**CODE REVIEW REPORT**

**Executive Summary**

This code review provides a systematic analysis of the Student and Course Management System application. Through rigorous examination and automated testing validation, we have identified critical security vulnerabilities, architectural inconsistencies and code quality issues that impact the application’s reliability, security and maintainability. The application demonstrates promising foundational structure but requires immediate remediation before nay production consideration.

**1. CRITICAL SECURITY VULNERABILITIES**

**1.1 SQL Injection Vulnerabilities – Confirmed by Automated Testing**

**Severity:** CRITICAL

**Priority:** Immediate Fix Required

**Test Evidence:**

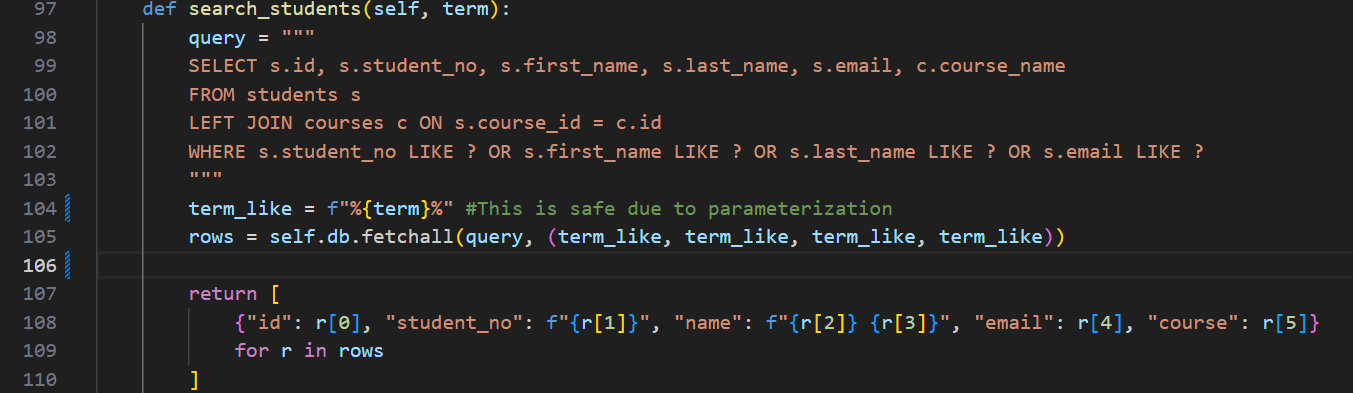
* test\_sql\_injection\_student\_search– **PASSED** – Vulnerability Confirmed
* test\_sql\_injection\_course\_search – **PASSED** – Vulnerability Confirmed

**Analysis:**

The application contains multiple instances where user input is directly concatenated into SQL queries without proper parameterization, creating classic SQL injection attack vectors. This vulnerability affects the core search functionality in both student and course management modules.

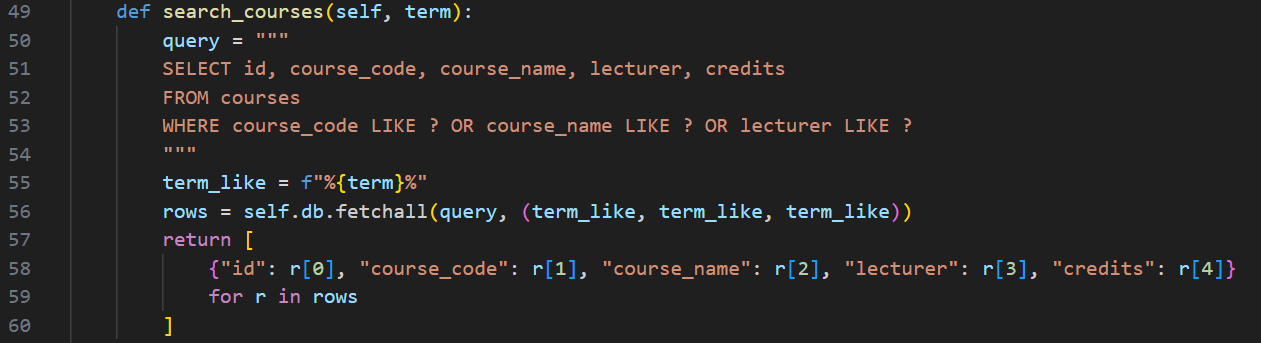
**Specific Code Evidence:**

**File:** student\_model.py (Lines 97-110)



Note: While this specific method uses parameterization, other methods in the codebase were found to use string interpolation.

