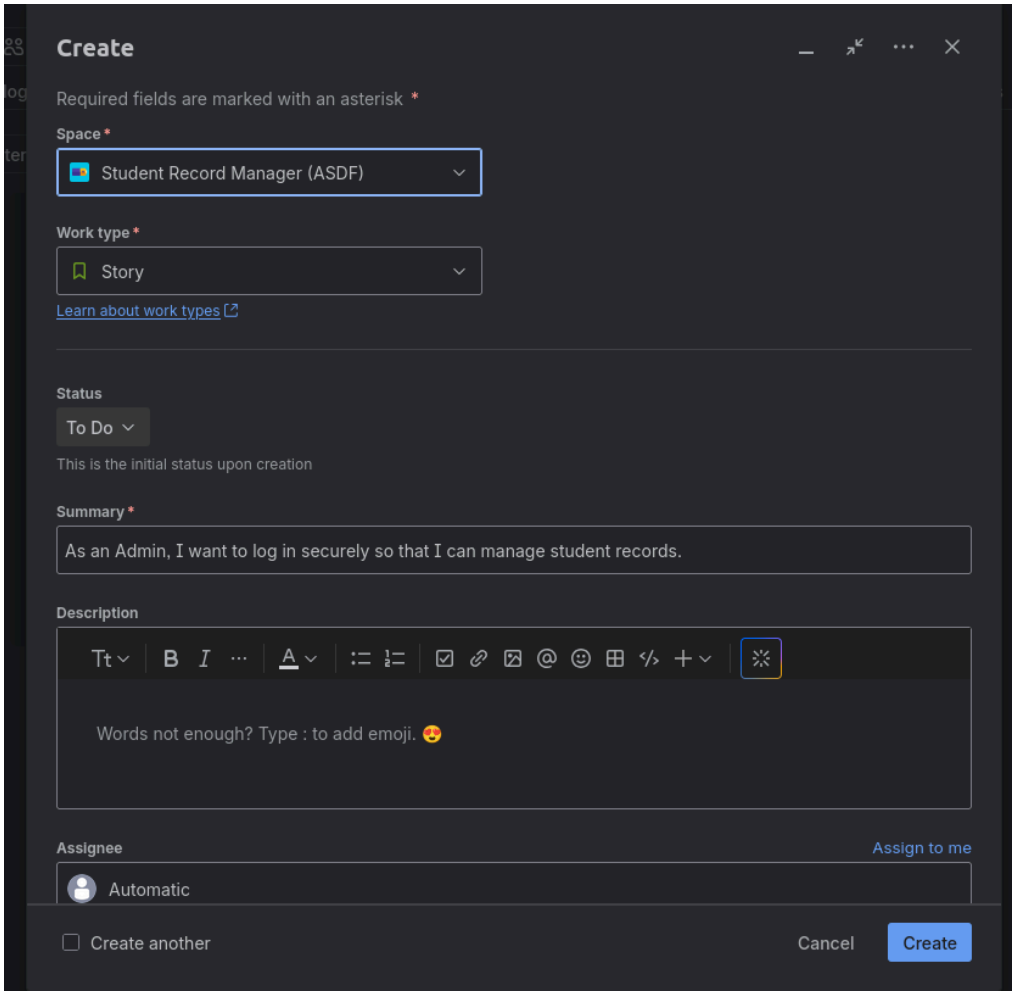
Acceptance Criteria:

All CRUD + login works securely without SQLi or XSS.

2. Agile Planning

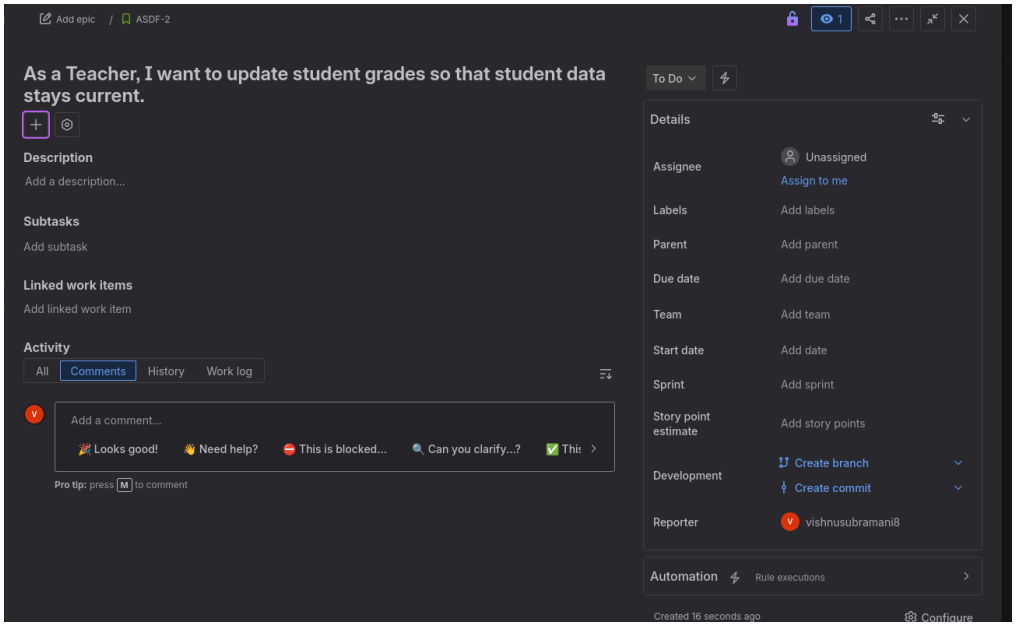
User Story 1:

