Testing Strategy (Manual & Automated)

Perform smoke testing, document defects

1. User Registration and Login

    1. Navigate to the registration page.
    2. Enter valid user details and submit.
    3. Verify that a confirmation email is received (if email verification is implemented).
    4. Navigate to the login page.
    5. Enter registered credentials and submit.

    Expected Result: User is registered and logged in without errors.

2. Homework Upload and Submission

    1. Log in as a student.
    2. Navigate to the homework submission section.
    3. Select and upload a file.
    4. Submit the homework.

    Expected Result: Homework is uploaded and submitted successfully.

3. Grade Viewing

    1. Log in as a student.
    2. Navigate to the grades section.
    3. View grades for various assignments.

    Expected Result: Grades are displayed correctly for each assignment.

4. Communication Between Students and Faculty

1. Log in as a student.
2. Navigate to the messaging or communication section.
3. Send a message to a faculty member.
4. Log in as the faculty member.
5. Check and read the received message.

Expected Result: Message is sent and received successfully.

5. Course Enrollment

1. Log in as a student.
2. Navigate to the course catalog.
3. Select a course and enroll.

Expected Result: Student is successfully enrolled in the course.

These scenarios cover the basic functionalities of the platform and should be executed to ensure that the core features are working as intended.

Conclusion: The following defects were found:

1. The password must contain exactly 6 characters
2. The date of news from the faculty is not updated
3. Student/Teacher status is displayed incorrectly

Develop automated tests (API testing)

Resources: API Tests in the `test2/apiTests` Folder

The folder contains **four main test files** focused on different non-functional testing aspects:

1. **LoadTests.cs** – Simulates multiple simultaneous users to test how the system handles traffic under stress. It evaluates response times and server behavior under heavy loads.

Test Coverage:

- 100 Parallel GETsSimulates 100 users fetching assignments simultaneously.
  50 Parallel POSTs: Simulates 50 users creating new assignments at the same time
- 500 Sequential GETs: Tests reliability and consistency under heavy sequential load.
  Bulk Create Then GET: Create 20 assignments, then fetch to confirm they exist.
  20 Parallel PUTs: Simulates simultaneous updates to 20 different assignments.
  Delete Multiple: Deletes assignments with IDs 1–10.
  Create–Read–Delete Cycle: Tests basic CRUD under load.

2. **PerformanceTests.cs** – Measures the speed and responsiveness of key API endpoints to ensure they meet performance benchmarks.

   Test Coverage:

- GetAssignments_ResponseTimeUnder200ms
  Checks that fetching the list of assignments responds within 200ms.
- PostAssignment_ResponseTimeUnder300ms
  Measures how quickly a new assignment can be created, with a threshold of 300ms.
  PutAssignment_ResponseTimeUnder300ms
  Ensures updating an existing assignment takes less than 300ms.
- DeleteAssignment_ResponseTimeUnder250ms
  Verifies the speed of deleting an assignment, with a cap at 250ms.
- GetAssignmentById_ResponseTimeUnder200ms
  Measures how quickly a specific assignment can be fetched.
- GetAssignments_AfterCreationIsFast
  Tests if system performance stays fast even after data is created (under 250ms).
- MultipleSequentialGets_Under500msTotal
  Checks the cumulative time of 5 consecutive GET requests stays under 500ms — testing sustained performance.

3. **ReliabilityTests.cs** – Assesses how consistently the system performs under repetitive or prolonged use, ensuring stability over time.

   Test Coverage:

- RepeatGet100Times_ShouldSucceed: Sends 100 consecutive GET requests to verify system stability under frequent reads.
- CreateThenGet_DeleteThenGet: Performs a create → read → delete → read sequence to ensure the system behaves consistently after state changes.
- LongRunningGetRequests: Executes 1,000 GET requests to test long-term reliability and memory stability.
- BulkCreateAssignments: Creates 50 assignments to check how the system handles a burst of write operations.
- ConsistentStatusCodesInLoop: Ensures every GET request in a loop returns a consistent 200 OK status.
- ParallelCreateAndDelete: Creates and deletes an assignment back-to-back to verify that transitions between states don't break behavior.
- RestartGetAfterFailure: Simulates a retry mechanism by checking if a failed GET can succeed after a short wait.

4. **SecurityTests.cs** – Tests for common vulnerabilities such as unauthorized access and insecure endpoints.

   Test Coverage:

- GetWithoutToken_ShouldFail: Ensures the API rejects unauthenticated requests (expecting 401 Unauthorized).
- PostWithInvalidToken_ShouldFail: Checks that requests with fake tokens are denied.
- SqlInjection_ShouldBeBlocked: Submits SQL injection payload and ensures the system does not crash (no 500 error).
- XssInjection_ShouldNotBeRendered: Sends an XSS payload and verifies the system handles it gracefully.
- AccessWithStudentRole_ToDelete_ShouldBeForbidden: Confirms that a user with limited privileges (student) cannot perform restricted actions like delete (expects 403 Forbidden).
- AccessToOtherUserData_ShouldBeRestricted: Tests that users cannot access resources belonging to others (403 expected).
- TamperedToken_ShouldBeRejected: Validates that a manipulated or corrupted JWT is not accepted by the server.