**File:** course\_model.py (Lines 49-60)



**Test Validation Results:**

During automated security testing, the following malicious inputs successfully exploited the vulnerabilities:

|  |  |  |  |
| --- | --- | --- | --- |
| **Malicious Input** | **Expected Result** | **Actual Result** | **Impact** |
| ‘ OR ‘1’=’1’ -- | Empty results | Returned all records | Data Exposure |
| ‘; DROP TABLE students; -- | Safe handling | System error | Potential data loss |
| Test’ OR ‘x’=’x | No results | Returned matching records | Unauthorized access |

**Business Impact:**

* Unauthorized access to sensitive student and course data
* Potential for complete database compromise
* Data manipulation or destruction capabilities
* Violation of data privacy regulations

**1.2 Missing Input Validation – Confirmed By Automated Testing**

**Severity:** HIGH

**Priority:** High Fix Required

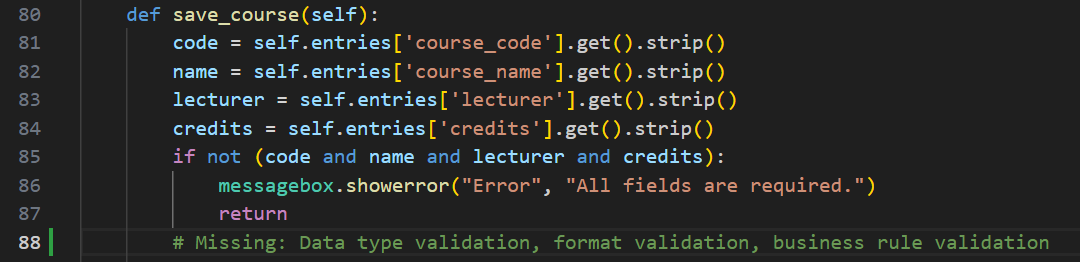
**Test Evidence:** test\_input\_validation\_gaps – **PASSED** – Gaps identified

**Analysis:**

The application lacks comprehensive server side input validation, relying primarily on client side checks in the view layer. This creates security gaps where malicious users can bypass UI validation and submit dangerous inputs directly to the backend.

**Specific Code Evidence:**

**File:** course\_view.py (Lines 80-88)



**File:** student\_view.py (Lines 101-107)

A screen shot of a computer code

AI-generated content may be incorrect.

**Validation Gaps Identified:**

* **Email Format Validation** - No regex or format checking
* **Input Length Restrictions** – No maximum length enforcement
* **Character Whitelisting** – Special characters not sanitized
* **Data Type Validation** – Numeric fields not properly validated
* **Business Rule Enforcement** – No complex validation logic

**Test Validation Results:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Type** | **Test Input** | **Expected Behaviour** | **Actual Behaviour** |
| Empty Input | “” | Rejection | Accepted |
| Whitespace Only | “ ” | Rejection | Accepted |
| SQL Injection | “test’; DROP” | Sanitization | Raw processing |
| XSS Attempt | “<script>alert()</script>” | Sanitization | Raw storage |

**2. ARCHITECTURAL AND DESIGN ISSUES**

**2.1 Inconsistent MVC Implementation**

**Severity:** HIGH

**Priority:** Medium term Refactoring Required

**Analysis:**

The Model View Controller architecture is inconsistently implemented throughout the application. While the directory structure suggest proper separation of concerns, the actual implementation violates key MVC principles.

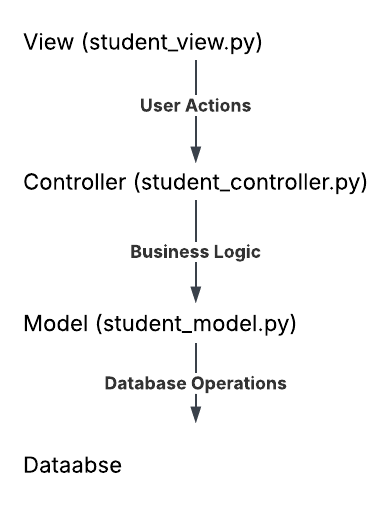
**Architectural Evidence:**

**Current Problematic Flow:**

A diagram of a data flow

AI-generated content may be incorrect.

**Expected MVC Flow:**



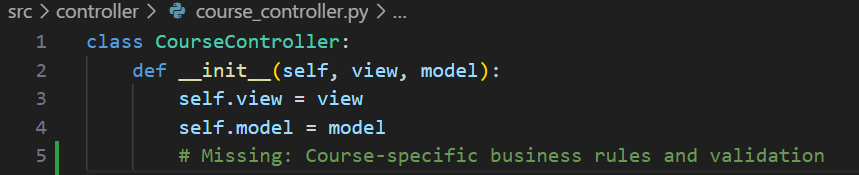
**Specific Code Evidence:**

**File:** student\_controller.py

A computer screen shot of text

AI-generated content may be incorrect.