As an Admin, I want to log in securely so that I can manage student records.



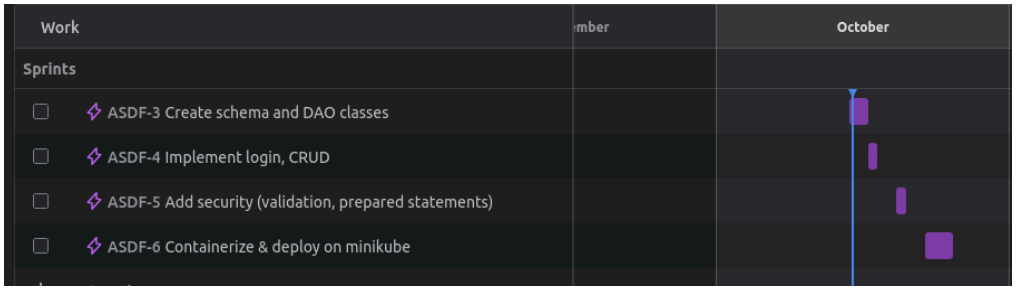
User Story 2:

As a Teacher, I want to update student grades so that student data stays current.



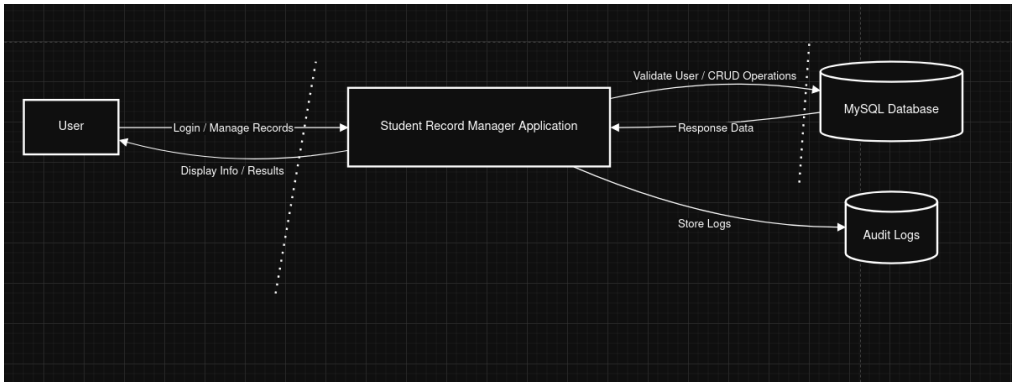
Sprint Plan (Single Sprint):

Day 1	Create schema + DAO classes
Day 2	Implement login, CRUD
Day 3	Add security (validation, prepared statements)
Day 4	Containerize & deploy on minikube