Conclusion:

```
Test summary: total: 28, failed: 7, succeeded: 21, skipped: 0, duration: 2.9s
Build failed with 7 error(s) and 8 warning(s) in 4.7s
```

```
  LoadTest_100ParallelGets (105ms): Error Message:   Expected: True
    But was:  False
```

```
LoadTest_500GetsInLoop (4ms): Error Message:   Expected: True
  But was:  False
```

Several automated API tests are failing due to a combination of performance, data consistency, and concurrency issues. Under high load or parallel execution, the backend struggles with:

- Unreliable data handling (e.g., creating or updating records in bulk).
- Missing or invalid test preconditions (e.g., updating or deleting non-existent records).
- Lack of proper response handling or synchronization (e.g., requests sent too fast).
- Performance bottlenecks during stress tests or large sequential loops.
- Server or database limitations under concurrent requests.

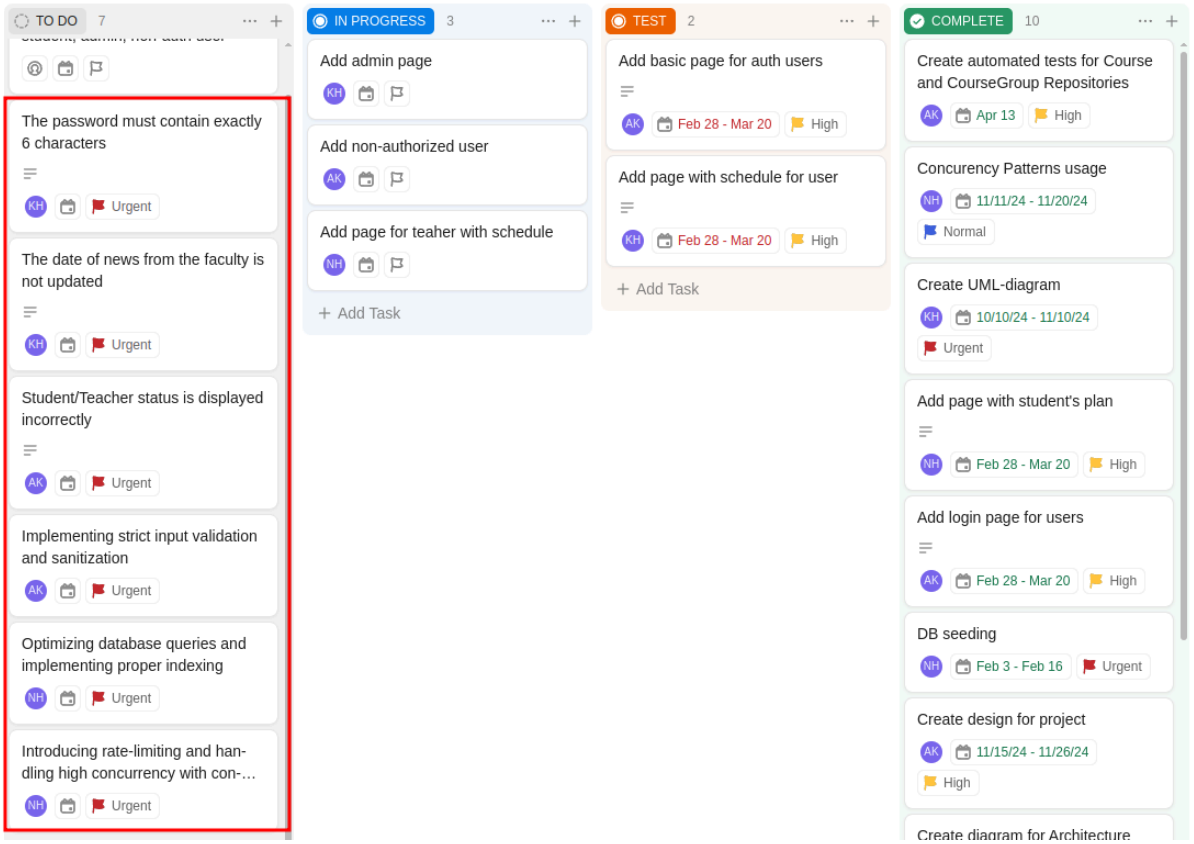
Areas for Improvement Because Tests Do Not Yield Expected Results:

- The database cannot handle multiple concurrent reads/writes, leading to timeouts or crashes.
- Lack of connection pooling or improper resource management can lead to server overload.
- API rate limiting or throttling is not implemented, which could cause resource exhaustion or downtime during peak traffic.
- Server or database limitations under concurrent requests.

Next Steps:
To address these potential failure points:

- Implementing strict input validation and sanitization.
- Optimizing database queries and implementing proper indexing.
- Introducing rate-limiting and handling high concurrency with connection pooling.

Tasks with the bug tag have been added to the board:



Task 86c35vww1 Ask AI

The password must contain exactly 6 characters

Ask Brain to create a summary · generate subtasks · find similar tasks · or ask about this task

Status	TO DO	Assignees	KH
Dates	Start → Due	Priority	Urgent
Time Estimate	Empty	Track Time	Add time
Tags	bug	Relationships	Empty

This task involves addressing a bug where the password requirement is not functioning as expected. The password must be exactly 6 characters long. Please investigate the issue, identify the cause, and implement a solution to ensure the system enforces this rule correctly.