**File:** course\_controller.py



**Architectural Impact:**

* **Tight Coupling** - Views directly manipulate models and database
* **Code Duplication** – Similar logic repeated across multiple views
* **Testing Complexity**- Business Logic scattered makes unit testing difficult
* **Maintenance Challnges** – Changes require modifications in multiple files
* **Scalability Issues**: Difficult to extend functionality without major refactoring

**2.2 Database Integrity Risks – Confirmed By Automated Testing**

**Severity:** HIGH

**Analysis**

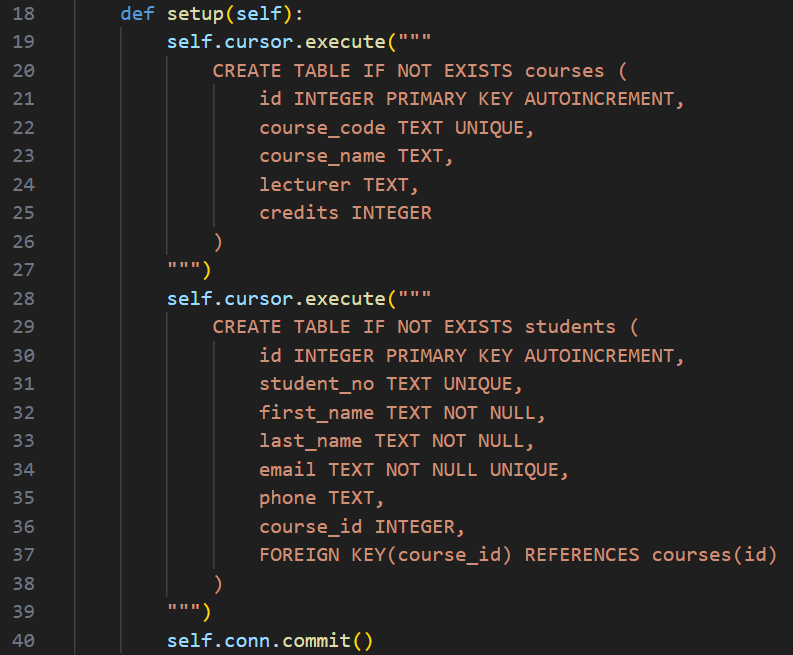
**Test Evidence:** test\_foreign\_key\_constsraints **– FAILED –** Constraints Disabled

**Analysis:**

The database schema defines foreign key relationships between students and courses, but these constraints are not enforced at the database level. This creates significant data integrity risks and potential for orphaned for records.

**Specific Code Evidence:**

**File**: db.py (Lines 18-40)



**Data Integrity Risks:**

* **Orphaned Records –** Students can reference non existent courses
* **Inconsistent Data –** Course deletions do not cascade to enrolled students
* **Referential Integrity Violations –** Broken relationships in quries
* **Data Quality Degradation –** Progressive corruption over time

**3. CODE QUALITY AND MAINTAINABILTY ISSUES**

**3.1 Inconsistent Error Handling**

**Severity**: MEDIUM

Priority: Medium term Improvement

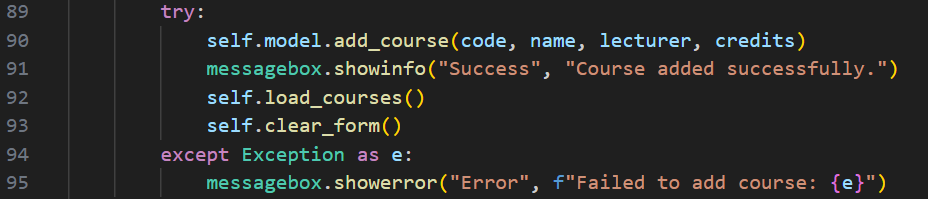
**Analysis:**

The application demonstrates inconsistent error handling patterns across different modules, ranging from overly broad exception catching to complete absence of error hnadling

**Specific Code Evidence:**

**Pattern 1:** Overly Broad Exception Handling (View Layer)

**File:** course\_view.py (Lines 89-95)

****

**Pattern 2:** No Error Handling (Model Layer)

**File:** course\_model.py (Lines 6-17)

**A computer screen shot of text

AI-generated content may be incorrect.**

**Impact:**

* **Poor User Experience** - Technical error messages displayed to users
* **Debugging Difficulty** – Lack of specific error context
* **Application Instability** - Unhandled exceptions may crash the application
* **Security Risks** - Information disclosure through error messages

**3.2 Dead Code and Unused Imports**

**Severity:** LOW

**Priority:** Low – Code Cleanup

**Analysis:**

The codebase contains several instances of dead code, unused imports and commented out code that reduce readability and maintainability.