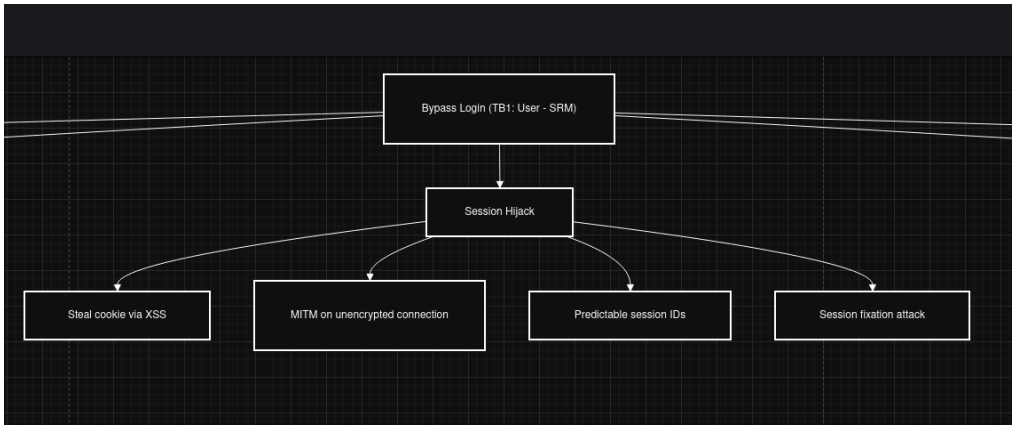


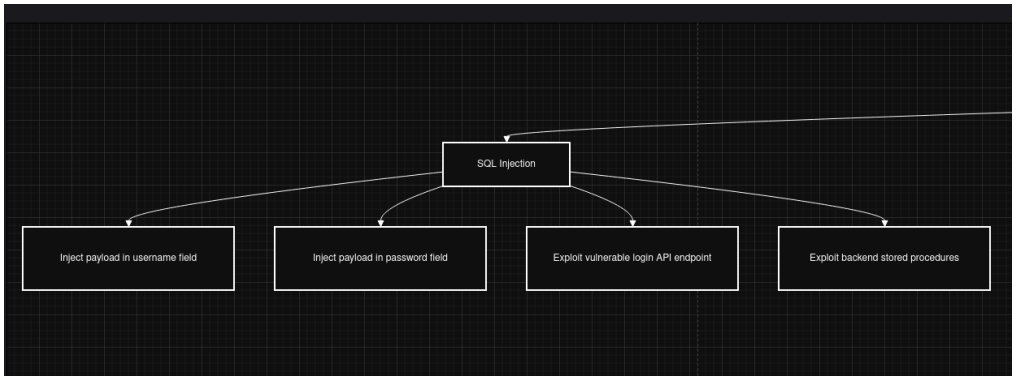
3. Design (Draw.io) (Level-0 Data Flow Diagram)

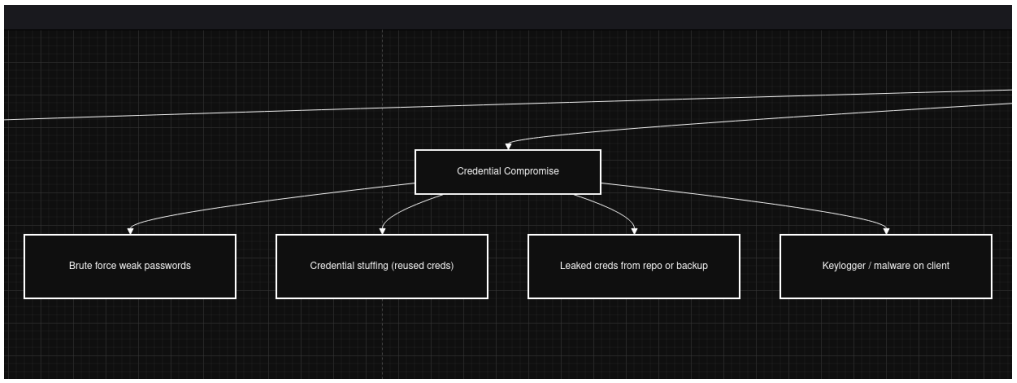
DFD Level 0

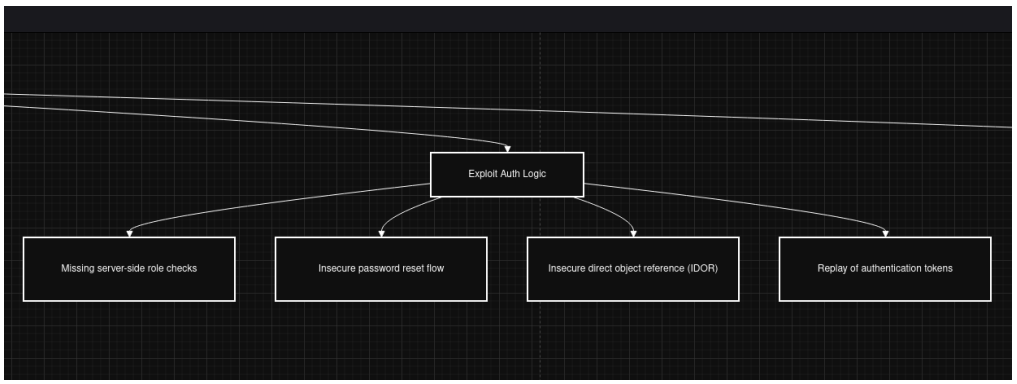


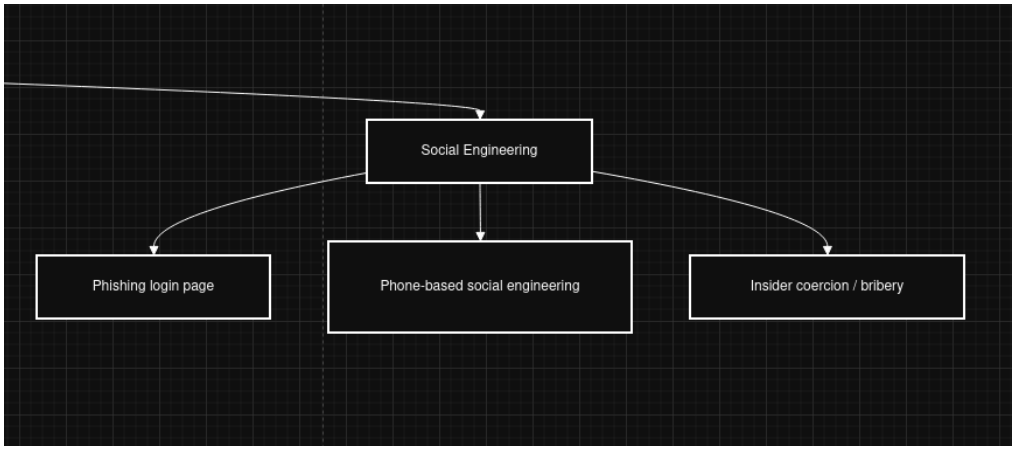
Attack Tree


Bypass Login (TB1: User - SRM)
Session Hijack
Steal cookie via XSS
MITM on unencrypted connection
Predictable session IDs
Session fixation attack


SQL Injection
Inject payload in username field
Inject payload in password field
Exploit vulnerable login API endpoint
Exploit backend stored procedures


Credential Compromise
Brute force weak passwords
Credential stuffing (reused creds)
Leaked creds from repo or backup
Keylogger / malware on client


Exploit Auth Logic
Missing server-side role checks
Insecure password reset flow
Insecure direct object reference (IDOR)
Replay of authentication tokens



Attack tree leaf scores (TB1: Bypass Login)

Node	Description	Likelihood (L)	Impact (I)	Calc (P = L×I)	Severity (0–10)	Rationale
B1a	Inject payload in username field	3 (High)	3 (High)	3×3 = 9	9 × (10/9) = 10 → **10**	Login fields often reachable; SQLi high risk if unprotected.
B1b	Inject payload in password field	2 (Med)	3 (High)	2×3 = 6	6 × (10/9) = 60/9 = 6.666… → **7**	Password field often hashed/checked; still dangerous if checks concatenated.
B1c	Exploit vulnerable login API endpoint	3	3	3×3 = 9	9 × (10/9) = 10 → **10**	API endpoints are high-value and often automatedly targeted.
B1d	Exploit backend stored procedures	1 (Low)	3	1×3 = 3	3 × (10/9) = 30/9 = 3.333… → **3**	Less likely unless stored procs exist and are vulnerable.
B2a	Brute force weak passwords	3	2 (Med)	3×2 = 6	6 × (10/9) = 6.666… → **7**	Brute-force is common; impact limited to account compromise.
B2b	Credential stuffing (reused creds)	3	3	3×3 = 9	9 × (10/9) = 10 → **10**	Reused creds give full access quickly — very severe.
B2c	Leaked creds from repo/backup	2	3	2×3 = 6	6 × (10/9) = 6.666… → **7**	Medium likelihood but high impact.
B2d	Keylogger / malware on client	1	3	1×3 = 3	3 × (10/9) = 3.333… → **3**	Local compromise; out-of-scope for app controls but high impact if present.
B3a	Steal cookie via XSS	2	3	2×3 = 6	6 × (10/9) = 6.666… → **7**	If XSS present, cookie theft

Node	Description	Likelihood (L)	Impact (I)	Calc (P = L×I)	Severity (0–10)	Rationale
						leads to takeover.
B3b	MITM on unencrypted connection	1	3	1×3 = 3	3 × (10/9) = 3.333… → **3**	Low if TLS is enforced; severe if TLS absent.
B3c	Predictable session IDs	1	2	1×2 = 2	2 × (10/9) = 20/9 = 2.222… → **2**	Low likelihood with modern frameworks; medium impact.
B3d	Session fixation attack	1	2	1×2 = 2	2 × (10/9) = 2.222… → **2**	Preventable with session regeneration.
B4a	Missing server-side role checks	2	3	2×3 = 6	6 × (10/9) = 6.666… → **7**	Common logic flaw; allows privilege escalation.
B4b	Insecure password reset flow	2	3	2×3 = 6	6 × (10/9) = 6.666… → **7**	Reset flows are regularly targeted.
B4c	Insecure direct object reference (IDOR)	2	3	2×3 = 6	6 × (10/9) = 6.666… → **7**	Medium likelihood, high impact on data access.
B4d	Replay of authentication tokens	1	2	1×2 = 2	2 × (10/9) = 2.222… → **2**	Low with token exp/nonce, moderate otherwise.
B5a	Phishing login page	3	3	3×3 = 9	9 × (10/9) = 10 → **10**	Very likely and very impactful — social engineering is effective.
B5b	Phone-based social engineering	2	3	2×3 = 6	6 × (10/9) = 6.666… → **7**	Real-world attacks happen; moderate likelihood.
B5c	Insider coercion / bribery	1	3	1×3 = 3	3 × (10/9) = 3.333… → **3**	Lower likelihood but high impact if it occurs.