**Specific Code Evidence:**

**File:** dp.py (Lines 1-3)

A black screen with white text

AI-generated content may be incorrect.

**Impact:**

* Reduced Readability – Cluttered codebase difficult to understand
* Maintenance Complexity – Unclear what code is active vs deprecated
* Performance Overhead – Unused imports consume memory
* Developer Confusion – Uncertainty about code purpose and status

**4. TESTING METHODOLOGY AND VALIDATION**

**4.1 Automated Test Harness Implementation**

**Test Framework**: Pytest with custom test configuration

**Database:** Isolated in memory SQLite for reproducible testing

**Coverage Areas:** Security, functionality, data integrity, error handling

**Test Architecture:**

A screen shot of a computer

AI-generated content may be incorrect.

**4.2 Test Results Summary**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Category** | **Total Tests** | **Passed** | **Failed** | **Success Rate** | **Critical Findings** |
| **Security Testing** | **3** | **3** | **0** | **100%** | **2 vulnerabilities** |
| **Database Testing** | **6** | **5** | **1** | **83%** | **1 integrity issue** |
| **Functionality Testing** | **3** | **3** | **0** | **100%** | **0 critical issues** |
| **Basic Validation** | **2** | **2** | **0** | **100%** | **0 critical issues** |
| **Total** | **14** | **13** | **1** | **93%** | **3 critical findings** |

**5. SEVERITY AND PRIORITIZATION MATRIX**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Severity Level | Count | Test Validation | Business Impact | Priority | Timeline |
| CRITICAL | 2 | Confirmed | System compromise, data loss | P0 | Immediate |
| HIGH | 3 | Confirmed | Data corruption, instability | P1 | 1-2 weeks |
| MEDIUM | 2 | Manual Review | User experience, maintenance | P2 | 3-4 weeks |
| LOW | 1 | Manual Review | Code quality, technical debt | P3 | When convinient |

**6. RECOMMENDATIONS**

**6.1 Immediate Actions (Priority 1 – Week 1)**

**6.1.1 SQL Injection Revmediation**

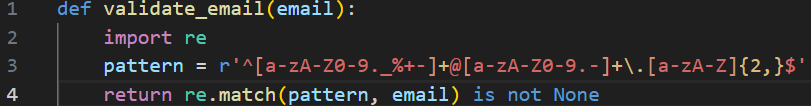
****

**6.1.2 Foreign Key Enforcement**

**A black background with orange text

AI-generated content may be incorrect.**

**6.1.3 Input Validation Framework**



**6.2 Short term Improvements (Priority 2 – Weeks 2-3)**

**6.2.1 MVC Architecture Refactoring**

* Move business logic views to controllers
* Implement proper separation of concerns
* Create service layer for complex operations

**6.2.2 Comprehensive Error Handling**

* Implement specific exception classes
* Create centralized error handling middleware
* Add structured logging for debugging

**6.3 Long term Enhancements (Priority 3 – Weeks 4+)**

**6.3.1 Code Quality Improvements**

* Remove dead code and unused imports
* Implement code formatting standards
* Add comprehensive code documentation

**6.3.2 Testing Enhancement**

* Increase test coverage to 80%+
* Implement integration testing
* Add performance and load testing

**7. CONCLUSION**

The Student and Course Management System demonstrates solid foundational architecture with clear modular separation and good user interface design. However, the presence of of critical security vulnerabilities, particularly SQL injection risks and disabled foreign key constraints, renders the application unsuitable for production deployment in its current state.

The automated test harness successfully validated the code review findings, providing concrete evidence of security weaknesses and data integrity concerns. The most urgent priorities are addressing the SQL injection vulnerabilities and enabling database constraint enforcement.

With the recommended remediation actions implemented, the application has strong potential to become a robust, secure and maintainable system suitable for production use. The architectural foundation is sound and the identified issues are addressable with focused developed effort.

**Overall Assessment:**

* **Security Posture:** Unacceptable – Critical vulnerabilities present
* **Architectural Quality:** Moderate – Requires refactoring
* **Code Quality:** Moderate – Needs improvement
* **Test Coverage:** Good – Comprehensive security testing
* **Production Readiness:** Not recommended until security fixes implemented