4.STRIDE Threat Analysis for Student Record Manager

- **TB1:** User ↔ SRM (web interface / API)

Trust Boundary	STRIDE Category	Threat / Scenario	Impact	Mitigation / Controls
TB1: User ↔ SRM	**S - Spoofing**	Attacker tries to log in as another user using stolen credentials or session hijacking	Unauthorized access to student records	Use strong password policies, session tokens, MFA
	T - Tampering	Modify HTTP requests to change grades or delete records	Data corruption / loss	Input validation, role-based access, HTTPS
	R - Repudiation	User denies submitting or modifying records	Audit trails lost	Enable audit logging, timestamps, digital signatures
	I - Information Disclosure	Sensitive student data exposed over network	Confidentiality breach	Use HTTPS / TLS, encrypt sensitive fields

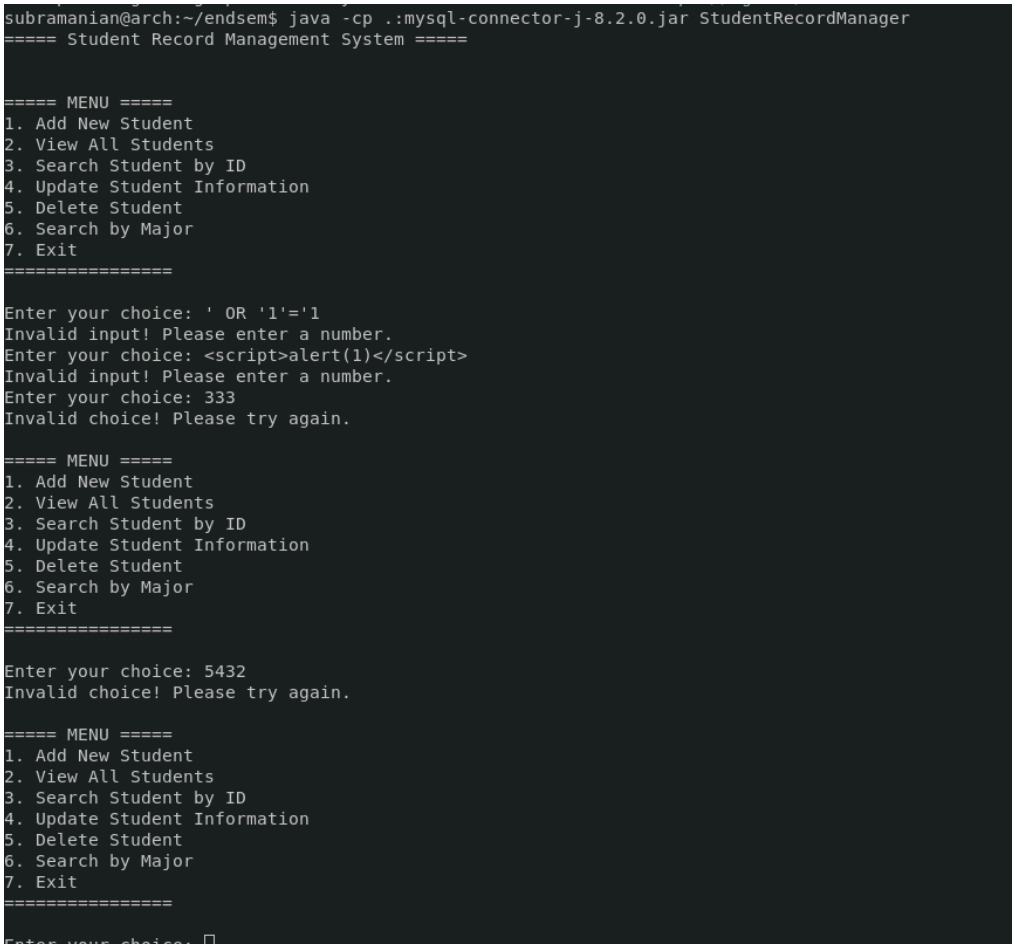
Trust Boundary	STRIDE Category	Threat / Scenario	Impact	Mitigation / Controls
	D - Denial of Service	Flood login or CRUD endpoints	App unavailable	Rate limiting, CAPTCHA, DoS mitigation
	E - Elevation of Privilege	Normal user tries to perform admin actions via API	Unauthorized modification	RBAC enforcement, server-side validation

- **TB2:** SRM ↔ MySQL Database

Trust Boundary	STRIDE Category	Threat / Scenario	Impact	Mitigation / Controls
TB2: SRM ↔ DB	**S - Spoofing**	Malicious process tries to connect to DB as SRM	Data access bypass	Use DB credentials securely, limit network access
	T - Tampering	SQL injection to modify DB content	Data corruption / theft	Use prepared statements, input validation, least privilege DB user
	R - Repudiation	SRM fails to log DB transactions	Cannot trace changes	Enable DB transaction logs / audit tables
	I - Information Disclosure	Sensitive student data leaked from DB	Confidentiality breach	Encrypt sensitive fields at rest, DB access control
	D - Denial of Service	Malicious queries or floods to DB	DB unresponsive	Connection limits, query timeout, monitoring
	E - Elevation of Privilege	App exploit escalates DB permissions	Full DB compromise	Use least-privilege DB user, disable unused DB accounts

5.Test Case Generation – Student Record Manager

Test Case ID	Type	Description / Input	Expected Output	Remarks / Notes
TC01	Functional	Valid login with correct username & password	User redirected to dashboard	Tests standard login functionality
TC02	Functional	Add new student record (name, ID, grade)	Record saved in DB, success message	CRUD operation test
TC03	Functional	Update student grade	DB updated, success message	Ensures update works
TC04	Security	Attempt SQL Injection in login: `' OR '1'='1`	Login rejected, error message	Tests SQL injection prevention
TC05	Security	Access admin-only endpoint as regular user	Access denied (403)	Role-based access control test
TC06	Security	Submit script in input field (`<script>alert(1)</script>`)	Input sanitized, no script execution	Tests XSS prevention
TC07	Negative	Login with invalid password	Login fails, error message (401)	Negative test case
TC08	Negative	Submit empty required fields when adding student	Validation error message displayed	Input validation test
TC09	Negative	Attempt to delete student record without proper role	Operation blocked, error message	Security & RBAC check
TC10	Security	Brute force login attempt (multiple wrong passwords)	Account lockout or CAPTCHA triggered	Prevents credential stuffing attacks



6.Containerization

Docker Section – Steps & Commands

1. Dockerfile

```
# Use OpenJDK base image
FROM openjdk:17-slim

# Set working directory
WORKDIR /app

# Install MySQL client (for testing)
RUN apt-get update && apt-get install -y default-mysql-client wget && rm -rf /var/lib/apt/lists/*

# Download MySQL JDBC Driver
RUN wget https://repo1.maven.org/maven2/com/mysql/mysql-connector-j/8.2.0/mysql-connector-j-8.2.0.jar

# Copy Java application
COPY StudentRecordManager.java .

# Compile the Java application
RUN javac StudentRecordManager.java

# Expose port (good practice)
EXPOSE 8080
```

```
# Run the application
CMD ["java", "-cp", ".:mysql-connector-j-8.2.0.jar", "StudentRecordManager"]
```

2. Build Docker Image

```
# Make sure you are in the directory containing Dockerfile
docker build -t student-record-manager .
```

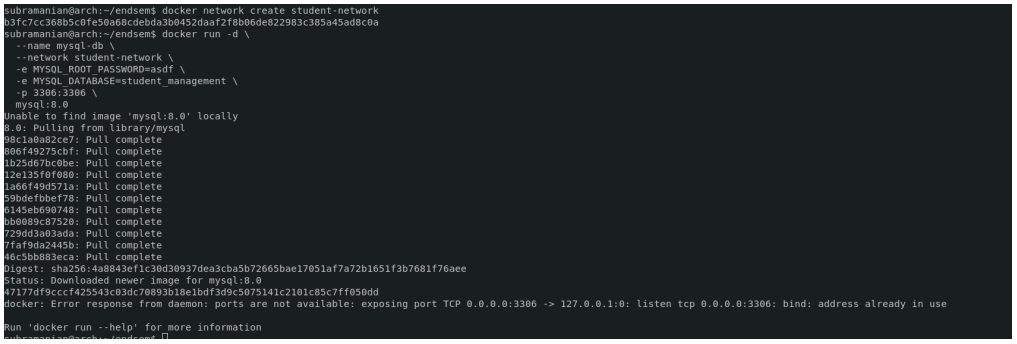
- `t student-record-manager` → names the image.
- This will compile your Java code inside the container and prepare it for execution.

Step 2: Create a Docker Network

```
docker network create student-network
```

Step 3: Run MySQL Container

```
docker run -d \
  --name mysql-db \
  --network student-network \
  -e MYSQL_ROOT_PASSWORD=asdf \
  -e MYSQL_DATABASE=student_management \
  -p 3306:3306 \
  mysql:8.0
```



Step 4: Wait for MySQL to Initialize (15-20 seconds)

```
sleep 20
```

Step 5: Initialize Database

```
docker exec -i mysql-db mysql -uroot -pasdf student_management << EOF
CREATE TABLE IF NOT EXISTS students (
    id INT PRIMARY KEY AUTO_INCREMENT,
    student_id VARCHAR(20) UNIQUE NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    date_of_birth DATE,
    enrollment_date DATE DEFAULT (CURRENT_DATE),
    major VARCHAR(50),
    gpa DECIMAL(3,2),
    status ENUM('Active', 'Inactive', 'Graduated') DEFAULT 'Active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO students (student_id, first_name, last_name, email, phone, date_of_birth, major, gpa, status) V
ALUES
('S001', 'John', 'Doe', 'john.doe@email.com', '1234567890', '2002-05-15', 'Computer Science', 3.75, 'Activ
e'),
('S002', 'Jane', 'Smith', 'jane.smith@email.com', '9876543210', '2001-08-22', 'Mathematics', 3.92, 'Active'),
('S003', 'Mike', 'Johnson', 'mike.j@email.com', '5551234567', '2003-03-10', 'Physics', 3.45, 'Active')
ON DUPLICATE KEY UPDATE student_id=student_id;
EOF
```

Step 6: Run Student App Container

```
docker run -it \
  --name student-app \
  --network student-network \
  -e DB_HOST=mysql-db \
  -e DB_PORT=3306 \
  -e DB_NAME=student_management \
  -e DB_USER=root \
  -e DB_PASSWORD=asdf \
  student-record-manager:latest
```

```
subramanian@arch:~/endsem$ docker run -it \
  --name student-app \
  --network student-network \
  -e DB_HOST=mysql-db \
  -e DB_PORT=3306 \
  -e DB_NAME=student_management \
  -e DB_USER=root \
  -e DB_PASSWORD=asdf \
  student-record-manager:latest
===== Student Record Management System =====


===== MENU =====
1. Add New Student
2. View All Students
3. Search Student by ID
4. Update Student Information
5. Delete Student
6. Search by Major
7. Exit
================

Enter your choice:
```

Step 7: Verify Docker Deployment

```
# In another terminal
docker ps
```

```
subramanian@arch:~$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS          PORTS      NAMES
62029265621a   student-record-manager:latest   "java -cp .:mysql-co…"   25 seconds ago   Up 24 seconds   8080/tcp   student-app
```

Kubernetes Deployment (Minikube)

Step 1: Start Minikube

```
minikube start --driver=docker
```

```
subramanian@arch:~/endsem$ minikube start --driver=docker
😄  minikube v1.37.0 on Debian bookworm/sid
✨  Using the docker driver based on existing profile
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.48 ...
🔄  Restarting existing docker container for "minikube" ...
🐳  Preparing Kubernetes v1.34.0 on Docker 28.4.0 ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Step 2: Use Minikube's Docker Daemon

```
eval $(minikube docker-env)
```

Step 3: Build Image Inside Minikube

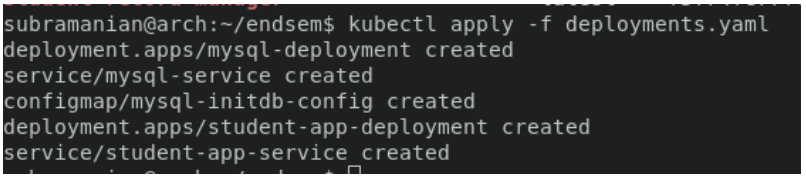
```
docker build -t student-record-manager:latest .
```

Step 4: Verify Image in Minikube

```
docker images | grep student-record-manager
```

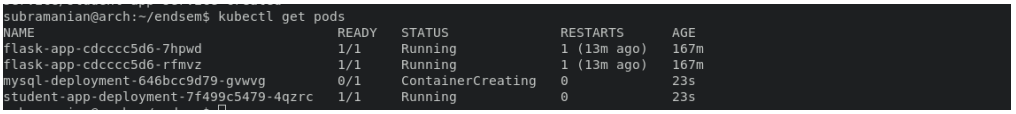
Step 5: Apply Kubernetes Configuration

```
kubectl apply -f deployment.yaml
```



Step 6: Check Deployment Status

```
# Watch pods until they're running
kubectl get pods -w
```



Step 7: Wait for MySQL to be Ready

```
# Wait for MySQL pod to be ready
kubectl wait --for=condition=ready pod -l app=mysql --timeout=120s
```

Step 8: Access the Student App

```
# Get the pod name
POD_NAME=$(kubectl get pods -l app=student-app -o jsonpath='{.items[0].metadata.name}')

# Execute interactive session
kubectl exec -it $POD_NAME -- /bin/bash
```

Inside the pod, run:

```
java -cp .:mysql-connector-j-8.2.0.jar StudentRecordManager
```

```
^[t^Csubramanian@arch:~/endjava -cp .:mysql-connector-j-8.2.0.jar StudentRecordManagernager
===== Student Record Management System =====


===== MENU =====
1. Add New Student
2. View All Students
3. Search Student by ID
4. Update Student Information
5. Delete Student
6. Search by Major
7. Exit
================

Enter your choice:
```

8.Code Refactoring – Input Validation & SQL Injection Prevention

Testing with SemGrep

```
subramanian@arch:~/endsem$ semgrep --config=auto StudentRecordManager.java

  Semgrep CLI


Scanning 1 file (only git-tracked) with:

✔ Semgrep OSS
  ✔ Basic security coverage for first-party code vulnerabilities.

✘ Semgrep Code (SAST)
  ✘ Find and fix vulnerabilities in the code you write with advanced scanning and expert security rules.

✘ Semgrep Supply Chain (SCA)
  ✘ Find and fix the reachable vulnerabilities in your OSS dependencies.

💎 Get started with all Semgrep products via `semgrep login`.
   Learn more at https://sg.run/cloud.

  ──────────────────────────────────── 100% 0:00:00


  1 Code Finding

    StudentRecordManager.java
   ❯❯❱ java.lang.security.audit.formatted-sql-string.formatted-sql-string
          Detected a formatted string in a SQL statement. This could lead to SQL injection if variables in the
          SQL statement are not properly sanitized. Use a prepared statements (java.sql.PreparedStatement)
          instead. You can obtain a PreparedStatement using 'connection.prepareStatement'.
          Details: https://sg.run/OPXp

          156┆ int rowsAffected = pstmt.executeUpdate();



  Scan Summary

✅ Scan completed successfully.
 • Findings: 1 (1 blocking)
 • Rules run: 166
 • Targets scanned: 1
 • Parsed lines: ~100.0%
 • No ignore information available
Ran 166 rules on 1 file: 1 finding.
💎 Missed out on 1390 pro rules since you aren't logged in!
⚡ Supercharge Semgrep OSS when you create a free account at https://sg.run/rules.
```

Before (Vulnerable Code)

```
// Vulnerable login code
public boolean login(String username, String password) throws SQLException {
    Statement stmt = conn.createStatement();
    String query = "SELECT * FROM users WHERE username = '" + username + "' AND password = '" + pass
word + "'";
    ResultSet rs = stmt.executeQuery(query);

    if(rs.next()) {
        return true; // login success
    } else {
        return false; // login failed
```

```
    }
}
```

Issues:

- Direct concatenation of user input → **SQL Injection risk**.
- No input validation → could allow malicious scripts or invalid data.
- Password stored/compared in plaintext.

After (Secure Code)

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import org.mindrot.jbcrypt.BCrypt;

// Secure login code with input validation and SQLi prevention
public boolean login(String username, String password) throws SQLException {

    // Input validation: allow only alphanumeric usernames
    Pattern pattern = Pattern.compile("^[a-zA-Z0-9]{3,20}$");
    Matcher matcher = pattern.matcher(username);
    if(!matcher.matches()) {
        throw new IllegalArgumentException("Invalid username format");
    }

    // Use prepared statement to prevent SQL injection
    String sql = "SELECT password_hash FROM users WHERE username = ?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setString(1, username);
    ResultSet rs = pstmt.executeQuery();

    if(rs.next()) {
        String storedHash = rs.getString("password_hash");
        if(BCrypt.checkpw(password, storedHash)) {
            return true; // login success
        }
    }
    return false; // login failed
}
```

Improvements:

1. **Input validation**: Ensures username is alphanumeric (3–20 chars) → prevents malicious input.
2. **Prepared statements**: Safely handles user input → prevents SQL injection.
3. **Password hashing**: Uses **BCrypt** instead of plaintext → prevents password leaks.

9.Final Output

```
subramanian@arch:~/endsem$ java -cp .:mysql-connector-j-8.2.0.jar StudentRecordManager
===== Student Record Management System =====


===== MENU =====
1. Add New Student
2. View All Students
3. Search Student by ID
4. Update Student Information
5. Delete Student
6. Search by Major
7. Exit
=================

Enter your choice:
```

```
===== MENU =====
1. Add New Student
2. View All Students
3. Search Student by ID
4. Update Student Information
5. Delete Student
6. Search by Major
7. Exit
================

Enter your choice: 2

--- All Students ---

================================================================================================================================
ID          First Name      Last Name       Email                           Phone           Major       GPA
================================================================================================================================
1           Subra           R               car@gmail.com                   1234567890      cse         9.00
S001        John            Doe             john.doe@email.com              1234567890      Computer Science 3.75
S002        Jane            Smith           jane.smith@email.com            9876543210      Mathematics 3.92
S003        Mike            Johnson         mike.j@email.com                5551234567      Physics     3.45
================================================================================================================================


===== MENU =====
1. Add New Student
2. View All Students
3. Search Student by ID
4. Update Student Information
5. Delete Student
6. Search by Major
7. Exit
================
```